FPGA-Vision

..... Accelerate Your Vision

Mustafa Gobulukoglu (A12045952)

Vipulkumar Lakhani (A10464467)

CSE 237D

Prof. Ryan Kastner

University of California, San Diego

June, 2019

Abstract:

With the emergence of machine vision, real time image processing applications, such as circuit inspection, template matching, and facial recognition, have become increasingly prominent. These applications demand low latency and high bandwidth to support real time processing and analysis of image data. FPGAs exhibit lower latency and power consumption along with better bandwidth, connectivity, and parallelism than instruction set architectures. Hence, FPGAs could potentially serve as good platforms for machine vision applications.

We have implemented several common machine vision techniques on both an instruction set architecture and an FPGA in an effort to benchmark and compare the performance of both platforms on such applications. For our benchmarks, we have used a Xilinx PYNQ FPGA and a 2018 MacBook Pro x86 ISA. In our experiments, heuristics on latency and accuracy were gathered from benchmarks for various template matching functions and a real time facial recognition application on both platforms using various image datasets. The FPGA exhibited 10x better performance than the general purpose CPU in real time facial detection.

FPGA, Instruction Set Architecture, Machine Vision, Facial Recognition, Template Matching, Real-Time Computing

Introduction:

Recently, machine vision has been gaining popularity in various sectors of the industry such as health care, defence, and autonomous driving. Using various image processing and learning algorithms, machine vision helps to interpret and understand the visual world. Template matching is an algorithm in which a machine accurately identifies specified objects from a source image. Facial detection is a machine vision problem which attempts to accurately identify and track the human faces from live video or source images. But most of these machine vision algorithms are computationally intensive and require large amounts of bandwidth and power to process on a CPU.

Now consider the field programmable gate array (FPGA). FPGAs are massively parallel in nature, allowing for image processing operations to be better optimized. FPGAs can offer much better bandwidth, less latency and less power consumption relative to general purpose CPUs for specific tasks that are suitable for acceleration; however, the advantages of an FPGA

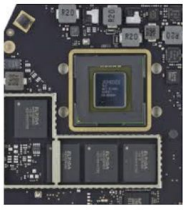
for image processing are dependent on the specific algorithms applied, latency or jitter requirements, I/O synchronization, and power utilization.

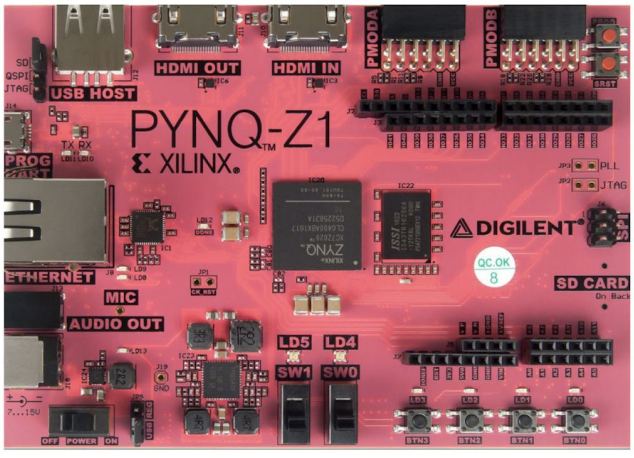
Often using an architecture featuring both an FPGA and a CPU presents the best of both worlds and provides a competitive advantage in terms of performance, cost, and reliability. Unfortunately, one of the biggest challenges in implementing an FPGA-based vision system is overcoming the programming complexity of FPGAs. The development of vision algorithms are, by its very nature, an iterative and complex process. We know up front that we would have to try a few approaches with any task. Most of the time, the question is not which approach works, but which approach works best, and "best" is different from application to application. For some applications, speed is paramount. In others, it's accuracy. At a minimum, we need to try a few different approaches to find the best one for any specific application. To maximize productivity, we need to get immediate feedback and benchmarking information on machine vision algorithms regardless of the processing platform we are using.

So, for this FPGA vision project, to assess the efficacy of FPGAs in accelerating machine vision applications, we have implemented various machine vision techniques on both general purpose ISA and FPGA platforms to compare their relative performances. The techniques we have chosen to benchmark are template matching and real time facial detection. Both techniques are implemented logically in an identical fashion on both platforms.

Methodology:

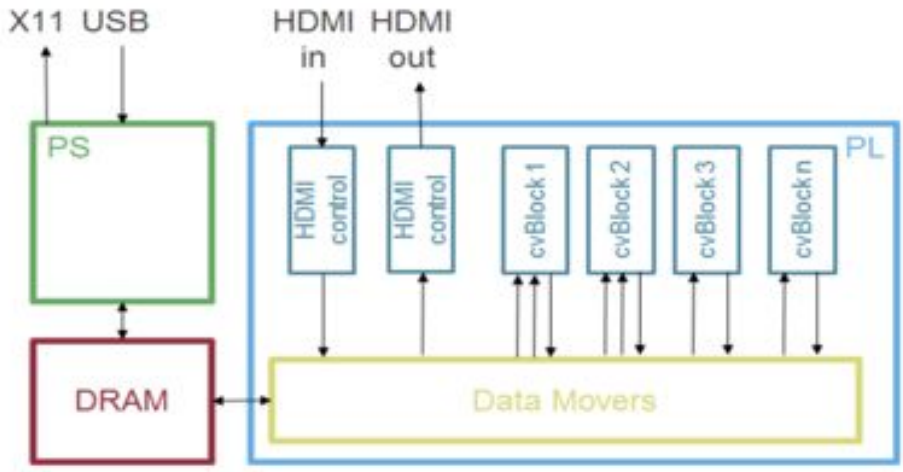
Hardware Components



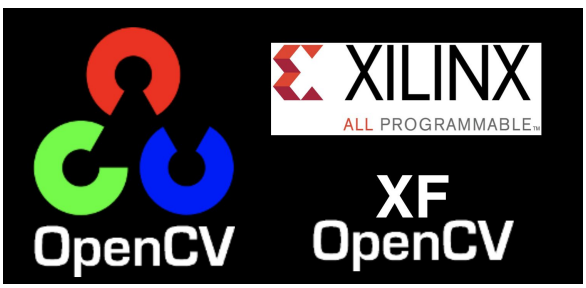


For our hardware components, in terms of our general purpose processors, we have employed the use of an Intel Core i7-8570H CPU, AMD Radeon Pro 555X GPU, and a dual-core Cortex-A9 ARM CPU.
The FPGA we have chosen to use is an Artix 7 family PL FPGA. Both the ARM core and FPGA are part of a PYNQ Z1 pSoC or programmable system on chip.

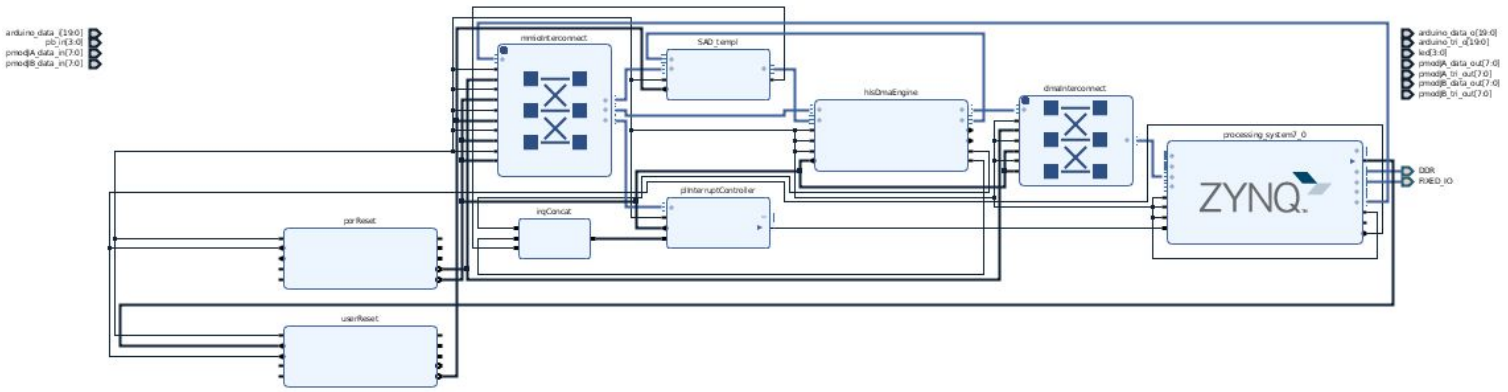
Furthermore, we have employed the use of two cameras, namely a MOKOSE 1080P HDMI Camera and a Logitech C170 for use in our machine vision algorithms.The motivation behind using two separate cameras is the peripheral interfacing supported by the PYNQ board. The PYNQ supports USB I/O for only the general purpose processor on board, while providing only HDMI I/O for the programmable logic of the FPGA. Hence, to minimize data transfer and latency, the HDMI camera was used for interfacing with the FPGA and the USB webcam was used for interfacing with the on board CPU.



The software components and design tools that were required for this project were VivadoHLS, Vivado, openCV, and xfopenCV. Both VivadoHLS and Vivado are tools provided by Xilinx, used in the development of FPGA logic. OpenCV is an open-source computer vision library licensed under BSD, which provides various functions for use in real time computer vision techniques. xfopenCV is a hardware synthesizable analog of the otherwise unsythnthsizable openCV library.
Software Components



FPGA Hardware Design and Streaming Protocol:



AMBA® 4 AXI4-Stream Protocol

An AXI Stream interface was deemed most appropriate for interfacing with our HLS core since image data is relatively large and the stream allows for high throughput. However, because of this, we had to employ the use of a DMA engine to convert stream data to memory mapped data to store in the PYNQ DRAM.

Template Matching on ISA Platforms:

We have implemented template matching algorithms using OpenCV in C++ for the below seven distinct similarity matching methods. Then, we have compiled and tested our code on ISA platforms and found that some matching methods are very accurate for certain kinds of images and not so accurate for other kinds of images. For example, figure 1 shows use of the CV_TM_SQDIFF matching method and accurately identifies the template image from the source, where as figure 2 shows the use of the CV_TM_CCORR matching method and completely misses the templated image from source image. Below are the seven template matching similarity functions.

Template Matching Similarity Functions

1. method=CV_TM_SQDIFF

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2$$

2. method=CV_TM_SQDIFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

3. method=CV_TM_CCORR

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))$$

4. method=CV_TM_CCORR_NORMED

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

5. method=CV_TM_CCOEFF

$$R(x,y) = \sum_{x',y'} (T'(x',y') \cdot I(x+x',y+y'))$$

6. method=CV_TM_CCOEFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

7. sum of absolute difference (SAD)

$$\mathrm{SAD} = \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} |I(x,y) - T(x,y)|$$

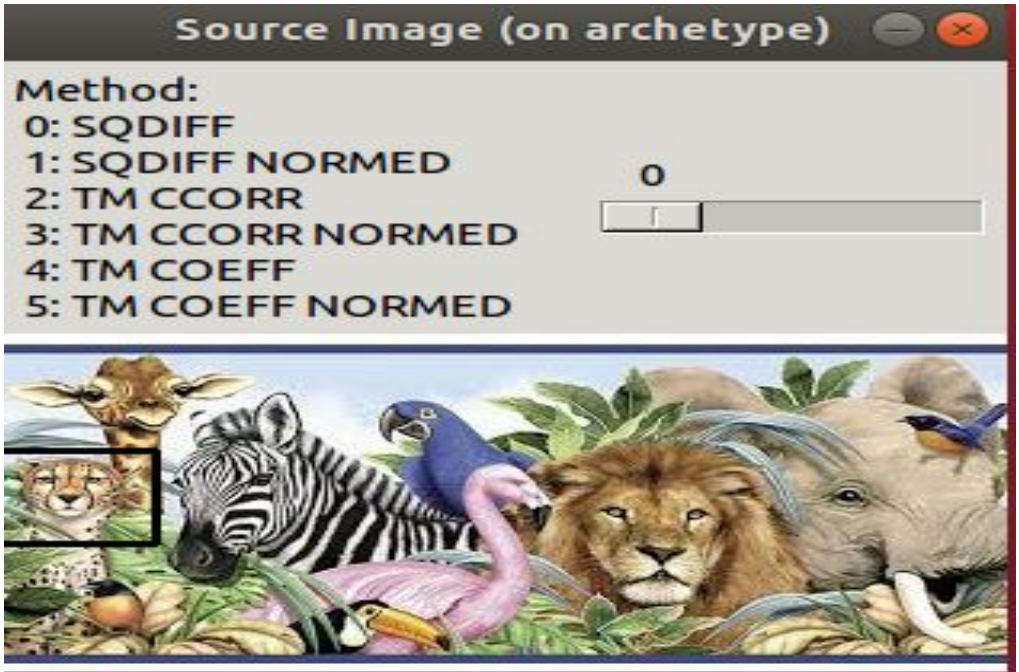


Figure 1: Template image correctly identified

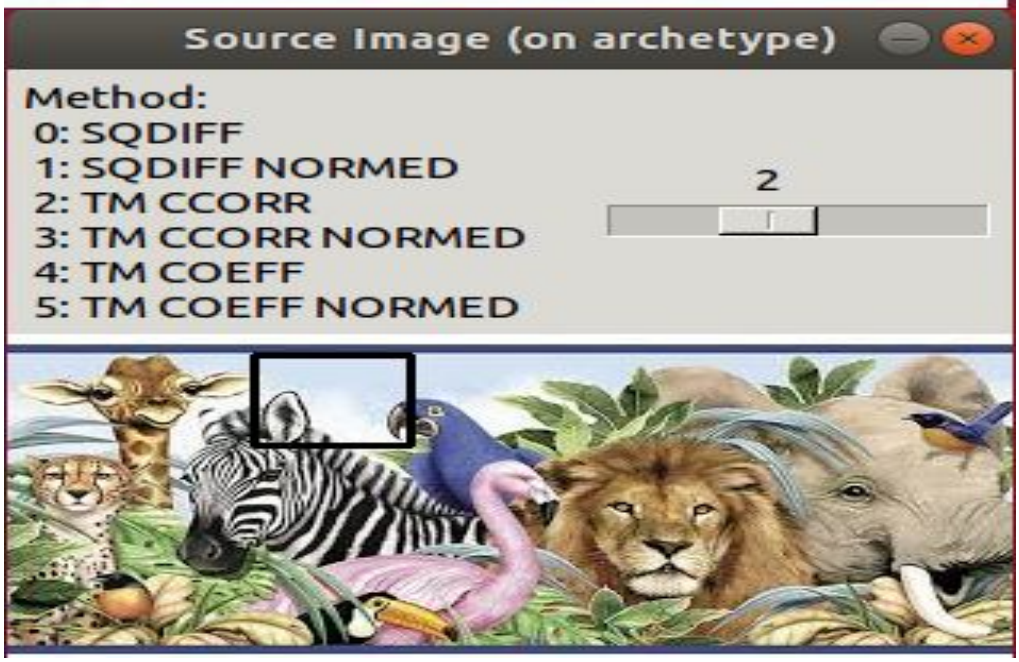


Figure 2: Template image missed

Template Matching on FPGA Platforms:

Once we started to synthesize the HLS code for the above OpenCV template matching methods, we came to realize that the OpenCV template matching methods are NOT synthesizable because these template matching methods use dynamic memory and resource allocation libraries. Furthermore, unfortunately there were not many template matching utilities

available in xfopenCV, which is the synthesizable analog of openCV. So, we had to implement the sum of absolute difference (SAD) method, sum of square difference (SSD) method and Normalized Cross Correlation method template matching methods on the FPGA platform from scratch and is implemented in 3 parts. (1) Pre-processing on CPU, (2) processing (template matching) on FPGA, and (3) post-processing on CPU.

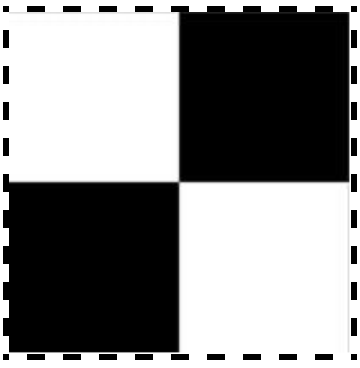
Pre-processing on CPU: The input to our template matching IP core is the template and source image RGB values contiguously stored in a 1D array. The preprocessing functions extract this data from our source and template images and format the input array such that it is compatible with our IP core. This is done in Python on the ARM core on the PYNQ board and is then sent to the template matching FPGA IP core.

Processing on FPGA: The template matching core has three key functions, grayscale, match, and similar. These functions grayscale the template and source image, slide a window the size of the template across the source image, and calculate the similarity of the window to the provided template image respectively. The core outputs a 1D array containing binary values indicating the potential matches of a given window to our template. The output is then redirected from the FPGA back to the ARM core.

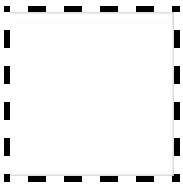
Post-processing on CPU: The output of the template matching core is then used to mark and identify the matching pixels in the source image. The matching pixels are marked in red and this new output image is then saved as the output of the template matching function. This image post-processing is also implemented in Python on the ARM core of the PYNQ.

The inputs to our template matching core are a 200x200 checkerboard as the source image and a 100x100 white square. The output of our template matching core correctly identifies both occurrences of this template image in the source image with minimal alignment error.

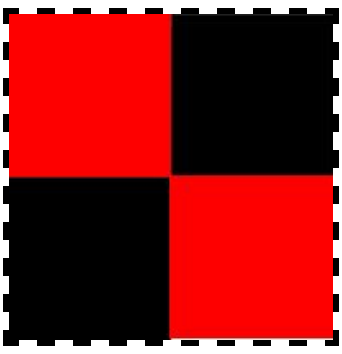
Source Image:



Template Image:



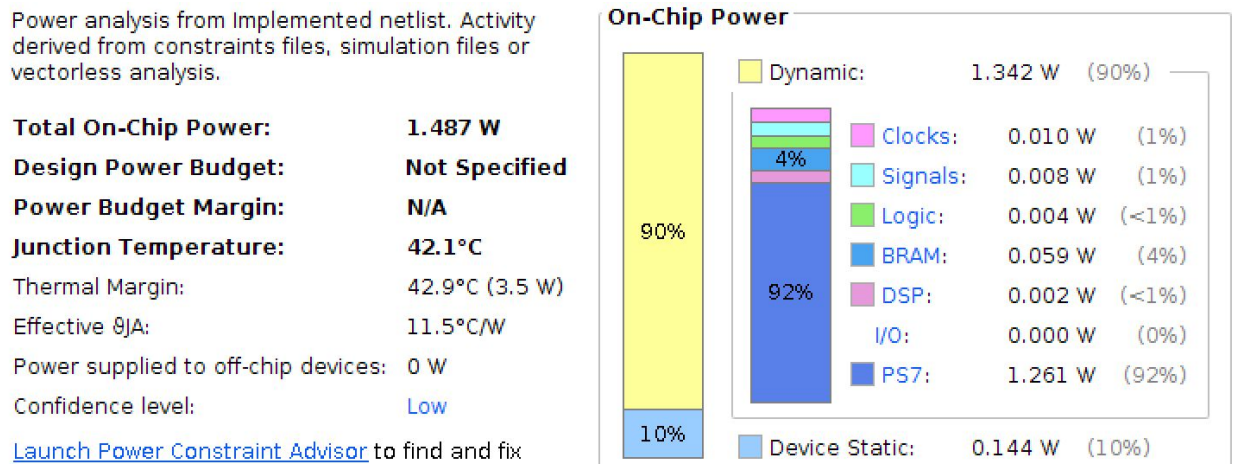
Output Image:



Results of Template Matching Experiment:

Both the ISA and FPGA platforms provided almost identical template matching accuracy; however the FPGA was vastly outperformed in terms of latency. The ISA completed the template matching benchmark in around 2 seconds, whereas the FPGA was an order of magnitude slower.

However, this performance disparity is attributable to the vast difference in expense between the two computing platforms, where the MacBook is an order of magnitude more expensive of a computing machine than the small PYNQ FPGA. Furthermore, the Macbook is also much more power hungry than the PYNQ board, with a CPU TDP of 45 W and GPU TDP of 75 W. The FPGA on the other hand consumed less than 1.5 W power, which is another advantage the FPGA has over the general purpose platform.



Furthermore, template matching code written on the MacBook is heavily optimized during compilation between both its six 2-way simultaneously multithreaded i7 CPUs and its AMD Radeon GPU core. Additionally the MacBook employed the use of the already refined vision functions provided by the openCV library. On the other hand, since xfopenCV provided no template matching utilities, the FPGA template matching logic was completely built from scratch in high-level synthesis, yielding less than optimal hardware RTL logic.

Real Time Real Time Facial Detection Experiment:

In order to make a more fair comparison for the next benchmark, we have implemented the ISA real time facial detection application on the ARM A9 processor found on board the PYNQ. Facial detection algorithms address the problem of identifying and tracking faces found in image data. The facial detection algorithm implemented here completely uses classical vision techniques and no machine learning models to aid in facial detection.

Image data is first denoised through Gaussian Blurring, after which thresholding is used to differentiate certain image features from one another like the background from the foreground. From here, dilation is used to exaggerate certain features of the image; then contouring is applied to finally deduce the location of the face. The major obstacle in implementing real time facial detection was the tuning of the various parameters used in thresholding, dilation, and the various other image processing functions employed.

Results of Facial Detection Experiment:

FPGA Frame Rate: ~ 2.5

CPU Frame Rate: < 0.25

FPGA Tracking Accuracy: ~ 100%

CPU Tracking Accuracy: N/A (CPU throughput cannot display frames fast enough to have any meaningful tracking accuracy)

FPGA Implementation Video: https://youtu.be/nyW51oxPwsA

CPU Implementation Video: https://youtu.be/BB4v9r35Oww

The ARM core does not do well in managing the large amount of processing power required by real time facial detection. It simply cannot keep up with the necessary throughput, misses frames and hence suffers in tracking accuracy. However, the same technique implemented on the FPGA exhibits much better performance, with up to 10x increase in frame rate and tracking accuracy.

Disadvantages of FPGA:

FPGAs are very difficult to program and design, and less flexible than general purpose architectures. Working with FPGAs requires in depth knowledge of both hardware and software. What is the right threshold value? How big or small are the particles to reject with a binary morphology filter? Which image preprocessing algorithm and algorithm parameters can best clean up an image? These are all common questions when developing a vision algorithm, and having the ability to make changes and see the results quickly is key. However, the traditional approach to FPGA development can slow down innovation due to the compilation times required between each design change of the algorithms.

Milestones:

We have successfully completed the milestones we had set out earlier, which were two separate template matching functions, initially template matching and stereo depth on both ISA and FPGA platforms. During the course of the quarter, we decided to change our second machine vision algorithm more interesting and pivoted from stereo depth to real time facial detection. Unfortunately, we did not have time to implement SLAM; however, we did not intend to implement the third algorithm as there was not simply enough time in 10 weeks with only 2 people, as initially the team was projected to be of 5.

Conclusion:

We were successfully able to accelerate the real time facial detection application using the FPGA; however, the FPGA failed to outcompete the i7 CPU and Radeon GPU in unison, which was to be expected. So, FPGAs indeed do have the capacity to accelerate image processing and can be considered as viable platforms for certain vision applications. However, it is also important to consider the time overhead and complexity required in programming FPGAs such as the necessary development tools along with the in depth hardware and software design knowledge required to interface with the board. FPGAs are best used in applications as accelerators for specific portions of logic that are suitable.

Video:

https://youtu.be/K-QgkE5LLRw

References

1. A brief introduction to OpenCV (https://ieeexplore.ieee.org/abstract/document/6240859)
2. Facial Recognition using OpenCV (https://www.researchgate.net/publication/267426877_Facial_Recognition_using_OpenCV)
3. High-level Synthesis Oriented Describing Method of Template Matching (https://www.researchgate.net/publication/301575148_High-level_Synthesis_Oriented_Describing_Method_of_Template_Matching)
4. HDMI / USB PYNQ Interfacing & Facial Detection https://github.com/ATaylorCEngFIET/Hackster
5. OpenCV Template Matching Documentation https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html