# Accelerating Face Detection Algorithm on the FPGA Using SDAccel

Jie Wang(✉) and Wei Leng

Dalian University of Technology, Dalian 116620, China
wangjiel003@163.com

**Abstract.** In recent years, with the rapid growth of big data and computation, high-performance computing and heterogeneous computing have been widely concerned. In object detection algorithms, people tend to pay less attention to training time, but more attention to algorithm running time, energy efficiency ratio and processing delay. FPGA can achieve data parallel operation, low power, low latency and reprogramming, providing powerful computing power and enough flexibility. In this paper, SDAccel tool of Xilinx is used to implement a heterogeneous computing platform for face detection based on CPU +FPGA, in which FPGA is used as a coprocessor to accelerate face detection algorithm. A high-level synthesis (HLS) approach allows developers to focus more on the architecture of the design and lowers the development threshold for software developers. The implementation of Viola Jones face detection algorithm on FPGA is taken as an example to demonstrate the development process of SDAccel, and explore the potential parallelism of the algorithm, as well as how to optimize the hardware circuit with high-level language. Our final design is 70 times faster than a single-threaded CPU.

**Keywords:** FPGA · Heterogeneous · Face detection · Architecture · High-level synthesis · SDAccel

## 1 Introduction

In the Internet industry, with the popularization of information technology, the explosion of data makes people have a new requirement on storage space. At the same time, the rise of machine learning, artificial intelligence, unmanned driving, industrial simulation and other fields makes the general CPU encounter more and more performance bottlenecks in processing massive computing, massive data and pictures, such as low parallelism, insufficient bandwidth and high delay. In order to meet the demand of diversified computing, more and more scenarios begin to introduce GPU, FPGA and other hardware for acceleration, resulting in the emergence of heterogeneous computing, which refers to the computing mode of a system composed of different types of instruction sets and computing units of the system architecture.

Two common heterogeneous computing platforms are CPU+GPU or CPU+FPGA architectures. The biggest advantage of these typical heterogeneous computing architectures is that they have higher efficiency and lower latency than traditional CPU parallel computing. CPU belongs to general computing and is good at management and


X. Chu et al. (Eds.): QShine 2019, LNICST 300, pp. 154–165, 2020.
https://doi.org/10.1007/978-3-030-38819-5_10

scheduling. Therefore, the algorithms-intensive part can be unloaded onto FPGA or GPU for parallel calculation, so as to realize algorithm acceleration. FPGA has hardware programmability, which means that FPGA can make customized design according to algorithm, which increases the flexibility of FPGA and enables FPGA to enter the market quickly. Compared with GPU, FPGA tends to show a better energy efficiency ratio, and the processing delay is much lower than GPU. This paper uses SDAccel to implement a heterogeneous computing platform for face detection based on CPU+FPGA. CPU transmits data to FPGA through PCIe bus. The face detection algorithm is accelerated by the FPGA. Finally, the results are transmitted back to CPU from FPGA through PCIe bus, detection results and pictures display are performed on CPU.

The SDAccel environment uses a standard programming language, providing a framework for developing and delivering FPGA accelerated data center applications. Nowadays, the application of FPGA has turned to the field of high-performance heterogeneous computing and massive data processing, and the object of using FPGA is not necessarily the traditional hardware engineer, it is likely that the programmer who works for developing software. SDAccel enables application developers to use familiar software programming workflows to accelerate with FPGA, even though there has been little experience in FPGA or hardware design before. Programs running on CPU are developed using c/c++ and opencl [1] APIs, while programs running on hardware can be developed using c/c++, opencl or RTL. When we have finished our design on SDAccel, we need to first carry out software simulation to verify the functionality of the design, then carry out hardware simulation to see the rationality of the resources and architecture needed for the design, and finally generate the FPGA bitstream. The hardware program of this paper is synthesized by Vivado HLS with c/c++ high-level language. The benefit of this approach is that it shortens the development cycle and allows developers to focus more on the architecture of the design.

The rest of the paper is organized as follows: Sect. 2 examines the related work; Sect. 3 provides an overview of face detection based on the Viola Jones algorithm; Sect. 4 describes the implementation of the real-time face detection system in an FPGA and the optimization methods; Sect. 5 presents performance and area results, followed by conclusions in Sect. 6.

## 2 Related Work

Face detection is the key technology of pattern recognition and computer vision. The improvement of algorithm speed is often at the cost of increasing hardware resources or power consumption. However, with the further development of face detection in various fields and the consideration of the trend of miniaturization and portability for energy saving, the requirement of hardware and power consumption for face detection has gradually become more and more important. Many researches are based on hardware-accelerated face detection algorithms [4–7, 13–16]. Most of which use Viola Jones face detection algorithm [3], because it not only has advantages in accuracy and speed, but also is more suitable for hardware implementation. Lai et al. [4] proposed a FPGA hardware architecture for face detection using feature cascade classifiers, which

achieved the detection speed of 143 frames per second (FPS) at VGA (640 × 480) resolution. But they only used 52 Haar feature classifiers, which greatly reduced the accuracy of face detection and could not be used in actual detection tasks. Hiromoto [5] made a thorough analysis of the algorithm, studied the effects of various parallel schemes, different image downscaling methods and fixed-point on the detection speed and resource consumption, and proposed a partial parallel face detection architecture. This greatly reduced the total processing time without greatly increasing the circuit area. Cho [6] used each detection window to generate integral image for face detection. Instead of upscaling Haar features, they used image pyramid to detect faces. Kyrkou [7] combined image downscaling and feature upscaling to realize viola jones face detection algorithm.

There are many studies on high-level synthesis (HLS) tools [2, 8–12]. Srivastava [13] explored how to implement viola jones face detection algorithm with high-level synthesis method. They implemented their design with SDSOC and achieved a rate of 30 frames per second (FPS) at 320 × 240 resolution. In this paper, we implement a face detection algorithm based on heterogeneous computing platform using high-level synthesis method and SDAccel tool.

## 3 Face Detection Algorithm

Haar features is first proposed by Papageorgiou [17]. But it is too much calculation to be applied. Later, Paul Viola and Michael Jones proposed a method of fast calculating Haar features by using integral image in [3], three types and four forms of Haar features are used. Haar-like features were widely used together with Adaboost algorithm. Common Haar features are shown in the Fig. 1.

**Fig. 1.** Five common haar features

The haar features is calculated by the sum of pixels in the white rectangle by their weight minus the sum of pixels in the black rectangle by their weight. Because of the translation and enlargement of variety of Haar features in the picture, there will be a huge number of single features. The numerical calculation of each feature involves the sum of many pixel values. If the calculation is carried out directly, the amount of calculation is very huge, which brings trouble to the practical application of Haar features. In order to solve this problem, Paul Viola and Michal Jones put forward a fast method of calculating Haar features by integral image. The pixel value of a point in the integral image is equal to the sum of all the pixel values corresponding to the upper left

of this point in the original image. As shown in Fig. 2, to calculate the total value of pixels in area D, we only need to subtract the total value of pixels in the upper left of point 2 and point 3 from point 4, and add the total value of pixels in the upper left of point 1. The calculation process is expressed as sum (x4, y4) - sum (x3, y3) - sum (x2, y2) + sum (x1, y1) by the pixel values in the integral image, where xi and yi represent the horizontal and vertical coordinates of point i. Using the integral image, we can quickly calculate the pixel values in any rectangular area of the image. A small detection window contains tens of thousands of Haar features. It is impractical to calculate all the Haar features by integral image. Moreover, not all Haar features can be used as classifiers. In order to find the most suitable features for constructing classifiers among the numerous Haar features and improve the accuracy of these classifiers, Haar features are combined with Adaboost algorithm. Adaboost algorithm is a kind of boosting algorithm, and boosting algorithm is an algorithm that upgrades weak learning algorithm to strong learning algorithm. Adaboost algorithm was proposed by Freund et al. [18] in 1995, namely, Adaptive Boosting learning. It can adjust the hypothesis error rate adaptively according to the learning results of the weak learning algorithm, so it does not need to get the lowest hypothesis error rate in advance. That is to say, Adaboost does not need to know the performance of the weak classifier in advance like other boosting algorithms, and the learning efficiency keeps the same as other boosting algorithms. This is why Adaboost algorithm is widely used.

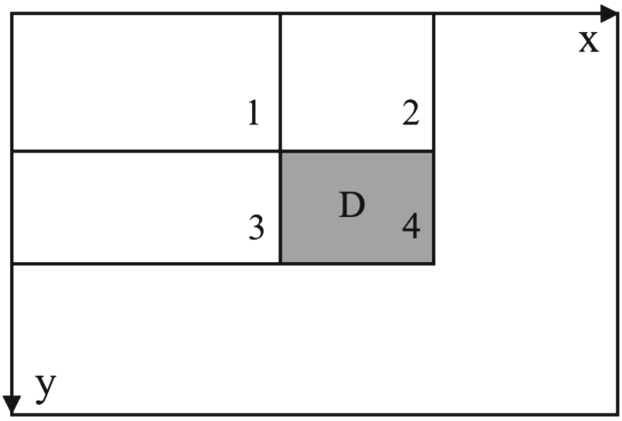


**Fig. 2.** Integral image.

## 4 Implementation

### 4.1 Data Transmission

The image used for detection is transmitted to DDR on the FPGA through the PCIe bus. The image pixel value needs to be extracted from DDR by AXI bus. In order to reduce the time of data transmission, we should adopt burst transmission method to

extract data from DDR through AXI bus. Because the lowest bit width transmitted by AXI bus is 32 bits in SDAccel, and the bit width of image pixel value is 8 bits, in order to make full use of the data width of AXI, we can combine four pixels into a 32-bit data. When the 32-bit data is transmitted to the FPGA chip, the data is divided into four pixels. The data transmission process is shown in Fig. 3.

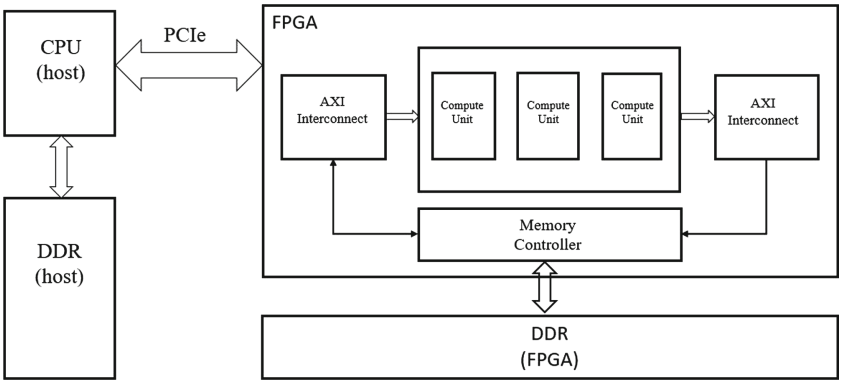


**Fig. 3.** Data transfer model

### 4.2 Image Storage

The module takes out the image transferred from PC to FPGA from DDR and stores it into BRAM on chip. Then pass the image from BRAM to the image scaling module and detection module. The advantage of storing images in BRAM is that the latency of data transmission is small and the speed is fast. The image resolution used in this paper is 320 $\times$ 240.

### 4.3 Image Scaling Module

When the sliding window method is used for face detection, in order to detect faces with different scales in the image, it is usually necessary to constantly expand the window size or reduce the image, and the scaling factor is usually 1.2 or 1.25. This paper uses the method of image pyramid, and scaling method is the nearest neighbor, the scaling factor is 1.25 (Fig. 4). This module is pipelined by adding a #pragma HLS pipeline instruction, which greatly speeds up image scaling.

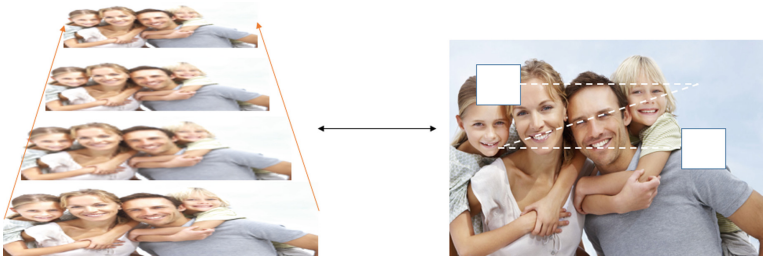

**Fig. 4.** Image pyramid and sliding window

### 4.4 Classification Module

**Integral Image**

In order to calculate the characteristic value of Haar quickly, we need to calculate the integral image of the image first. The calculated Haar feature values often need to be normalized by dividing the standard deviation of the pixels in the current window, so a square integral image is needed. The square integral image is the sum of squares of all the pixel values in the image. Taking the image of 320 × 240 resolution as an example, if we want to store the whole integral image on the FPGA, the data bit width of each storage unit of the integral graph is $\log_2(320 \times 240 \times 255)$, at least 25 bits are needed to store one data in the integral image, and the whole integral image needs 320 × 240 × 25bits, that is, 2 M storage unit. But in the process of each detection of sliding window, only the data in the window is needed, every data in the integral image is useless in most of the time. Moreover, with the improvement of image resolution, the space occupied by the integral image will increase rapidly. Therefore, we used the method of dynamically generating integral image, which only generates one integral image of detecting window size at a time.

The integral image detection window in this paper consists of 25 × 25 registers, each of which stores the integral values corresponding to each pixel position in the detection window. The width of the register is 18 bits ($\log_2(255 \times 25 \times 25)$). The reason why register is used instead of RAM or ROM is that the integral image data at any position in the detection window can be accessed and updated at the same time, so as to improve the detection efficiency. Because it only stores the integral image of the current detection window, the consumption of hardware is not large. By taking advantage of the relevance between adjacent windows and the read-on-call characteristics of registers, we design a pipeline based quick update method of single clock integral image. Only the first detection window requires several clock delays to fill the empty integral image window. After that, only one clock can get the integral image of adjacent window, and the data can be updated very quickly. The update of integral image is divided into two parts: get the column integral value of the detection window, and get the integral value of the whole window from the column integral value. Taking

$3 \times 3$ window as an example, the updating process of integral image is shown in Fig. 5. To compute the squared integral image, the same procedure is followed.

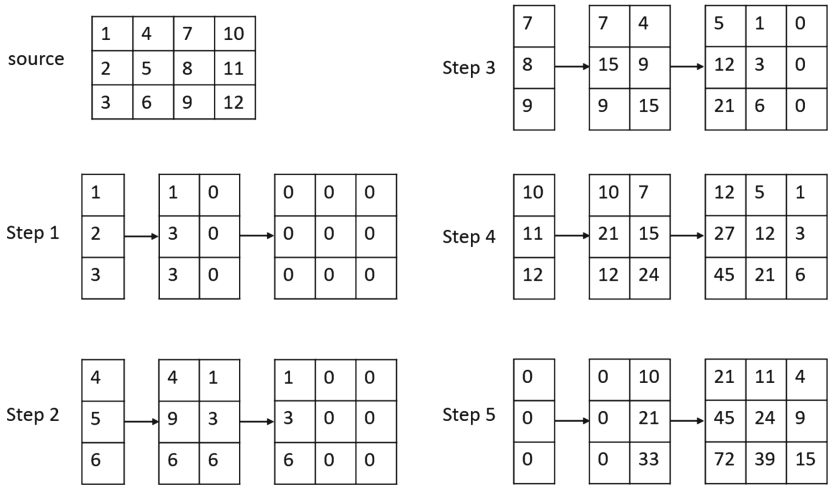


**Fig. 5.** Illustration of the integral image computation and data movement

**Classifier**

The classifier used in this paper comes from opencv. This classifier is trained by Adaboost algorithm with $24 \times 24$ frontal face, in which each haar feature is a weak classifier, multiple weak classifiers are combined to form a strong classifier, and multiple strong classifiers are cascade to form the final cascade classifier. The classifier finally trained consists of 25 strong classifiers and 2913 weak classifiers, and its specific components are shown in the Table 1.

**Table 1.** Number of weak classifiers in each stage

| Stage | Classifier | Stage | Classifier | Stage | Classifier |
|---|---|---|---|---|---|
| 0 | 9 | 9 | 91 | 18 | 169 |
| 1 | 16 | 10 | 99 | 19 | 196 |
| 2 | 27 | 11 | 115 | 20 | 197 |
| 3 | 32 | 12 | 127 | 21 | 181 |
| 4 | 52 | 13 | 135 | 22 | 199 |
| 5 | 53 | 14 | 136 | 23 | 211 |
| 6 | 62 | 15 | 137 | 24 | 200 |
| 7 | 72 | 16 | 159 | Total | 2913 |
| 8 | 83 | 17 | 155 | | |

The formula for calculating the value of weak classifier is shown in (1).

$$F_{Haar} = \frac{E(R_{white}) - E(R_{black})}{w \cdot h \cdot \sqrt{\left|E(R_u)^2 - E\left(R_u^2\right)\right|}} \tag{1}$$

$E(R_u)^2$ represents the square of the average pixel values in the window, and $E\left(R_u^2\right)$ represents the average sum of squares of pixel values in the window. $w \cdot h \cdot \sqrt{\left|E(R_u)^2 - E\left(R_u^2\right)\right|}$ can be converted to $\sqrt{w \cdot h \cdot sqsum - sum \cdot sum}$, sqsum is expressed as the sum of squares of the pixels in the window, and sum is expressed as the sum of the pixels in the window. These two values can be obtained by the square integral image and the integral image respectively. Each weak classifier has a threshold, a left value and a right value. If the Haar feature value is greater than the threshold, the right value is output, otherwise the left value is output. The input of weak classifier module has coordinate of the upper left corner of the rectangle, length, width, weight, left and right values. Since these values are fixed and do not need to be changed, we can store these values in each ROM of the FPGA according to the category, thus eliminating the initialization time and extracting all attributes of a weak classifier in parallel in one clock cycle. In SDAccel, we can define arrays through static int when writing kernel code, and fill the values of various attributes of the weak classifier into different arrays, so that vivado HLS will synthesize these arrays into ROM in FPGA when compiling kernel code.

These weak classifiers will eventually form a strong classifier. Each strong classifier also has a threshold, which we store in ROM. If the sum of the output of all weak classifiers in the strong classifier is less than this threshold, the current detection window is excluded, otherwise the current detection window is passed. These strong classifiers are cascaded to form the final face detection classifier. The working mode of the classifier is shown in the Fig. 6. We also store the number of weak classifiers in each strong classifier in ROM. Thus, in SDAccel, we can complete the detection module through two for loops. But it does not give full play to the characteristics of FPGA parallel computing. We can make vivado HLS design this code into pipeline mode on hardware by adding # pragma HLS pipeline instruction in the inner loop. By pipelining 2913 weak classifiers, the speed of image detection is greatly accelerated.

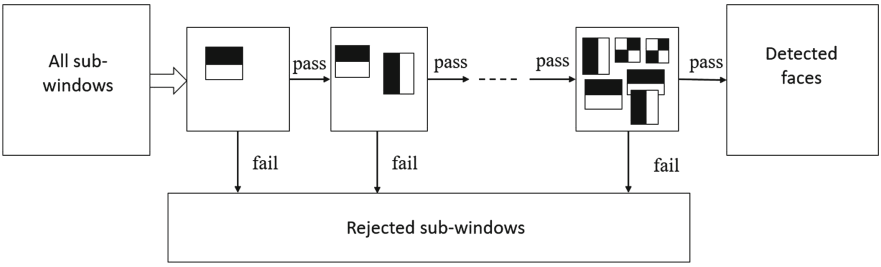


**Fig. 6.** Illustration of the Classification process

### 4.5 Optimization

There are many factors that affect the speed of face detection, such as image scaling factor, sliding window step size, initial sliding window size and so on. Although the values of these factors can be changed, they can only be changed in a certain range, otherwise, the accuracy of face detection will be affected. The image scaling factor used in this paper is 1.25, the sliding window step is 1, and the window size is 24 × 24. For a 320 × 240 resolution image, a total of 157433 detection windows can be generated. In the worst case, each detection window will be detected by 2913 classifiers, which shows that the computation of the detection algorithm is very large. However, except for the target area in the image, very few areas will be detected by all the classifiers, most of which are usually excluded in the first few layers of the cascade classifier. This is also the advantage of cascaded classifiers. If we want to speed up the algorithm, we usually use hardware area in exchange for time. In VJ face detection algorithm, the most extreme example is to parallelize all weak classification detection. But this approach requires a lot of hardware resources, increases power consumption and increases the cost of products. We made a compromise between area and speed. Since most of the detection windows are excluded by the first three layers of cascade classifier, we only expand the first three layers of cascade classifier. The latter 22-layer classifier still adopts pipeline parallel method and achieves good results. In order to expand the first three layers of strong classifiers, the 12 coordinate values required by weak classifiers should be written into the code manually when writing the kernel code in SDAccel, rather than from ROM. Because ROM can only read two values in one clock cycle at most, it is obviously not able to fully expand a strong classifier.

We should also note the various floating-point operations in fpga. Because FPGA is not suitable for floating-point operation, we should convert floating-point into fixed-point as much as possible, such as the weights and thresholds of classifiers. This saves hardware resources and speeds up computing.

When writing software algorithms, we are often used to defining integer variables as int types. When we want to implement an algorithm on hardware, we should try to be as accurate as possible about the bit width of the data. Vivado HLS provides ap_ [u] int, ap_[u]fixed type so that we can accurately declare the data bit width. For example, in our algorithm, we use an integral image of 25 × 25, which has a bit width of 18bits ($\log_2$ (25 × 25 × 255)), much less than 32bits. This saves hardware resources, improves clock frequency, speeds up algorithms, and makes vivado easier to route.

In addition to the hardware architecture, we can also consider increasing the speed of the algorithm by changing the algorithm. Variance is the necessary data for VJ face detection algorithm. Through the variance statistics of face frontal images, we find that the variance of face is mostly in the range of 200 to 6000. So we also put the variance as a strong classifier into the first layer of the cascade classifier. Since the variance calculation does not need to consume extra time to calculate, and the classification process is only compared with the threshold value, this method speeds up the face detection algorithm. The speed comparison between adding variance classifier and not adding variance classifier will be given in the results section.

## 5 Experimental Results

Our design is implemented on inrel i7-8700 CPU and KCU1500 FPGA. The software environment is SDAccel 2018.2. SDAccel generates kernel programs by calling Vivado HLS 2018.2, and develops host programs by calling opencl API. We designed a single thread cpu-based face detection algorithm for comparison with our hardware implementation. We tested the performance of the software algorithm on the Intel i7-8700 CPU, and the hardware performance was tested at the clock frequency of 100 MHz.

**Table 2.** Performance of proposed face detection system

| | Resolution | Variance classifier(No) | Variance classifier(Yes) |
|---|---|---|---|
| SW Classifier | 320 × 240 | 0.78fps(1279 ms) | – |
| HW Classifier | 320 × 240 | 58.8fps(17 ms) | 66.6fps(15 ms) |

Table 2 shows the performance of the implemented face detection algorithm in software and hardware. It can be seen that CPUs based single thread face detection algorithm is extremely time consuming, with a speed of less than one frame per second. After acceleration by FPGA, the speed can be increased 70 times to 58 frames per second (FPS), which meets the demand of video real-time processing. We can also see that when adding variance classifier, our speed can be increased by at least 10%, and in some cases, the variance classifier can achieve quite better performance. Table 3 shows the resource utilization of the system.

**Table 3.** Resource Utilization of our proposed face detection system

| Logic | Total used | Total available | Utilization (%) |
|---|---|---|---|
| LUT | 84137 | 663360 | 12 |
| FF | 52735 | 1326720 | 3 |
| DSP48E | 68 | 5520 | 1 |
| BRAM_18K | 194 | 4320 | 4 |

## 6 Conclusions

In this paper, we implement VJ face detection algorithm by using SDAccel, and implement a heterogeneous computing platform of CPU+FPGA. We discuss the parallelism of the algorithm and the hardware architecture, and try some optimization methods. According to the experimental results, our design can meet the requirements of real-time face detection. We demonstrate the efficiency of SDAccel, and it is clear

that SDAccel's products are of good quality and greatly reduce the development cycle of FPGA products. But there is still room for improvement in the synthesized fine control of design.

## References


1. Stone, J.E., Gohara, D., Shi, G.: OpenCL: a parallel programming standard for heterogeneous computing systems. Comput. Sci. Eng. **12**(1–3), 66–73 (2010)
2. Guidi, G., et al.: On how to improve FPGA-based systems design productivity via SDAccel. In: Proceedings of 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016, pp. 247–252 (2016)
3. Viola, P., Jones, M.J.: Robust real-time face detection. Int. J. Comput. Vis. **57**(2), 137–154 (2004)
4. Lai, H.C., Savvides, M., Chen, T.: Proposed FPGA hardware architecture for high frame rate (>00 Fps) face detection using feature cascade classifiers. In: IEEE Conference on Biometrics: Theory, Applications and Systems, BTAS 2007 (2007)
5. Hiromoto, M., Sugano, H., Miyamoto, R.: Partially parallel architecture for AdaBoost-based detection with haar-like features. IEEE Trans. Circuits Syst. Video Technol. **19**(1), 41–52 (2009)
6. Cho, J., Benson, B., Mirzaei, S., Kastner, R.: Parallelized architecture of multiple classifiers for face detection. In: 2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, pp. 75–82. IEEE (2009)
7. Kyrkou, C., Theocharides, T.: A flexible parallel hardware architecture for AdaBoost-based real-time object detection. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **19**(6), 1034–1047 (2011)
8. Casseau, E., Gal, B.L.: High-level synthesis for the design of FPGA-based signal processing systems. In: International Symposium on Systems, Architectures, Modeling, and Simulation (SAMOS) (2009)
9. Skalicky, S., Wood, C., Łukowiak, M., Ryan, M.: High-level synthesis: where are we? A case study on matrix multiplication. In: International Conference on Reconfigurable Computing and FPGAs (ReConFig) (2013)
10. Winterstein, F., Bayliss, S., Constantinides, G.A.: High-level synthesis of dynamic data structures: a case study using Vivado HLS. In: International Conference on Field-Programmable Technology (FPT) (2013)
11. Neuendorffer, S., Li, T., Wang, D.: Accelerating OpenCV applications with Zynq-7000 all programmable SoC using Vivado HLS video libraries. Xilinx Inc., August 2013
12. Edwards, S., et al.: The challenges of synthesizing hardware from c- like languages. IEEE Des. Test Comput. **23**(5), 375–386 (2006)
13. Srivastava, N.K., Dai, S., Manohar, R., Zhang, Z.: Accelerating face detection on programmable SoC using C-based synthesis. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA 2017, pp. 195–200 (2017)
14. Zemcik, P., Juranek, R., Musil, P., Musil, M., Hradis, M.: High performance architecture for object detection in streamed videos. In: Proceedings of 2013 23rd International Conference on Field Programmable Logic and Applications, FPL 2013, pp. 4–7 (2013)
15. Musil, P., Juranek, R., Musil, M., et al.: Cascaded stripe memory engines for multi-scale object detection in FPGA. IEEE Trans. Circuits Syst. Video Technol., 1–1 (2018)

16. Kyrkou, C., Bouganis, C., Theocharides, T., Polycarpou, M.M.: Embedded hardware-efficient real-time classification with cascade support vector machines. IEEE Trans. Neural Netw. Learn. Syst. **27**(1), 99–112 (2016)
17. Papageorgiou, C.P., Oren, M., Poggio, T.: General framework for object detection. In: Proceedings of the IEEE International Conference on Computer Vision, February 1998, pp. 555–562 (1998)
18. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting BT. In: Proceedings of Computational Learning Theory: Second European Conference, EuroCOLT 1995, Barcelona, Spain, 13–15 March 1995, pp. 23–37 (1995). Journal of Computer & System Sciences