A Novel Low-Power Embedded Object Recognition System Working at Multi-Frames per Second

ANTONIS NIKITAKIS, SAVVAS PAPAIOANNOU, and IOANNIS PAPAEFSTATHIOU,
Technical University of Crete

One very important challenge in the field of multimedia is the implementation of fast and detailed Object Detection and Recognition systems. In particular, in the current state-of-the-art mobile multimedia systems, it is highly desirable to detect and locate certain objects within a video frame in real time. Although a significant number of Object Detection and Recognition schemes have been developed and implemented, triggering very accurate results, the vast majority of them cannot be applied in state-of-the-art mobile multimedia devices; this is mainly due to the fact that they are highly complex schemes that require a significant amount of processing power, while they are also time consuming and very power hungry. In this article, we present a novel FPGA-based embedded implementation of a very efficient object recognition algorithm called Receptive Field Cooccurrence Histograms Algorithm (RFCH). Our main focus was to increase its performance so as to be able to handle the object recognition task of today's highly sophisticated embedded multimedia systems while keeping its energy consumption at very low levels. Our low-power embedded reconfigurable system is at least 15 times faster than the software implementation on a low-voltage high-end CPU, while consuming at least 60 times less energy. Our novel system is also 88 times more energy efficient than the recently introduced low-power multi-core Intel devices which are optimized for embedded systems. This is, to the best of our knowledge, the first system presented that can execute the complete complex object recognition task at a multi frame per second rate while consuming minimal amounts of energy, making it an ideal candidate for future embedded multimedia systems.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems; I.4 [**Computing Methodologies**]: Image Processing and Computer Vision; I.5.1 [**Pattern Recognition**]: Applications—*Computer vision*

General Terms: Design

Additional Key Words and Phrases: Multimedia, embedded design, computer vision, object detection, performance, FPGAs

ACM Reference Format:
Nikitakis, A., Papaioannou, S., and Papaefstathiou, I. 2013. A novel low-power embedded object recognition system working at multi-frames per second. ACM Trans. Embedd. Comput. Syst. 12, 1s, Article 33 (March 2013), 20 pages.
DOI: http://dx.doi.org/10.1145/2435227.2435229

This work is partly funded by "A Highly Efficient Adaptive multi-processor framework" research project (HEAP, project number: 247615), within the Seventh Framework Program and financed by Community Funds and by "Increasing EU citizen security by utilizing innovative intelligent signal processing systems for euro-coin validation and metal quality testing" research project (SAFEMETAL, project id: 262558), implemented within the Seventh Framework Program and financed by Community Funds.
Author's address: I. Papaefstathiou; email: ygp@synelixis.com.

1. INTRODUCTION

Multimedia is a technology which has been enjoying considerable attention for many years. Multimedia involves the use of multiple forms of media such as audio, images and video in an integrated manner. A computer user interacts in a daily basis with a huge amount of multimedia data mostly through the internet. Multimedia usually carries useful but raw and unsorted information. It is an emerging need for the user to be able to search in a content base through multimedia content such as images and video. For this reason object recognition gives today multimedia applications new potentials that enable the previously poorly sorted media data to be classified and unified with the rest of today information.

In the Content-based image retrieval applications we try to find all images in a larger set of images which have a specific content. The content can be specified for example in terms of similarity relative a target image (e.g., find all images similar to image X or find all images showing the specific object depicted in image X).

Google recently introduced image search based on content as well as the Google's Goggles android application which enables the user to search real world objects by taking a picture. In a desktop environment already Apple's iPhoto and Google's Picasa featuring face detection and recognition technology in order to classify user's photos according to the persons depicted in the photo.

By extending the above object recognition approaches we are also now able to search objects or persons in large video streams; this is called Content based Video Retrieval and it is a very important application for numerous future multimedia systems. This problem can also be defined as a generalized Video Mining problem where we are trying to discover patterns and objects, in an unsupervised way, within video streams. In such video streams the amount of data that has to be analyzed can be tremendous; this is the case for example for YouTube or for all the TV broadcast networks. In general, the classification of thousands of video hours is impossible to be done in a supervised way and even in an unsupervised way the processing time is a crucial factor. Moreover, such a classification is also needed in state-of-the-art mobile multimedia systems which can display and store several Gigabytes of high-quality video; in such systems low energy consumption is also crucial.

Moving to the game industry we recently met alternative input systems based around a webcam-style peripheral such as the Microsoft Kinect [XBOX].[1] Kinect enables users, by recognizing their gestures (i.e., performing certain object recognition tasks), to control and interact with the Xbox 360 game console without the need to touch a game controller. Such gesture recognition systems are also being introduced in different embedded multimedia systems including interactive shop windows, portable game consoles and smart TVs.

The critical factors in all those discrete yet interrelated applications are the accuracy of the system and well as its response time. Moreover, since in many cases those object recognition systems are utilized in mobile environments the energy consumption is also a critical issue. The object recognition problem itself is a complex and computational intensive task for today microprocessors no matter if they are power-hungry desktop CPUs or low-power embedded ones. This complexity further increases since totally or slightly different objects can appear anywhere in the image (in different light conditions) and the system should also report where those objects are (i.e., localize them). As a result, even when executed on high-end mobile CPUs, such applications cannot be performed at multi-frame per second rates, whereas when optimized for speed they typically give only target specific responses (i.e., a car found in coordinates x,y) .

[1]http://www.xbox.com/kinect.

This article presents a complete generalized object detection system implemented on a reconfigurable device that can execute a very efficient such algorithm at a rate of more than one frame per second while consuming about 60 times less energy than a low-power CPU executing the exact same algorithm.

We clearly demonstrate that such a complex task can, probably for the first time, be addressed by a single chip solution running on minimal power; this is achieved by exploiting the heterogeneity of custom hardware and a low power embedded CPU. We present such a single chip prototype in this article while our ideal target single-chip platform is the already announced by Xilinx[2] Zynq-7000 single chip device featuring a dual-core ARM CPU and FPGA reconfigurable logic in the same silicon.

Moreover, due to the programmability features of the FPGAs, the system can support the requested object recognition tasks only when needed; based on the multimedia applications' requirements, at any given time, this same FPGA can also perform other similar tasks that are executed efficiently on reconfigurable devices such as 3D image reconstruction [Hadjitheophanous et al. 2010] or face detection [Gao and Lu 2008] also met in various embedded multimedia applications.

2. RELATED WORK

2.1. Frequency Assignment

Even though no complete low-power multi-frame per second object recognition system exists, there are several FPGA-based systems implementing certain face recognition algorithms as well as some hardware systems executing specific sub-parts of the object recognition algorithms that are related, in a certain manner, to our work.

Gao and Lu [2008] the authors present a novel approach, utilizing a state-of-the-art FPGA, so as to accelerate the Haar-classifier face detection algorithm. By utilizing a large number of parallel arithmetic units in the FPGA they achieved real-time performance, with very high detection rates and very low false positives. Their implementation is tailored to a HiTech Global PCIe card that contains a Xilinx XC5VLX110T FPGA device. Moreover in Kyrkou and Theocharide [2011] another Haar-based face detection scheme is described which outperforms all the existing such schemes implemented in FPGAs. However, all those systems are optimized for face-detection and cannot be efficiently applied to general object-recognition.

He and Yuan [2008] proposed a novel self-adaptive Canny edge detection scheme while they also present an FPGA implementation optimized for mobile robotic systems. Their system utilizes an Altera Cyclone EP1C60240C8 and can detect the edges of a certain, pre-defined, object on a grey-scale image at an analysis of 360x280 in 2.5ms (or in other words at a speed of 400 frames per second). Gentsos et al. [2010] present another implementation of the Canny edge detector that processes 4-pixels in parallel; this approach increases the throughput of the design without increasing the required on-chip cache memories. By increasing the parallelism of their scheme, their system can process high resolution images (up to 1.2Mpixels) in 3.09ms (i.e., at about 300 frames per second) when their scheme is implemented on a Xilinx Spartan-6 FPGA clocked at 200MHz. However, their system implements only the edge detection task while the rest of the object recognition process is not supported or even discussed.

In Bhowmik et al. [2006] a hardware implementation of an object classification system based on moment invariants and Kohonen neural networks is presented capable to classify objects in realtime. The authors implemented the classification phase in hardware while leaving the training of the Kohonen network into software; in particular the

[2]Xilinx,DS190,Zynq-7000 Extensible Processing Platform Overiew.

computation of the moment invariants has been implemented in hardware along with a set of sixteen parallel Kohonen neurons for the classification of an unknown object, demonstrating a possible real-time solution for object classification; unfortunately no specific performance numbers are given.

Nair et al. [2005] present an FPGA-Based People Detection System. They use JPEG-compressed frames from a network camera which after pre-processing (i.e., feature extraction), are sent to a machine-learning detector, implemented on a Virtex-II 2V1000; the FPGA executes the actual detection process. The system is demonstrated on an automated video surveillance application detecting people accurately at a rate of about 2.5 frames per second when clocked at 75 MHz.

Goshorn et al. [2010] present an object detection system that can detect a single object at a rate of 266 frames per second. However, they did not present any data about its accuracy and since they use a very poor correlation method based on the sum of absolute differences (SAD), the accuracy of their system is heavily questioned; moreover, their device can detect only a pre-defined single object in a single scene while they only roughly localize it (i.e., localize only the center of the object and they do not report any bounding box).

Finall, Shotton et al. [2011] the authors propose a new method to predict 3D positions of body joints from a single depth image at up to 200fps on consumer hardware. However they use a depth camera such as Microsoft Kinect, which consists of an infrared laser projector combined with a monochrome CMOS sensor. They also do not generalize their method to other object detection tasks that may be useful in multimedia systems.

When compared with all these existing systems, our approach has certain significant advantages such as the following.

(1) It is the only one supporting the complete general, multi-object recognition and localization task at more than one frame per second.
(2) This is, to the best of our knowledge, the only embedded system that has been specifically designed so as, not only to be real-time, but also to consume as less energy as possible, in order to address the needs of today's embedded multimedia devices.
(3) It is the only system that can work simultaneously on multiple features (i.e., 7 features) which significantly increase the robustness of the system while still supporting a multi frame per second rate in real-world environments.
(4) Even when compared with the different face detection systems, it is the only one performing efficiently in hardware the on-line training phase utilizing only a single training sample per object; the Haar-based systems need hundreds of training samples per object and thus they do the training off-line which severally limits their efficiency.
(5) The algorithm utilized is probably one of the most accurate generalized object recognition algorithms presented so far as described in Ekvall and Kragic [2005].

Based on the above, we believe that this is the first system addressing all the needs of the real-time embedded multimedia devices, recently introduced, that involve complex object recognition tasks. The rest of the article is organized as follows: Section 3 presents the algorithm that has been implemented while Section 4 demonstrates how we ended up with the optimal hardware/software partioning for our final embedded system. Section 5 presents, in detail, the high-level as well as the micro-architecture of the system while it also highlights how the complete embedded device has been verified. In Section 6 we reveal the silicon cost of the embedded system whereas in Section 7 we demonstrate the real-world performance results of the end-device. Finally, Section 8 discusses the limitations of the current system as well as some direction for future work and Section 9 concludes our article.

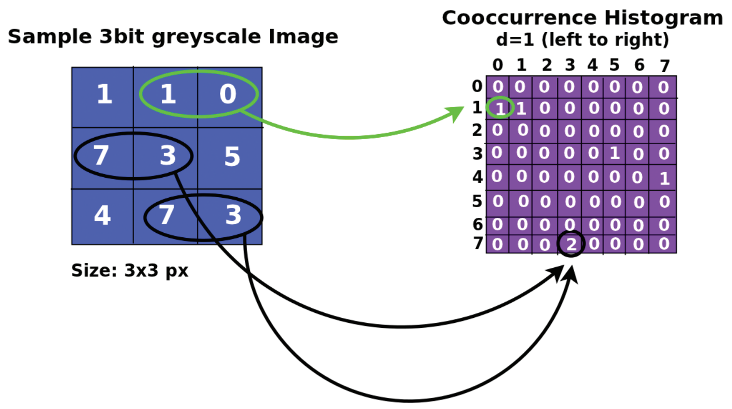

Fig. 1. High level view of algorithm.

3. RECEPTIVE FIELD COOCCURRENCE HISTOGRAM FOR OBJECT DETECTION

A Receptive Field Histogram is a statistical representation of the occurrence of several descriptor responses within an image. Examples of such image descriptors are color intensity, gradient magnitude and Laplace response. If only color descriptors are taken into account, the histograms produced are called regular color histograms.

A Receptive Field Cooccurrence Histogram (RFCH) is able to capture most of the geometric properties of an object. Instead of just counting the descriptor responses for each pixel, the histogram is built from pairs of descriptor responses. The pixel pairs can be constrained based on, for example, their relative distance. In this way only pixel pairs located within a maximum certain distance, dmax, are considered. Thus, the histogram represents not only how common a certain descriptor response is in the image but also how often certain combinations of descriptor responses occur close to each other. In other words, an RFCH is a representation of how often pairs of certain filter responses and colors lie close to each other in the image. This results in a representation of the image in which most of the geometric information is preserved thus allowing for more accurate object recognition. Figure 1 presents the concept of the cooccurrence histogram, of a 3bit (8-color) greyscale image, where we search for cooccurrences from left to right with dmax = 1.

3.1. Receptive Field Cooccurrence Histogram for Object Detection

One of the main advantages of this algorithm is that it can work with numerous different types of image descriptors such as Color, Gradient magnitude, Laplacian, Gabor as well as any mixture of them. As it is has been proved in Ekvall and Kragic [2005] for object recognition the optimal choice is to utilize rotationally invariant image descriptors such as Color, Gradient magnitude and Laplacian descriptors and the actual choice can depend, among others, on the image characteristics.

3.2. Image Quantization

When utilizing histograms in the recognition process, the computational complexity of the algorithm increases exponentially with the dimensions of the histogram. In order to alleviate this problem the algorithm firstly clusters the input data, so as to reduce the histogram dimensions. Hence, by altering the number of clusters the histogram size may be controlled. The cluster centers (N) have a dimensionality equal to the number of image descriptors used. The adopted algorithm is using the K-Means clustering algorithm [MacQueen 1967] for the dimension reduction. In particular, after quantization, each object ends up with its own cluster scheme which is used together with the RFCH which has been calculated on the quantized training image. When searching for a certain object in a scene, the whole image is quantized with the cluster scheme that has been applied in the quantization of this search object.

3.3. RFCH-Based Object Detection

After the clustering step, the algorithm creates the object's cooccurrence histograms in the clustered descriptor space. In the testing phase the image is scanned using a small search window and the RFCH of the window is calculated at any given instance. In each scan the RFCH of the window is compared with the object's RFCH.

The similarity between two normalized RFCHs is computed as the histogram intersection:

$$\mu(h_1, h_2) = \sum_{n=1}^{N^2} \min\ (h_1[n], h_2[n]),$$

where $h_i[n]$ denotes the frequency of receptive field combinations in each discrete interval (bin) n for image i, when quantized into N cluster centers. The higher the value of the $\mu(h_1, h_2)$ the better the match between the histograms, and as a result, the better the match between the search object and this specific part of the image.

As a summary the algorithm works in two phases and performs the following steps in order to detect a certain object in an image:

Training Phase:

—Extract Features from the Object
—Calculate Feature Clusters
—Quantize Object
—Create object's RFCH

Detection Phase:

—Quantize image with Object's cluster scheme
—Calculate the RFCH for a small image window (for all possible image windows)
—Match Object and Image RFCH with histogram intersection (for all windows)
—Report the best match

4. HARDWARE/SOFTWARE PARTIONING

In order to create an efficient embedded system, we first analyzed the RFCH application so as to be able to perform the optimal hardware software partioning. In order to profile the software implementation of the algorithm we have used Intel's VTune[3] Amplifier XE 2011. The profiling was performed on an Intel SU7300 Dual Core ULV CPU working at 1.3GHz since this is a low-power CPU found in embedded multimedia systems (e.g., MSI WindBox)[4]. The same profiling results were also produced when executing the same code on an ARM placed in a Gumstix device[5] [GUMSTIX]. All of our experiments were conducted using the original optimized software provided by the inventors of the underlying algorithm [Ekvall et al. 2005] along with images from the most widely used Image Database, the CVAP[6] Object Detection Image Database, which we have rescaled to 640×480.

[3]Intel VTune, Ampiler XE Documentation.
[4]MSI WindBox III (MS-9A35) Core2Duo Fanless Embedded System.
[5]http://www.gumstix.com/.
[6]CVAP Object Detection Image Database, http:/www.nada.kth.se/ekvall/codid.html.

After running various tests combining different scenes and objects we concluded that, functions CalculateClusters(), ClusterFeatures() and CalculateRFCH() are taking 97.8% in average (and at least 96%) of the total execution time. By making the above 3 functions faster, we can significantly improve the performance of the overall algorithm; according to Amdahl's Law the maximum theoretical speedup in that case is 45x. We have also analyzed the interconnection needed if those functions are implemented in hardware and the rest of the functions for Feature Extraction (i.e., Create Image Gauss, Create BW Image etc) and Histogram Intersection (i.e., MatchRFCHs) are executed in the CPU and found it to be minimal as described in the next section. In particular, even though CalculateRFCH takes only 8% of the total time we implemented it in hardware so as to minimize the data transactions between our hardware modules and the embedded CPU.

Another important reason for implementing the Feature Extraction as well as the Histogram Intersection Algorithms in software is that it allows us to easily change those parts of the algorithm depending on the image characteristics (e.g., change the actual descriptor used) thus heavily increasing the applicability as well as the accuracy of the end system. Before we have actually implemented those functions in hardware, and in order to be able to fully dimension the problem, we have also measured the computational complexity of those 3 functions.

CalculateClusters. This function implements an iterative version of the K-Means algorithm, and it has been identified as the major hot-spot during the profiling procedure. The computational complexity of the above algorithm is $\mathbf{O(nfNT)}$ where $\boldsymbol{n}$ is the number of samples, $\boldsymbol{f}$ is the number of features, $\boldsymbol{N}$ is the number of clusters and $\boldsymbol{T}$ is the number of iterations until convergence.

ClusterFeatures. This function is responsible for the quantization of the image according to the pre-calculated cluster centers. The function has a complexity of $\mathbf{O(nfN)}$. The function takes as input the Feature Array and the Cluster Point Array and produces the Binned Image Array.

CalculateRFCH. The complexity of this function is approximately $\mathbf{O(nd^2)}$, where $\boldsymbol{n}$ is the Image Size and $\boldsymbol{d}$ is the maximum distance ($\mathbf{d}_{\max}$).

5. SYSTEM ARCHITECTURE

Moving to the implementation of the previously identified hot-spots of the presented scheme, we have decided to use a Xilinx Virtex-6 FPGA, which resides on the ML605 Xilinx Evaluation Board[7] [UG534]. Those designs have been implemented manually in VHDL and we have synthesized, mapped, placed and routed them using Xilinx ISE 13.3.

The main concept of our approach is that the three HW accelerated functions are placed one next to the other in such a way so as to minimize the data being sent from and to the CPU executing the rest of the functions in software. By adopting this approach we don't need to have 3 independent data transactions to the reconfigurable fabric which will trigger a significant communication overhead. In the proposed architecture, demonstrated in Figure 2 we transfer data from the CPU to the FPGA practically only when loading the Feature memory; then our hardware modules process those data until the complete image slice is fully processed. The loading time for the feature memory is very low and up to about 0.05msec for the 640×480 images (for a typical 100MHz bus as in [PCI BUS][8]) while the software processing does not need more than 1msec at any experiment conducted. Moreover, as it is demonstrated in the next section, we have utilized a double buffering scheme in order to pipeline the loading

[7]Xilinx, UG534 ML605 Hardware User Guide.
[8]Datasheet PCI Bus Bridge Memory Controller, 100 MHZ

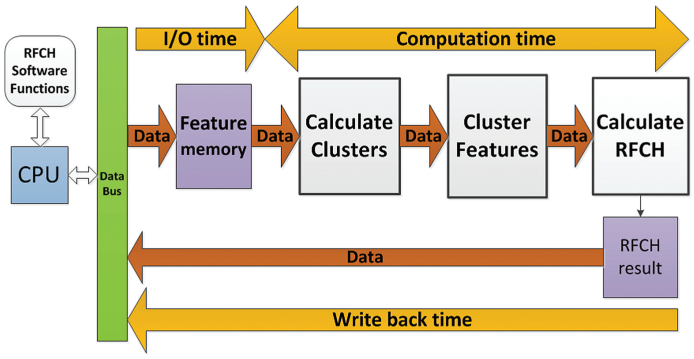

Fig. 2. High level architecture.

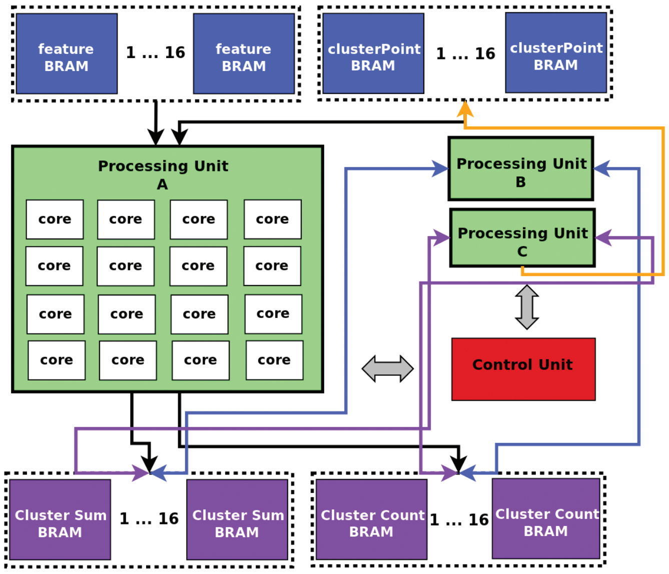

Fig. 3. Calculate Cluster HW implementation.

and software processing time of the different slices of every image with the actual hardware processing of them. The write back time is negligible as the only thing we need to transfer is the RFCH result which is a 80x80x11 bits datum and this is only needed once for each complete image. 5.1 also demonstrates the data flow through the implemented modules.

5.1. Calculate Clusters Module

The CalculateClusters function is responsible for the clustering of the features array, and it mainly implements an iterative version of the K-Means algorithm. It is applied during the training phase of the algorithm and it works in 3 distinct phases: Phase 1 and 2 perform the actual calculations while phase 3 updates the cluster centers. The overall micro-architecture of this module is demonstrated in Figure 3. Processing Unit A (PUA) is calculating the cluster centers and it utilizes 16 cores. Each core can perform the necessary processing on a small image slice of size 640 $\times$ 2 (which consists actually of 2 lines of the feature image). The whole feature image (640 $\times$ 480 $\times$ 7 features-8bit)

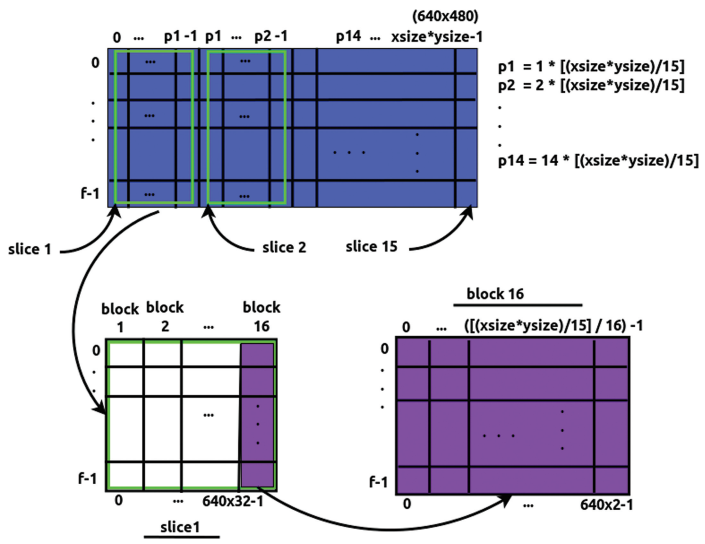
(640x480)
0 ... p1 -1 p1 ... p2 -1 p14 ... xsize*ysize-1
0
f-1
p1 = 1 * [(xsize*ysize)/15]
p2 = 2 * [(xsize*ysize)/15]
p14 = 14 * [(xsize*ysize)/15]
slice 1
slice 2
slice 15
block 1
block 2
block 16
640x32-1
slice1
block 16
0 ... ([(xsize*ysize)/15] / 16) -1
640x2-1

Fig. 4. Feature Array decomposition.

is pre-segmented by the software and it is sent in slices (i.e., 32 lines per processing cycle) to the hardware module. As a result the Feature Array/Memory utilized in each core is 640 × 2 × 7 bytes, and with the proposed configuration, each module can process 16 feature image blocks (640 × 2 × 7) simultaneously. In each processing core we have to execute a critical multiply-and-accumulate (MAC) operation; in order to speed up this function we have utilized a pipelined Digital Signal Processor (DSP) built-in core, found in those Xilinx Virtex6 devices.

When the processing is completed, Processing Unit B (PUB) sums all the intermediate results produced by the 16 cores. When the sum is fully calculated, PUB triggers Processing Unit C which is responsible for updating the cluster centers as well as the clusterPoint array; the latter is also split into 16 slices. The clusterPoint array holds the calculated clusters information needed for the clusterFeatures module as it is described in the next paragraph.

5.2. Cluster Features Module

This module implements the image quantization task and it also utilizes 16 parallel cores. The inputs of this module are a) the feature array and b) the clusterPoint array calculated from the CalculateClusters module. Concerning how the data are decomposed and processed in parallel the exact same technique with the one described in the last section is applied.

In particular, we used images of sizes 640 × 480 and utilized 7 distinct features. This means that our feature array is equal to 640 × 480 × 7(×8bits) = 2.15 Mbytes which cannot fit in the on-chip RAM. In order to be able to load the feature array on-chip, while also processing a sub-part of it, we have split it in 15 slices; in this way we load a certain slice to the FPGA while simultaneously we process the previous slide. Those 15 slices are of size 2.15/15 = 0.13 MB each. In that way in order to process the whole feature array, we have to process 15 slices.

Then we split further the on-chip slice into 16 blocks with size (640 × 480 × 7)/240 = 8.75 KB each and then we pass each block to a distinct processing core (i.e we utilize all 16 parallel processing cores in order to process one slice). The above procedure is depicted in Figure 4.

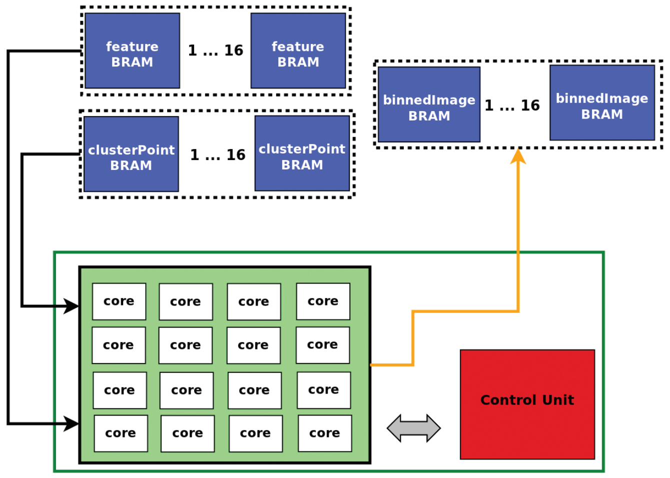

Fig. 5. Cluster Features module.

The high level architecture of the cluster feature module is demonstrated in Figure 5. Each slice of the feature image array (640 × 32 × 7) is fitted in 16 distinct feature Memories (RAMs). Each Feature RAM can hold a block equal to 640 × 2 × 7 (using 8-bit color). The clusterPoint array of size 7 x 80 is initially loaded into the 16 distinct clusterPoint RAMs each of size 7 × 80 × 8 bits. The actual processing comprises of each core quantizing an image block of 640 × 2 pixels (i.e., 2 lines) as follows: The first core quantizes the image pixels 0 to 1279, the second core quantize the pixels 1280 to 2559 and so on. Again we process each image block simultaneously thus fully utilizing the 16 distinct processing cores. Each core also has a dedicated BRAM for storing the results. This BRAM is the binnedImage memory with a total size of 640 × 2 × 8 bits. Each core performs the same MAC operation, as in the CalculateClusters case, so we also utilize here a fully pipelined built-in DSP core; in total we need 16 DSP slices to support this module.

When all the cores have completed the corresponding processing, an image slice has been fully quantized and the results reside in the 16 binnedImage RAMs. The control unit for the ClusterFeatures module is quite simple. It just monitors when all the cores have finished their processing and then it loads the next block.

5.3. Calculate RFCH Module

This module calculates the required receptive fields' cooccurrence histograms. The overall architecture of the module is presented in Figure 6. The module utilizes 8 processing cores as it is less demanding, in terms of processing time, than the other two modules.

The RFCH is calculated based on the BinnedImage data. The binnedImage array is of size 640 × 480 and, as we presented in the last section, it is the quantized version of the image. In order to calculate the RFCH for each binnedImage slice of size 640 × 32 we need to process the data coming from two continuous blocks; the additional block/slice is needed since we need 4 extra lines in order to serve the dmax = 4 condition (i.e., each core should look up to 4 lines ahead thus utilizing the data of the next slice). The last block of the current slice is paired with the first block of the next slice in order to keep the dmax condition valid between image slices. The above procedure is shown in the Figure 7.

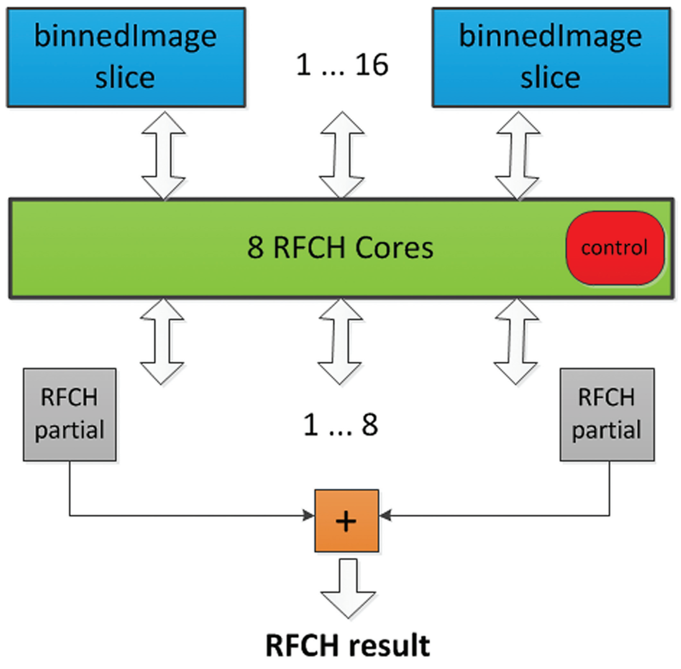

Fig. 6. Calculate RFCH module organization.

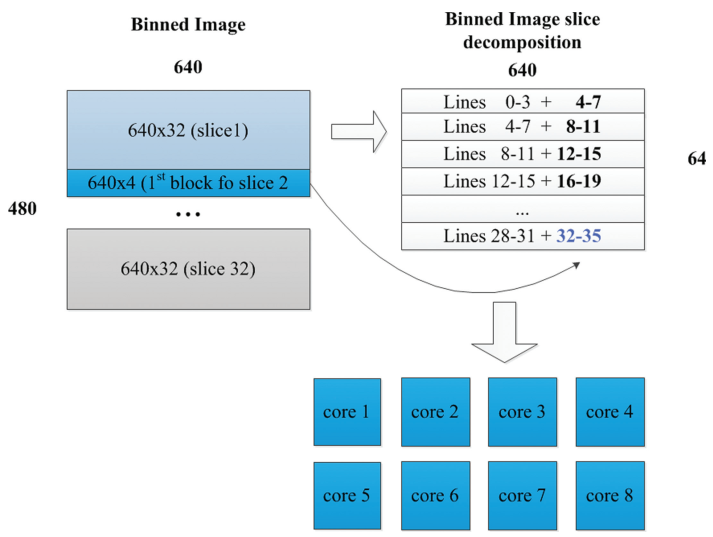

Fig. 7. Binned Image Array Decomposition.

Each of the eight cores calculates the cooccurrences of a binnedImage block of size 640 $\times$ 4. This means that the 8 cores together can calculate the cooccurrences of a binnedImage slice equal to 640 $\times$ 32. As previously mentioned, in the current version of the system, the cooccurrenses are calculated based on a specific value of dmax (dmax = 4) which gives very high accuracy as described in Ekvall and Kragic [2005]. If, for any reason, we decide to use a larger value for dmax, we will have to utilize more memory since the RAM blocks needed, in the presented architecture, are equal to 2dmax. Each processing core maintains a dedicated memory block for its output in which a partial RFCH is stored; the required memory size is 80 $\times$ 80 $\times$ 11 bits. In order to calculate the RFCH of the whole binnedImage of size 640 $\times$ 480 we have to process 15 binnedImage slices of 640 $\times$ 32 each and update the corresponding memories. After we process all the slices we store the results in the corresponding 8 RFCH memories; then those stored results are summed in order to form the final RFCH for the image and this is the end-result sent to the CPU for further processing.

The control of this module is quite complicated since it contains various termination criteria and jump conditions involved in the cooccurrence histogram computation. In particular it is implemented as a 14-state Finite State Machine and it has been fine-tuned in order to achieve the maximum possible clock rate.

5.4. Verification of the Complete System

In order to verify the correct functionality of our approach and mainly the feasibility of our complete system being implemented in a single chip, we integrated all the hardware modules as well as the software code in an embedded design platform based on a Virtex 6 FPGA board[9] . For this proof of concept we have selected the on-chip Xilinx Microblaze[10] softcore processor for the software execution, connected to the AXI / AMBA bus. We used the AXI Block Ram Controller provided by the Xilinx EDK 13.3 environment to make all the necessary data transfers between the CPU and the reconfigurable modules. The controller occupies the first port of a dual port configuration RAM leaving the second port to run on the custom-made hardware side at a different clock domain (i.e the Frequency of the Microblaze CPU was fixed at 150 MHZ while our custom hardware works at 350MHz).

We also used the AXI IP Interface modules (IPIF) to utilize user specific software accessible registers. In that way we were able to control and monitor the status of our custom accelerator from the software side. In this particular platform a soft core CPU was the only choice as the Virtex 6 family is not equipped with any hard core CPUs (like the ARM found in the already announced Virtex-7 FPGA under the codename Zynq-7000[11] [DS190]). Unfortunately the Microblaze is performing poorly when implementing the software functions and as a result it degrades the overall performance of the system (to an overall speedup when compared with the reference Intel CPU, of about 3.5x); thus we used it only for verification and feasibility purposes. We also used the compact flash peripheral in order to load the test images and verify the results. Figure 8 shows an overview of the proof-of-concept embedded platform.

The Microblaze talks to an external DDR SDRAM in which we place images from a well known image database [CVAP][12] as well as real-world multimedia data (i.e., real-world videos). The Microblaze executes the part of the RFCH algorithm that we mapped to the CPU, when performing the hardware/software partioning, while it additionally controls all the system's peripherals. In this initial single-chip approach the

[9]Xilinx, UG534 ML605 Hardware User Guide.
[10]UG081 Microblaze processor reference guide.
[11]Xilinx, DS190, Zynq-7000 Extensible Processing Platform Overview.
[12]CVAP Object Detection Image Database, http://www.nada.kth.se/ekvall/codid.html.

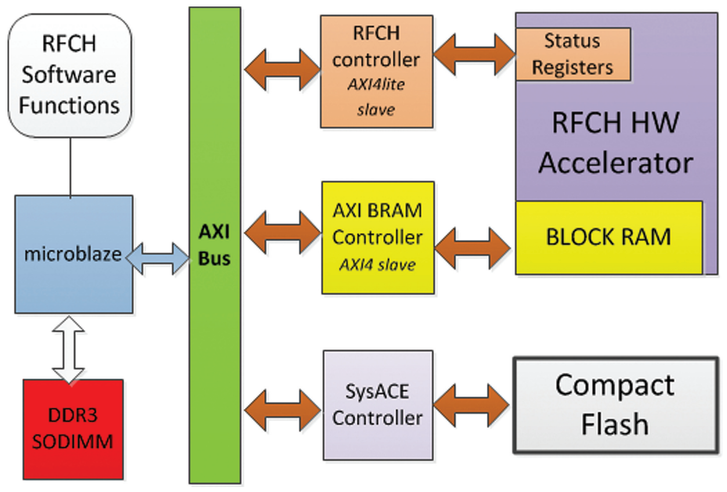

Fig. 8. Micro-Architecture of our proof of concept single-chip approach.

Table I. Hardware Cost on a Virtex-6 VLX75T device (16 Cores C.Clusters and Cluster Features, 8 cores for Calculate RFCH)

Module	Slice Registers	Slice LUT	Block RAM	DSP Slices
Calculate Clusters	26,082/93,120 28%	23,691/46560 50%	112/312 35%	17/288 5%
Cluster Features	4,176/93,120 4%	5,840/46,560 12%	112/312 35%	16/288 5%
Calculate RFCH	1,066/93,120 1%	3,465/46,560 7%	16/312 5%	0/288 0%
Total	31,324/93,120 33%	32,996/46,560 70%	240/312 76%	33/288 11%

functions that were implemented in hardware have been called by the corresponding software drivers independently (i.e., 3 different calls, one for each hardware module) since that allowed for much faster and easier debugging of the hardware cores. The RFCH controller is the hardware module supervising the interconnection between the Microblaze and the RFCH hardware accelerator, and it is attached as an AXI slave IPIF module. This module is basically an AXI4lite slave[13] which provides a bi-directional control/status interface between the RFCH hardware accelerator and the AXI bus. After executing the complete algorithm in the specified FPGA platform we compared the end results residing in the DRAM with those triggered by the pure software solutions and they were identical.

It should also be stressed that in numerous current embedded devices the image size is 640 × 480. As a result our prototype hardware implementation, as well as the performance results of the next sections, are all based on that image size. However, our system can seamlessly (i.e., with a simple change in a couple of control modules) process any images up to 1920 × 1080 (High Definition-HD) pixels; and this is an analysis that will soon been incorporated even in mobile embedded multimedia systems. The only difference will be that those HD images will consist of more slices and therefore more time will be needed in order to process a complete such image.

[13]Xilinx, DS768AXI Interconnect.

Table II. Hardware Cost on a Virtex-6 VLX130T device (32 Cores C.Clusters and Cluster Features, 8 cores for Calculate RFCH)

Module	Slice Registers	Slice LUT	Block RAM	DSP Slices
Calculate Clusters	5,4642/160,000 34%	48,121/80,000 60%	112/528 21%	32/480 6%
Cluster Features	8,769/160,000 5%	11,986/80,000 15%	112/528 21%	32/480 6%
Calculate RFCH	1,066/160,000 1%	3,465/80,000 4%	16/528 3%	0/480 (0%)
Total	64,477/160,000 40%	63,572/80,000 79%	240/528 45%	64/480 13%

6. HARDWARE IMPLEMENTATION

The Table I shows the utilization of each function on both a Virtex-6 VLX75T and a Virtex-6 VLX130T device. The total utilization of the three functions leaves room for more cores to be implemented in both devices if needed.

As demonstrated in Table I by using small image slices (as described in Section 5) we minimized the RAM usage while exploiting high levels of parallelization. The small footprint of our design, in terms of Slice LUTs, is giving as the choice to double the cores in the current design and assign 1 image line per core with neither increasing the amount of utilized Block RAMs nor the control complexity. The utilization for this system is demonstrated in Table II. This approach almost doubles the performance of our system, as the next section clearly demonstrates, while making the LUT-to-BRAM factor very close to the LUT-to-BRAM factor of the available resources in the middle Virtex-6 FPGAs.

Since our system can process different images totally independently of one another, by adding more memory and logic resources we will trigger a further speedup and we expect that the performance will scale linearly with the device utilization. In order to further support this case we have placed two instances of the system described in the last paragraph to a Virtex-6 VLX240T (the total utilization for both BRAMs and Slices was close to 95%) and since the I/O was not the bottleneck, we managed to eventually double the supported bandwidth. As will be explained further in the next section, even if the loading time is doubled due to the sharing of the bus bandwidth among the two cores it is still far less than the processing time. This fact in addition with a double buffering scheme eliminates any I/O overhead. However, since we mainly focus on low-energy, relatively low-cost multimedia embedded applications, the performance results demonstrated in the next sections, cover the implementations of our system on both a Virtex-6 VLX75T and a Virtex-6 VLX130T low-cost devices connected to a low-cost ARM core.

Since our aim was to implement a low-power embedded multimedia system, special care has been taken so as to reduce the overall power consumption of our novel device. In particular, we implemented, in each module, numerous parallel simple cores working at smaller speeds instead of creating complex cores with large and complicated control working at higher speeds. In order to achieve that we decomposed the actual processing, in each module, in smaller parts; special care has been taken so to reduce the, triggered by this decomposition, overhead in terms of memory repetition and/or memory reads/writes. The resulted reduction in the energy consumption by adopting this technique was up to 17%.

We have also introduced the pipeline of Figure 2 so as to minimize the data moving to/from the CPU; for example, by implementing the “Calculate RFCH” module in

Table III. Typical Speedup Achieved for Processing of a CODID image - 16 cores

Module	SW Time (sec)	HW Time (sec) -	Speedup[1]
Calc.Clust./obj	0.3200	0.0170	18.8
ClusterFeat. /obj	0.5000	0.0175	28.8
ClusterFeat./scene	1.8100	0.0927	19.5
Calc.RFCH/obj	0.0500	0.0074	6.72

[1] Without the I/O time and CPU processing time.

hardware we heavily decreased the intercommunication needed with the CPU and that resulted to a 12% reduction in the overall power consumption.

Additionally, with the aim of the FPGA design tools, we have placed the logic as close as possible to the Memory Banks therefore minimizing the necessary routing while we have also hand-designed from scratch all the processing cores (so as to eliminate any unnecessary silicon), instead of using the ready-made IP cores provided by third parties; those techniques reduced the overall power consumption by an additional 11%.

Moreover, we have utilized a double buffering scheme, as demonstrated in the next section, which, together with the introduced pipeline of Figure 2, allow us to utilize all the hardware resources at any given time. Since, certain hardware units consume power even when they are idle (i.e., do not produce any useful results), by effectively removing any idle states we decreased the actual energy consumed when the complete application is executed by a further 20%.

7. EVALUATION AND PERFORMANCE RESULTS

In this Section we demonstrate the performance of our FPGA-based hardware and we compare it with the optimized single threaded software provided by the inventors of the algorithm when executed on a state-of-the-art low power Intel SU7300 dual-core ULV CPU @ 1.3GHz; such a CPU can typically be used in an embedded multimedia device.

The configuration of the embedded system demonstrated is as follows:

—Number of Calculate Clusters cores: 16/32
—Number of Cluster Features cores: 16/32
—Number of Calculate RFCH cores: 8
—Image Size: 640 × 480
—Number of Clusters(N): 80
—Number of features(f): 7
—Maximum distance (dmax): 4

Those configurations trigger the best performance-to-silicon results while the selected algorithm's parameters trigger the optimal accuracy-to-processing ratio based on Ekvall and Kragic [2005]. The modules are clocked at 350 MHz on both FPGAs. Table III and Table IV show the average time needed for each function in order to process and classify a common object/scene of the CODID dataset [CVAP] in our FPGA-based system. The performance variance when processing different images is negligible (in terms of processing time for each function invocation). The speedup triggered over the optimized software is about 20–60 times for the first two functions, whereas for CalculateRFCH is only 6.7 times; however this last function takes less than 10% of the total processing time and, as it has been described in Section 4, the

Table IV. Typical Speedup achieved for processing of a CODID image - 32 cores

Module	SW Time (sec)	HW Time (sec)	Speedup[2]
Calc.Clust./obj	0.3200	0.0086	37.6
ClusterFeat./obj	0.5000	0.0088	57.6
ClusterFeat./scene	1.8100	0.0463	39.0
Calc.RFCH /obj	0.0500	0.0074	6.7

[2] We keep the same 8 cores between the two implementation sice each such core must be able to process the next 4 lines (e.g., 32 lines in total), based on the fact that we have selected the distance parameter to be equal to 4, and we process in both cases 32 lines at any given time.

main reason for implementing it in the reconfigurable hardware was to minimize the amount of data that should be sent to and from the CPU. It should be noted that in those numbers the I/O overhead has not been taken into account but this has been separated from the critical path due a double buffering scheme we have implemented; this scheme is analytically described in the next paragraph.

Figure 9 demonstrates our proposed double buffering scheme which isolates the intercommunication overhead, between the CPU and our hardware modules, from the critical path. In order to measure the communication cost we calculate the transactions needed in order to fully load the Feature memory with 144Kbits, assuming a typical 32bit bus with 4-byte burst mode clocked at 100MHz. In a typical AMBA bus those 4480 transaction needed a total of less than 0.05msec. Moreover, the measured 0.002msec write back time is indeed very low as we need to transfer the RFCH result only 1 time per image (not per image slice). The internal hardware is always clocked at 350Mhz. Since our processing time can take, in the very best case, 1.39msec and we use double buffering in the Feature memory (i.e., when we load a certain slice in one part of the Feature memory we simultaneously process the previous slice from the other part), our critical path consists only of the actual processing on our cores.

By trying various dataset configurations with different numbers of objects and different scenes (thus changing the number of times each module processes a certain slice of the image), the performance triggered is listed in Table V.

The slight decrease when we go from 1 scene to 10 scenes is due to the fact that the time consumed by the ClusterFeatures and the CalculateRFCH functions is growing while the time consumed by the CalculateCluster stays stable between the two cases. This is because Calculate Clusters is executed only during training; in the testing case only the two other functions are executed (several times each one), and as CalculateRFCH has a lower speed up over the others it slightly decreases the overall performance. The average frame per second number demonstrated includes both the detection and the training of the subjects in different scenes. Moreover, it should be noted that the variance between the different experiments was insignificant.

Those results clearly demonstrate that our system which can trigger a speed of about two fps can be utilized in a state-of-art multimedia system whereas no existing object recognition system running in software can be utilized even at those relatively low object recognition speeds.

Moreover, our system is significantly less power and energy hungry than the conventional software approach. The specified Intel CPU consumes, on average, about 8W, when executing the specified three tasks on a single core and the other core is disabled,

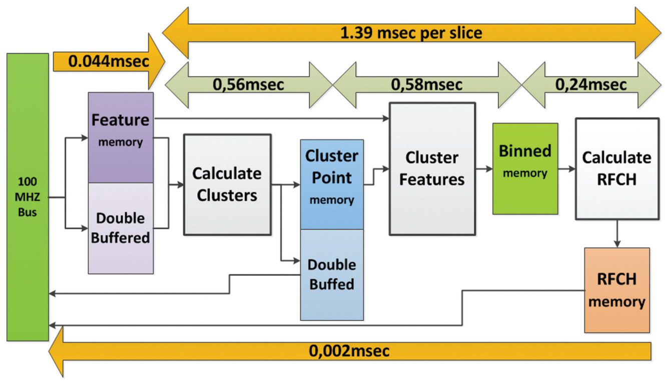

Fig. 9. Timing Distribution when a 32bit Bus clocked at 100Mhz is used for I/O.

Table V. Overall Speedup at 350 MHz Including the Software Execution

Configuration	Overall Speedup at 350 MHz (16 cores)	fps	Overall Speedup at 350 MHz (32 cores)	fps
3 objects - 1 scene	11.5	1.1fps	19.65	1.87 fps
10 objects - 1 scene	15.9	1.2fps	27.12	2.04 fps
10 objects - 10 scenes	15.1	1 fps	25.67	1.7 fps

whereas its maximum power consumption is up to 10W[14]. All the software power measurements are based on Intel's Power Gadget 2.0 tool. On the other hand, our larger Virtex-6 VLX130T consumes about 2.7W on average (with a peak of less than 3.5W) (based on the Xilinx XPower Estimator (XPE) [UG440])[15] while it is also from 20 to 25 times faster than the CPU. As a result, the total energy consumed by our FPGA system is 60 to 80 times less than that of the existing single-threaded, purely software based approach.

In order to investigate how our algorithm behaves on a multi-core, multi-threading environment we have also developed an openMP version of the original algorithm which we have hand-optimized so as to get the best possible parallelism. As clearly demonstrated in [Youssef et al. 2010] and [Saravanan et al. 2011] the power consumption is increased when more cores and/or more threads are utilized in a multi-core, multi-threaded system. As a result we investigated if the increased speedup triggered by utilizing numerous cores and threads can counterbalance the increased demands in terms of power; if this is the case the total energy needed for the execution of our application can be reduced.

In our experiments we have utilized two Intel multi-cores each with a nominal power consumption of 80W per CPU. "*Talos*" is a dual processor machine hosting a 4-core CPU on each socket (Intel® Xeon® Processor E5620 with 12M Cache, 2.40 GHz, 5.86 GT/s Intel® QPI) while it also supports hypertheading (i.e., it has 8 physical cores while it can support 16 active threads in total). The other machine "*Iraklis*" hosts two CPUs

[14]ULV. http://ultrabooknews.com/2012/02/07/intel-core-ulv-vs-lv/. SU7300. http://ark.intel.com/products/42791/Intel-Core2-Duo-Processor-SU7300-(3M-Cache-1_3 0-GHz-800-MHz-FSB).
[15]Xilinx®, UG440 XPower Estimator User Guide.

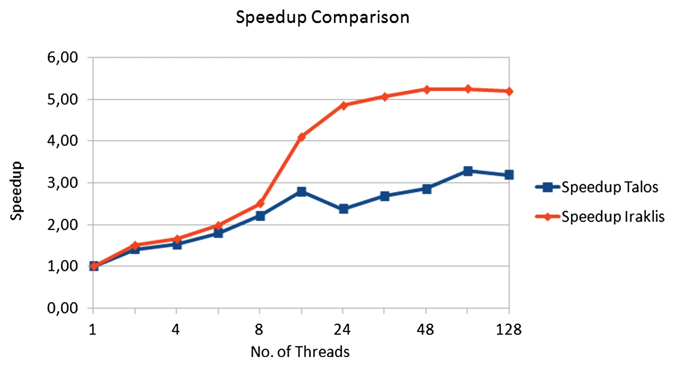

Fig. 10. Speedup vs Number of Threads for our experiments.

with 4 cores each that do not support hyperthreading (Intel® Xeon® Processor E5430, 12M Cache, 2.66 GHz, 1333 MHz FSB).

In order to investigate how the system behaves when the number of utilized cores and threads increases we executed a number of experiments each exploiting a different number of threads. The speedup is expressed relative to the performance of the single-threaded software; our aim is to demonstrate how efficiently the reference software can be parallelized in terms of both power and performance. As Figure 10 clearly demonstrates, the maximum speedup achieved in our experiments is about 5x; this speedup is triggered when we utilized more than 32 threads which were executed on the 8 available cores. From those results, it is clear that the multi-threaded software is faster than the single-threaded one, since the application very frequently moves data from the memory to the CPU, and the utilization of multiple threads hides this memory latency.

Another interesting conclusion is that *Iraklis* performs much better than *Talos* (as shown in Figure 11) although it features an older CPU without hyperthreading while both systems feature the same amount of cache (12MB per CPU, 24MB in total). The difference in the performance comes from the fact that the memory sub-system of *Iraklis* is faster than that of *Talos* due to a faster CPU-memory interconnect system.

Moving to the power consumption both CPUs consume very close to their nominal power (i.e., 80W) when triggering the maximum speedup (i.e., 5x for *Iraklis* and 3x for *Talos*), whereas the power consumption in all the experiments was more than 60 W. As a result the processors consume about 150W in total while processing, in the best case, a single CODID image every about 1.7 sec. When comparing those numbers with the corresponding ones achieved by our FPGA system (2.7Watts for 0.5 sec per CODID image) our novel device is about 188 times better in terms of energy consumption.

At the same time the recently introduced 8-core power efficient Xeon CPU (E5-2448), which is optimized for embedded applications, has a nominal power of 70W while it includes 8 cores working at 1.8GHz each. So even if this new 1.8Ghz device has the same performance with our reference 2.66GHz multi-core system, our FPGA-based device will still be 88 times more energy-efficient than this state-of-the-art Intel multi-core.

The recently introduced ARM-based Cortex-A9, implemented in the newest CMOS technology, has a processing time which is very close to that of the reference Intel CPU; on the other hand this power-optimized ARM consumes about 400mW per core[16]. As a result our novel reconfigurable system is from 10 to 12 times more power efficient than

[16]http://www.arm.com/products/processors/cortex-a/cortex-a9.php.

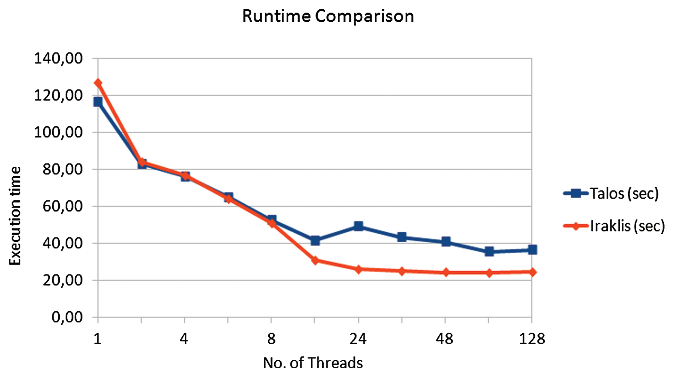

Fig. 11. Execution time (sec) for 14 CODID images vs Number of Threads.

even this state-of-the-art ARM CPU when executing the exact same object recognition scheme. Also looking at the future ARM-based multi-cores, they are expected to be "up to an order of magnitude more power efficient" than the current Intel-based ones [Jag 2009]; as a result, and based on our real-world measurements on the Intel multi-core, our FPGA-based device will still be at least 8 times more energy efficient that those future power-optimized ARM-based multi-cores.

8. LIMITATIONS AND FUTURE WORK

Although our novel embedded approach can achieve a significant speedup over the conventional purely-software approach, the single-chip validation platform we have used proved to be inappropriate for high performance embedded multimedia systems; this is due to the fact that the Microblaze was very slow when executing the specified double-precision floating point software routines. In particular, Microblaze's low clock rates combined with a slow Floating Point Unit, degraded the overall performance and finally undermined the accelerated functions effect. In the near future we are going to upgrade our system to the upcoming Zynq FPGA[DS190], which employs a hard core ARM CPU. Our system is already fully compatible with any ARM-based embedded framework as it is built around ARM's AXI bus which is also the standard bus for the Zynq Platform. The performance of the software functions is expected, based on our initial measurements on a stand-alone Cortex A8 CPU, to be equal with that of the Intel processor used as a reference, therefore the final single chip solution will exploit the full speedup of our hardware implementation. Furthermore, we are about to convert the floating point arithmetic to fixed point arithmetic since, based on our experiments, the dynamic range of all the internal variables is limited; this will certainly increase even more the performance on both the software and the hardware sides.

9. CONCLUSIONS

This article describes a complete low-power embedded object recognition system that can support multi frames per second speeds and can efficiently be utilized in multimedia systems performing complex object recognition (such as fixed and mobile game consoles or tomorrow's smartphones). The system implements one of the most efficient Object Recognition Algorithms, the Receptive Fields Cooccurrence Histograms (RFCH) one. This system is highly flexible, since it is not fixed to a specific image size, but instead it is designed to support image sizes up to High Definition (HD), allowing it to be utilized in numerous distinct multimedia applications. Additionally, the number of features and the number of clusters that are supported are not fixed either, and they can be altered by just changing the software part of the system which offers even greater flexibility. Moreover, the programmability feature of the FPGA allows our system to be efficiently instantiated in numerous distinct multimedia applications; for example in a high-accuracy multi-object mode it can utilize our algorithm whereas the reconfigurable sub-system can be rapidly re-programmed to execute a simple face detection algorithm when such a demand is triggered by the end-application. The presented system, when implemented on a relatively low-cost Virtex-6 FPGA connected to an ARM, can be up to 27 times faster than the conventional software approach whereas it consumes about two orders of magnitude less energy than either a low-power CPU executing the exact same algorithm or the recently introduced low-power multi-cores that are optimized for embedded applications.

REFERENCES

Bhowmik, D., Amavasai, B. P., and Mulroy, T. J. 2006. Real-time object classification on FPGA using moment invariants and Kohonen neural networks, In *Proceedings of the IEEE SMC UK-RI 5th Chapter Conference on Advances in Cybernetic Systems*. 43–48.

Ekvall S. and Kragic, D. 2995. Receptive field coocurrence histograms for object detection. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 84–89.

Gao, C. and Lu, S.-L. 2008. Novel FPGA based HAAR classier face detection algorithm acceleration, In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 373–378.

Gentsos, C., Sotiropoulou, C.-L., Nikolaidis, S., and Vassiliadis, N. 2010. Real-time canny edge detection parallel implementation for FPGAs, In *Proceedings of the International Conference on Electronics, Circuits, and Systems*. 499–502.

Goshorn D., Cho, J., Kastner, R., and Mirzaei, R. S. 2010. Field programmable gate array implementation of parts-based object detection for real time video applications. In *Proceedings of the International Conference on Field Programmable Logic and Applications*. 582–587

Hadjitheophanous, S. Ttofis, C. Georghiades, A. S.; and Theocharides, T. Towards hardware stereoscopic 3D reconstruction a real-time FPGA computation of the disparity map. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*. 1743–1748.

He, W. and Yuan, K. 2008. An improved canny edge detector and its realization on FPGA. In *Proceedings of the 7th World Congress on Intelligent Control and Automation*.

Jag, B. 2012. Intel, ARM Battle for Microservers. *Processor Watch*, May 17.

Kyrkou, C. and Theocharides, T. 2011. A flexible parallel hardware architecture for Ada Boost-based real-time object detection. *IEEE Trans. VLSI Syst. 19*, 6. 1034–1037.

MacQueen, J. B. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, 281–297.

Nair, V. Laprise, P. O. and Clark, J. J. An FPGA-based people detection system. *EURASIP J. Appl. Signal Process. 7*, 1–15.

Saravanan, S., Chandran, S. K., Punnekkat, S., and Kothari, D. P. 2011. A study on factors influencing power consumption in multithreaded and multicore CPUs, *WSEAS Trans. Comput. 10*, 3, 93–103.

Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. 2011. Real-time human pose recognition in parts from a single depth image. In *Proceedings of the Computer Vision and Pattern Recognition Conference*.

Youssef, A., Zahran, M., Anis, M., and Elmasry, M. 2010. On the power management of simultaneous multithreading processors. *IEEE Trans. VLSI Syst. 18*, 8

Received June 2012; revised August 2012; accepted September 2012