Histogram of Oriented Gradients front end processing: an FPGA Based Processor Approach

Colm Kelly[1], Fahad Manzoor Siddiqui[2], Burak Bardak[2], Roger Woods[2]
[1]Thales UK, Belfast
[2] ECIT, Queens University, Belfast
colm.kelly@uk.thalesgroup.com, (f.siddiqui, b.bardak, r.woods)@qub.ac.uk

***Abstract*—The Field Programmable Gate Array (FPGA) implementation of the commonly used Histogram of Oriented Gradients (HOG) algorithm is explored. The HOG algorithm is employed to extract features for object detection. A key focus has been to explore the use of a new FPGA-based processor which has been targeted at image processing. The paper gives details of the mapping and scheduling factors that influence the performance and the stages that were undertaken to allow the algorithm to be deployed on FPGA hardware, whilst taking into account the specific IPPro architecture features. We show that multi-core IPPro performance can exceed that of against state-of-the-art FPGA designs by up to 3.2 times with reduced design and implementation effort and increased flexibility all on a low cost, Zynq programmable system.**

***Index Terms*—FPGA, Memory, DSP, Video Processing, HOG**

I. Introduction

Intelligent cameras used to be the preserve of factory inspection but are now increasingly used in public places and medical diagnostics. However, increased intelligence needs to be deployed near image capture to reduce the need to transfer large volumes of data to the host system, whilst preserving the perceived useful information of the captured video [1]. An example is in immersive computing where situational characteristics are decoded from the environment locally by sensors and either appended to, or sent to, a central monitor, instead of a processed video stream [2].

Video analysis is now predominantly automated requiring little operator involvement. In commercial video domain, algorithms are typically validated in software e.g. MATLAB and then implemented in FPGA hardware. There is reluctance to iterate the functionality due to perceived difficulty in making design changes as FPGAs are seen to offer high performance but with limited programmability compared to software. As system designers are usually not willing to invest this design effort, software implementations tend to be more prevalent.

In this paper, we propose applying a processor design approach to the design of a HOG algorithm. This is achieved by applying a reprogrammable FPGA-based, scalable, multicore SIMD processor approach called IPPro which has been developed within our research group**.** It has a reconfigurable datapath and memory structure and has low power consumption and is scalable to a size which is appropriate to the task in hand. This allows the designer to trade resources with performance to achieve the required solution in a suitable form. This paper gives details of how the entire HOG algorithm has been decomposed to allow optimization of the newly developed architecture benefits and limitations. The design involves generation of the IPPro instructions and scheduling. The work challenges the preconceptions around the flexibility of FPGA-based video processing by offering impressive FPGA performance but with the software programmability.

The paper is organized as follows. Section II outlines the HOG algorithm and defines how it is suited as a good test for our framework. Section III details our new FPGA framework and Section IV bounds the method of deploying an algorithm onto the framework. Section V evaluates our implementation and summarizes the papers findings.

II. HOG and its suitability to IPPro

Dalal and Triggs [3] illustrates that human form can be characterized rather well by the distribution of local intensity gradients or edge directions. Their algorithm converts the pixel intensity information in Fig. 1(a) to gradient information, as illustrated by the image of gradients for the pedestrian's heel (see Fig. 1(b)). Gradients have both magnitude and direction.

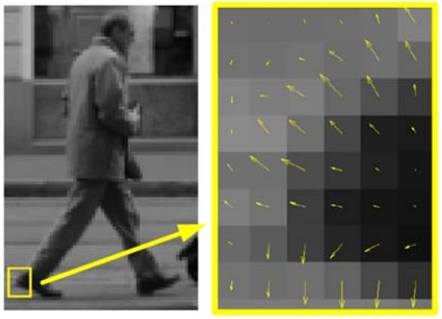

Fig. 1(a) Image of Intensities (b) Image of Gradients [4]

For this example, the detection window is fixed at 64 x 128 pixels and is divided up into small 8x8 pixel spatial regions or cells, which consist of histograms of gradient directions (edge orientations) over the 64 pixels of the cell as shown in *Fig. 2*. The algorithm shown in Fig. 3, comprises of 6 stages. In the 1st stage, the camera image is normalized for human vision and luminosity values are extracted from the RGB values.

Table 1: Image processing operations categorization [5]

Operation Type	*Domain*	*Definition*	*Examples*
Point	Spatial	Output depends on single input	Intensity change by factor, Negative image – inversion
Neighborhood/ Local	Spatial	Output depends on single input & neighbours	Convolution functions: Sobel, Sharpen, Emboss
Global	Spatial	Output depends on whole input image	Histogram (peak, top-hat, valley or well analysis), Thresholding of entire image
Geometric	Spatial	Output requires a whole input image	Rotate, Scale, translate, reflect, perspective & affine
Temporal/ Frame based	Frequency	Output calculated as a result of several input images	Frequency domain filtering, Target tracking, Motion Estimation

The gradient of each pixel relative to its surrounding pixels is calculated in the 2nd stage. The 3rd stage involves calculation of the magnitude of each *x* and *y* gradient pair and addition of the result to the relevant cell bin. Over an 8x8 pixel cell, a single 9-element vector is produced which is referred to as the histogram of gradients. In the 4th stage, blocks are generated by locally normalising groups of four cells i.e. 2x2 cells, in order to improve the invariance to illumination and shadowing. The resultant block vector has 36 elements. Collation of the blocks over the full detection window (7x15 blocks) is carried out in the 5th stage to produce HOG descriptors. In the final stage, a Support Vector Machine (SVM) classifier receives the 3780 (7x15x36) vectors and multiplies with its set weights to achieve the human detection chain.

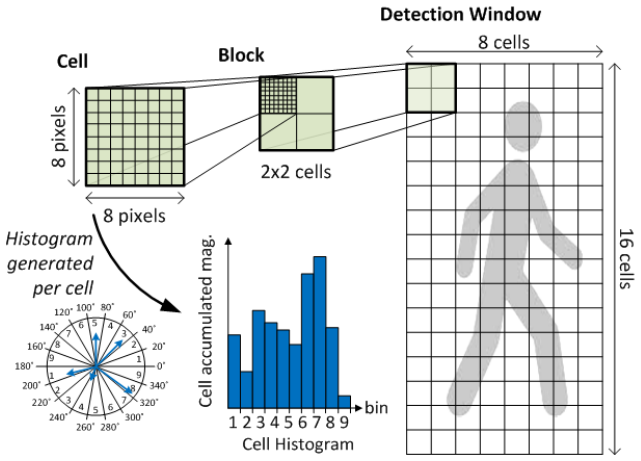

Fig. 2: Detection window divided into blocks and cells producing an array of histograms per window

In Benkrid et al.'s categorization of image processing operations which is given in Table 1 [5], it is shown how memory and computational requirements vary. As the HOG descriptor for a single HD frame requires all operations listed by Benkrid with the exception of Temporal/Frame based operations, it represents a good test case for assessing the architecture approach.

The results of our datapath analysis are illustrated in Fig. 3. It shows that the most intensive calculations are performed in the *Compute Gradients* and *Weighted Vote into Spatial and Orientation Cells* blocks, so their acceleration would provide the largest processing performance gain. The algorithm reduces the data transmission needs from 1.5Gbs^{-1} to a humble 30bs^{-1} (Fig. 3) which is important in remote surveillance applications where long range RF communications have very low bandwidth and minimal energy levels. Fig. 3 also indicates that computation hotspots are to be found between the 2nd and 3rd blocks where a peak bandwidth of 1990Mbs^{-1} is required. Conventional software approaches struggle to achieve real-time implementations with these computational hotspots but as we shall demonstrate, the IPPro approach can. Initially, this algorithm was the preserve of high performance computing systems but now have been realized in real–time FPGA implementations [6], [7], [8]. However these fixed implementations are created using hand written HDL which is a time consuming process and it is believed that this work will be the first to implement the HOG descriptor using a 'processor only' orientated FPGA framework providing FPGA performance in a truly programmable environment.

III. IPPro System Architecture

The IPPro processor has been developed as part of the Rathlin research project [9]. The IPPro is a soft processor solution that allows the full potential of the FPGA fabric to be exploited by utilizing the incorporated dedicated FPGA processing blocks and memory resource. Implementing processor architectures on FPGA is not new and earlier instances include a reconfigurable data path processor in an early Xilinx FPGA [10]. However, IPPro outperforms other examples giving a clock rate of 509MHz. Compared to earlier examples [9], the HOG algorithm was deemed to be a challenging exemplar for this emerging design due to its computational throughput. In order to achieve dependable high performance, we have firmly fixed the IPPro core architecture and interconnects between cores is also tightly controlled. This permits the device utilization and the overall system performance to be accurately

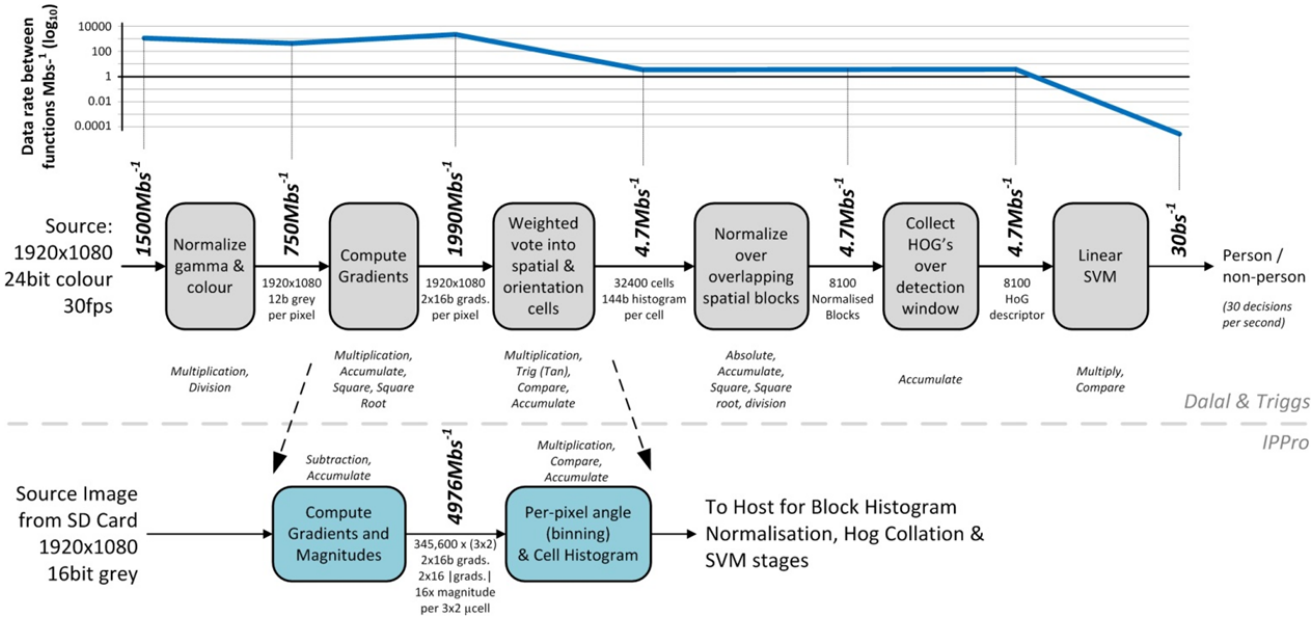

Fig. 3. Original and new definition of HOG algorithm [3] annotated with inter-function data rates and their characteristics

estimated prior to synthesis enabling more accurate exploration of the mapping of the algorithm to hardware. We have invested considerable effort up front to achieve the most from the Xilinx architecture and as such make it possible to outperform some hand written implementations with our soft programmable solution. The following text highlights some of the IPPro defining features but the reader is referred to work in [9] for more detail.

A. IPPro Core

The IPPro *core* is at the heart of the IPPro system and it is based around the FPGA signal processing unit, the Xilinx DSP48E. It has a number of key features:

- Two pipelining stages
- Different arithmetic and logical functions employed at each cycle period
- Optimal use of Dynamic Random Access Memory (DRAM) for registers and BlockRAMs (BRAMs) for Instruction Memory (IM)
- Data forwarding techniques employed to reduce latency during successive computations

It has the following memories, the sizes of which have been carefully chosen to ensure the highest clock rate:

1.	*Instr. Memory/ Instr. Register*	(IM/IR)	34 bits
2.	*Register File*	(RF)	32x16 bits
3.	*Data Memory*	(DM)	32x16 bits
4.	*Kernel Memory*	(KM)	32x16 bits

B. IPPro Framework Architecture

The IPPro *system* comprises multicore *Processors* connected as 3x3 arrays of IPPro cores, each with nearest neighbor connections by either FIFO or by direct addressing between cores. Depending on the target device and the algorithm implementation demands, an array of IPPro Multi Core Processors or IPPro *Platform* can be created.

This platform and essentially the routing of high level data are managed by an overseeing host. In this scenario, the two ARM Cortex A9 processors in the Xilinx Zynq are used as they have good support for Ethernet and DDR interfaces.

IV. Algorithm Deployment onto IPPro

The IPPro System functions as the computational stage between the video source, e.g. the surveillance camera and the distribution of the processed video data to a host over a network. For present, our implementation is deployed on a Xilinx Zed board with images from the onboard SD memory card in lieu of an actual camera source. Image data is fed directly into the DDR and stored as frames. The ARM Cortex A9 host processor manages the flow of data from the frame buffer to the programmable fabric that accommodates the IPPro cores.

A. Algorithm Partitioning

The mapping to the IPPro begins with a functional breakdown of the algorithm, coded as a Simulink Model or M-Code of the algorithm. We started with the partition in Dalal and Triggs but modified it to facilitate the reuse of already computed gradient, absolute gradient and magnitude data. We passed this additional data between the two IPPro functional blocks, giving an increase in required bandwidth from 1990Mbs^{-1} to 4976Mbs^{-1}. As within the FPGA fabric, we have access to several Tbs^{-1} of bandwidth, it is deemed to be more beneficial to trade bandwidth for reduction in computation time.

The algorithm segments are further broken down to sequences of fundamental mathematical operations such as multiply, add subtract etc., which in turn are mapped to the IPPro cores instructions. The IPPro core Program Memory (PM) code is generated to produce an efficient ratio of read/write to ALU instructions whilst maximizing the quantity of input data to process. Partitioning of the input image data is such that smaller blocks and their computation fit within the IPPro core registers. In order to determine the highest throughput, all PMs must be coded and different configurations examined. The configuration with the highest throughput of 209 frames per second (fps) was chosen.

In FPGAs memory is limited, so the optimization of the memory access/computation ratio is essential particularly as each IPPro Core has very limited local storage in the form of high speed registers on which to carry out its calculations. This encourages the designer to re-use intermediate results rather than recalculate them again later. Temporal parallelism is exploited by the reuse of local data in the IPPro core registers. For example, we found that we could append the *Pixel Gradients* function with the *Magnitude* (*M*) function thus saving costly additional load and store of the *Gradient* values as the IPPro core already had the pixel intensities loaded with gradients stored in its register file; there were enough unused register locations to store the results. This increased the throughput by 6% which in real terms for our multi-IPPro core is a real increase of 13fps.

The following mathematical optimization was also incorporated when calculating the gradient values $G_x = [-1, 0, 1]$ and $G_y = [-1, 0, 1]^T$; as the gradient calculation kernel values use only -1, 0 and 1 as factors, we only need to do one subtraction instruction to generate each gradient. Factors other than -1, 0 and 1 for G_x and G_y would each require three multiply-accumulate i.e. MULACCK instructions. Negi [6] and Xie [7] have already shown that these optimizations have negligible effect on the accuracy of the algorithm.

The next stage is that of binning which allocates each of the derived pixel magnitudes to one of the nine available bins as shown in the lower left corner of Fig. 2. Each bin has a 20˚ range but rather than perform a trigonometric calculation to determine its bin, a series of comparisons are run to check if the angle is greater than a threshold [4]. *Cell histograms* are generated by accumulating M value of each pixel at the appropriate bin for that pixel over a cell.

B. Instruction Mapping and scheduling Single IPPro

Currently the step of translating the algorithm's mathematical functions to the available IPPro instruction set is a manual activity. Care was taken to maximize the ratio of ALU instructions to memory access instructions to ensure a high IPPro ALU core activity.

Our approach is summarized by the four steps in Fig. 4. Continuing on from the initial algorithm exploration and design stage, it is essential that functional p*artitioning* identifies a linear path or paths through the algorithm. This allows the host to manage the coarse grained flow of processed data more efficiently.

At this point we are faced with *allocating* the functions to hardware. As the register size is tightly constrained, we need to work out *whether* or it is more efficient to heavily process small amounts of data (complex PM) or lightly process large amounts of data (simple PM). Complex PMs will result in longer instruction lengths and reduce the granularity of the high level functional blocks. This reduction in granularity will reduce the ability to explore the mapping across multiple IPPro cores.

The next step is the *translation* of the image processing functions to the IPPro instruction set. In order to increase efficiency, no operation (NOP) instructions are reduced by interleaving the sub-tasks within the functional blocks. Sub-tasks begin when sufficient registers are loaded; the remaining input data for subsequent sub-tasks is loaded by paying attention to the pipeline delays during sub-tasks. Interleaving of IPPro processors is also employed in a multi-IPPro core implementation such that scheduling of each of the processors is initialized serially; this reduces the bandwidth needs on the higher level memory accesses.

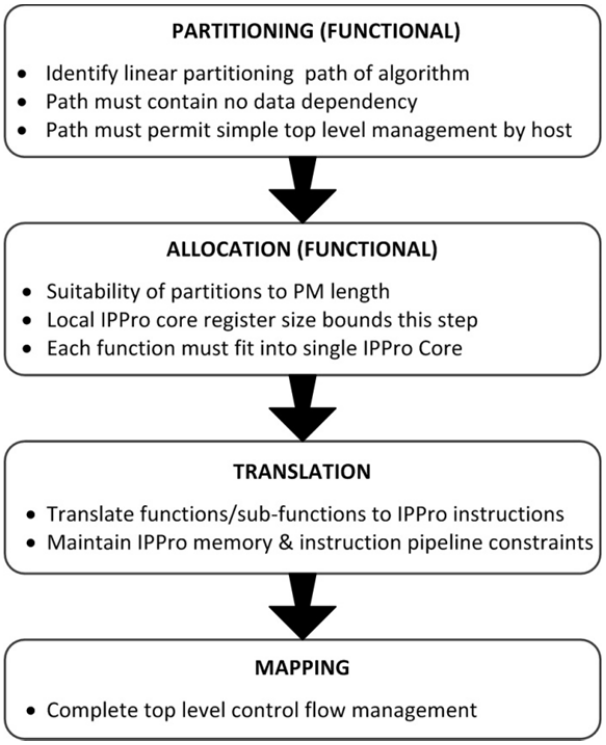

Fig. 4. Key Features in IPPro Mapping and Scheduling

The final stage of the process involves mapping the generated PMs onto the available hardware. With a single IPPro Core there are two approaches. The first approach relies upon each of the PMs carrying out tasks that are not data-dependent and thus events are scheduled to execute sequentially on a small volume of data. However, if each functional stage is dependent on a group or entire frame of results from the current PM, then the second approach is adopted to perform the function across the entire frame, producing intermediate results which are stored in the higher

level memory. The subsequent PM then retrieves the data from the high level memory and executes the next stage of dependent processing from the entire frame or block of intermediate results. Whilst this process is very similar to the traditional mapping of functions, we put an emphasis on maximizing the utilization of the DSP48E whilst minimizing the overhead caused by the memory accesses.

C. Instruction mapping and scheduling Multiple IPPro

Whilst we have undertaken the mapping of the HOG algorithm onto a single core, we also consider the performance of a 3x3 core array. The same IPPro PM code and functional decomposition as earlier is used but *spatial* and *temporal* parallelism is also considered (see Fig. 5). *Mapping* involves allocating the number of processing steps to processor elements and generating a *schedule*. In the instance of one processor element, hardware reuse is used whereas parallelism is exploited in the multi-core case by duplicating the functionality across single IPPro cores as illustrated in Fig. 5. During the many core IPPro implementation, data will be streamed, so careful scheduling is essential to balance the computation phases thus avoiding blocking. For now, this is achieved in a deterministic manner by inserting NOP instructions.

D. Results generation

Two versions of the functional blocks were explored, one a hand-coded VHDL description and the other an IPPro implementation. Both designs were coded in and taken through Xilinx ISE 14.6 Place and Route (PAR) tools with the results recorded in Table 2. In each case, the target platform was the programmable fabric within the Zynq 7020 used in the ZED board. After place and route, the single IPPro core operated at 509MHz for both functions (PMs) and the hand coded implementation operated at 288MHz for the Gradients and Magnitude design and at 164MHz for the Binning & Cell Histogram.

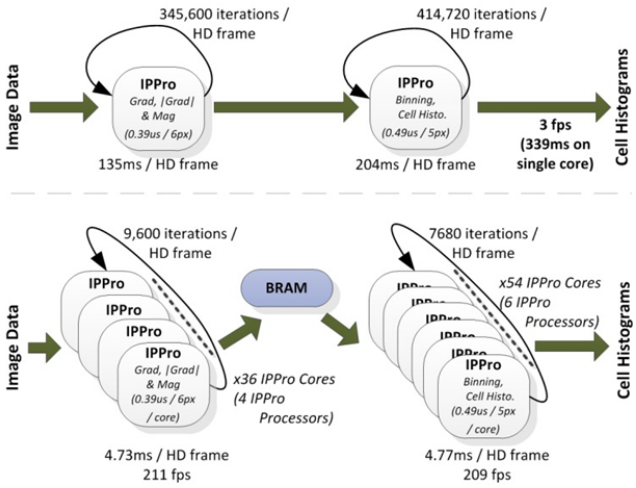

Fig. 5. HOG Architecture for single-IPPro(top) and multi-IPPro (bottom) implementations

The PM for the Gradients and Magnitude functions required 199 instructions to generate the values for 6 output pixels. The PM for the Binning and Cell Histogram required 251 instructions to generate the values for 5 output pixels. Implementation and results were verified on the ZED board. For the architecture in Fig. 5, it is possible to achieve a maximum throughput of 264 fps at a 1920x1280 pixel resolution. This architecture consumes 90 IPPro cores and can be implemented on the smallest Xilinx 7 Series FPGA (XC7A35TCPG236) which has 90 DSP48E available – the clock speed is limited to 404MHz on the Zynq 7020. To make a fair comparison with respect to resources between hand coded VHDL and IPPro, we have used 16 IPPro cores. Our total processing time per frame is 41.92ms for IPPro versus 19.78ms for the hand coded designs.

Table 2: Algorithm resource usage on a single IPPro versus a hand coded approach

Function	***Resource type***	***Single IPPro***		***Hand coded (Single Core)***	
		Usage	***Single Frame***	***Usage***	***Single Frame***
Gradient, |Gradient| & Magnitude	LUTs DSPs BRAMs	140 1 0	135ms	422 0 0	7.18ms
Binning & Cell Histogram	LUTs DSPs BRAMs	140 1 0	204ms	1463 0 0	12.6ms

E. Comparison with other work

FPGAs have been used to implement elements of the HOG algorithm [6] and [7] but these have been hand coded in VHDL, requiring a considerable design effort and major redesign effort should the algorithm change. Other FPGA-based processor design approaches have also been explored. The iDEA processor [11] is based around the DSP48E1 primitive but has some differences in that it uses Block RAM rather than Distributed RAM for the RF and DM. iDEA has a maximum frequency of 407MHz (compared to 509MHz here) for a single core implementation and also uses a 9-stage pipeline which can be a problem when NOP instructions are required as discussed above. iDEA consumes 335 LUTs, 2 RAMB36E1, 1 DSP48E1 compared to 140 LUTS and 1 DSP48E1 here.

An approach which deals with the difficulties of multi-core programming and the dissemination of tasks across many processing elements is FlexGrip [12]. By creating a soft GPGPU which can be controlled by a Xilinx MicroBlaze soft core microprocessor on the AXI bus, they are able to leverage the existing programming environments established for GPUs namely CUDA but the resource usage is high and it has lower operating frequency.

Table 3 compares some recent FPGA implementations of the HOG algorithm. The Hahnle design [13] only quotes the resource and performance for HOG descriptor generation, so this is the closest to our implementation. Whilst the ratio of fps to resource usage is very similar for the 90 core IPPro solution and Hahnle design, our solution is achievable

quickly and without any HDL knowledge. Our solution is also very amenable to reconfiguration should the algorithm evolve. The frame rate of the IPPro 90 core solution is comparable to Xie's solution [7] but the number of pixels processed per frame by Xie is 27 times less than our multi-IPPro solution. The clock rates for considered solutions are much less than IPPro, which achieves 404MHz.

We used Xilinx's Power Analyzer 14.6 tool to determine the power consumption for our multi-IPPro implementation. The resultant 3.6W total power consumption is considerably lower than that of any comparable GPU or CPU solution, indeed more than an order of magnitude lower. Our designs power consumption is in line with other current HOG FPGA implementations.

Table 3. Comparisons of the resources for different recent FPGA implmentations

Ref (Device)	***Clock (MHz)***	***Resource***		***Performance***	
		Type	***Use***	***Frame size***	***Frame/s***
[7] (Spartan-3e XC3S500E)	67.75	LUTs DSPs BRAMs	3,379 no data 6	320 x 240	293
[6] (Virtex 5 VLX-50)	44.85	LUTs DSPs BRAMs	17,383 no data 36	640 x 480	112
[13] (Virtex 5 VFX200T)	270	LUTs DSPs BRAMs	3642 12 26	1920 x 1080	64
Hand Coded (Zynq 7020)	164	LUTs DSPs BRAMs	1885 0 0	1920 x 1080	79
IPPro - 90 cores (Zynq 7020)	404	LUTs DSPs BRAMs	10694 90 24	1920 x 1080	209

V. Summary and Conclusions

A processor-based approach for implementing HOG which avoids the recursive design loops associated with HDL-based FPGA design is presented. The multi-core IPPro approach achieves high performance on a Zynq, comparable to a highly optimized implementation [7] which makes many approximations to achieve high throughput. Our multi-IPPro design achieves 2.6 times the throughput of our hand coded design and 3.2 times the closest recent work [13].

The work demonstrates that our architecture is highly scalable and using mapping and scheduling, we can easily employ more parallelism without impeding levels of complexity. Reuse of the same PM from single core to multicore implementations allows scaling with only minor adjustments to the interconnection and collation of data outputs. This higher dataflow is managed at the highest i.e. platform level by the ARM host and efficiently controlled by simple software. Our IPPro architecture implementation effort was 10 man days as opposed to 40 days for the hand crafted VHDL implementation. Whilst only considered for FPGA in this study, the IPPro architecture could suit ASIC implementation permitting high performance and reconfigurability on an even lower powered platform. Currently, a programming environment based on dataflow to allow efficient extraction of parallelism from a high level description and optimization of the allocation of the functions to the IPPro Platform, is being created. This will allow deployment of image processing algorithms onto FPGA in a cost effective and efficient manner.

Acknowledgements

This work has been undertaken in collaboration with Heriot-Watt University in a project funded by the Engineering and Physical Science Research Council (EPSRC) through the EP/K009583/1 grant.

References

[1] F. Catthoor, K. Danckaert, K. K. Kulkarni, E. Brockmeyer, P. G. Kjeldsberg, T. Achteren and T. Omnes, *Data Access and Storage Management for Embedded Programmable Processors*, Springer, 2010.

[2] Y. Gat, I. Kozintsev and O. Nestares, "Fusing image data with location and orientation sensor data streams for consumer video applications," in Computer Vision and Pattern Recognition Workshops, 10.1109/CVPRW.2010.5543781.

[3] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *IEEE Conference on Computer Vision and Pattern Recognition, San Diego*, CA, 2005, pp886 - 893.

[4] S. Bauer, U. Brunsmann and S. Schlotterbeck-Macht, "FPGA Implementation of a HOG-based Pedestrian Recognition System," in *MPC-Workshop*, Karlsruhe, 2009.

[5] K. Benkrid and D. Crookes, "High Level Programming for FPGA Based Image and Video Processing Using Hardware Skeletons," in *IEEE Ninth Annual Symposium on FPGA Custom Computing Machines*, 2001, 0-7695-2667-5.

[6] K. Negi, K. Dohi, Y. Shibata and K. Oguri, "Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm," in *International Conference on Field-Programmable Technology*, 10.1109/FPT.2011.6132679.

[7] S. Xie, Y. Li, Z. Jia and L. Ju, "Binarization based implementation for real-time human detection," in *23rd International Conference on Field Programmable Logic and Applications*, 2013, 10.1109/FPL.2013.6645590.

[8] R. Kadoto, H. Sugano, M. Hiromoto, H. Ochi, R. Miyamoto and Y. Nakamura, "Hardware Architecture for HOG Feature Extraction," in *Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2009.

[9] F. Siddiqui, B. Baradak, R. Woods and K. Rafferty, "IPPro: FPGA based Image Processing Processor," in *IEEE International Workshop on Signal Processing Systems*, Belfast, 2014.

[10] G. Maki, S. Whitaker and G. Ganesh, "A Reconfigurable Data Path Processor," in *Fourth Annual IEEE International ASIC Conference and Exhibit*, 1991.

[11] H. Cheah, S. Fahmy and D. Maskell, "iDEA: A DSP Block Based FPGA Soft Processor," in *International Conference on Field-Programmable Technology*, 2012.

[12] K. Andryc, M. Merchant and R. Tessier, "FlexGrip: A soft GPGPU for FPGAs," in *International Conference on Field-Programmable Technology (FPT)*, Kyoto, 2013.

[13] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann and K. Doll, "FPGA-based Real-Time Pedestrian Detection on High-Resolution Images," in *IEEE Conference on Computer Vision and pattern Recognition Workshops*, 2013.