# IPPro: FPGA based Image Processing Processor

Fahad Manzoor Siddiqui, Matthew Russell, Burak Bardak, Roger Woods, Karen Rafferty
School of Electrical Engineering, Electronics and Computer Science
Queens University, Belfast
(f.siddiqui, mrussell16, b.bardak, r.woods)@qub.ac.uk
k.rafferty@ee.qub.ac.uk

*Abstract*—**the paper presents IPPro which is a high performance, scalable soft-core processor targeted for image processing applications. It has been based on the Xilinx DSP48E1 architecture using the ZYNQ Field Programmable Gate Array and is a scalar 16-bit RISC processor that operates at 526MHz, giving 526MIPS of performance. Each IPPro core uses 1 DSP48, 1 Block RAM and 330 Kintex-7 slice-registers, thus making the processor as compact as possible whilst maintaining flexibility and programmability. A key aspect of the approach is in reducing the application design time and implementation effort by using multiple IPPro processors in a SIMD mode. For different applications, this allows us to exploit different levels of parallelism and mapping for the specified processing architecture with the supported instruction set. In this context, a *Traffic Sign Recognition* (TSR) algorithm has been prototyped on a Zedboard with the colour and morphology operations accelerated using multiple IPPros. Simulation and experimental results demonstrate that the processing platform is able to achieve a speedup of 15 to 33 times for colour filtering and morphology operations respectively, with a reduced design effort and time.**

**Keywords—FPGA, processor architecture, embedded systems, heterogeneous computation, image processing, traffic sign detection.**

## I. INTRODUCTION

*Embedded Vision* (EV), *Video Analytic* (VA) and *Smart Camera* applications are being widely used; the Embedded Vision Alliance estimates a market worth $25b by 2018 [1]. Typically, video streams are captured from multiple cameras and then sent to a *centralized processing* server, requiring high transmission bandwidth and memory. Alternatively, a *distributed* approach can be used where data-intensive *front-end* pre-processing is carried out inside the camera module as illustrated in Fig. 1; this gives a reduction in data bandwidth requirements from MB/s to kB/s.

Typically, customized application specific hardware accelerators based on Field Programmable Gate Arrays (FPGA) are ideal platforms for real-time image processing applications. They offer high levels of parallelism and improved performance over DSP and CPU-based platforms, and provide much lower power consumption than GPUs [2]. The challenge is to build a front-end FPGA solution that gives *high performance* but which can be easily *programmed*. This needs careful design as FPGAs have considerable processing resources but limited memory.

There are number of FPGA-based image processors in the open literature. These include: a vector processing approach [3] which uses fixed, pipelined functional units (FUs) that can be inter-connected; the soft vector processor VESPA architecture [4] which employs vector chaining, control flow execution support and banked register file to reduce execution time. However, both approaches are limited to a clock rate less than 200MHz. VENICE [5] is a processor–based solution that provides support for operations on unaligned vectors and FlexGrip [6] is a FPGA-based multicore architecture that allows mapping of pre-compiled CUDA kernels which is scalable, programmable and flexible. However, both solutions operate only at 100MHz.

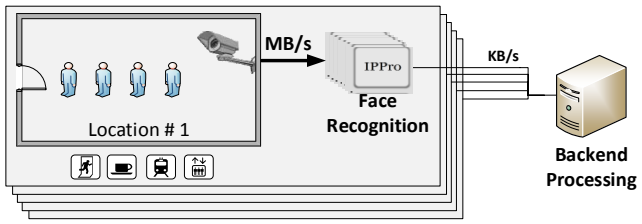


Fig. 1 Distributed computing approach to Video Analytics

The key is to build a much faster soft-core, processor which can meet the performance requirements of image processing. This can be achieved by carefully crafting a multicore *Single Instruction Multiple Data* (SIMD) or a *Multiple Instruction Multiple Data* (MIMD) processor architecture which efficiently utilizes FPGA resources, e.g. DSP blocks, embedded memory. Chu et.al [7] has created such programmable, heterogeneous parallel soft-core processor architecture, but it was focused towards telecommunication applications. Similarly, iDEA [8] is a 9-stage, pipelined DSP48E1 based soft-core processor which supports basic arithmetic and logical instructions by utilizing limited FPGA resources. The design runs at 407MHz which is 1.93 times faster than Xilinx MicroBlaze and significantly faster than previous work.

Whilst these solutions offer flexibility and scalability, they are difficult to extend/adapt to multicore architectures due to limited connectivity or use deeply pipelined 9-stage architecture [8] and can further be improved. We propose the IPPro soft-core processor which operates at 526MHz but with a much smaller pipeline and extendable to multicore architectures. It has been specifically targeted for image processing applications and in this paper, we demonstrate its performance for a *Traffic Sign Recognition* (TSR) algorithm implemented on a Xilinx Zynq Zedboard.

The paper is organised as follows. Section 2 describes the IPPro processor architecture outlining the key design

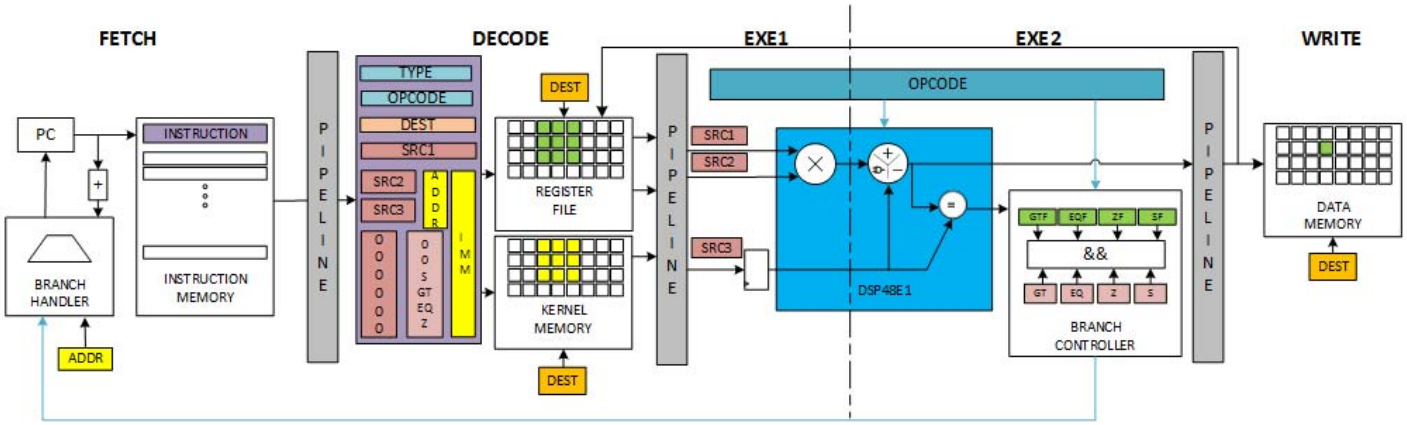


Fig. 2. IPPro Processor Datapath

decisions. The system architecture is described in section III and is followed by a TSR implementation using Zedboard in section IV. Finally conclusions and future work are outlined.

## II. IPPRO ARCHITECTURE

Numerous hard and soft FPGA implementations exist in the research and industrial domains [3-8] to simplify the hardware design flow. *High Level Synthesis* (HLS) tool approaches such as Xilinx's Vivado and Altera's OpenCL, allow the creation of efficient hardware realizations. However, a gap exists between efficient FPGA realization (resource utilization and performance) and programmability (reduced design time). The aim of a major research programme at Queen's University Belfast is to fill this gap by the creation of a completely programmable platform consisting of FPGA-based programmable hardware and a software tool-chain to makes it easier to use and reconfigure the underlying hardware with less design effort without compromising performance. Central to this philosophy is the creation of a flexible, programmable and reconfigurable platform which is presented here.

To achieve performance and take advantage of FPGA parallelism, multicore heterogeneous processors supporting SIMD and MIMD execution models capable of exploiting *Data Level Parallelism* (DLP) and *Instruction Level Parallelism* (ILP) are required. The challenges for the complete approach include:

1) *A suitable high-level representation* such as the CAL *Data Flow Graph* (DFG) representation [9] which allows expression of DLP.
2) Capability to apply *decomposition, mapping, resource binding and scheduling* onto a hardware platform in an efficient manner using tools such as ORCC [9].
3) *High performance FPGA-based hardware* in the form of a processor that offers high performance and software programmability. It has a flexible memory hierarchy, performance and data communication model which is able to adopt to different high level DFG models.

This section describes the IPPro soft-core processor which is a 16-bit signed fixed-point *scalar processor* based on RISC *load/store* architecture but customizable to 8/16/32 bits. It is hand-coded using Xilinx primitives and designed to be scalable and flexible enough to easily extend for multi-core operation (see section III). A key consideration is to create an architecture which has been optimized for image processing algorithms but also achieves high performance. The complete datapath is illustrated in Fig. 2. It uses *distributed memory* to build the lowest level of memory hierarchy available to multicore architecture, supported by *addressing modes* to make it flexible. This helps to match the IPPro memory with the algorithmic characteristics and execution flow. Whilst linear operations are common, it is important to support a non-linear data dependent execution which includes *conditional execution* and *branches*. This requires careful consideration as how the execution flow is decomposed and mapped to the underlying FPGA architecture. The following design decisions were made to optimize FPGA performance and image processing needs:

*1)* *High processing capability* (350-450MIPS) is required to handle the large data (30-40MB/s) needed for real-time video streaming. This is achieved by explicit mapping of operations and logic to resource primitives.
*2)* *Efficient memory utilization* by *distributing memory* to hide data transfer overheads between main and local memory to keep IPPro busy in processing data. *Dedicated kernel memory* accelerates the linear filter operations. But also reduces the code size by avoiding excessive load/store instructions and maximize memory reusability.
*3)* *Optimized instructions/addressing modes* and *reduced branch penalty* by reducing number of pipeline stages as unpredicted branches degrade performance: *special instruction sets* to allow the acceleration of image processing operations; a*ddressing modes to* give flexibility to the programmer and; c*onditional execution* in the form of a customizable and flexible branch controller (BC) to support mask-based conditional execution out-of-box without need of significant architectural changes.

### A. Memories

The IPPro contains small, fast and efficient memory to locally store data and keep *Arithmetic and Logic Unit* (ALU) busy in processing data. This helps to hide data transfer overheads between the main and local memories. The following memories are listed below with their respective memory sizes:

| | |
|---|---|
| *Instruction Memory* (IM)*/Instruction Register* (IR) | 34 bits |
| *Register File* (RF) | 32x16 bits |
| *Data Memory* (DM) | 32x16 bits |
| *Kernel Memory* (KM) | 32x16 bits |

Sizes were chosen to allow an effective balance between image processing requirements and efficient utilization of FPGA resources. Memories are implemented using FPGA *distributed memory* resources except the IM which is based on *BRAM*. The design choice was made to allocate a size of 32 registers to facilitate *area operations*. The RF is a quad-port RAM with 1 synchronous write and 3 asynchronous reads used to store data locally and allows 3 operands operations e.g. multiply-add (MULADD) to be accommodated as they are efficient and commonly used in image processing applications. To accelerate *area* operations, a dedicated KM is included into IPPro datapath near ALU. It allows programmer to store frequently used constants e.g. filter coefficients, permitting memory reuse and reduced program code size by avoiding unnecessary *load/store* instructions. The DM is designed to store data/pixels before/after processing and can be used to store temporary results. Considering the mentioned memory distribution, IPPro supports following four addressing modes:

1. *Register File – Register File* *(R-R)*
2. *Register File – Data Memory* *(R-D)*
3. *Register File – Kernel Memory* *(R-K)*
4. *Register File – Immediate* *(R-I)*

TABLE. I
IMAGE PRE-PROCESSING OPERATIONS

| | | |
|---|---|---|
| **Area Operation** | *3-D Conv.* | [A B C; D E F; X Y Z]<br>e.g. Gaussian, Sobel, Erosion, Dilation |
| **Point Operations** | *Threshold* | O = (A > threshold)? 1 : 0 |
| | *Contrast* | O = (A – 0.5) * contrast + 0.5 |
| | *RGB to Grey* | O = (( R<<2)+((G <<2)+G)+ B)>>3 |
| **Geometric Operations** | *Reflect* | $x2 = x1\ ;\ y2 = -y1 + (2 * y0)$<br>$y2 = y1\ ;\ x2 = -x1 + (2 * x0)$ |
| | *Translate* | $x2 = x1 + \beta x$<br>$y2 = y1 + \beta y$ |

### B. Instruction set

Most pre-processing applications require *point* and *area* operations which comprise arithmetic and logical operations as shown in Table. I. As, the IPPro supports basic arithmetic and logical operations, a series of IPPro instructions are needed to implement them, as incorporation of special instruction impacts clock rate and affect performance. Table. I have been used to drive the instruction set listed in Table II and lists all 53 supported instructions, along with their respective addressing modes. BZF, BEQF, BGTF and BSF are *Branch* instructions that require IPPro flags which are discussed later.

### C. Datapath

The complete IPPro datapath is shown in Fig. 2. It has a *load-store* 5-stage *balanced pipelined* architecture giving a fixed latency of 5 clock cycles. It exploits the features of the Xilinx DSP48E1 to implement all of the supported instructions and provides a balance between hardware resource utilization, performance, throughput, latency and branch penalty. A balanced pipeline simplifies the development of compiler tool-chain compared to variable pipeline architecture. The deep pipeline comes at the cost of larger latency and branch penalty which adversely affects the overall performance. Various techniques predict branches but none of them was deemed to give a shorter latency. The five pipeline stages are as follows:

1. *Fetch* (IF)
2. *Decode* (ID)
3. *Execute 1* (EXE1)
4. *Execute 2* (EXE2)
5. *Write Back* (WB)

TABLE. II
INSTRUCTION SET

| R-R | | R-K | | R-I | Misc. |
|---|---|---|---|---|---|
| ADD | LOR | ADDK | LORK | ADDI | LD |
| SUB | LNOR | SUBK | LNORK | SUBI | ST |
| MUL | LNOT | MULK | LNANDK | LANDI | BZF |
| MULADD | LNAND | MULADDK | LANDK | LXORI | BEQF |
| MULSUB | LAND | MULSUBK | STK | LXNRI | BGTF |
| MULACC | LSL | MULACCK | LSLK | LORI | BSF |
| LXOR | LSR | LXORK | LSRK | LNORI | JMP |
| LXNR | MIN | LXNRK | MINK | LNANDI | CMP |
| | MAX | | MAXK | MINI | NOP |
| | | | | MAXI | |

Each stage takes a single clock cycle to execute. IF and ID stages read the instruction and two execution stages (EXE1 and EXE2) support the logical and multiply accumulate operations e.g. MULADD, MULSUB. The *programmer/compiler* takes care of possible data and control hazards during scheduling and register allocation.

The execution model follows a traditional processor approach where the instruction is first fetched from the IM and decoded by the ID; this generates corresponding control signals to modify/control the data-path needed to execute respective instruction. During ID, data operands are read asynchronously from the *RF* or *KM* and stored into pipeline registers. In next clock cycle, they are forwarded to the EXE1 stage; both EXE1 and EXE2 are internal to the DSP48E1 block. At this stage, the DSP48E1 is dynamically reconfigured on a cycle-to-cycle basis utilizing the control signals generated by ID. This process has explained in detail in reference [10].

### D. Flags& Branches

Flags are important status indicators in processors and used to handle exceptions encountered during data computation. IPPro currently supports data flags but flexible to define new flags by defining them in the *Branch Controller* (BC) as shown in Fig. 2.

1. GTF (Greater than)
2. EQF (Equal)
3. ZF (Zero)
4. SF (Sign Flag)

Flags in IPPro are generated using the *pattern detector* function embedded inside the DSP48E1. It compares the two operands available at the input of DSP48E1 and sets the PATTERNDETECT (PD) bit in the very same clock cycle if both operands are equal. Therefore no additional clock cycle is needed to compute the flag bit which is important in the case of conditional/data dependent instructions being executed in the multicore architecture. BC is flexible and scalable defined as combinational logic.

TABLE. III
IPPro SYNTHESIS RESULTS

| IPPro | Virtex-6 | | Virtex-7 | Zynq SoC |
|---|---|---|---|---|
| | *XC6VLX240T -3* | | *XC7VX550T -3* | *XC7Z020-3* |
| *Slice Reg's* | *330* | *<1%* | *330* | *330* |
| *Slice LUTs* | *273* | *<1%* | *273* | *273* |
| *DSP48E1* | *1* | *<1%* | *1* | *1* |
| *BRAM* | *1* | *<1%* | *1* | *1* |
| *Freq. (MHz)* | *509* | | *526* | *526* |

### *E. Conditional Branch and Jump*

IPPro supports both conditional and unconditional branch instructions to support loops, which in turn reduces the program code size. IPPro handles branch instructions with the help of the BC and *Branch Handler* (BH) as illustrated in Fig. 2. The BC compares the IPPro flags against the branch instruction. If the condition is met, it instructs the BH to change the *Program counter* (PC) value to the target address in IM, and branch is executed. With the IPPro pipeline constraints, the branch penalty is 5 clock cycles. The *Jump* instruction is handled in the same way except it is unconditional.

The reason to have a separate BC and BH has been driven to make the IPPro *scalable* and *adaptable* for SIMD operation for handling *conditional execution* using a core masking technique. In this case, the BC generates the mask value and sends it to the *SIMD Controller (SIMDC)* to decide whether the respective processor need to be halted or not. Therefore, IPPro is extendable and can be used to develop multicore processor designs.

### *F. IPPRO Results*

The complete design is synthesized, implemented and verified on the Xilinx ZYNQ XC7Z020 SoC using Xilinx ISE v14.1 and Mentor Graphics ModelSim SE v10.2. Table III presents the synthesis results and the percentage of each hardware resource of different FPGAs.

## III. MULTICORE PROCESSOR

Fig. 3 shows the block diagram of the *front-end* processor architecture. It was prototyped on a Zedboard platform using a Xilinx Zynq SoC which comprises on-chip dual-core ARM processors and programmable logic. The *SIMD-IPPro* is comprised of a number of IPPro cores connected together thus giving an *n-way SIMD-IPPro*. Each SIMD-IPPro has two levels of memories i.e. *local* and *shared.* The main challenge is to connect these multiple IPPro processors interconnected together sharing data to meet the algorithmic characteristics without compromising performance.

Multicore operation is achieved by modifying the *fetch* pipeline of IPPro datapath by replacing the IM with the IR and letting the *SIMD Program Memory cache* hold the main SIMD program code controlled by SIMDC. Moreover, the PC and BH module have to be removed from IPPro datapath whilst keeping the BC and incorporating it into the SIMDC. These modifications result in the multiple SIMD-IPPro, processor architecture shown in Fig. 3. Since all processors in SIMD-IPPro are connected in SIMD mode, there are hardware resources that can be shared among them allowing reduced area utilization (see Table. IV).

The Zedboard contain external memory interfaces including DDR3 to buffer captured images in real-time before processing. The dual-ARM cores are responsible for streaming unprocessed data to SIMD-IPPro processor and handling control scheduling, synchronization, intra SIMD-IPPro data transfer etc. *Direct Memory Access* (DMA) is used for fast transfer of data between ARM and *Programmable Logic* (PL). Similarly, *SIMD Controller* is responsible for handling the control decisions inside SIMD-IPPro which include fetch, decode instructions and control the execution of SIMD-IPPro processors. Right down in the processing chain there are IPPro processors that are presented in this paper. They are simple scalar highly efficient programmable processors having a special instruction set designed to accelerate basic data intensive operations on FPGA's PL.

| Hardware Resource | IPPro | SIMD-IPPro |
|---|---|---|
| *Program Memory* | *NO* | *YES* |
| *Kernel Memory* | *NO* | *YES* |
| *Instruction Decoder* | *NO* | *YES* |
| *Register File* | *NO* | *NO* |
| *Data Memory* | *NO* | *NO* |
| *ALU (DSP48E1)* | *NO* | *NO* |

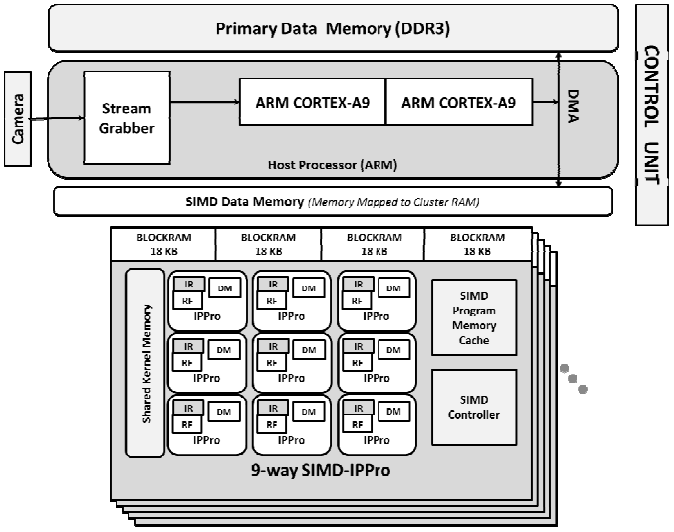


Fig. 3 Front-End Processor architecture

## IV. CASE STUDY: TRAFFIC SIGN RECOGNITION

This section presents an implementation and assessment of TSR algorithm acceleration using SIMD-IPPro as it is a typical image processing application. Fig. 4 illustrates the different stages of TSR algorithm. Colour filtering is the most data intensive operations and used to separate red and blue traffic signs from the background of the image. Morphological operations are then applied to clean up small holes in objects, thus avoiding unnecessary processing on objects which are too small to be successfully recognised even if they are traffic signs.

Some FPGA-based accelerator solutions exist [10-13]. The work in [11] uses a MicroBlaze processor on a Virtex-5 with hardware accelerators for colour filtering and morphological operations on images of 320x240 pixels; it computes in 777ms. In [12], LEON3 CPUs are used in a Virtex4 FPGA, but this leaves little room for custom accelerators; their design completes within 600ms, but the image size is not discussed. The Altera-based Cyclone II based design in [13] does not consider circular traffic signs. In [14], a Zynq SoC is used to accelerate the colour and morphology operation of TSR using a 1920x1080 image size, whole algorithm is performed in the ARM processors; it takes 5s to process it. The key advantage here is the use of soft processors for the complete software based acceleration.

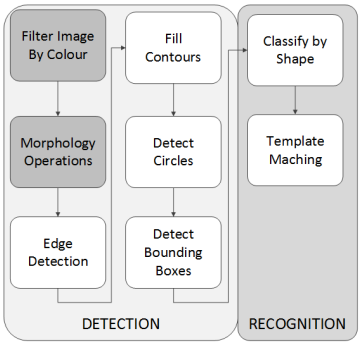

Fig. 4 Traffic sign detection algorithm

Fig. 5 shows the adopted system level architecture. The image size used is 600x400 pixels. The *host* is used to send commands to the PS using the UART interface that gives console access to Linux operating system. It gives access to OpenCV libraries which are used to perform rest of the TSR stages inside the ARM. The Ethernet connection can be used for larger data transfer to/from the Zedboard. Data communication between the *Processing System* (PS) and the PL is provided by the *High Performance* (HP) ports, as they give much higher throughput than *General Purpose* (GP) ports. The GP ports are used but only to read/write to the AXI Lite register space inside the DMA Engine. In the PL, there is a SIMD-IPPro controlled by a defined finite state machine. In our design, the c*olour* and *morphology* operations were accelerated, as they represent the most computationally complex as shown in Fig. 6 while rest of the stages were implemented using on-chip dual ARM core (see Fig. 5). To accelerate *colour* and *morphology* stages, SIMD-IPPro consist of 32 and 16 processors are used respectively.

Table. V present the results obtained by processing set of real images using IPPro running on Zedboard. The execution time taken by the targeted colour and morphology stages using 32 and 16 IPPro cores is ≈2*x* smaller (19.710ms and 41.361ms) provided that difference in computational intensity was ≈4.5*x* higher (88.865ms and 399.793ms) previously achieved by [14]. Fig. 6 clearly shows that proposed architecture achieved better acceleration maintaining higher throughput 30-92MPixels/s. Moreover, this gain indirectly accelerated the edge/contours detection and bounding boxes stages implemented in PS, as they are using colour and morphology filters.

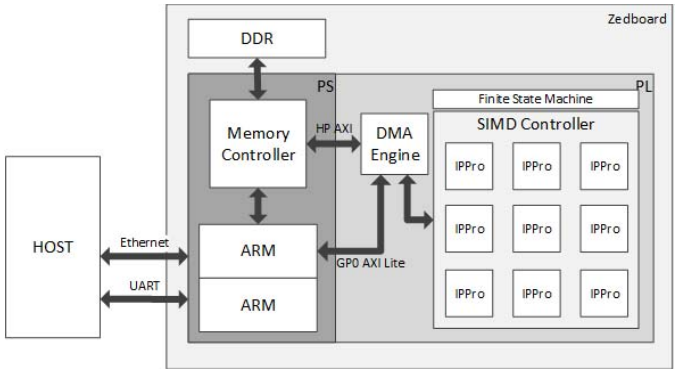

Fig. 5: System Level Architecture of TSR acceleration

The synthesis result presented in Table. V did not share resources for quick prototyping as discussed in Table. IV. Therefore the design has not been optimised for area. Also, the IPPro is capable of operating at 526MHz rather than 150MHz due to limited clock support in Zedboard. In this case the results presented in Table. VI will have 3.51 times improvement without any additional optimisation in the datapath. This gives a throughput of 105–323 MPixels/s, means the IPPro is capable of processing HD video at almost 155 *frames per second* (fps).

TABLE. V
ACCELERATION OF TSR USING SIMD-IPPro

| | SIMD-IPPro @ 150MHz | | SIMD-IPPro @ 526MHz | |
|---|---|---|---|---|
| | **Colour** | **Morph.** | **Colour** | **Morph.** |
| *# of cores* | *32* | *16* | *32* | *16* |
| *Freq.(MHz)* | *150* | *150* | *526* | *526* |
| *Slices Regs.* | *38622 (36%)* | *42066 (40%)* | *38622 (36%)* | *42066 (40%)* |
| *LUTs* | *23318 (44%)* | *30207 (57%)* | *23318 (44%)* | *30207 (57%)* |
| *DSP48E1* | *32 (15%)* | *48 (22%)* | *32 (15%)* | *48 (22%)* |
| *BRAM* | *40 (29%)* | *75 (54%)* | *40 (29%)* | *75 (54%)* |
| *Cycles/pixel* | *160* | *26* | *160* | *26* |
| *MPixels/sec* | *30* | *92.30* | *≈105.3* | *≈323.973* |
| *Exec. Time (ms)* | *19.710* | *41.361* | *≈5.615* | *≈11.787* |
| *Speed up* | *4.51x* | *9.66x* | *≈15.83x* | *≈33.90x* |

TABLE. VI
PERFORMANCE COMPARISON

| Design | Family | Fps | Processor | | Image Size |
|---|---|---|---|---|---|
| *IPPro* | *Zynq* | *2.38/3.62** | *Hard* | *ARM Cortex* | *600 x 400* |
| *[14]* | *Zynq* | *0.2* | *Hard* | *ARM Cortex* | *1920 x1080* |
| *[11]* | *Virtex-5* | *1.29* | *Soft* | *Microblaze* | *320 x 240* |
| *[12]* | *Virtex-4* | *2* | *Soft* | *LEON* | *Not Avail.* |
| *[12]* | *Cyclone-II* | *0.058* | *Soft* | *Nios II* | *320 x 240* |

*SIMD-IPPro running @ 526MHz

Table. VI compares performance of different software based accelerators in fps. It shows how the proposed architecture can achieve better performance. However, it can further improved

as limiting factor is the overhead of data communication infrastructure as it spends approximately 49.7% of the processing time transferring data to/from DDR to DMA.

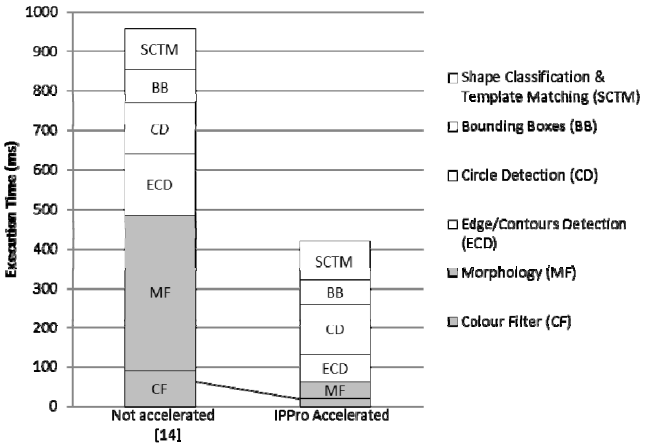


Fig. 6: Comparison of TSR acceleration (Stage-wise)

## V. CONCLUSIONS AND FUTURE WORK

In this paper, a high performance and scalable soft-core processor design has been presented. The scalable design has been fully implemented and tested on Zedboard, with the implementation of a TSR algorithm. Given the results of the TSR acceleration, the IPPro accelerated the reference design by *15*-33 times. Given that the design involves producing hand written IPPro code, the implementation stage only involves decomposing the algorithm and mapping to the instruction set. The design effort and time is significantly reduced without compromising throughput. This shows the considerable potential of being able to approach the performance of hand-crafted FPGA designs with the advantages of fast software design. With the custom-made toolset and compiler, which is currently under development, it will be possible for one to decompose and parallelize the interested algorithm without much effort and reduced design time.

The following has been proposed to optimize the work:

- Modify the system level clock management in the Zedboard to enable the IPPro to run at full speed i.e. 526MHz.
- A shared resource based SIMD-IPPro architecture.
- Substitution of the *finite state machine* with a *SIMD Controller*.
- Acceleration of the *template-matching* stage using IPPro as it is taking 22.5% of overall execution time.

## ACKNOWLEDGMENT

The work has been funded by the UK Engineering and Physical Science Research Council's ICT grant (EP/K009583/1), a collaborative grant with the University of Heriot-Watt.

## REFERENCES

[1] Embedded Vision. (2013). *Image Recognition Market Worth $25.65 Billion by 2018*. Available:http://www.embedded-vision.com/industry-analysis/market- analysis/2014/03/26/image-recognition-market-worth-2565-billion-2018. Last accessed August 2014.

[2] Srinidhi Kestur, John D. Davis, and Oliver Williams, *"BLAS Comparison on FPGA, CPU and GPU*", in Proceedings of the 2010 IEEE Annual Symposium on VLSI (ISVLSI '10). IEEE Computer Society, Washington, DC, USA, Jul. 2010, pp. 288-293.

[3] R. M. Russell, *"The CRAY-1 computer system",* in Communications of the ACM - Special issue on Computer Architectures, Jan. 1978, vol. 21, pp. 63–72.

[4] P. Yiannacouras, J.G. Steffan and J. Rose, *"Portable, Flexible, and Scalable Soft Vector Processors"*, in IEEE Trans. on VLSI Systems, Aug. 2012, vol. 20, pp. 1429-1442.

[5] A. Severance and G. Lemieux, *"VENICE: A compact vector processor for FPGA applications"* in Int'l Conf. on Field-Programmable Technology (FPT), Seoul, Korea, Dec. 2012, pp. 261–268.

[6] K. Andryc, M. Merchant, and R. Tessier, *"FlexGrip: A soft GPGPU for FPGAs",* in Int'l Conf. on Field-Programmable Technology (FPT), Kyoto, Japan, Dec. 2013, pp. 230–237.

[7] X. Chu and J. McAllister, "*FPGA based soft-core SIMD processing: A MIMO-OFDM Fixed-Complexity Sphere Decoder case study*", in Int'l Conf. on Field-Programmable Technology (FPT), Beijing, China, Dec. 2010, pp. 479–484.

[8] H. Y. Cheah, S. Fahmy, and D. Maskell, *"iDEA: A DSP block based FPGA soft processor",* in Int'l Conf. on Field-Programmable Technology (FPT), Seoul, Korea, Dec. 2012, pp. 151–158.

[9] H. Yviquel, *"From Dataflow based Video coding tools to dedicated embedded multicore platforms",* Ph.D. dissertation, Universite de Rennes 1, France, 2013.

[10] Xilinx Inc, *"Xilinx Virtex-6 FPGA DSP48E1 Slice"*, User Guide, UG369, 2011

[11] S. Waite and E. Oruklu, *"FPGA-Based Traffic Sign Recognition for Advanced Driver Assistance Systems",* in Journal of Transportation Technologies, Jan. 2013, vol. 3, pp. 1-16.

[12] M. Mu□ller, A. Braun, J. Gerlach, W. Rosenstiel, D. Nienhuser, J.M. Zollner and O. Bringmann,, *"Design of an automotive traffic sign recognition system targeting a multi-core SoC implementation",* in Int'l Conf. on Design, Automation & Test in Europe (DATE), Dresden, Germany, Mar. 2010, pp. 532-537.

[13] M.A. Souki, L. Boussaid and M. Abid, *"An embedded system for real-time traffic sign recognizing"* 3rd Int'l Conf. on Design and Test Workshop, Monastir, Tunisia, Dec. 2008, pp. 273-276.

[14] M. Russell and S. Fischaber, *"OpenCV based road sign recognition on Zynq"* in 11th IEEE Int'l Conf. on Industrial Informatics (INDIN), Bochum, Germany, Jul. 2013, pp. 596-601.