Dataflow toolset for soft-core processors on FPGA for image processing applications

Burak Bardak, Fahad Manzoor Siddiqui, Colm Kelly, Roger Woods
Institute of Electronics, Communications and Information Technology (ECIT)
Queen's University Belfast, Belfast, Northern Ireland, UK
{*b.bardak, f.siddiqui, r.woods*}@*qub.ac.uk*

***Abstract*—This paper proposes a design tool chain that uses the Cal Actor Language (CAL)[2] as a starting point for an image processing algorithm and targets to custom design soft-core processors on FPGA. CAL is a high level programming language that allows the definition of algorithm with data-flows.**

The main purpose of the design tool is to exploit the task and data parallelism in order to achieve the same parallelism as HDL implementation minus the required design, verification and debugging steps of HDL design, which increases the time to market, and design effort.

s

I. Introduction

The emerging need for processing bigger-data sets such as high-resolution video footage demands faster, configurable, high throughput systems with high-energy efficiency. Field-Programmable Gate Arrays play a big role in this demanding market, where FPGAs can provide configurability, scalability and concurrency to match the required throughput. FPGAs can potentially allow the use of distributed image processing where processing platform is located as close as possible to the the image source. The use of distributed processing can reduce the need for bandwith, and s power in a large scale. This reduces the communication overhead, the amount of data that needed to be stored, latency and power requirements. FPGAs works very well with the applications that requires high concurrency, bandwitdh and reprogrammability. However FPGA design and verification is a very delicate and time-consuming process, where the conventional FPGA design involves hand-written Hardware Description Languages (HDLs) like VHDL, Verilog, SystemC and many others. HDL allows defining precise description of digital circuit with the timing information to the design tools can synthesise, map, place and route the HDL design accordingly. However this design process involves numerous verification and debugging steps, which increases the time to market from weeks to months, depending on the complexity of the interested algorithm. In order to reduce the required design time, and effort new High Level Synthesis (HLS) tools are being used which allows the one to use high level languages like C or OpenCL [1], to design algorithms for FPGA implementation.

Despite the use of HLS tools, still the design route involves numerous verification and debugging steps, which causes a gap between efficient FPGA realization and programmability. In comparison to the conventional FPGA Design which now includes HLS tools as well, this paper proposes a dataflow based designed toolset that targets reprogrammable custom designed soft processors based on current FPGA technology. The benefits for using an adopted approach which uses bespoke designed soft processors, are guaraneted performacne, and resource usage, easily reprogrammability, even potential support for runtime reconfigurability.

Proposed tool-chain shown in Figure 1, accepts the algorithms in dataflow language CAL, and compiles them with exploiting task and data parallelism [2], [3], to be used in reconfigurable small soft-core RISC architectures on FPGA. Our approach is to create a reconfigurable high-level medium that keeps the flexibility of FPGA logic and without affecting the performance/throughput of the FPGA logic with the use of hardened DSP logics and block rams within the FPGA.

The paper is organised as follows. Section 2 reviews related background work in the area of FPGA Design tools concentrating on High Level Synthesis (HLS) tools. Section 3 briefly describes the soft-processor architecture and its capabilities. Section 4 outlines the toolset, how the programming paradigm is achieved. Section 5 presents the case studies where the toolset is used with the designed soft-processor architecture, and their performance comparisons. Section 6 concludes and reviews the proposed approach.

II. Background

The "Reprogrammable" design methodology proposed in this paper aims to remove the requirement of HDL design, sythnesise, place and route processes by replacing the reconfigurability property of FPGAs with the proposed "Reprogrammable" model. In order to do this, an intermediate medium formed by programmable multi-core processors is proposed. The proposed system consists RISC architectures that support Single Instruction Multiple Data (SIMD), and various interprocessor communication methodologies, to provide the required flexibility and programmability. This Reprogrammable medium is designed to be as compact as possible, to increase the efficiency of the use of the available FPGA logic.

As stated the main processing platform is a custom desgined reprogrammable multi-core processors and the toolset that supports this platform uses the CAL Dataflow Language. Dataflow languages, in general have the ability to express the parallelism, and they are easy to identify and resolve data dependencies to exploid concurrency as much as possible.

A. CAL Dataflow Language

A dataflow program consists of actors and its firing rules, where every actor describes the required arithmetic/mathematical operation to process the input stream before passing it to the output streams. The representation of actors in dataflow programming models are done by directed graphs where the nodes represents computations and the arcs in general represents the movement of data. The main principles behind the dataflow design methodology are the concurrency, scallability, modularity and data-driven approach. The term data-driven is used to express the execution control of dataflow with the availablity of the data itself. In this context an actor is a standalone entity, which defines an execution procedure. Actors communicate with other actors by passing data tokens, and the execution is done through the token passings. Combination of a set of actors with a set of connections between actors constructs a Network. Within the defined networks the communication is formed with infinite size of FIFO components.

B. Dataflow Development Environment - ORCC

ORCC is an open source dataflow development environment and compiler framework, that allows the trans compilation of actors and generates equalivant codes depending on the chosen backends. ORCC is developed within the Eclipse-based Integrated Design Environment(IDE) as a pluging with graphical interfaces to ease the desing of dataflow applications [4].

C. Soft-Core processors

In the current state of the art there are number of soft-core processors for FPGA architecture. These include FlexGrip [5], IDEA [6], and DSP48 based processor for MIMO [7]. In detil FlexGrip maps pre-compiled CUDE kernels on soft-core processors which are programmable, flexible and scallable, can operate at 100 MHz. IDEA and Chu et.al processor has a very similar structure with IPPro, all of these processors uses DSP48 processing unit from Xilin FPGA's as their Arithmetic Logic Unit. IDEA processor uses a 8-stage pipeline to achive 407MHz clocking frequency, and Chu et.al supports a very specific instruction set for Multiple Input Multiple Output (MIMO) communication systems and able to work clock frequency at 265MHz.

III. RISC ARCHITECTURE – IPPRO

This section describes the custom designed DSP48 based Reduced Instruction Set Computing (RISC) architecture, which supports a wide range of instructions and various memory accesses. The Image Processing PROcessor (IPPro) is a hand-coded ISA architecture, which uses Xilinx primitives especially DSP48 block as an Arithmetic Logic Unit (ALU), for more efficient and faster processing. IPPro is cabaple of processing 16 bit operations, and uses distributed memory to build three different memory hierachy, which can be listed as Register File, Data Memory, and Kernel Memory. The IPPro architecture uses a 5 stage balanced pipelined architecture shown in Figure 1.

IPPro is capable of running at 526 MHz on Xilinx SoC in particular XC7Z020-3 using one DSP48E, one BRAM and 330

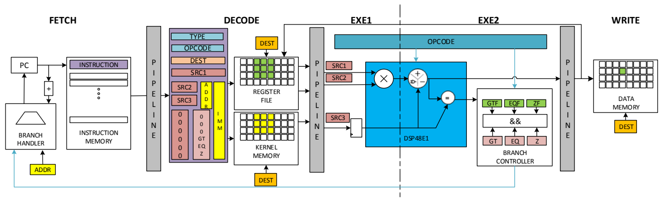


Fig. 1: IPPro Architecture

Slice Registers per processor. The main idea of the processor was to keep it compact, reprogrammable, and scalable as much as possible to achieve high throughput rates compared to custom-made HDL designs.

Overall IPPro has 3 addressable memory locations within the processor core:

- Register File – Register File (R-R)
- Register File – Data Memory (R-D)
- Register File – Kernel Memory (R-K)
- Register File – Immediate (R-I)

Register file is used for regular memory locations, where seperate data's can be stored and processed. Data memory is the main data input and output location for IPPro where the input, and output streams are stored. Kernel memory is a specilazed location designed for window and filter operations for coefiicient storage. Immediate memory is used to reduce number of the register file ans as well as load and store operations, for operations which one operand is constant.

An overview of the supported instructions can be seen in Table I. IPPro instruction set capable of processing basic arithmetic and logical operations for different addressing modes. In addition to unary operations IPPro instruction set, also have support for trinary expressions such as MULADD, MULSUB, MULADDK, and others shown in Table I. The support for trinary expressions added to the tool chain in order to benefit this feature.

TABLE I: 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	

Given the limited instruction support and requirements from the application domain, the overall system is capable of adding coprocessor(s) to support more complex processes. Ongoing research is being done for adding coprocessors like Division and Square Root.

IV. Dataflow toolset

This section describes the toolset that allows to represent interested image processing algorithms in CAL Dataflow Langugage [8], and compiling the actors to work with the custom multi-core architecure.

The overall algorithm design and compilation scheme involves the following steps:

1) Algrotihm implementation in CAL Dataflow Language
2) Profiling the algorithm, and spotting the required changes within the CAL description of the chosen algorithm
3) Compiling the algrotihm targeting the IPPro backend.
4) Loading the generated binary file to the development board through host operating system.

The design approach shown in Figure 2, shows the mapping of actors to multi-core processors. During the compiling process each actor is compiled and mapped to a processing element and the interconnection between processing elements are assigned as FIFO channels. Given the architectural limitation, ie lack of stack infastructure, support for functions calls are limited to the size of the Instruction Register of the IPPro. As a consequence lexical nested routines are not implicitly supported, the detailed limitations and assumptions are explained in the limitations section (B).

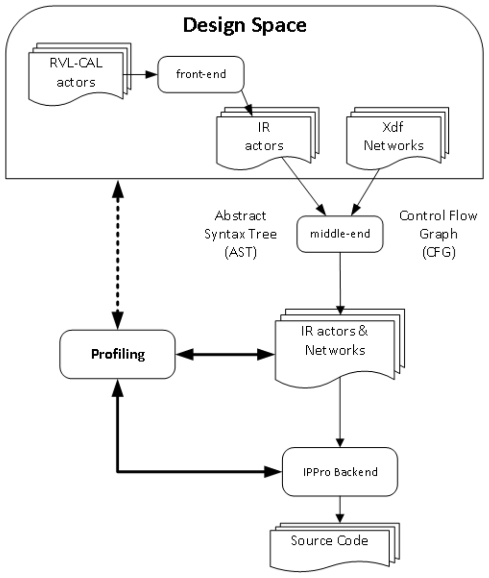


Fig. 2: Dataflow Toolset for soft-core processors

A. Compilation Flow

The compilation flow within the ORCC tools is composed of two distintive steps. The first step of the compilation trans compiles each actor to its Intermediate Representation (IR). The IR is used within the compiler to keep the modularity and be able to target different backends. The latter step of the trans compilation is the conversion of the IR to IPPro assembly code. It is notable that required transformation for a specific backend should be done before the latter trans compilation.

In this concept, revisiting the IR should requires for optimizing the trans compilation for a specific backend. For istance IPPro is able to process MUL & ADD operation in single clock cycle and the IR should be revisited to replace consecutively MUL and ADD operations with a single MULADD operation.

B. Limitations

Given the CAL Language Lexical semantic properties, it is not possible to fully support the compilation process from CAL to IPPro. Consequently this imposes some limitation on the trans compilation process within the toolflow.

Given that, A_x represents the every actor within the network, where $0 < x \leq N, N \in \mathbb{Z}$. Assuming that every A_x may contain variable number of actions,$ac_{x,y}$,due to the purpose of the interested algorithm, where x is the actor, and y is the action index. As previously stated, since the stack infrastructure is not supported the function calls are limited with the size of the instruction memory(IM) and only possible access to IM is done by sequentially and in-order. This limited functionality arises the problem to re-factor and serialize $ac_{x,y}$ to one ac_x per A_x. At the current state, this re-factoring is being done by the algorithm designer, where one should consider the target hardware limitation and re-factor the actor structure.

C. Profiling

Efficient implementation of interested algorithms on a specific computing platform requires niche expertise and knowledge. Especially in embedded platforms algorithm designer should be aware of the capabilities and memory structure in order to achieve high performance implementations. In order to realize efficient algorithm implementations, toolset must be aware of the target embedded platform. In our current case studies all the algorithms are profiled by hand and optimized for the targeted platform according to the limitations and supported instruction set. However Simone et.al [9] proposed a very beneficial design space framework for profiling and optimizing algorithms. ♠**BB:** Should I say in future we will integrate TURNUS?♠

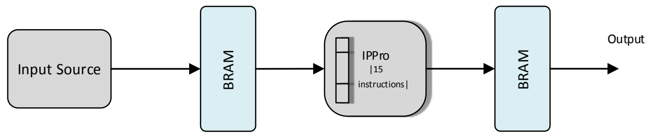
V. Case Studies

In this section case studies have been used to demonstrate the applicability of our approach. CAL dataflow language is used to describe the designs and designs are compiled using the proposed tool-set.

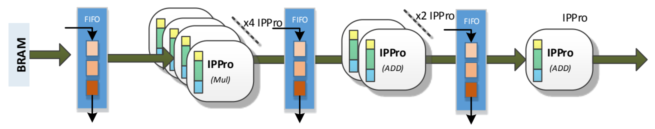
A. Finite Impulse Response (FIR) Filter

In this case study basic FIR filter is targeted for proof of concept. In detail this particular FIR filter is a 3 tap fir filter, with 8-bit fixed point coefficients and 16-bit datapath. FIR filter implemented in two cases, one with single core, and latter with multiple IPPro in streaming mode.

In case 1 shown in 3a, single core IPPro is used for FIR filter, for this particular case input values are loaded to Register File and then processed by MUL and ADD instructions. The need for load and store for every input reduces the througput of the design, which is 430b/s at 404MHz clocking frequency. In order to increase the throughput same FIR filter is implemented in a streaming mode of the IPPro, where Load and Store instructions are by-passed and FIFO's are used for processor communication, shown in 3b).



(a) Single IPPro



(b) Multiple IPPro

Fig. 3: FIR filter with IPPro.

In the streaming mode 7 IPPros are being used to increase the throughput of the design, where every IPPro instructions are transcompiled from one actor, and communication is done through the FIFOs. In this case it is possible achive a througput of 6Mb/s with clocking frequency of 404MHz, which is limited by the Zynq 7020 FPGA.

B. *Histogram of Oriented Gradients (HOG) Algorihtm*

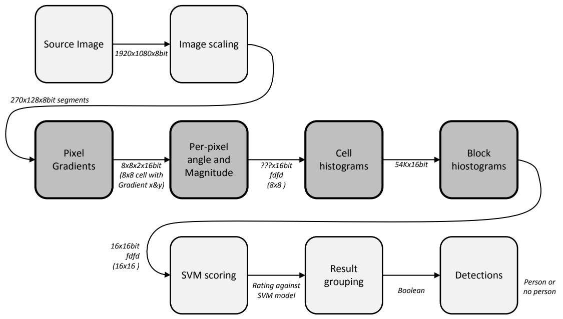
HOG algorithm is a well known algorithm used for human detection, used in pedestrian detection using the gradient orientation [10]. HOG algorithm converts the pixel intensity information to gradient information, where gradient consist of magnitude and direction. After this step the vectors are formed with the extracted information from gradients. At the last stage of the algorithm Support Vector Machine (SVM) is used to achieve human detection from extracted vector information. The overall algorithm's processing blocks shown in 4a, where parts that have been chosen for IPPro implementation has been identified in grey blocks.

Colm et.al [11] implemented HOG algorithm using the IPPro platform and compared the design with hand-written design and proved that the HOG algrotithm can be accelerated up to 3.2 times to give better performance, if 90 cores used within the system. Colm et.al work demonstrated that our approach is highly scalable and reduces the design effort and time.

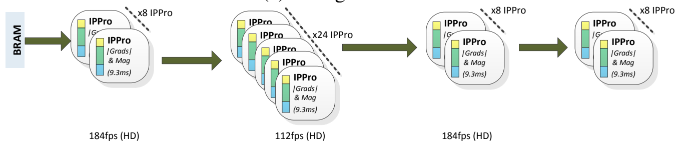
TABLE II: HOG Algorithm Comparison

	IPPro(Single Core)		IPPro(16 Core)		Hand Coded	
Gradient & Magnitude (GM), Bin. & Cell Hist. (B&CH)	GM	B&CH	GM	B&CH	GM	B&CH
Frequency	404 MHz	404 MHz	404 MHz	404 MHz	288 MHz	164 MHz
Throughput[fps] (1920 x 1280 image size)	7.4 fps	4.9 fps	118 fps	78 fps	139 fps	79 fps

Colm et.al identified four processing blocks as a candidate for IPPro implemenation which are Gradient & Magnitude, and Binning and Cell histogram calculation. The performance metrics for these blocks with comparison to hand-written HDL code is shown in II. In this particular study interested blocks are implemented using for 1 core and then desined scaled up according to the algorithms throughput requirements. This IPPro implementation is done by hand, and optimizations are made as a result of algorithm profilings.



(a) Single IPPro



(b) Multiple IPPro

Fig. 4: HOG algortihm with IPPro.

VI. Conclusion

The work proposed in this paper demonstrates a dataflow tool-set to control soft-core processors, and its capabilities to reduce the design time and effort. The overall design tool-set with limited optimizations and limited memory access is in a working order, and many optimizations and profiling are planned to comply the design tool-set in order to match the same perforamce of any hand-crafted design.

Future Work

Given the target application area, which is distibuted image pre-processing, one of the main limitations for current version of the processor is the lack of division and square root instructions. The support for these instructions is in our agenda for future work, along with the optimization and profiling support within the design tool-set.

Acknowledgement

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] D. Singh, "Implementing fpga design with the opencl standard," *Altera whitepaper*, 2011.

[2] M. Arslan, J. Janneck, and K. Kuchcinski, "Partitioning and mapping dynamic dataflow programs," in *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, Nov. 2012, pp. 1452–1456.

[3] S. S. Bhattacharyya, G. Brebner, J. W. Janneck, J. Eker, C. Von Platen, M. Mattavelli, and M. Raulet, “OpenDF: a dataflow toolset for reconfigurable hardware and multicore systems,” *ACM SIGARCH Computer Architecture News*, vol. 36, no. 5, pp. 29–35, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1556449

[4] H. Yviquel, A. Lorence, K. Jerbi, G. Cocherel, A. Sanchez, and M. Raulet, “Orcc: Multimedia development made easy,” in *Proceedings of the 21st ACM International Conference on Multimedia*, ser. MM ’13. New York, NY, USA: ACM, 2013, pp. 863–866. [Online]. Available: http://doi.acm.org/10.1145/2502081.2502231

[5] K. Andryc, M. Merchant, and R. Tessier, “FlexGrip: A soft GPGPU for FPGAs,” in *Field-Programmable Technology (FPT), 2013 International Conference on*. IEEE, 2013, pp. 230–237.

[6] H. Y. Cheah, S. A. Fahmy, and D. L. Maskell, “iDEA: A DSP block based FPGA soft processor,” in *Field-Programmable Technology (FPT), 2012 International Conference on*, 2012, pp. 151–158.

[7] X. Chu and J. McAllister, “FPGA based soft-core SIMD processing: A MIMO-OFDM fixed-complexity sphere decoder case study,” in *Field-Programmable Technology (FPT), 2010 International Conference on*, 2010, pp. 479–484.

[8] J. Eker and J. Janneck, “Cal language report,” *University of California at Berkeley, Tech. Rep. UCB/ERL M*, vol. 3, 2003.

[9] S. Brunei, M. Mattavelli, and J. Janneck, “Turnus: A design exploration framework for dataflow system design,” in *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, May 2013, pp. 654–654.

[10] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005*, vol. 1, 2005, pp. 886–893 vol. 1.

[11] C. Kelly, F. M. Siddiqui, B. Bardak, R. Woods, and K. Rafferty, “Histogram of oriented gradients front end processing: an fpga based processor approach,” in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, Oct 2014.