AN HETEROGENEOUS COMPILER OF DATAFLOW PROGRAMS FOR ZYNQ PLATFORMS

Endri Bezati[1] *Simone Casale-Brunet*[2] *Romuald Mosqueron*[3] *Marco Mattavelli*[2]

[1]EPFL VLSC, [2]EPFL SCI STI MM, École Polytechnique Fédérale de Lausanne, Switzerland
[3]School of Business and Engineering Vaud HES-SO, REDS Institute at HEIG-VD,
University of Applied Sciences Western Switzerland, Yverdon-les-Bains, Switzerland

ABSTRACT

In recent years, the number and variety of heterogeneous multiprocessor system-on-chip MPSoCs, such as for instance Zynq platforms, has sensibly increased. However, today all design flow solutions capable of programming the different components of such platforms require to the designer either to modify the software or hardware based designs to obtain higher performance implementations. Thus, the developer needs to either rewrite functional blocks in HDL or to use high-level synthesis of C-like sequential languages with platform locked extensions. In this paper, a compiler infrastructure that takes as input a single behavioral representation, expressed in high-level dataflow RVC-CAL language, is proposed for programming any of the components of heterogeneous Zynq MPSoCs platforms without the need of modifying any line of code on the design.

Index Terms— dataflow programing, heterogeneous computing, RVC-CAL, Zynq

1. INTRODUCTION

The emergence of reconfigurable Multiprocessor Systems on Chip (MPSoCs) platforms with several processing cores and reconfigurable logic resources such as the Xilinx Zynq-7000, the Ultrascale+ series, the Altera/Intel Stratix and Arria SoCs, provide to designer powerful means to implement large and complex system designs. Nonetheless, to fully exploit the potential capabilities of such devices, it is required to execute multiple tasks in parallel using both processing cores and programmable logic resources. Xilinx and Altera devices are supported by CAD tools for programming the MPSoCs by using C/C++ designs based on proprietary pragmas or OpenCL APIs, as a consequence the design incorporates platform locked attributes. An alternative method for programming MPSoC is to use more general dataflow programming approaches, like RVC-CAL, which is platform agnostic by design.

The contributions of this work are the following: how to express the system architecture of a dataflow program in a heterogeneous platform, the necessary interfaces for enabling bidirectional communication between heterogeneous processing elements (PE), and the code generation for processing cores and programmable logic. The mapping of the actors to each processing element of the heterogeneous platform is statically assigned before the code-generation. For an automatic mapping procedure based on the profiling of the dataflow program, the interested reader can refer to [1].

Dataflow programming models have a long and rich history dating back to the early 1970s [2, 3]. A *dataflow program* is intended as a (hierarchical) directed graph in which nodes (called *actors*) represent the computational kernels and directed edges (called *buffers*) represent the lossless, order preserving, and point-to-point communication channels between actors. Buffers are used to communicate sequences of data packets (called *tokens*). In literature, several variants of dataflow have been introduced [3, 4, 5], often referred to as different dataflow Model of Computation (MoC). One common property shared by all dataflow MoCs is that individual actors encapsulate their own state, and do not share their internal states. Instead, actors communicate with each other exclusively by sending and receiving tokens by means of buffers connecting them. The absence of race conditions makes the behavior of dataflow programs more robust to different execution policies. The paper is organized as follows: Section 2 gives a summary of related work of tools for dataflow programming, Section 3 describes the compiler infrastructure for the Zynq platform and the original contribution provided by this work. Section 4 demonstrates the capabilities of the compiler by partitioning a design into multiple arbitrary configurations, and finally, Section 5 concludes the paper.

2. RELATED WORK

PREESM [6] based design flows are based on decidable parametric synchronous dataflow (PiSDF) [7] MoC, recently [8] PREESM has been extended to support MPSoC by using Xilinx SDSoC. A tool supporting mapping Kahn Process Network based designs onto heterogeneous MPSoCs is MAPS [9], it uses C to describe actors. In [10] , the author are using parametric SDF with actors expressed as C++ classes for a heterogeneous platform without reconfigurable logic. The DAL framework [11] is based on KPN MoC

and an extension [12] targeting heterogeneous systems with OpenCL enabled devices is also available. DKPN such as the ones reported in [13], support mixed Dataflow Process Network(DPN)/KPN execution models for mapping streaming applications on heterogeneous MPSoC. Compared with the previous works, the approach of this paper is to use as MoC [14] the DPN with firing, specifically expressed by the ISO/IEC standardized programming language called RVC-CAL [15]. With the RVC-CAL expressiveness, it is possible to describe dataflow networks composed by processing elements implementing DPN, KPN, SDF and CSDF(cyclo-static dataflow) MoCs.

3. CONTRIBUTIONS ON EXELIXI FOR ZYNQ

The Open RVC-CAL compiler [16] (Orcc) is based on Model-Driven Engineering (MDE) by representing the Intermediate representation (IR) with meta-models. The compiler infrastructure used as main compiler backbone for the design flow is made of four main components: the front-end, the IR, the back-end and the interpretation stage. In this work a tool call Exelixi [17] is used as middle-end and back-ends of Orcc. A illustration of the compiler infrastructure and its main elements is depicted in Fig. 1.

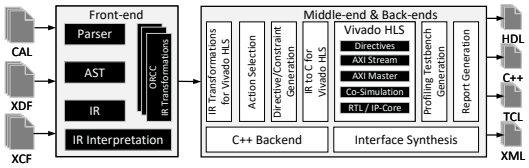


Fig. 1: The compiler infrastructure that we use in this work.

Exelixi accepts as input the Orcc IR built from RVC-CAL networks. For a given a platform, Exelixi transforms the IR of each actor by generating C++ and/or HDL source code. A dataflow program can thus be partitioned according to the specification provided by an XML configuration file (XCF) or directly by interacting with the user interface provided by Orcc user interface. Exelixi reads the mapping configuration and depending on how the actors of the network are mapped on a hardware (FPGA programmable logic) or a software processing element (CPU eg. x86 or ARM), generates the appropriated source code. For software mapped actors it generates C++ code as described in [18]. Specifically for the Zynq platform, Exelixi generates a set of baremetal I/O drivers/functions for controlling the peripherals of the ARM processor. For hardware mapped actors it generates C/C++ code for Vivado HLS. The actor I/O uses the Vivado HLS stream interface and the action selection as described in [17]. Every actor is exported as a Vivado IP core. The hardware network is represented as a Vivado TCL script file which interconnects the IP core actors with an Accelerator FIFO IP core. The RTL interface between the actors and the FIFOs is called *Accelerator Handshake Interface* or ACI. Finally, the partitioned network is represented in Vivado as a hierarchical IP core, depicted as PL Partitioned Network in Fig. 2.

In this work, the capabilities described in [19] have been extended by creating a new multiple input/output interface for mapping dataflow programs on Zynq platform. Zynq MP-SoCs are separated into two parts, a Processing System (PS) and a Programmable Logic (PL) section. The on-chip PS is connected to the on-chip PL through multiple ARM AMBA AXI ports. There are two 32-bit AXI master interfaces (GP Master), two 32-bit AXI slave interfaces (GP Slave), four 64-bit high-performance AXI slave interfaces (HP Slave), and one 64-bit AXI Accelerator Coherency Port (ACP slave) interface. HP slave interfaces have a direct connection to the DDR memory controller of the PS.

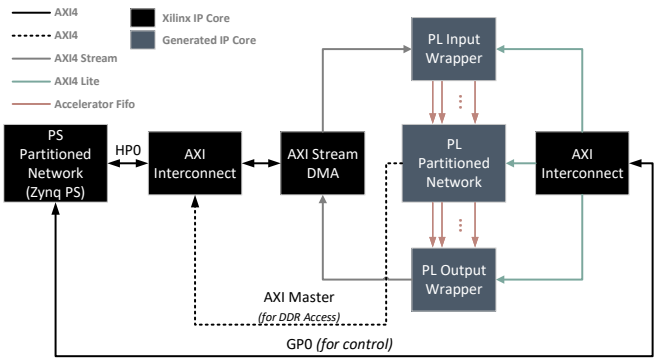


Fig. 2: A high-level representation of how Exelixi interfaces the PL and the PS parts of the Zynq platform.

The HP0 is used for data communication and the GP0 for the control of the IP cores. Fig. 2 depicts a high-level representation of how Exelixi connects the PS and PL sections, for simplicity not all connections are represented. The data communication between PS and PL is obtained by using the AXI Stream DMA IP core. This core is enabled for read and write channels. All the incoming and outgoing FIFOs of the PL shares a single AXI Stream DMA IP Core. Exelixi generates two IP cores for the communication with the PL partitioned network: the PL Input wrapper and the PL Output wrapper, for converting the AXI Stream data from/to a set of ACI interfaces of the DMA IP core. The AXI Stream DMA IP core is connected to the HP0 slave interface through an AXI interconnect. All bidirectional communications are controlled through the PS. The PS controls, from the GP0 interface, in which Accelerator FIFO IP Core the data should flow from or to the AXI Stream DMA IP core. This is achieved by baremetal drivers in PS, which are generated automatically depending on the partitioning configuration, that controls the AXI DMA IP Core and the PL I/O wrappers. If an actor in the PL requires a large state variable (e.g. a framebuffer for a video decoder), then an AXI Master interface from the PL actor is connected to the HP0. Thus, it is possible for any actor

to access directly the DDR memory. The compiler does not automatically attribute large state-variables to the DDR, is up to the designer to provide a directive to Exelixi (*@external*) linked to the RVC-CAL state variable to enable such a feature. Finally, given a partitioning configuration, the compiler generates a Vivado TCL file that represents all the interconnection between the generated IP cores and the Xilinx ones.

4. EXPERIMENTAL RESULTS

The purpose of following experiments is to demonstrate that from the same behavioral representation without changing any line of code, but just giving a mapping configuration, it is possible to partition the actors of the dataflow program into the heterogeneous PEs of the MPSoC.

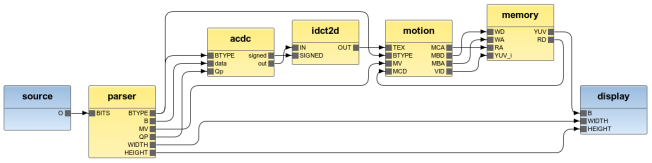


Fig. 3: MPEG-4 SP Decoder, yellow boxes represents a network of actors, blue boxes represents actors.

To demonstrate the compiler capabilities we use the same dataflow MPEG-4 SP decoder as in [20]. The design uses the RVC-CAL behavioral description and its fully compliant with the ISO/IEC 14496-2 standard. Fig.3 represents a snapshot of the decoder as displayed in the user interface of Orcc. The blue boxes represent actors and the yellow boxes compositions of actors (or a network of actors). The source and display actors implement platform based functions for reading a file from the system (e.g. from an SD card) and displaying results on a screen respectively. The entire description of the decoder is a composition of 34 actors, connected in five networks: *parser*, *acdc*, *idct2d*, *motion*, and *memory*. The second row of Table 1 specifies the number of Actors per network.

The decoding process of the MPEG-4 SP can be summarized as follows. The *source* actor reads MPEG4-SP stimuli from a file and feeds the parser with a stream of bytes. Then, the *parser* operates the entropy decoding and sends the data to the rest of the decoder. The *acdc* block performs the AC/DC prediction for Intra macroblocks. The *idct2d* performs the inverse discrete cosine transform on those macrbolocks and then the motion compensation (*motion*) block selectively adds the macroblocks by issuing from the IDCT the blocks taken from the previous frame. Consequently, the motion compensation needs to store the entire previous frame of video data, which it also needs to address into with a certain degree of random access. This operation is effectuated by the actor *ddr* included in the *memory* network. Finally, the decoded frames are displayed by the *display* actor.

We are using the Zedboard embedded board for the experimental results. The Zedboard contains a Xilinx XC7Z020 MPSoC with two ARM cores (PS) clocked at 667Mhz, an FPGA (PL) with 85k slices, 512Mb DDR3 memory, and different I/O peripherals. Exelixi currently supports only baremetal applications on the PS and only one ARM core is used. The generated code includes any optimized function for the ARM core or optimized IP cores in the PL. For all the partitions reported below on the PS, the source code is compiled with *-O3* flag. All actors on the PS are executed sequentially with a non-preemptive round-robin scheduler. All actors in PL are executed in parallel.

	src	parser	acdc	idct2d	motion	memory	display	PL Connections	
Actors	1	5	7	12	6	2	1	**Inputs**	**Outputs**
P1	PS	PS	PS	PS	PS	PS	PS	0	0
P2	PS	PL	PS	PS	PS	PS	PS	1	6
P3	PS	PL	PL	PS	PS	PS	PS	1	6
P4	PS	PL	PL	PL	PS	PS	PS	1	5
P5	PS	PL	PL	PL	PL	PS	PS	2	6
P6	PS	PS	PL	PL	PL	PL	PS	4	1
P7	PS	PS/PL	PL	PL	PL	PL	PS	7	1
P8	PS	PL	PL	PL	PL	PL	PS	1	3

Table 1: Partitioning configuration of actors and networks between PL and PS.

The compiler supports most of the Xilinx 7th series development boards as long as they are supported by Vivado. The user of Exelixi can select the mapping configuration either by the user interface of Orcc or by specifying an XML configuration file (XCF) which actor or network (as a subnetwork of the top network) is going to be executed either in the PS or the PL. To demonstrate the capabilities of the compiler, 8 partitioned configurations have been created as depicted in Table 1. From P1 to P6, one network after the other has been gradually included in the PL. For configuration P7 the *parser* network is divided between the PS and the PL, and for P8 all networks are placed in the PL. In addition, Table 1 shows the number on incoming and outgoing connections to PL for each partition. Each FIFO of has a depth of 1024 tokens even for the border FIFOs that connect the PS and PL partitions. The PL part of the XC7Z020 has limited availability of BRAMs which restricts the usage of large depth FIFOs. To optimize the performance of the PS partition, the actor scheduler is a non-preemptive round robin, since decreasing the FIFO depths will decrease its output throughput [1]. It can also be mentioned that the decoder does not need such large FIFO depths [20]. Finally, the *source* and *display* actors are always included on the PS partition, as they use native Xilinx baremetal functions.

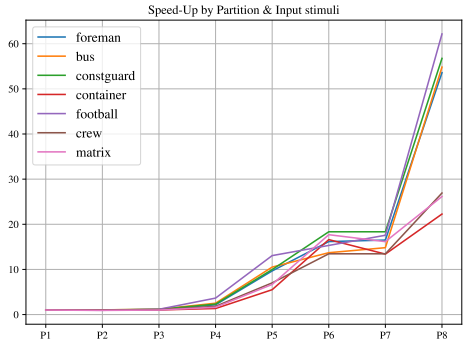
	Resources %								
Paritition	P1	P2	P3	P4	P5	P6	P7	P8	Available
LUT	0	18	22.03	27.81	31.83	29.64	32.23	34.89	53200
LUTRAM	0	4	4.66	4.63	4.82	6.37	6.4	6.4	14700
FF	0	9	10.78	14.19	16.07	16.47	17.25	17.84	106400
BRAM	0	3.62	12.44	30.8	38.57	34.93	48.93	54.29	140
DSP	0	0	1.36	4.55	4.55	4.55	4.55	4.55	220

Table 2: FPGA Resources for different partitioning configurations.

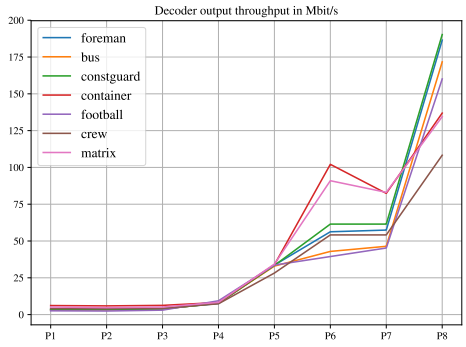
Exelixi permits to set the clock configuration for the PS and the PL of the Zynq platform. For the PS the maximum clock frequency is set to 667Mhz. For the PL we use a clock frequency of 100Mhz. Table 2 depicts the post implementation FPGA resources used for each mapping configuration. As we include gradually the networks to the PL partition, the hardware resources are increased. The most demanding resource usage in the FPGA is the BRAMs on which the FIFO are placed. The DSPs are used in the DC prediction actor of the *acdc* network which uses a 13-bit integer division and the multiplications of the *idct2d*. It is to be mentioned that the ddr actor of the *memory* network, which stores the frame-buffer (16MBytes) of the decoder, for P6, P7, and P8 has a direct connection to the DDR3 memory of the board from the AXI HP0 port of the Zynq.

		Frames/sec							
Stimuli	**Format**	P1	P2	P3	P4	P5	P6	P7	P8
foreman	QCIF	12	12	13.7	26	116	194	198	643
bus	CIF	2.7	2.8	3.2	6.8	28.4	37	40	148
costguard	CIF	2.89	3	3.3	6.7	28.8	53	53	164
container	CIF	5.3	5.1	5.4	7.1	29.2	88	71	118
football	CIF	2.22	2.1	2.6	8.1	29	34	39	138
crew	4CIF	1.04	1.02	1.1	1.9	7.3	14	14	28
matrix	480p	1.3	1.2	1.3	2.1	8.68	23	21	34

Table 3: Frames per seconds for different partitioning configurations.



(a) Speed-Up



(b) Throughput in Mbit/sec

Fig. 4: Speed-up and output throughput in Mbit/s of the decoder given the partitioning configuration and input stimuli.

Table 3 reports the frame per second obtained by the decoder for the different partition configuration running the eight input stimuli. The stimuli samples has low, medium, and high motion and coding complexity. In addition, these samples are commonly used among researchers apart from the matrix stimuli which has been encoded by the authors. The performance of P1 and P2 (even with the *parser* on the PL) are among the lowest in our experiment. This due to the processing power of the ARM core and number of actors that needs to be executed on the single core. The performance of the decoder is gradually increasing for all input stimuli as more actors migrates from the PS to PL.

Furthermore, Fig. 4a reports the speed-up by partition and input stimuli. We observe that P1, P2, and P3 have relatively the same performance. Once the *idct2d* network migrates to PL, the speed-up is doubled. Input stimuli such as the football benefit the most as it uses only Intra frames. Once the *motion* migrates to the PL the speed-up drastically increases by a factor of 10 and more for P5, P6 and P7 compared to P1. Partition P6 and P7 have similar performance. The parser network of the P7 partition is split between the PS and the PL. A serialize actor that transforms bytes into bits and the entropy decoding actor is located in PS, the macroblock expansion, the motion vectors, and motion vector reconstruction parsing actors are located in the PL. All the other actors of P7 are partitioned as defined in Table 1. P6 is faster for container and matrix input stimuli because of the PS partition in P7 schedules more data transmission than executing the entropy parser actor. This effect can be observer for those partitions on Fig. 4b which represents the output throughput of the decoder in Mbit/s. For P6 vs P7, it can be concluded that the handling of the communication by the PS slows down the execution of the actors on the ARM and limits the input throughput of the PL. Finally, the P8 has the highest speed-up and output throughput than the other partitioning configurations as all networks executes in parallel in the FPGA.

5. CONCLUSION

The paper presents a compiler infrastructure of dataflow programs for heterogeneous platforms such as the Zynq. It offers to its user the possibility to partition dataflow networks into different processing units without the need of adapting the original behavioral description to each targeted execution component as required by other methodologies or design flows. The implementation of complex signal processing applications on MPSoCs is greatly simplified by taking away the hurdles of coding low-level drivers, peripheral interconnection and HDL code for FPGAs. We demonstrated that our compiler can partition a dataflow network to varied hardware with heterogeneous communications channels. Furthermore, our tests revealed that using this compiler we can accelerate by large factors a design once the actors can be executed on both the processing system and in the programmable logic.

6. REFERENCES

[1] Malgorzata Maria Michalska, *Systematic Design Space Exploration of Dynamic Dataflow Programs for Multi-core Platforms*, EPFL, Lausanne, 2017.

[2] Jack B. Dennis, "First version of a data flow procedure language," in *Symposium on Programming*, 1974, pp. 362–376.

[3] Gilles Kahn, "The Semantics of Simple Language for Parallel Programming," in *IFIP Congress*, 1974, pp. 471–475.

[4] Edward A. Lee and David G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. 36, no. 1, pp. 24–35, 1987.

[5] E.A. Lee and T.M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773 –801, may 1995.

[6] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J. Nezan, and S. Aridhi, "Preesm: A dataflow-based rapid prototyping framework for simplifying multicore dsp programming," in *2014 6th European Embedded Design in Education and Research Conference (EDERC)*, Sept 2014, pp. 36–40.

[7] S. S. Bhattacharyya and W. S. Levine, "Optimization of signal processing software for control system implementation," in *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, Oct 2006, pp. 1562–1567.

[8] L. Suriano, A. Rodriguez, K. Desnos, M. Pelcat, and E. de la Torre, "Analysis of a heterogeneous multi-core, multi-hw-accelerator-based system designed using preesm and sdsoc," in *2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, July 2017, pp. 1–7.

[9] J. Castrillon, R. Leupers, and G. Ascheid, "Maps: Mapping concurrent dataflow applications to heterogeneous mpsocs," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 527–545, Feb 2013.

[10] M. Dardaillon, K. Marquet, T. Risset, J. Martin, and H. Charles, "A compilation flow for parametric dataflow: Programming model, scheduling, and application to heterogeneous mpsoc," in *2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Oct 2014, pp. 1–10.

[11] L. Schor, A. Tretter, T. Scherer, and L. Thiele, "Exploiting the parallelism of heterogeneous systems using dataflow graphs on top of opencl," in *The 11th IEEE Symposium on Embedded Systems for Real-time Multimedia*, Oct 2013, pp. 41–50.

[12] J. Boutellier and T. Nylanden, "Programming graphics processing units in the rvc-cal dataflow language," in *2015 IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2015, pp. 1–6.

[13] P. Arras, D. Fuin, E. Jeannot, and S. Thibault, "Dkpn: A composite dataflow/kahn process networks execution model," in *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, Feb 2016, pp. 27–34.

[14] J. Eker and J.W. Janneck, "CAL Language Report," Tech. Rep. ERL Technical Memo UCB/ERL M03/48, University of California at Berkeley, Dec. 2003.

[15] ISO/IEC 23001-4:2011, "Information technology - MPEG systems technologies - Part 4: Codec configuration representation," 2011.

[16] Herve Yviquel, Antoine Lorence, Khaled Jerbi, Gildas Cocherel, Alexandre Sanchez, and Mickael Raulet, "Orcc: Multimedia development made easy," in *Proceedings of the 21st ACM International Conference on Multimedia*. 2013, MM '13, pp. 863–866, ACM.

[17] E. Bezati, S. C. Brunet, M. Mattavelli, and J. W. Janneck, "High-level system synthesis and optimization of dataflow programs for mpsocs," in *2016 50th Asilomar Conference on Signals, Systems and Computers*, Nov 2016, pp. 417–421.

[18] E. Bezati, R. Thavot, G. Roquier, and M. Mattavelli, "High-level dataflow design of signal processing systems for reconfigurable and multicore heterogeneous platforms," *Journal of Real-Time Image Processing*, vol. 9, no. 1, pp. 251–262, 2014.

[19] E. Bezati, S. Casale-Brunet, M. Mattavelli, and J. W. Janneck, "High-level synthesis of dynamic dataflow programs on heterogeneous mpsoc platforms," in *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, July 2016, pp. 227–234.

[20] Jörn Janneck, Ian Miller, David Parlour, Ghislain Roquier, Matthieu Wipliez, and Mickaël Raulet, "Synthesizing Hardware from Dataflow Programs: An MPEG-4 Simple Profile Decoder Case Study," *Journal of Signal Processing Systems*, vol. 63, no. 2, pp. 241–249, 2009, 10.1007/s11265-009-0397-5.