Dataflow-Functional High-Level Synthesis for Coarse-Grained Reconfigurable Accelerators

Claudio Rubattu, Francesca Palumbo, Carlo Sau, Rubén Salvador, Jocelyn Sérot, Karol Desnos, Luigi Raffo, Maxime Pelcat

Dataflow-Functional High-Level Synthesis for Coarse-Grained Reconfigurable Accelerators

Claudio Rubattu, Francesca Palumbo, *Member, IEEE*, Carlo Sau, Rubén Salvador, *Member, IEEE*, Jocelyn Sérot, Karol Desnos, Luigi Raffo, *Member, IEEE*, and Maxime Pelcat

***Abstract*—Domain-specific acceleration is now a "must" for all the computing spectrum, going from high performance computing to embedded systems. Unfortunately, system specialization is by nature a nightmare from the design productivity perspective. Nevertheless, in contexts where kernels to be accelerated are intrinsically streaming oriented, the combination of dataflow models of computation with Coarse-Grained Reconfigurable (CGR) architectures can be particularly handful. In this paper we introduce a novel methodology to assemble and characterize virtually reconfigurable accelerators based on dataflow and functional programming principles, capable of addressing design productivity issues for CGR accelerators. The main advantage of the proposed methodology is accurate IP-level latency predictability improving Design Space Exploration (DSE) when compared to state-of-the-art High-Level Synthesis (HLS).**

***Index Terms*—Coarse-Grained Reconfiguration, Dataflow MoC, Functional Programming, HLS, CAPH, MDC, FPGA, Design Predictability, Design Productivity.**

I. Context, Background and Motivation

Flexibility and performance are two highly valued properties of processing systems in many application domains. Modern systems include ubiquitous and upgradable devices for IoT or cyber-physical applications that also communicate with cloud infrastructures. These systems must adapt to mutable conditions, while avoiding unpredictable performance degradation.

To boost performance, High Performance Computing (HPC) and embedded systems designers are pushed to opt for Domain-Specific Accelerators (DSAs), built from ad-hoc hardware-coded kernels exploiting parallelism for optimizing performance. Two representative examples of this trend are Amazon Web Services (AWS) and Movidius. On the HPC side, AWS offers FPGAs in a cloud environment [2]. On the embedded side, the Intel® Movidius™ Vision Processing Unit (VPU) is a low-power DSA used in smartphones and drones for computer vision and artificial intelligence applications.

In terms of system flexibility, hardware DSAs based on CGR architectures offer flexibility by adapting to variable application parameters. CGR architectures are traditionally composed of a mesh of Processing Elements (PEs) whose interconnections are reconfigured over time [16] to offer flexibility. CGR architectures inherit from their hardware nature the capacity of executing compute-intensive kernels in an energy-efficient way. However, efficiency and flexibility offered do not come for free. Hardware design is a complex and error-prone task, leading to productivity losses, especially for heterogeneous and irregular targets. High-Level Synthesis (HLS) approaches have been proposed to cope with these losses, obtaining design productivity gains by separating functional system verification, performed from a time-agnostic high-level language, from timed system verification, performed after automatically inferring hardware-specific code [7].

Many HLS tools are now available, such as Xilinx Vivado HLS [17] and Intel FPGA SDK for OpenCL [4]. These tools are based upon imperative, C-like, languages. The algorithm to be implemented is formulated as a *sequence* of instructions operating on mutable data. This choice is motivated by the very large number of developers trained to manipulate imperative languages that have been dominating computer sciences for decades. But, from a hardware perspective, this choice presents two major drawbacks: 1) imperative formulations generally do not distinguish iterations over time from iterations over space, which do not translate uniformly in hardware (the latter do not imply causality and can therefore be parallelized using replication); 2) they implicitly rely on the concept of global memory at the implementation level, which immediately leads to a "bottleneck" on memory accesses [1]. These drawbacks can be circumvented by relying upon so-called applicative or functional languages in which algorithms are described as a (mathematical) composition of side-effect free functions. This approach naturally fits DataFlow (DF) Models of Computation (MoCs). An application is decomposed into independent processing actors, communicating with First-In First-Out data queues (FIFOs), with no global storage or synchronization. This is particularly true for *stream processing* applications, processing data "on the fly" and which benefit significantly from CGR architectures-based acceleration, as found in signal/image processing, media coding/compression, cryptography, video analytics, etc.

This paper, in Section II, introduces a novel methodology for the optimal characterization of virtually reconfigurable DSAs exploiting a DataFlow-Functional (DFF) HLS approach as an alternative to traditional HLS tools based on imperative languages. The proposed methodology is targeted for heterogeneous and irregular CGR architectures, leveraging on application specific PEs and tailoring the interconnect to minimize FIFOs. For prototyping purposes, and to compare with commercial flows, experiments target FPGA technologies and demonstrate that, thanks to its modularity and abstraction capabilities, the proposed approach guarantees latency predictability (see Section III). Beside we also show (Section

"Thanks to the projects CERBERO (H2020,732105) and PROSSIMO (POR FESR 2014/20-ASSE I), and to HiPEAC (H2020,687698) for a collaboration grant."

C. Rubattu and F. Palumbo - University of Sassari (e-mail: crubattu@uniss.it). C. Sau and L. Raffo - University of Cagliari. R. Salvador - Universidad Politécnica de Madrid. J. Sérot and M. Pelcat - the Institut Pascal. C. Rubattu, K. Desnos and M. Pelcat - INSA Rennes.

IV) that our DFF approach leads to improved performance for DSAs with CGR architectures.

II. Proposed Dataflow-Functional HLS

The proposed framework is built from two existing tools developed by the authors: the CAPH compiler and the Multi-Dataflow Composer (MDC) toolset. CAPH [12] is an open-source, domain-specific framework for the specification, simulation and implementation of stream-processing applications based on a dynamic dataflow MoC. Applications are specified as DataFlow Networks (DFNs) using a higher-order, polymorphic functional language. The behavior of each DF actor is defined as a set of transition rules using pattern matching on structured data, resulting in improved abstraction capabilities. The CAPH toolchain provides graphical visualization of DFNs; code simulation with trace facilities; and HLS producing SystemC code for simulation and resource-monitoring purposes, as well as platform-agnostic ready-for-synthesis VHDL code. CAPH provides 1:1 DFN PEs synthesis, but no support for reconfiguration. MDC is a suite for the design and development of heterogeneous and irregular DSAs based on a set of DFNs to specify the desired behavior of the system [6]. It builds reconfigurable datapaths, leveraging on a two-step approach: 1) a *model-to-model compiler*, the Multi-Dataflow Generator (MDG), derives a multi-functional DFN [10] starting from a set of disjointed DFN specifications; 2) a *dataflow-to-hardware mapper*, the Platform Composer (PC), deploys an HDL description of the CGR datapath. Up to this work, MDC was specific to RVC-CAL language and the actor communication protocol was hardwired on the resulting CGR architecture. Now users define custom communication protocols, and MDC accordingly implements the handshake among PEs. MDC provides N:1 DFNs mapping on a CGR DSA, but it does not support PE synthesis.

Based on the described complementaries, this work proposes a fully automated toolchain, integrating MDC and CAPH, for specifying and deploying CGR DSAs. The toolchain architecture is depicted in Fig. 1. The proposed DFF HLS flow consists of three main phases:

1) *Composition* - Model-to-model compilation performed by MDC MDG, taking as inputs generic DFNs. Output: high-level multi-dataflow DFN of the DSA.
2) *Optimization* - Optimal sizing of the actor-connecting FIFOs . Optimization starts from an estimation produced by the CAPH Compiler SystemC backend, which is adapted to the multi-functional DFN case by worst case analysis.
3) *Generation* - Deployment of the CGR DSA. The MDC PC component outputs a top-level HDL module, corresponding to the optimized multi-functional DFN, using the CAPH generated Hardware Component Library (HCL) actors.

Figure 1b summarizes the flow steps. (1) Users are required to provide the input set of specifications (N different DFNs) to be accelerated using the CAPH language. Starting from these inputs, three parallel steps take place to i) make CAPH DFNs compliant with MDC through the CAPH-to-XDF parser (2), ii) optimize sizing of the buffer connecting DF actors (3), and iii) generate the target-independent HCL (4). Three more steps

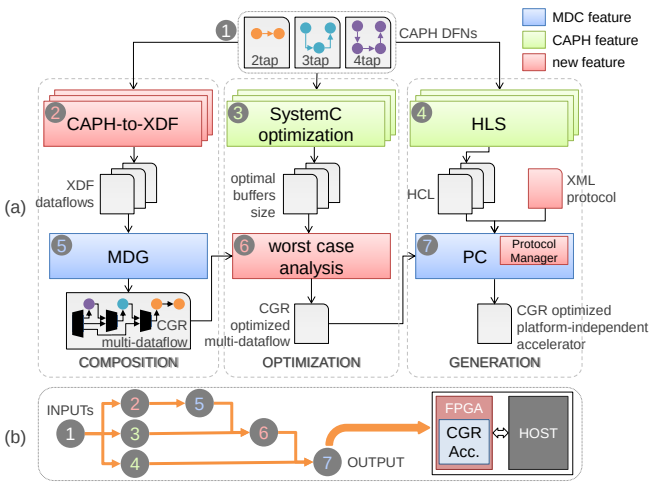


Fig. 1. Proposed flow: components (a); and execution sequence (b).

come in succession. The N XDF networks produced by (2) are merged in a multi-dataflow network (5), which is optimized (6) using the worst case analysis results of (3). Finally, the CGR platform-independent DSA is deployed in (7). During the merging process (5), common actors are identified and connections minimization is guaranteed. In the CGR DSA, combinatorial switching elements are used to access shared modules, this may affect frequency but not latency.

III. Proposed Approach: Features and Novelty

CGR architectures provide the flexibility needed by modern signal processing applications, while achieving high efficiency through specialization. Their features and granularity are highly varied [3], and several works in literature address their design issues. Raffin et. al [8] propose the ROMA framework for the scheduling, binding and routing of reconfigurable accelerators for multimedia applications. The ROMA architecture is composed of custom reconfigurable PEs and the framework is specific to them, following an approach similar to general purpose devices, thus loosing specialization and, in turn, efficiency. Voros et. al [15] leverage on a fixed CGR structure that comes along with its synthesis flow, and where 3 different types of PEs can be used for accelerating purposes. Our effort focuses on a more specific problem: we discuss in this paper a flow to characterize and synthesize circuits to be embedded in a DSA infrastructure. MDC has been combined in previous work with the Xronos HLS to offer DFF HLS to build a Xilinx-specific DF-to-hardware design environment [9]. Table I reports a qualitative comparison among our work and other frameworks available in literature for CGR DSAs.

TABLE I
Proposed flow vs. state-of-the-art.

	[8]	[15]	[9]	this work
Framework: Input Specs	C	C	dataflow	dataflow
Framework: (Re-)mapping	Y	Y	Y	Y
Framework: PE generation	N	Y	Y	Y
DSA: technology	custom device	custom device	Xilinx FPGA	ASIC, FPGA
DSA: heterogeneity	N	Y	Y	Y
DSA: topology	fixed	fixed	custom	custom

CGR DSAs can also be obtained with sole HLS, while modeling reconfiguration by hand. State-of-the-art HLS frameworks are typically imperative-based, and most of them are also target-dependent [5]. As we are going to demonstrate in Section IV, imperative-based HLS is not the best choice for developing CGR DSAs, differently from DFF HLS. Synflow Studio [14] provides a proprietary HLS framework for multi-vendor FPGAs using the Cx DF language, which does not support ASICs. Platform-agnostic HDL specification is challenging, as the resulting hardware has difficulties to compete with platform-specific code performance. To the best of our knowledge, only two academic flows support platform-agnostic DF-oriented HLS: the Orcc HDL backend [13] leveraging on the RVC-CAL language, and CAPH [12]. CAPH, adopted in this work, has the advantage of using functional programming semantics that, contrary to the RVC-CAL language, removes imperative semantics from actors. The Orcc HDL backend is no longer available, thus it is not possible to compare performance with [13].

Performance predictability is important in a design flow. Indeed, knowing the properties, and in particular the latency, of the implemented system at early design stages improves productivity and reduces design effort. This is a key feature of our approach. HLS tools based on imperative languages can only provide IP latency estimates *after synthesis* when a reconfigurable system is designed. In Vivado HLS [17] this estimate takes the form of a range (min to max latency): the actual value of each single configuration in a reconfigurable case is obtained by running the corresponding mode over the deployed CGR accelerator. The Intel FPGA SDK for OpenCL [4] pursues automatic system-level integration (and parallelization) of hardware accelerators complying the OpenCL computing model. As a result, low-level, cycle-accurate pipeline/scheduling information, typical in other IP-based HLS tools, is sacrificed in favor of an annotated dependence graph describing the generated kernel as a combination of blocks. The estimation provided in the graph refers to the types/sizes of load/store units, stalls and latencies, but a non-trivial calculation of the IP latency is left to designers. Finally, optimization is performed by profiling kernel execution, requiring re-synthesizing an equivalent kernel, automatically instrumented with performance counters, to analyze memory access behavior (stall, occupancy, bandwidth).

The key selling feature of the proposed DFF flow is the ability to compute early and accurate latency[1] estimations *before synthesis*. Such values depend on the critical path length in the DFN and on the maximum number of cycles required for each actor firing. For Synchronous DataFlow (SDF) graphs, both can be computed statically. For non-SDF graphs, CAPH estimates latency running the cycle-accurate code generated by its SystemC backend. The latency of the CGR-based DSA generated by the DFF flow depends on the selected mode only, corresponding to the selected stand-alone networks, since merging does not affect latency. As a result, the DSA can be optimized in terms of latency *before synthesis*. To the best of our knowledge, neither [8] nor [15] present this predictability feature, while [9] handles SDF inputs only.

In general, *pre-synthesis* prediction offers an additional advantage: developers can focus on the application to be accelerated with an *algorithmic-oriented approach* rather than a synthesizer-based one. Imperative-based HLS tools require DSE to improve latency. Developers have to deal with the mentioned lack of information and to understand how to exploit both the available `#pragmas` and their combinations towards best performance. Moreover, code refactoring may also be necessary to achieve optimal latency values. For these reasons, developers must have a thorough knowledge of the synthesizer and a considerable effort is required to obtain an optimized architecture. To conclude, when compared to imperative HLS, the DFF nature of the proposed HLS provides earlier explorations, while offering system predictability and guaranteeing performance, as demonstrated in the next section.

In summary, with respect to the context of CGR DSAs, the main benefits/features of the proposed integrated flow are:

1) *Custom PE Generation* - Hardware generation considers heterogeneous, HLS-generated PEs for each DF actor, favoring flexibility with respect to [8].

2) *Reconfigurability Management* - DF-based mechanism maximizes and controls resource re-use, leveraging on datapath merging techniques. Moreover, reconfiguration management is guaranteed by MDC.

3) *Predictability* - Modular specifications facilitate the predictability of system properties. *Before synthesis* latency estimations can be carried out on the basis of the pre-processed CAPH DFNs.

4) *Target Independence and Availability* - Both MDC and CAPH are platform-agnostic and open source.

5) *Code Readability* - MDC and CAPH preserve the correspondence among DF actors and hardware PEs. This is important for example in case of post-HLS enhancements. Vivado HLS acts differently since it assigns IPs to functional units (see Fig. 2).

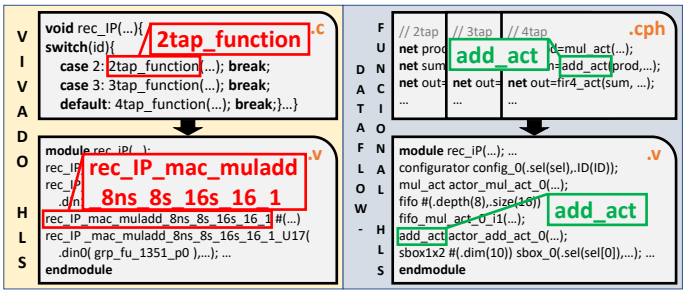


Fig. 2. Proposed Flow vs. Vivado HLS: starting and generated codes.

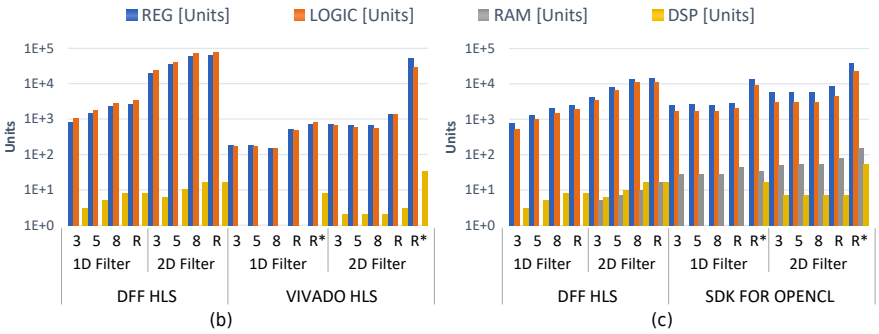
IV. Experimental Results

This section compares results obtained with three design flows: i) the proposed DFF one, using Vivado v2015.2 and Quartus v17.1 to synthesize the CGR DSAs; ii) Vivado HLS v2015.2; and iii) Intel FPGA SDK for OpenCL v17.0. Target devices are Xilinx Virtex 7 and Intel Cyclone V FPGAs. The test case chosen is a 1D/2D HEVC 3/5/8-tap hardware accelerator for approximate image interpolation filters, in fixed and reconfigurable configurations (with runtime filter switching)

[1]Number of clock cycles required to compute all outputs as in [17].

FILTER	# TAP	DFF HLS [CC estimation]	VIVADO HLS [CC estimation]
1D	3	76	1615
	5	78	2403
	8	81	3479
	R (R*)	76,78,81 ---	2171-5771 (68-69)
2D	3	4365	504557
	5	4369	1135201
	8	4375	2409163
	R (R*)	4365,4369,4375 ---	543782-3681252 (15106-38999)

(a)



(b) (c)

Fig. 3. Proposed flow vs. (a)/(b) Xilinx (XC7VX485T) and (c) Intel (5CSEMA5) target FPGAs. Intel/Xilinx resources: REG=register/FF, LOGIC=ALUT/LUT, RAM=M10K/BRAM, DSP=DSP/DSP. Data are shown on a logarithmic scale with base of 10. Latency values in clock cycles (CC) are reported in (a).

[11]. In the case of Xilinx/Intel HLS tools, a moderate effort has been put into code refactoring for optimization using shift register inference, line buffers, resource reuse when possible, and pragmas (loop unrolling and pipeline). For this reason they involve two reconfigurable designs: baseline (R) and optimized (R*). A comparison among the different flows is proposed in terms of resource utilization and early latency estimation.

DFF HLS vs. Vivado IP-Level HLS: Fig. 3a shows that for standard implementations (indicated by 3/5/8/R) the DFF flow achieves better latency results. The very large latencies obtained with Vivado HLS are explained by the default area-driven synthesis (which required adding `#pragma` and code refactoring). Moreover, as stated in Sect. III, our flow preserves the latency values of the reconfigurable IPs, meaning that merging does not alter latency as opposed to other synthesizers and, on top of that, before sythesis estimations are equal to actual post-synthesis results. With respect to the optimized reconfigurable versions (R*), only for the 1D filter a lower use of resources (see Fig. 3b) and latency has been obtained using Vivado HLS (-72.0% register count, REG; -76.1% logic elements, LOGIC; -10.5% for minimum and 14.8% for maximum latency). In the 2D filters case our performances are superior. Latency results are more than 3 times larger in the best Vivado HLS case, and again their value is (i) not preserved and (ii) highly variable in the reconfigurable case.

DFF HLS vs. OpenCL System-Level HLS: As in the Vivado HLS case, a significant effort on code refactoring is needed for optimization, which is driven by an optimization report offered at early design phases (it contains highly inaccurate resource and system-level pipeline latency/stalls estimations, rather than individual kernel latency values). Hence, there is no sane way to accurately calculate kernel latency and have insights into the generated datapath; therefore, no latency results are reported in Fig. 3a. For resource usage results refer to the kernel instantiation (See Fig. 3c). The support for the OpenCL model even at kernel level (automatic pipeline to support various threads on the fly, memory accesses optimization, replications, etc.) introduces significant resource overhead.

V. Conclusion

The results presented in this paper suggest that DFF HLS is a promising alternative to classical imperative HLS to build flexible and predictable CGR DSAs. Competitive resource usage are achieved with respect to commercial HLS methods, while allowing pre-synthesis and exact datapath latency predictions. By trying to ease hardware design to software developers, mainstream HLS tools have probably missed a key point in hardware design that is critical to embedded systems: having precise information on the generated datapaths to better optimize hardware accelerators through improved DSE. We demonstrated this issue on latency, proposing an alternative design flow that overcomes traditional HLS tools under this aspect. DFF HLS is able to tackle both issues: raising the abstraction level while keeping low-level performance estimates available for further hardware tuning. In this work, for the sake of brevity, we have provided the complete assessment of one single accelerator. Future works will extend the proposed analysis to other system features and more test cases.

References

[1] J. Backus. Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs, Com. ACM, 21(8), 2007.
[2] J. Barr. EC2 F1 Instances with FPGAs - Now Generally Available, Available: https://aws.amazon.com, 2017.
[3] K. Compton et al.. Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys*, 34(2), 2002.
[4] Intel. Intel FPGA SDK for OpenCL, Available: https://www.intel.com.
[5] R. Nane et al.. A survey and evaluation of FPGA high-level synthesis tools, *Trans. on CAD of Int. Circ.s and Sys.*, 35(10), 2016.
[6] F. Palumbo et al.. Power-Awareness in Coarse-Grained Reconfigurable Multi-Functional Architectures: a Dataflow Based Strategy, *JSPS*, 2017.
[7] M. Pelcat et al.. Design productivity of a high level synthesis compiler versus HDL, *SAMOS 2016*.
[8] E. Raffin et al.. Scheduling, binding and routing system for a run-time reconfigurable operator based multimedia architecture, *DASIP 2010*.
[9] C. Sau et al.. Automated design flow for coarse-grained reconfigurable platforms: an RVC-CAL multi-standard decoder use-case, *SAMOS 2014*.
[10] C. Sau et al.. Reconfigurable coprocessors synthesis in the MPEG-RVC domain, *ReConFig 2015*.
[11] C. Sau et al.. Challenging the Best HEVC Fractional Pixel FPGA Interpolators With Reconfigurable and Multifrequency Approximate Computing, *Embedded Systems Letters*, 9(3), 2017.
[12] J. Sérot et al.. CAPH: a language for implementing stream-processing applications on FPGAs, *Embedded Systems Design with FPGAs*, 2013.
[13] N. Siret et al.. A codesign synthesis from an MPEG-4 decoder dataflow description, *ISCAS 2010*.
[14] Synflow. Synflow studio, Available: www.synflow.com.
[15] N. S. Voros et al.. MORPHEUS: A heterogeneous dynamically reconfigurable platform for designing highly complex embedded systems, *Trans. Embedded Comput. Syst.*, 12(3), 2013.
[16] M. Wijtvliet et al.. Coarse grained reconfigurable architectures in the past 25 years: Overview and classification, *SAMOS 2016*.
[17] Xilinx. Xilinx Vivado HLS, Available: www.xilinx.com.