TEMPLATE-BASED GENERATION OF STREAMING ACCELERATORS FROM A HIGH LEVEL REPRESENTATION

Nikolaos Bellas, Sek M. Chai, Malcolm Dwyer, Dan Linzmeier

Embedded Systems Research, Motorola Inc.,
email: bellas@labs.mot.com

1. INTRODUCTION

The availability of a tool flow that abstracts out the FPGA hardware structures and presents a software-only front end interface to the application developer is a necessary step to precipitate the acceptance of FPGAs as SoC platforms. Such a tool can be used by a larger pool of engineers, and not necessarily experts in system architecture and hardware design. Furthermore, an architectural automation tool should combine interactive architectural exploration, automatic hardware-software partition and an efficient mapping of one or multiple kernels to the reconfigurable fabric.

We have developed an automation process which maps streaming data flow graphs (sDFG) to accelerators of the main scalar core. The streaming programming model assumes that the kernels process streams of data with a relatively limited lifetime, and deterministic memory access pattern. The streaming model decouples the description of memory access sequences from the computation within a kernel, thus making the customization of each of these two components (computation and memory access) easier and more re-usable.

Programs that follow the streaming paradigm are an interconnect of filters that communicate using streams [1]. The streaming programming model separates communication from computation, and favors data intensive applications with a regular memory access patterns.

2. TEMPLATE-BASED HARDWARE GENERATION

The problem we are addressing in this paper is the automatic generation of synthesizable accelerators from the streaming representation. shows the iterative design flow. The main points of the tool flow, which are shown in Figure 1, are the following:

- a common template based on a regular architecture that accesses and processes streaming data,
- an iteration engine that instantiates system parameters that meet system and user constraints to initiate the next iteration of space search,
- a scheduler that performs sDFG scheduling and hardware allocation based on the parameters set by the iterator,
- an RTL constructor engine that produces optimized Verilog code for the data path and the stream units,
- and an evaluation phase that synthesizes and maps the designs in FPGA and produces quality metrics such as area, and clock speed

Each data path and stream interface unit have their own acceleration generation process. The rest of the section details each one of these engines and their interfaces. For brevity, we will only outline the main points of the accelerator templates, without detailing the hardware generation algorithms (Figure 2).

The data path template of is an interconnect of reconfigurable functional units that produce and consume streaming data, and communicate via reconfigurable links. The stream unit transfers streams from a system memory or peripheral, through a system bus and present them in-order to the accelerator. It also transfers processed output streams back to the memory.

The scheduler receives as input the sDFG along with the user and system constraints and schedules the operation of the sDFG to optimize throughput. The constraints include resource constraints for the data path, bus bandwidth and latency, etc. The scheduler uses modulo scheduling to overlap multiple iterations in each cycle and exploits all the

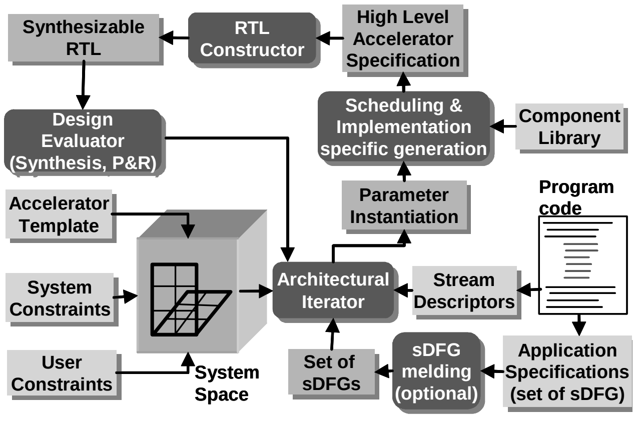

Figure 1 Template-based accelerator generation

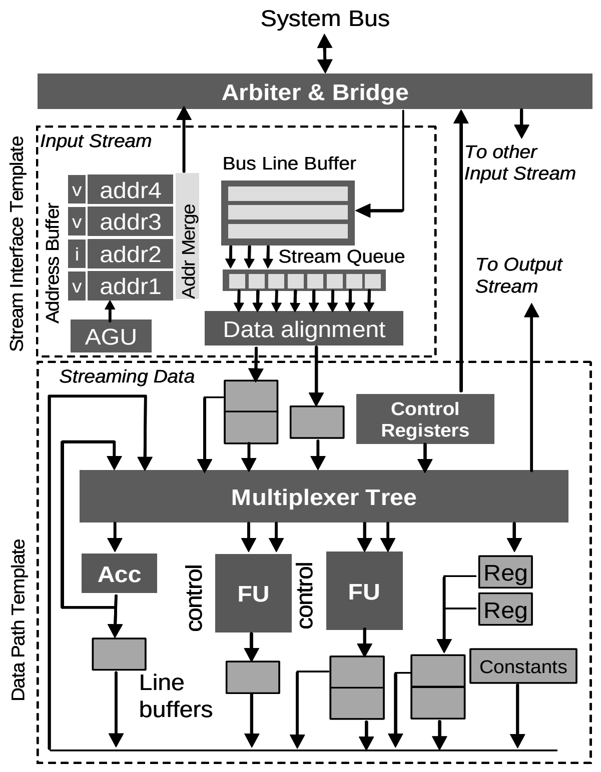

Figure 2 The accelerator template consists of the data path and the stream interface unit templates.

available parallelism under the resource constraints and data dependencies. The output of this stage is a hardware representation of the data path of the accelerator at a higher specification level than an RTL specification. The next stage produces the synthesizable RTL code of the accelerator.

The RTL for the stream interface is being generated by instantiating a series of parameters that aim at optimizing the effective throughput in and out of the data path.

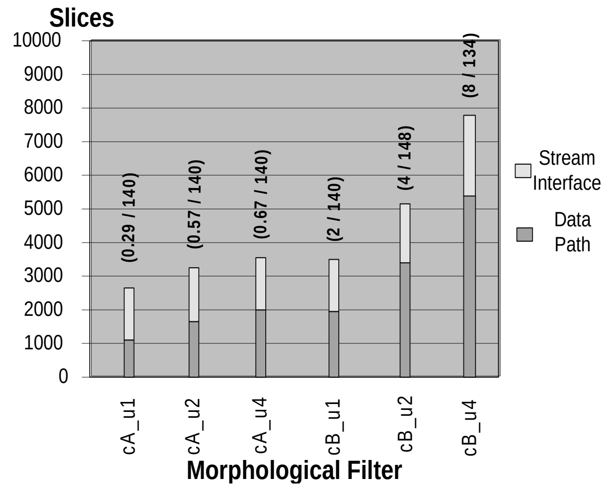

Figure 3 Number of slices for each configuration of the benchmark. The maximum bandwidth in bytes/cycle and the clock frequency in MHz, and are shown on top of each bar.

3. RESULTS

Figure 3 shows the synthesis results for a morphological filter benchmark that is used for an automatic license plate recognition application under several configurations (c_i,u_j) [2]. The c_i parameter refers to user constraints in terms of maximum number of computational resources that the tool is allowed to utilize to schedule the sDFG. For this experiment, c_B corresponds to a very wide configuration with an unlimited number of functional units while c_A corresponds to the intermediate configuration with fewer functional units, similar to the RSVP-2™ accelerator [3]. The u_i parameter shows the degree of unrolling for the sDFG to achieve higher throughput.

The generated hardware is synthesized and mapped onto a Xilinx Virtex-4 FPGA, and the quality metrics of the produced bitstream (area, clock frequency) are recorded to assess the quality of the design.

Figure 3 shows the total number of FPGA slices for each configuration of a morphological filter benchmark, and how the slices are distributed among the data path and the stream interface. The average I/O bandwidth in bytes per cycle between the data path and the stream interfaces, and the clock frequency in MHz after synthesis are also indicated at the top of each bar. The I/O bandwidth shown is an upper limit on the achievable bandwidth between the accelerator and the external bus.

In general, wider designs require more resources because the template design requires a larger number of functional units and queuing elements at the output of each functional unit to store live variables at each cycle.

4. CONCLUSION

A design methodology and prototype tool to automate the design and architectural exploration of hardware accelerators are described in this paper. In comparison to other approaches, we utilize a well-engineered template to enable fast convergence to an area and speed efficient design. We show how this methodology is used for an application set with various architectural configurations.

5. REFERENCES

[1] Amarasinghe S., Thies B. Architectures, Languages and Compilers for the Streaming Domain. *Tutorial at the 12th Annual International Conference on Parallel Architectures and Compilation Techniques*, New Orleans, LA

[2] N. Bellas, S. M. Chai, M. Dwyer, and D. Linzmeier, FPGA implementation of a license plate recognition SoC using automatically generated streaming accelerators. *Reconfigurable Architecture Workshop*, April 25-26, 2006.

[3] S. M. Chai, S. Chiricescu, R. Essick, A. López-Lagunas, B. Lucas, P. May, K. Moat, J. Norris, M. Schuette, "Streaming Processors for Next Generation Mobile Imaging Applications," IEEE Communications Magazine, Dec 2005