Enabling Automatic Module Generation for FCCM Compilers

Andreas Koch
Tech. Univ. Braunschweig (E.I.S.), Gaußstr. 11, D-38106 Braunschweig, Germany
koch@eis.cs.tu-bs.de, Ph +49-531-391-2386, Fx +49-531-391-5840

We present a standardized interface for the integration of high-performance module generators into automatic FCCM compilation flows. An API and a common data model allow the compiler to retrieve module characteristics and instantiations in an efficient and vendor-independent manner.

1 Introduction

High-performance design flows for FPGAs rely on automatic module generation [1] [2] [3] to quickly create fast and dense circuits. This structured circuit generation becomes even more crucial when FPGAs are used as compute elements in configurable computing machines (FCCM), instead of just implementing glue logic. Many research efforts on automatic compilation to FCCM targets include module generation as a fundamental step [4] [5] [6].

However, no standardized interface currently exists that allows the main flow tools (compiler/synthesis, floorplanning, place and route) access to generator libraries. With the flexibility of today's generators that are, e.g., able to restructure a circuit exploiting constant inputs [3], the total number of design alternatives covered by a single generator makes the simple static enumeration of all variations (e.g., in a file) infeasible.

2 FLAME

The Flexible API for Module-based Environments (FLAME) solves these problems with a two-pronged approach. First, it provides a standardized design data model expressing generator capabilities and module characteristics to client tools. Second, it replaces the common file-based data exchange by an active interface (API), allowing an interactive dialog between client tools and module generators. In this manner, a module is instantiated by successive refinement: The client tools incrementally tighten constraints, while the generators reply with increasingly accurate area/time/power/... estimates, culminating at the highest refinement level in the generation of layout.

Note that FLAME *wraps* existing module libraries, it has no generation capabilities of its own. Furthermore, since it aims at the integration of automatic design flows, it does not contain a GUI. Instead, it defines multiple data representations covering a spectrum of efficiency vs. portability for the exchange of information between EDA tools.

3 Active Interface

A sample for a dialog between client tools and generators is shown in Fig. 1. Computation times can be reduced since results need only be computed to the abstraction level of the current query. E.g., when requesting area and delay estimates for synthesis, it is not necessary to place and route the circuit down to the layout level.

Figure 2 shows the internal FLAME architecture. The Manager collects queries from clients and distributes them among the servers (generators). Replies are routed in the reverse direction. Additional functions can include the automatic translation between different FLAME representations and gatewaying between various communication mechanisms. The latter allows design flows that are wholly or partially distributed over a network.

The communications overhead is reduced by a memoization (caching) mechanism in the Manager. Previously encountered queries (e.g., for the same 8-bit AND) are directly answered from this cache without consulting the original generator again.

4 Data Representations

FLAME data can be represented in multiple formats to match the specific environment. A human-readable text version is easiest to process for simple tools (e.g., Perl scripts). A pre-tokenized format can be processed and transferred more efficiently between tools, but still remains portable (machine and language independent). For use in tightly integrated flows, an implementation language-specific pointer-based representation (e.g., [7]) offers the highest performance, but has only limited portability. All three representations have the capability to wrap existing formats. E.g, for netlists, simulation models, or layout, existing descriptions in EDIF, Verilog, or XNF can be encapsulated in FLAME.

5 Views

The concept of a "view" is used in FLAME to group related data. For example, a client only has to query for a "synthesis" view to receive a collection of characteristics such as timing, area, control interface, and power estimates. It is the view mechanism that is used to restrict the scope of generator computation to the information that is needed at a single step in the design flow. This avoids computing *all* data, and only have it discarded when the module is not selected early on in the synthesis process.

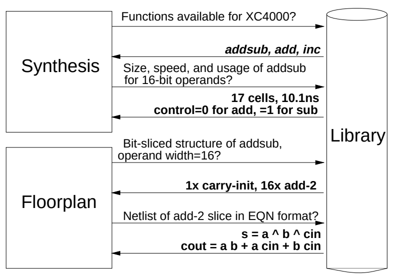
Synthesis
Floorplan
Library
Functions available for XC4000?
addsub, add, inc
Size, speed, and usage of addsub for 16-bit operands?
17 cells, 10.1ns control=0 for add, =1 for sub
Bit-sliced structure of addsub, operand width=16?
1x carry-init, 16x add-2
Netlist of add-2 slice in EQN format?
s = a ^ b ^ cin
cout = a b + a cin + b cin

Fig. 1: Sample dialog

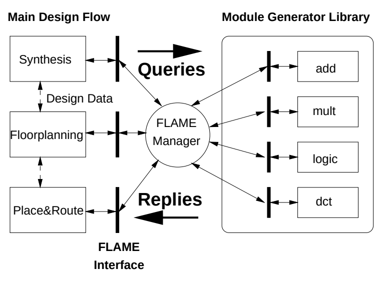

Fig. 2: FLAME system architecture

6 Design Hierarchy

The amount of data exchanged between clients and servers is also controlled by strictly following a hierarchy of design entities (Figure 3), where lower levels (more detail) are only accessed when required.

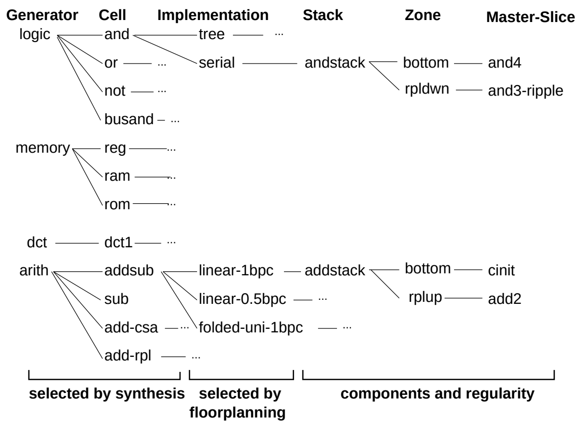

Fig. 3: FLAME design entities

To illustrate the hierarchy, consider the following example: A generator *arith* might provide the cells *addsub* (switchable adder-subtractor), *sub* (subtractor), *add-csa* (adder), and *add-rpl* (adder). The adder-subtractor is available in three implementations (*linear-1bpc*, *linear-0.5bpc,* and *folded-uni-1bpc*) that realize it in different physical layout styles. In the implementation *linear-1bpc*, the circuit consists of a single stack *addstack* defining two zones, *bottom* and *rplup*. The zone *bottom* holds a single iteration of the master-slice *cinit* (carry initialization), while the zone *rplup* contains multiple (up to the desired operand width) iterations of the master-slice *add2* (full-adder bit-slice).

7 Target Technology

The capabilities of storage elements and tri-state buffers as well as available routing and logic resources are abstracted by FLAME in a portable manner. Design tools are thus presented with a uniform view of the different underlying FPGA architectures, allowing both the easy retargeting of designs between architectures as well as the development of portable CAD tools supporting multiple technologies.

8 Cell Characteristics

The function(s) of a cell in FLAME are described using either an expression in infix notation (such as $Y = A\&B$ for a bitwise AND), or using a procedure prototype (e.g., FIR(Y,A,COEFFS) for a FIR filter). Primitive modules (AND, ADD, MUX, ...) will be instantiated automatically by the compiler when covering the user program's data-flow graph, while complex modules (e.g., FIR/IIR, FFT, DCT, ...) must be explicitly instantiated by the user as a function call.

In addition to the cell function, FLAME describes its logical and physical interfaces. E.g., while the logical interface of a serial adder might just list the operand inputs and the sum output, the physical interface could also reveal the clock and Start (=clear stored carry) inputs.

Specifying the control interface completes the information required to automatically use a cell in a synthesized circuit. Control specifications might range from a simple addition/subtraction switch by changing the value of a control input from 0 to 1, to complex multi-cycle sequences of simultaneously loading and unloading data into and from a computation unit that signals its completion after a variable number of cycles. FLAME relies on six control instructions to provide the information required by synthesis to create the appropriate FSM.

Timing characteristics can be described in FLAME using both path- and slack-based models. They cover not only combinational delays, but also latency values for pipelined execution. For units with variable execution times, best-case, average-case, and worst-case timing can be indicated to guide the module selection by the compiler.

For regular logic optimization and floorplanning [8], the FLAME design data model supplies constructs to describe a regular composition (e.g., bit-sliced) as well as topological information such as the port pitch and shape of the final layout.

9 Results

FLAME currently consists of a comprehensive specification [9] and a technology demonstrator [7] containing the base library and a sample FLAME Manager. It offers unified access to generators both developed ad-hoc as well as to modules in the Xilinx CoreGen package [10].

10 Summary

We presented a brief overview of the capabilities of FLAME, a new method for tool integration in generator-based compilation flows for FCCMs. By allowing the compiler to automatically access powerful module libraries, the full flexibility of a generator-based implementation method may be harnessed to create highly optimized circuits without human intervention.

References

[1] Xilinx Inc., "X-BLOX Reference", *EDA tool documentation*, San Jose (CA) 1995

[2] Dittmer, J., Sadewasser, H., "Parametrisierbare Modulgeneratoren für die FPGA-Familie Xilinx XC4000", *Diploma thesis*, Tech. Univ. Braunschweig (Germany), 1995

[3] Chu, M., Weaver, N., Sulimma, K., DeHon, A., Wawrzynek, J., "Object Oriented Circuit Generators in Java", *Proc. IEEE Symp. on FCCM*, Napa Valley (CA) 1998

[4] Gokhale, M.B., Stone, J.M., "NAPA-C: Compiling for a Hybrid RISC/FPGA Architecture", *Proc. IEEE Symp. on FCCM*, Napa Valley (CA) 1998

[5] Harr, R., "The Nimble Compiler Environment for Agile Hardware", *Proc. ACS PI Meeting,* `http://www.dyncorp-is.com/darpa/meeting/acs98apr/Synopsys\%20for\%20WWW.ppt`, Napa Valley (CA) 1998

[6] Hall, M., "Design Environment for ACS (DEFACTO)", *Proc. ACS PI Meeting,* `http://www.dyncorp-is.com/darpa/meeting/acs98apr/defacto.ppt`, Napa Valley (CA), 1998

[7] Koch, A., "FLAME/Java Release 0.1.1", `http://www.icsi.berkeley.edu/~akoch/research.html#FLAME`, Berkeley (CA), 1998

[8] Koch, A., "Regular Datapaths on Field-Programmable Gate Arrays", *Ph.D. thesis*, Tech. Univ. Braunschweig (Germany), 1997

[9] Koch, A., "FLAME: A Flexible API for Module-based Environments – User's Guide and Manual", `http://www.icsi.berkeley.edu/~akoch/research.html#FLAME`, Berkeley (CA), 1998

[10] Xilinx Inc., "CORE Generator System User Guide", *EDA tool documentation,* San Jose (CA) 1998