A Generic Library for Adaptive Computing Environments

Tilman Neumann and Andreas Koch

Tech. Univ. Braunschweig (E.I.S.), Gaußstr. 11, D-38106 Braunschweig, Germany
`neumann,koch@eis.cs.tu-bs.de`

Abstract. The Generic Library for Adaptive Computing Environments (GLACE) consists of a comprehensive set of module generators currently targeting Xilinx XC4000 and Virtex devices. In contrast to other research efforts in this area, it provides detailed meta-data *about* the generated circuits (behavior, area, timing, topology etc.) using the active FLAME interface. All of the modules adhere to a common layout scheme which allows the efficient automatic composition of high-performance data paths.

1 Introduction

While well known for decades, the use of module generators in VLSI design flows has recently been exploited with renewed interest. In the ASIC field, commercial products such as Module-Compiler [1] are achieving good results in terms of design time and quality. For FPGAs with their limited interconnect resources and coarse-grain logic blocks, module generators have traditionally been the tool of choice to quickly provide fast and dense circuits [2] [3] [4] [5] [6].

However, most of the existing generator systems for FPGAs generate only structural circuit descriptions (e.g., netlists or pre-placed layouts). What they sorely lack is meta-data *about* the generated instance: Main design flow tools (such as compilers and synthesis) rely on accurate area and timing data for making optimization decisions. More advanced flows including automatic floorplanning steps [9] additionally require topological information (layout shape, port placement and pitch) as the base of their operation.

This work introduces the Generic Library for Adaptive Computing Environments (GLACE). It offers a comprehensive suite of parameterized module generators suitable for automatically composing data paths on adaptive computers. In addition to placed layouts and simulation models, it also makes a broad spectrum of meta-data available to the client tools.

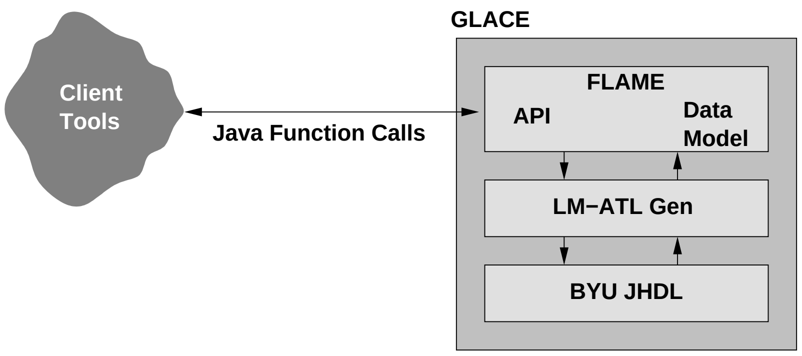
2 GLACE Architecture

The GLACE architecture (Figure 1) is not monolithic. Instead, it encompasses a number of other technologies which remain hidden from clients in the main design flow. These are being presented with a single consistent tool- and device-independent interface.

2.1 JHDL

The BYU JHDL package [6] is used as the foundation for actually creating circuits. JHDL consists of a Java class library that allows the composition of primitives to describe designs at the structural level. These circuit elements may then be annotated with device-specific mapping and

Figure 1. GLACE Architecture



placement directives. The environment, which also allows for seamless simulation of the design, currently supports the Xilinx XC4000 [7] and Virtex FPGA [8] families.

Since the full expressive power of Java including features such as inheritance and polymorphism is available to the designer, very powerful and flexible module generators can be implemented and verified with relative ease.

Unfortunately, JHDL does not exploit the structural, placement, and device data of a design to automatically derive timing, area and topology information from the circuit description. Instead, these characteristics have to be calculated manually in explicitly coded sections of the generators (Section 3.2).

2.2 Gen

The Gen package was implemented by Lockheed-Martin Advanced Technology Laboratories as part of DARPA ACS project “A Nimble Compiler for Agile Hardware” [10]. It uses JHDL to provide a complete set of parameterized basic operators suitable for use by automatic hardware compilation. The behaviors and interfaces of the functions follow the FLAME Library Specification (Section 2.3). Thus, instead of offering just basic gates, Gen also includes higher-level operations such as absolute value, arithmetic negation, signed and unsigned shifters, etc. Parameters now extend beyond simple bit widths to data types, optional registering of outputs, and architectural choices (e.g., pipelining depth for a multiplier).

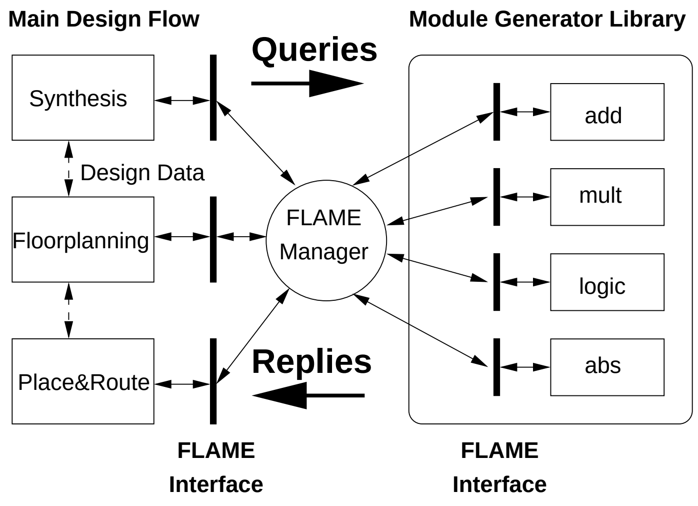
In addition to the specific cells, Gen adds some area and timing estimation capabilities to JHDL. However, even though Gen can create circuits for both XC4000 and Virtex, the estimation is only supported for the XC4000 series.

2.3 FLAME

The very flexibility of modern module generators can cause significant difficulties when they are to be integrated with the main design flow. Since cell characteristics vary with the actual parameter values used (such as bit widths, data types, absence or presence of optional inputs), the standard approach of static “library files” simply enumerating all alternatives is doomed to fail.

The Flexible API for Module-based Environments (FLAME) [11] solves these problems. It consists of three major components: The API itself, the design data model, and the library specification. Currently, GLACE uses a Java-based FLAME implementation. However, using the Java Native Interface (JNI) [12], its services can be accessed from languages different from Java.

Figure 2. FLAME Architecture Overview



API The API and communications infrastructure provided by the FLAME Manager (Figure 2) replace static library files with an active function call-based interface. Clients in the main design flow can thus enter into a dialog with the module libraries and retrieve data specific to the actual parameter values of the current instance. In GLACE, the client queries accepted by the FLAME Manager will be forwarded down the chain of Gen and JHDL to result in the computation of estimated characteristics or the creation of actual cicuits (netlists, placed layouts).

Data Model The information exchanged in this manner is represented using the FLAME design data model. This model is partitioned into a number of task-specific views: A front-end compiler might request a “behavior” view to determine which functions are available for a given target technology. Later on, it could query for a “synthesis” view to retrieve area and timing characteristics for a specific module instance. Additional views include, e.g., “topology” for layout shapes and port pitch (crucial for efficiently laying out regular data paths). “netlist”, “placed”, and “mapped” views contain the circuit itself. Instead of defining yet another netlist format, FLAME seamlessly encapsulates existing formats such as EDIF [13] or XNF.

Library Specification The FLAME Library Specification describes a set of behaviors and interfaces. One or more of these can be attached to a hardware cell to precisely define its function for automatic use by a main flow tool. For example, the cell of a switchable adder/subtracter might have both the addition and subtraction behaviors attached. The interface carefully distinguishes between the logical (e.g., the operands of the adder) and the physical perspective (e.g., clock ports and clock enable signals). Furthermore, in FLAME, an interface extends beyond port specifications, such as width and data type, to the control characteristics of the cell. This could cover “start” and “done” signals as well as mode switches (e.g., alternating between addition and subtraction). By considering all of these aspects, a main flow tool can choose the cell most applicable to a given task and automatically drive it correctly from the central data path controller.

Description	Behavior Name and Logical Interface
Addition	`add(sum, [cout,] [ovfl,] a, b [, cin])`
Subtraction	`sub(diff, [bout,] [ovfl,] a, b [, bin])`
Multiplication	`mul(prod, [ovfl, ] a, b)`
Division	`div(quot, [zerodiv,] a, b)`
Modulus	`mod(rem, [zerodiv,] a, b)`
Negation	`neg(neg, [cout,] a [,cin])`
Absolute	`abs(abs, [cout,] a)`
Logical Shift Left	`lsl(lsl, din, bits)`
Logical Shift Right	`lsr(lsr, din, bits)`
Arithmetical Shift Right	`asr(asr, din, bits)`
Less-Than	`lt(lt, a, b)`
Less-Than-Equal	`le(le, a, b)`
Equal	`eq(eq, a, b)`
Not-Equal	`ne(ne, a, b)`
Greater-Than-Equal	`ge(ge, a, b)`
Greater-Than	`gt(gt, a, b)`
Logic	`logic(y, a, b, c, d, [e,] [f,] [g,]  ttable)`
Multiplexing	`mux(mux, a, b, [c,] [d,] [e,]  ..., sel)`
Register	`reg(q, d [,clk, en] [,lt])`

Table 1. Sample FLAME behavior names and interfaces

3 Cell Library

The following sections deal with the capabilities of the current version of GLACE and their implementation.

3.1 Behaviors

Table 1 shows a selection of FLAME behaviors including their logical interfaces. Optional ports are marked by square brackets. For example, a simple addition behavior without carry ports is expressed as `add(sum,a,b)`. Optional input ports have well-defined default values (e.g., 0 for a carry-in `cin`). Unused output ports may result in more compact circuits when their driving logic can be completely suppressed during module generation.

Individual logical ports in the behaviors are constrained further in a FLAME query to parameterize the specific module instance. E.g., the logical inputs `a,b` in the behavior `lt(lt,a,b)` can be annotated with the constraints `(WIDTH 7) (SIGNED_2)` to create a 7-Bit Less-Than comparator using two's complement signed arithmetic.

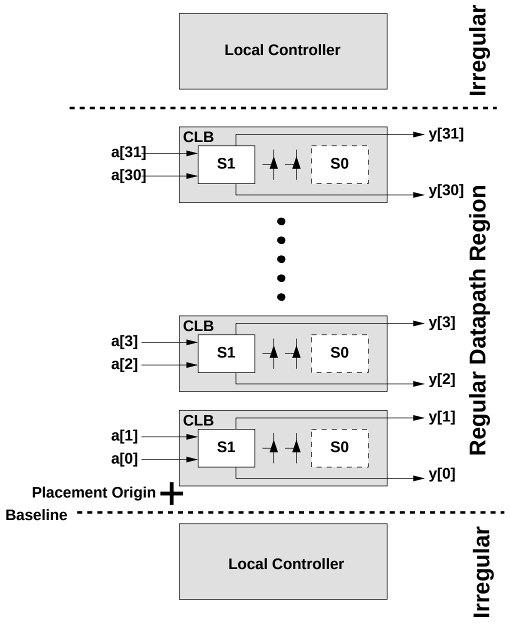
With these behaviors, all data flow graphs resulting from high-level synthesis (such as compilation from C using the Nimble Compiler) can be mapped to their corresponding hardware operators.

3.2 Implementation

The behaviors described in the previous section are decoupled from their actual implementation. For that, we have to consider both technology-specific features as well as a general module architecture. Often, these two issues are closely related.

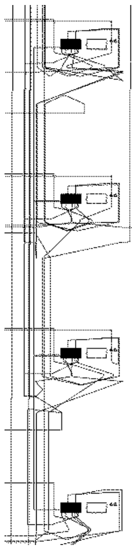
Technology-specific features Many FPGA architectures support registering the outputs of combinational logic within the same block. Since this is an extremely useful capability for RTL-style circuits and efficient pipelining, the FLAME data model has dedicated constructs for expressing the capabilities of the device registers (flip-flops, latches, clock enables) and to actually request their use. The tri-state buffers available on certain devices are handled in a similar manner. In the current implementation, the Gen package is able to create registered versions of all combinational behaviors when a client tool sends the appropriate FLAME query.

Figure 3. GLACE module architecture for Virtex devices



General module architecture The module architecture is heavily influenced by the underlying device architecture. For the Xilinx XC4000 and Virtex chips, the orientation of the on-chip carry chains (vertical) determines the direction of data flow (horizontal). Furthermore, efficient data-path layout relies on aligning busses with matched pitch and avoiding corner-turns in the routing. To this end, all of the GLACE modules targeting Virtex are laid-out as shown in Figure 3, which depicts a very simple 32-bit instance that has only one operand a and one result y. Modules consist of one or more columns of CLBs, with each vertical CLB processing two bits of each operand word (thus having a data path pitch of two). In order to balance logic outputs (2 per Virtex *slice*) with available tri-state buffers (2 per Virtex *CLB*), only slice S1 of the column containing the module outputs is used. This approach guarantees the availability of nearby buffers when a client requests the outputs to be tri-stateable. Again, note that this waste of area only occurs in the output columns. Inside of a module, all slices may be used.

Figure 4. Timing/area characteristics of an 8-bit adder



```
(TECHNOLOGY "Xilinx" "Virtex" "XCV50PQ240I" "-4"
 (STATUS QUERYOK "technology ok. area unit is 'CLB's...")
 (DPEXTENT
  (RECT 24 16 0 0 0)                 Geometry of target device
 )

 (UNIT
  (TIMESCALE -10)                    Time expressed as units of 0.1ns
 )

 (TIMING
  ( ("addition" )
   (FIXED
    (ARRIVAL
     (("sum" 7 0) )                  2.4ns propagation delay
     0 24)
    (CYCLETIME 24)
    (THROUGHPUT 1)
   )
  )
 )

 (AREA
  ("CLB" 4 4 384)                    Requires 4 CLBs (of 384 available)
 )
```

3.3 Module Characteristics

Figure 4 shows the layout and an excerpt from the FLAME timing and area data for an 8-bit adder on a Xilinx Virtex XCV50 device (speed grade -4). Note that only the left (S1) column of the CLB slices has been used in the generated circuit. The DPEXTENT attribute specifies the maximal area for the data path in CLBs. TIMESCALE sets the time unit to $10^{-10}\text{s} = 0.1\text{ns}$. This is used in the TIMING attribute: The addition result on the output port sum will be valid 2.4ns after the operand inputs stabilize. The circuit can process one datum per clock cycle. Finally, the description states that the adder will use 4 of the 384 CLBs available.

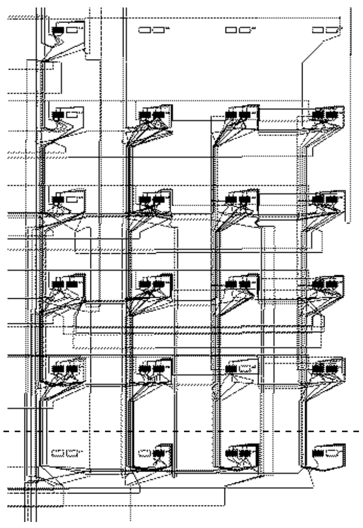
The topology information for an unsigned 8-bit multiplier is shown in Figure 5. The instance has been pipelined to have a register for every two shift/add stages. First, note that the density within the module is higher than for separate modules: In many cases, both slices of a CLB are used. Furthermore, this module has two irregular components which are placed below and above the regular data path region. All busses are spaced so 2 bits of each word are processed per CLB of module height.

3.4 Performance Evaluation

Table 2 lists area and performance data for a number of GLACE cells on a Virtex device with speed grade -4 (the slowest). The instances have been created with 32 bits of width, unsigned operands and registered outputs. The topology data is expressed as a CLB rectangle (rows x columns). Furthermore, note that the speed in MHz is the *system* speed measured when instantiating a single module and includes the delay for routing all of its ports to the chip pads.

For some of the cells, additional comments are in order: Mul1 to Mul4 differ in their degree of pipelining. Mul1 has a register inserted after each shift/add stage, Mul2 only after every second one etc. While more stages lead to a slower clock rate, the latency in clock cycles to compute the result drops. In this manner, the area/performance of the multiplier can be matched to the clock rate of the rest of the data path. The current divider implements a very simple iterative

Figure 5. Topology data for an 8-bit unsigned multiplier



```
(TECHNOLOGY "Xilinx" "Virtex" "XCV50PQ240I" "−4"
 (STATUS QUERYOK "technology ok. area unit is ’CLB’s...")

  (MATRIX                    ] Target device has matrix architecture

   (SHAPE                    ] Layout is a single 4x6 CLB rectangle
    (RECT 4 6 1 0 0)         ]        extending 1 unit below baseline
   )

   (PORTLOC
    (PORTS
     ( ("a" 7 0) ("b" 7 0) ("start" 0 0) ("out" 7 0) ("done" 0 0) )
     (PITCH 2 1)          ] Port spacing for busses is 2 bits per CLB
     (COORD 0 0)

     (FOLDING LINEAR)]                       Layout is not folded
    )
   )
  )
```

Function	Area in CLBs	Max. Clock in MHz
Abs	1x16	116.9
Add	1x16	128.3
Eq	1x16	128.3
Gt	1x16	128.3
Div	4x18	34.9
Logic	1x16	167.5
Mod	4x18	39.1
Mul1	3x18	79.1
Mul2	4x18	62.9
Mul3	5x18	53.4
Mul4	6x18	45.4
Mux	1x16	173.2
Neg	1x16	135.5
Reg	1x16	200.6
Shift	1x16	72.4
Sub	1x16	128.3

Table 2. Area and performance data for 32-Bit functions

add/subtract scheme. Later GLACE versions will replace it with a faster circuit. The shifter can perform any left shift in the range of 0 to 31 bits.

Figure 6. Sample GLACE datapath

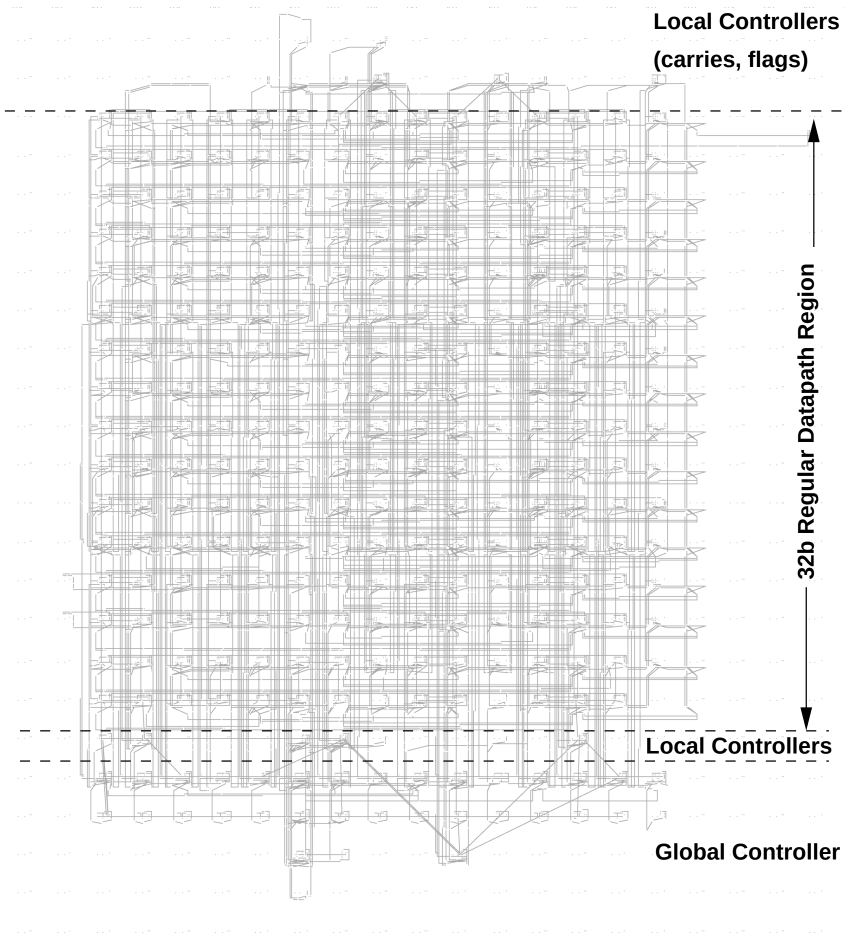


Figure 6 shows a complete data path composed from 14 GLACE operators. This example (created by the Nimble Compiler [10]) realizes the first loop in the `block_quantize` function of the Versatility benchmark from Honeywell's ACS benchmark suite [14]. The loop searches an integer array for its minimum and maximum elements. The circuit consists mainly of 32-Bit comparators, registers, muxes and adders. In the non-pipelined version shown here, it runs at 54.8 MHz on the -4 speed grade. Note the very regular placement and routing resulting from exploiting the module architecture described in Section 3.2.

4 Future Work

While already in a practically usable state, the current GLACE version has much potential for improvement and expansion. Providing a greater number of generators with different time/area trade-offs is just one route of advance. Additionally, the productivity of a module implementor would be considerably improved if JHDL could be extended to evaluate the structural, mapping, and placement data it has available anyway to automatically generate the FLAME meta-data views. Realizing this functionality is the major item on our mid-term agenda.

5 Summary

With GLACE, we have introduced a module generator system for adaptive computing systems that goes beyond simple macro-cell creation. Instead, it also offers a comprehensive set of meta-data views that present the front-end tools with sufficient instance-specific characteristics to base optimization decisions on.

Cell layouts created by GLACE have a consistent layout style which allows their efficient composition to form entire data paths. By using the general FLAME interface for all interactions and data representations, the internals of the system are abstracted. The resulting infrastructure is thus easily extended by adding other generator cores without affecting the client-tools.

The first public distribution of GLACE will be available in Summer 2001. In addition to all the software components, it will be accompanied by a comprehensive set of documentation and usage examples.

References

1. Synopsys Inc., “Module Compiler User Guide”, *EDA software documentation*, Mountain View (CA) 1997

2. Xilinx Inc., “X-BLOX Reference”, *EDA software documentation*, San Jose (CA) 1995

3. Dittmer, J., Sadewasser, H., “Parametrisierbare Modulgeneratoren für die FPGA-Familie Xilinx XC4000”, *Diploma thesis*, Tech. Univ. Braunschweig (Germany), 1995

4. 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

5. Mencer, O., Morf, M., Flynn, M.J., “PAM-Blox: High Performance FPGA Design for Adaptive Computing”, *Proc. IEEE Symp. on FCCM*, Napa Valley (CA), 1998

6. Hutchings, B., Bellows, P., Hawkins, J., Hemmert, S., “A CAD Suite for High-Performance FPGA Design”, *Proc. IEEE Symp. on FCCM*, Napa Valley (CA), 1999

7. Xilinx, Inc., “XC4000E and XC4000X FPGA Series”, *device datasheet*, http://www.xilinx.com/partinfo/4000.pdf, 2000

8. Xilinx, Inc., “Virtex 2.5V FPGAs”, *device datasheet*, http://www.xilinx.com/partinfo/ds003.pdf, 2000

9. Koch, A., “Regular Datapaths on Field-Programmable Gate Arrays”, *Ph.D. thesis*, Tech. Univ. Braunschweig (Germany), 1997

10. Li, Y.B., Harr, R., et al. “Hardware-Software Co-Design of Embedded Reconfigurable Architectures”, *Proc. Design Automation Conference*, 2000

11. Koch, A., “Enabling Automatic Module Generation for FCCM Compilers”, *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, 1999

12. Gordon, R., “Essential JNI”, *Prentice-Hall*, 1998

13. Electronics Industry Association, “EDIF Version 4 0 0”, *ANSI/EIA 682-1996 Standard*, Washington (DC) 1996

14. Kumar, S. et al, “A Benchmark Suite for Evaluating Configurable Computing Systems” *Proc. Intl. Symp. on FPGAs*, Monterey (CA), 2000