Bambu: An Open Source Framework for High Level Synthesis of Complex Applications

Christian Pilato and Fabrizio Ferrandi
Politecnico di Milano – Dipartimento di Elettronica e Informazione – Italy
{pilato,ferrandi}@elet.polimi.it
http://trac.ws.dei.polimi.it/panda/wiki/Bambu

This paper proposes bambu[1], a semi-automatic open-source framework to assist the designer during HLS, aiming at supporting most of the C constructs and directly interfacing with commercial tools for the synthesis to take technology aspects into account. Our framework receives as input the C description of the specification to be implemented and an XML configuration file, as shown in Figure 1. As output, it produces the HDL description of the corresponding hardware implementation and the scripts for the synthesis with the desired synthesis flow. At the moment, it is possible to support most of the C constructs, such as:

- function calls and sharing of the corresponding modules;
- pointer arithmetic and dynamic resolution of the memory addresses;
- accesses to arrays/structs and any combination of them;
- variables and structs passed either by reference or copy to the functions;
- floating point arithmetic (single/double precision) and data types with different bit-width.

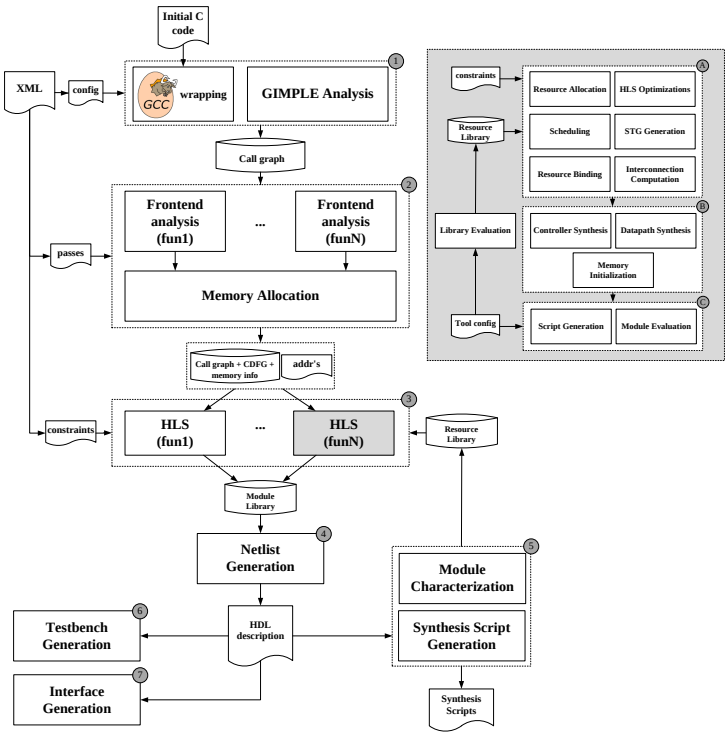

Fig. 1. Overall structure of the bambu framework and details of the HLS flow.

As front-end, bambu uses a customized interface to GCC ver. 4.5 since it provides the possibility of exporting the internal representation of the source code after the target-independent optimizations. This allows to integrate several compiler optimizations into our framework, such as loop unrolling, constant propagation, dead code elimination that can be easily enabled/disabled with command-line options or through the input configuration file. The call graph of the input application is then derived starting from this syntax tree structure (step **1** in Figure 1).

The resulting call graph is then analyzed to perform specific analysis, such as the memory allocation (step **2** in Figure 1). In details, this compile-time analysis determines the data (e.g., scalar variables, arrays, structs) to be allocated in memories. Then, this information is combined with the decisions provided by the designer about the physical allocation of the data, such as, for example, the constraints on the space available for internal allocation or the physical addresses of the variables which the designer decides to allocate in the external memories.

[1]bambu is written in C++ and its source code can be freely downloaded under GPL license at http://trac.ws.dei.polimi.it/panda/wiki/Bambu.

At this point, bambu generates all the modules necessary to implement the specification, producing the classic datapath, the controller modules (based on the FSM paradigm) and the memory interface for each of them (step **3** in Figure 1). The HLS part is built in a modular way, as shown in Figure 1, and it can be easily extended with different algorithms for each of the synthesis steps. We implemented different algorithms for scheduling and resource binding, as well as optimizations for reducing the number of multiplexers. The user can decide which algorithms have to be used by command-line options or by configuring an XML file. As a result, complex applications (e.g., the CHStone benchmarks – JPEG, ADPCM, GSM) can be thus generated taking the technology effects into account. In fact, considering the part **C** of Figure 1, we adopt specific wrappers to synthesis tools to characterize the resource library. Then, for each module/function, it is possible to generate different area/time trade-off by performing a multi-objective design space exploration [1], [2], taking into account the interconnection effects and the target device. It is thus possible to adopt the proper implementation for each of the different functions contained in the specification. The FloPoCo library is integrated for supporting floating-point operations.

A novel architecture [3] is then generate (step **4** in Figure 1) to build the modules and to deal with the different memory interfaces (one for each of them), avoiding to use three-states for its implementation. In particular, bambu implements the decisions resulting from the memory allocation as follows: internal variables are allocated on heterogeneous and distributed memories, which addresses are determined at compile time. On the other hand, for the variables allocated on external memories, the methodology is able to follow the decisions suggested by the designer by providing the proper addresses to the memory interface and access the data. This architecture is thus able to dynamically resolve the addresses. Such a memory interface allows a seamless integration with software tasks, opening new possibilities for hardware/software co-design for heterogeneous platforms. Moreover, the possibility of generating a Pareto-set of solutions allows to deploy high-performance applications onto such systems [4]. For this reason, there is also the possibility to create different interfaces (step **7** in Figure 1) to connect the resulting accelerators to software processors (e.g., PLB, FSL) and to memory controllers (e.g., Xilinx NPI).

We also integrate a toolflow (step **5** in Figure 1) with different wrappers to commercial synthesis tools (e.g., Altera Quartus, Xilinx ISE, Synopsys Design Compiler), based on a common XML configuration schema, to generate the scripts for targeting the related devices.

Finally, bambu offers the possibility to generate testbenches (step **6** in Figure 1) starting from the initial C specification and a dataset represented in XML file. Then, after generating the HDL description and the resulting testbench, it compares the produced results with the corresponding software counterpart to verify the execution. We adopt the GCC regression test suite for verifying the different aspects of our framework and the supported constructs. Moreover, we are able to synthesize all the CHStone benchmarks with different configurations.

References

[1] C. Pilato, A. Tumeo, G. Palermo, F. Ferrandi, P. L. Lanzi, and D. Sciuto, "Improving Evolutionary Exploration to Area-Time Optimization of FPGA Designs," *Journal of Systems Architecture - Embedded Systems Design*, vol. 54, no. 11, pp. 1046–1057, 2008.

[2] F. Ferrandi, P. L. Lanzi, D. Loiacono, C. Pilato, and D. Sciuto, "A multi-objective genetic algorithm for design space exploration in high-level synthesis," in *Proceedings of ISVLSI '08*, 2008, pp. 417–422.

[3] C. Pilato, F. Ferrandi, and D. Sciuto, "A design methodology to implement memory accesses in high-level synthesis," in *Proceedings of CODES+ISSS '11*, 2011, pp. 49–58.

[4] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant Colony Heuristic for Mapping and Scheduling Task and Communications on Heterogeneous Embedded Systems," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 29, no. 6, pp. 911–924, June 2010.