ScienceDirect

High Level Synthesis

The objective of HLS is to extract parallelism from the input description, and construct a micro architecture that is faster and cheaper than simply executing the input description as a program on a microprocessor.

From: DSP for Embedded and Real-Time Systems, 2012

Related terms:

Energy Engineering, Field Programmable Gate Arrays, Convolutional Neural Network, Application Specific Integrated Circuit, Data Path, Memory Access

Modern Control Architectures and Implementation

Óscar Lucía, ... José I. Artigas, in
Control of Power Electronic Converters and Systems, 2018

29.3.2.2 High-Level Synthesis

HLS is an automated design process that takes as input an algorithmic description in order to create the digital hardware that implements the desired function. Typically, the control algorithms are written in a high-level programming language such as C or variants (SystemC, OpenCL framework, among others), and the automated tool provides the register transfer level (RTL) hardware description. Whereas implementing a complete system using a HDL [8] is a discouraging task for regular DSP or μP user, the use of high-level programming languages is a well-known and easy-to-use tool. For this reason, high-level synthesis tools (HLSTs) [9–12] arise as an alternative to HDLs when using FPGAs or ASIC for data-path implementation.

In the last decades, HLSTs have been extensively researched as a remarkable CAD tool for engineers from different disciplines leading to three main generations [9]. HLSTs were first developed in the early 1980s as a research-oriented tool with little impact in industry [13]. The lack of a real need, obscure input languages, and a problematic performance limited the adoption of these tools. The next generation was developed in mid-1990s and was fostered by the major computer-aided design (CAD) companies present in the market, i.e., Cadence, Synopsys, and Mentor Graphics, among others. Several commercial tools were deployed, but they still lacked of good performance and reduced learning curve, making it not interesting for current designers. Finally, the current generation of HLSTs started at the beginning of 2000s. This generation of tools offers an improved performance and user interface, leading to a significant reduction in the design times. Besides, most of them have adopted high-level languages commonly used by design engineers, such as Matlab or C, reducing the learning time. Currently, most of the manufacturers of FPGAs are providing HLST tools with high integration in their development environment.

In the past, HLSTs have been used mainly for communications and signal-processing developments [9,14]. However, there is a big potential on using these tools for power converter control algorithm development, especially for high demanding FPGA-based controllers [15]. Among these applications, HiL implementations, advanced controls based on estimators, or real-time identification systems are examples of complex control systems with a data-path that encourages the use of HLSTs [16,17]. Some reports have highlighted the benefits of HLSTs in control [18] and power electronic applications [19,20].

The HLST design process workflow is summarized in Fig. 29.8. First, the desired control algorithm is described using a high-level programming language, typically C. At this point, it is important to note that any existing library

containing generic components can be used, which enables a high grade of portability and complex algorithm integration.

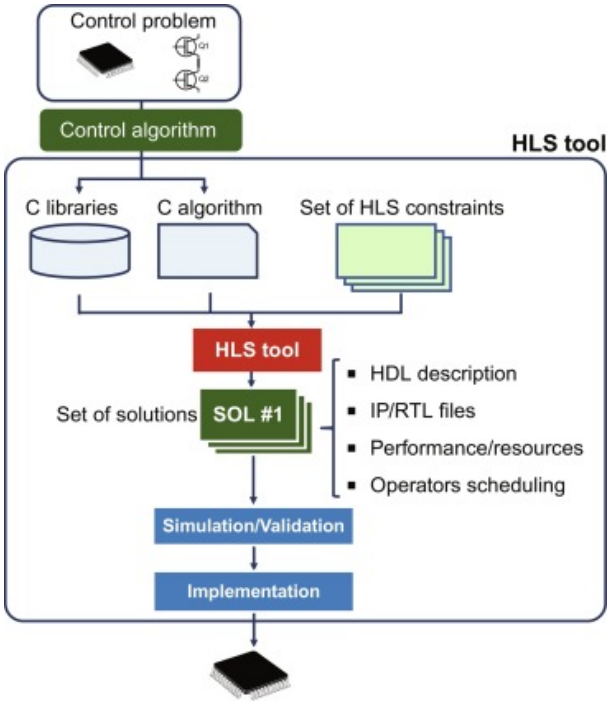

Fig. 29.8. High-level synthesis design flow for power electronic control system design.

Together with the algorithm to implement, the HLST takes as an input a set of constraints defined by the designer in order to perform the RTL synthesis. This is the most critical point in the design flow, since the defined constraints will define the final performance, resource usage, and energy consumption, leading to optimum or subpar implementations. Typically, when using HLS tools for C-to-VHDL translation the most important constraints are those related to loop unrolling, pipelining, and memories partition. By combining correctly both constraints optimum implementations can be achieved considering the required performance and available resources/cost.

The output of the HLST includes the set of solutions created as a result of the set of constraints. For each solution, the HDL description (either VHDL or Verilog), operator scheduling graphs, and performance reports are provided. Consequently, the different implementations can easily be compared, and, more important, the simulation of the RTL description can be performed under the same test-bench, enabling a straight-forward architecture verification.

One of the main benefits of the HLST approach is that a set of algorithms and HLS constraints can be directly evaluated. This enables new design possibilities and opens the window to wide design space exploration leading to unprecedented optimization. It is important to note that, for complex algorithms, it is unfeasible to test a high number of implementations as well as unrolling/pipelining possibilities by using hand-coded solutions. This aspect, together with the easy portability, easy arithmetic implementation, and the use of well-known programming languages, makes the HLS an excellent option for future power electronic control implementations.

Read less