Research Paper High Level Synthesis From C/C++/SystemC to FPGA

Catalin Rizea

POLITEHNICA University of Bucharest, Faculty of Electronics, Telecommunications and Information Technology.Master degree, Advanced Computing in Embedded Systems Email: catalinrizea96@gmail.com

Abstract

High-level synthesis (HLS) is an automated design process that interprets an algorithmic description of a desired behavior and creates digital hardware that implements that behavior. The flow of High-level synthesis is as follows: the code is analyzed, architecturally constrained and then scheduled to transcompile into a register-transfer level (RTL) design in a Hardware Definition Language like Verilog or VHDL, which can then be synthesized to gate level by a logic synthesis tool. HLS was introduced to speed up the design process as well as to optimize design architectures, thus allowing designers to implement designs at a higher level of abstraction. The result of High-level synthesis usually offers many solutions and a tradeoff between hardware resources used and the number of clock cycles in which the algorithm is performed must be made. As High-level synthesis methods evolved for designing circuits, new methodologies evolved for verification too. These new methodologies are included in the High-level verification paradigm. High-level verification is the task to verify electronic system-level designs at high abstraction level (above register-transfer level). The role of High-level verification has two important areas under development: to validate that High-level Synthesis is correct when doing the translation process, usually done through formal methods, and to verify a design in C/C++/SystemC code is conforming to a specification, typically through logic simulation. One of the first uses of High-level synthesis was C to HDL or C to RTL which is used to convert the C language code into hardware description language (HDL) such as VHDL or Verilog. The converted code can then be synthesized and translated into a hardware device such as an FPGA. Although the equivalent designs in hardware consume less power and execute faster with lower latency and more parallelism, the system design and functional verification in a hardware description language can be time-consuming, so critical modules in HDL are often written in high level languages and then synthesize these into HDL through C to HDL or High-level Synthesis tools. As mentioned earlier, C to HDL techniques, are mostly used in applications with high execution times like bioinformatics, financial processing, data analysis and even embedded application or real-time data processing. Another important aspect of C to HDL is that it is easier to modify a few lines of C code compared to changing the HDL code directly.

Keywords: HLS, HDL, C, VHDL, RTL, tool.

1 C to HDL

As previously mentioned, C to HDL tools convert C language code into HDL code like Verilog and VHDL. Although performance isn't the same as writing HDL code directly, if offers support to the designers by offering a higher level of abstraction and thus reducing complexity. The early development of C to HDL have been made by Ian Page and the first language for HLS was developed in the 1990s and was called Handel-C.

- **Problems of developing High-level synthesis**
  - C language follows a sequential execution model while Hardware is intrinsically parallel from where its efficiency comes.
  - C has a linear vision of memory compared to FPGAs where memory is distributed in memory blocks

Electronics and Computers (2020)

Edited by
Catalin Rizea

  - C supports a limited number of data types. In Hardware you can customize word length and use special arithmetic
  - C has no notion of time and most hardware circuits need synchronization for enabling connections.

  Having this restrictions the development concluded to using classes to model circuits component and structure. The basic flow was to instantiate components and then interconnect them. Automatic templates helped reducing the complexity of the circuits. The timing problem was modelled using loops (ex: one loop iteration equals one clock cycle). Regarding the memory, the use of pointers was necessary in enabling memory blocks, but still, memory management had strong restrictions as it didn't allow memory allocation, global variables and offered limited support for pointer arithmetics.
- **High-level synthesis tools architecture**
  - High-level synthesis tools operate at function level
    * Functions can call other functions but recursion is not allowed
    * Every hardware component is a function
    * Inputs and outputs are inferred from the prototype of the function
  - High-level synthesis tools decompose the program in hardware blocks
    * Blocks are synthesized in FSMs with datapaths
    * A top level FSM controls which FSM is active at a specific time
    * Depending on the tool, some data paths can be merged to reduce the hardware resources needed.
  - Arrays are mapped to local or external memory
  - External memory access management
    * Access to external memory takes an indefinite number of cycles
    * Overlapping is not allowed
    * Burst mode is supported only in specific conditions
  - Local memory management
    * Memory access takes one cycle
    * The number of ports can be configured
    * Memory data type can be any type depending on the High-level synthesis tool
  - High-level support for conditional branches
    * Each branch is considered a different block
    * Executes both branches and selects the result at the end (only supported if branches don't contain loops)
  - High-level support for loops
    * The loops are the most important components to be optimized by the HLS tools
  - No control on the clock speed
  - Hardware reuse rate is controlled by the user (trade-off between parallelism (performance) and cost (reuse)). HLS tools optimizes for area or speed according to user constraints

2 High-level synthesis tools and languages

- **CoDeveloper with Impulse C**

  Impulse C is a subset of the C programming language enhanced with additional libraries to support parallel programming. The targeted applications are FPGA based devices. The High-level synthesis tool CoDeveloper includes an Impulse C compiler and also has compatibility with ANSI C, thus allowing standard C tools to be used for designing and debugging applications targeting FPGA. The Impulse C based tools include C to RTL optimizing technology used to map application elements to hardware via FPGA logic synthesis tools.

  Compared to other languages, Impulse C is dataflow oriented, streaming applications and also has

support for shared memory. The data buffering implemented using dual-clock FIFOs which makes writing parallel applications relatively easy thanks to the high level of abstraction it provides. Regarding the software side of the application, the functions are usually used to read or write streams of data, send status messages or poll for results. On the hardware side, the Impulse C functions are compiled to generate the equivalent parallel hardware implementation in the form of synthesizable HDL files. The targeted FPGA platforms are from Xilinx and Altera.
In recent applications developments High-level synthesis programs have surpassed traditional HDL applications in code writing complexity and achieve almost the same performance. In the work "Impulse C vs. VHDL for Accelerating Tomographic Reconstruction" published by IEEE, the authors demonstrate that Impulse C designs achieved over 61x improvement over multi-threaded software (8 threads).

- **Catapult C**
  Catapult C is a High-level synthesis tool from Mentor Graphics. It can take ANSI C/C++ and SystemC code and generate the equivalent RTL code for FPGAs and ASICs. It offers hierarchical design support for synthesizing pipelined and multi-block subsystems from untimed ANSI C/C++ descriptions. It also allows the user to introduce constraints for timing and area. Further improvements to the tool allowed the system to automatically create SystemC transaction-level models and wrappers. Later, interface synthesis to map the data transfer was added to the tools. In 2009 the software was enhanced, allowing the synthesization of control logic,creating power-optimized RTL netlists with automatic multi-level clock gating and automated verification flow to enable debugging between the original HDL and the C code. Another important feature of Catapult C is that it introduced pointers, classes and operator overloading which facilitates design reuse methodology over RTL code.
  The performance of Catapult C can be seen in the VP9 Video decoder project. The High-level synthesis version was completed in under 6 months with a total number of lines of code of 69K while the RTL version the development took 1 year and a total number of 300K lines of RTL. The further implementation of HEVC took only 3 more months.
- **Vivado High-level Synthesis**
  The Vivado High-level synthesis is one of the most commercialized products in the High-level synthesis tools domain. It provides advanced algorithms used in today's wireless, medical, defense and consumer applications. The Vivado High-level synthesis tools supports both the ISE and Vivado design environments with a faster way of creating IP with its:
  – Abstraction of the algorithmic description, data type specification and interface support
  – Extensive libraries for configurable precision data types
  – Accelerated verification using C/C++ test bench simulation and test bench generation
  – Automatic use of Xilinx on-chip memories, DSP elements and floating-point library
  – Directives to execute multiple tasks in parallel
  – The possibility to restructure the physical implementation of arrays, functions, loops and ports to improve data availability

3 Comparison between handwritten HDL and HLS

In the "Two FPGA Case Studies Comparing High Level Synthesis and Manual HDL for HEP applications" research paper, written by Marc-Andre Tetrault, a comparison between HDL written code and HDL code generated by HLS is performed. The module developed in this case was a real-time data acquisition for nuclear science and the target device was the Zynq-7000 family. The data acquisition module was split into two separate modules: a crystal identification module and a sorter module. The crystal identification module is comprised of 6 steps: baseline estimation and correction, maximum value search with interpolation, normalization, phase identification, Wiener filter multiply-accumulate block and Wiener filter matrix inversion.

Table 1. Crystal identification design outcomes

Category	Manual HDL	HLS
Clock period	5.30 ns	10.09
Flip Flops	1736	3865
LUT	1437	3453
RAM Blocks	3	3
Multipliers	23	25
Event Interval	36 clocks	36 clocks
Latency	252 clocks	294 clocks

Table 2. Sorter design outcomes

Category	Manual HDL	HLS
Clock period	3.30 ns	3.30 ns
Flip Flops	187	150
LUT	75	351
RAM Blocks	1	2
Interval	100 clocks	104 clocks

In the case of the crystal identification module, the following observations have been made:

- The event processing interval was the same in both cases
- The HLS implementation consumed twice as many lookup tables and flip flops compared to the HDL version
- The HLS implementation took only half the time to write and debug

The sorter module is used in a distributed coincidence detection engine. It uses a memory block as an iterative shift register and inserts timestamps one by one in chronologically correct locations.

In the case of the sorter module, the following observations have been made:

- The HLS implementation took about 4 times the hardware resources to develop
- The processing interval was slightly longer
- The HLS implementation took only half the time to write and debug

4 Conclusions and the future of HLS

The current improvements that HLS tools producers are currently working on are:

- The availability of multiple input programming languages to allow more accessibility when designing hardware
- The introduction of more formal techniques so that the tools generate more efficient HDL code
- The adaptability of the HLS methodologies need to be raised so that it can be integrated in more engineering environments
- The need of specialized tutorials for HLS developing. This is an important factor because the HLS approach is very close to the software world and the ability to develop good hardware designs requires good hardware design knowledge.

Supplementary Material

(This is dummy text) For supplementary material accompanying this paper, please visit https://doi.org/10.1017/pan.xxxx.xx.

References

[1] Coussy, P.; Gajski, D. D.; Meredith, M.; Takach, A. (2009). "An Introduction to High-Level Synthesis". IEEE Design and Test of Computers.

[2] Philippe Coussy • Adam Morawiec - (2008)“High-Level Synthesis From Algorithm to Digital Circuit”

[3] Jimmy Xu ; Nikhil Subramanian ; Adam Alessio ; Scott Hauck - “Impulse C vs. VHDL for Accelerating Tomographic Reconstruction”

[4] https://www.mentor.com/hls-lp/

[5] Richard Langridge - High Level Synthesis with Catapult 8.0

[6] https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html

[7] Marc-André Tétrault - Two FPGA Case Studies Comparing High Level Synthesis and Manual HDL for HEP applications

[8] Department of Informatics Engineering, TEI of Western Macedonia, Kastoria Campus, Fourka Area, Kastoria - High-Level Synthesis: A Practical Perspective