Using C based logic synthesis to bridge the productivity gap

Chris Sullivan
Celoxica Ltd
chris.sullivan@celoxica.com

Alex Wilson
Celoxica Ltd
alex.wilson@celoxica.com

Stephen Chappell
Celoxica Ltd
stephen.chappell@celoxica.com

Abstract

Digital circuits from software designs and formal executable specifications can be automatically synthesized using hardware compilation or ‘C based logic synthesis’. Designs can be verified using that same formal specification and coupled with the increasing deployment of higher-level C based languages and IP reuse in hardware design and system codesign, C based logic synthesis is enabling new methodologies and levels of designer productivity. In this paper we discuss the rationale for such a synthesis approach, the required semantics and compilation technology and offer a contrast with RTL synthesis. Design examples are used to provide case studies of practical experience.

1. The design productivity dilemma

The semiconductor roadmap\(^1\) predicts that the complexity of digital system design will continue to increase according to Moore’s law.\(^{ii}\) Others suggest design complexity\(^{v}\) will increase 10x every 5 years\(^{vi}\). Single dies with microprocessor cores, memory, DSPs, Programmable Logic, connectivity and tens of millions of transistors, vis-à-vis SoC, will soon be standard. However whilst chip capacity grows in accordance with Moore’s Law, the support provided by EDA vendors’ increases at only 21% per annum\(^{v}\) and this disparity creates a productivity gap (Figure 1) between design complexity and design capacity.

Figure 1. The design productivity gap.

2. Reuse

From a system perspective, components of Intellectual Property (IP) that are pre-designed and pre-verified may be used to accelerate and economize the design process. They introduce ‘standard’ components into a system and can include special purpose IP, memory IP or processor IP. Such reuse strategies offer design benefits in terms of reducing design time and are supported by standards organizations such as the VSIA\(^{vi}\).

The overriding prerequisites for successful IP reuse are component matching and integration\(^{vii}\) with little or no room for modification. At the point of integration the designer must verify that the component is accurately connected and controlled so that it interacts correctly with other system components. This can be difficult and IP reuse, by its very nature, is constrained by the number of available components. To ‘fill in the gaps’ and to maximize final product differentiation, custom IP is developed, and in the case of logic, synthesized to a physical implementation.

3. Synthesis

Synthesis is a computer driven process that transforms a circuit description from one level of abstraction to a lower level, usually towards physical implementation. Synthesizing a HDL description to the Register Transfer Level (RTL) traditionally drives hardware synthesis. An alternative and increasingly attractive synthesis technology is C based logic synthesis.

There are a number of C based languages propagating within electronics design; Cyber-C\(^{viii}\), Esterel-C\(^{ix}\), Handel-C\(^x\), SpecC\(^xi\) and SystemC\(^xii\) are examples. The languages can be distinguished via their semantics, level of abstraction and their function, e.g. for modelling, specification, simulation or implementation. In particular Cyber-C and Handel-C are distinct in their suitability for synthesis and physical implementation. Clearly there will be differences between RT level HDL based logic synthesis and behavioral/algorithmic level C based logic synthesis and we can contrast synthesis technologies and design languages using the Y-Chart.

3. Y Chart

Gajski and Kuhn’s Y-Chart\(^{viii}\) (Figure 2) is a conceptual framework that coordinates abstraction levels in different domains and can be used to explore different design methodologies.
The concentric rings represent levels of abstraction from ‘System’ on the outer edge to ‘Transistor’ at the centre. Evidently the number of components at the system level is many orders of magnitude less than the lower, more detailed levels. The upper axes represent ‘behavioral’ and ‘structural’ descriptions of the system components and the lower axis ‘physical’ implementation.

Figure 2. Gajski and Kuhn’s Y chart

Starting from a System Level Model we can represent synthesis, placement, optimisation and refinement on this diagram as paths. In Figures 3 and 4 respectively we use the Y chart to position RT Level synthesis (using HDLs) with C based logic synthesis (using Handel-C).

Figure 3. Y chart positioning RTL synthesis

3.1. Register Transfer Level (RTL) synthesis

A traditional design process starts from the behavioral domain that specifies the behavior-only of the system, without any implementation details. The design is then mapped onto an architecture in the structural domain. Traditionally our only path to synthesis is from the RT level. This requires the designer to incrementally refine higher-level descriptions or the system specification to the RT level for synthesis. As VHDl and Verilog were originally designed for simulation, and only a subset of the HDL may be synthesized, HDL descriptions must be carefully designed to ensure that they are both simulatable and synthesizable. An EDIF File or GDSII library is output and this is either placed and routed directly into the FPGA or sent to the Foundry for ASIC.

3.2. C based logic synthesis

Using C based logic synthesis the refinement effort for any System Level Model can be reduced by keeping the level of design abstraction to the outer rings of the model (Figure 4). However the corollary of this reduction in hand refinement is an increase in the sophistication of the enabling synthesis technology.

Figure 4. Y chart positioning C based logic synthesis

To enable this higher-level synthesis, design flow and methodology the C language employed should be totally synthesizable to a physical implementation and supported by synthesis tools. The language should provide determinism and effective control over registers and timing, and whilst the complexity of the synthesis increases, tools that make the designer’s task less complex and time consuming should manage this.

Synthesis has traditionally focused on RTL while the higher level of behavioural abstraction is used for initial system modelling or simulation. Attempts have been made to synthesize RTL behavioral code, mostly without success, though the Cyber tool and BDL (Behavior Design Language) from NEC is a notable exception.

4. Handel-C

Handel-C is a synchronous programming language where the notion of time is fundamental. It builds synchronous circuits using global clocks that tick continuously and carries the capability to define more than one global clock. It captures programs at the level of simple transformations on data, and movement of data, between variables (registers) and parallelism can be explicitly defined using the ‘par’ construct. The semantics of the language are based on Occam\textsuperscript{xiv} and CSP\textsuperscript{xv} and the syntax is based on the C language\textsuperscript{xvi}. Occam is an ideal source language for hardware compilation due to its simplicity, static compilation properties, minimal run-time demands and its expressive power for parallelism. Moreover its well-defined semantics, based on CSP, make it suitable for checking the equivalence between widely varying designs and is amenable to formal proof techniques.
The level of design abstraction of Handel-C is algorithmic (above RTL but below behavioral). Behavioral code does not specify where registers should be in the circuit or the details of the cycle timing; such decisions are left up to the behavioral synthesis tool or the designer. In Handel-C each assignment infers a register and one clock cycle, so it is below the common definition of behavioral abstraction.

5. Case study I: JPEG2000

Xilinx Inc. selected Handel-C and the DK design suite of system design tools to benchmark C based synthesis and design with HDL synthesis and design. The design example was JPEG2000 with HW/SW implementation in a Virtex II Pro platform FPGA. To drive system verification the ANSI-C specification was run through an appropriate target, in this instance the IBM PPC 405GP. The design functionality was then simulated and verified to establish a test bench that remained consistent throughout the design.

Code profiling was then performed to establish where the program spent its execution time and which functions called other functions during execution. Using Wind River’s WindView diagnostic tool, it was determined that the DWT (Discrete Wavelet Transform) and Tier 1 encoder were the processor-intensive functions, consuming 87% of processing time. They were selected for HW/SW trade-off analysis.

The final JPEG2000 system performance was greatly dependent upon partitioning decisions and validating the system partition against the requirements of the design was critical; it made little sense to invest time, money and effort optimizing and refining an incorrect partition. To enable this validation and analysis the Data Streaming Manager (DSM), a portable co-design API supplied with Celoxica’s DK design suite was used.

DSM (Figure 5) coupled with C based design for HW and SW allowed the designer to iteratively explore, test, and verify multiple partition alternatives and to analyze the data flow, burst length and frequency between the HW and SW. The designer was able to quickly create and easily move ports, used to send data between the SW and HW, using the API standard (Figures 6 & 7).

Before implementation the designer was able to verify the partitioning, as each was explored, with the software used as a test bench throughout.

With the optimal partition determined and verified, the design-optimization phase of the project began. With the system specification described in ANSI-C, the design was implemented in ANSI-C and hardware language extensions defined in Handel-C were used to describe the hardware. These hardware extensions enable, for example, efficient control over area, timing, clocks, RAMs, ROMs, and interfaces.

Optimizations such as combining multiple function calls into single calls were made to the software. Hardware optimization techniques such as increased parallelism, replacing for() loops with while() loops, pipelining and syntax duplication were applied. Design re-use was supported in the JPEG2000 design through the re-use of a novel lifting algorithm that performed two-dimensional DWT. The IP was available in VHDL and was integrated into the C based design using the ‘interface’ declaration defined in Handel-C. (Figure 8)

Software object code for the PowerPC processor was compiled directly from C into the PPC405GP under VxWorks. The hardware implementation was...
synthesized directly from Handel-C using the EDIF output generated by the Celoxica DK Design Suite. This EDIF netlist was optimized for Virtex II Pro for maximum efficiency and best Quality of Results (QoR). The DWT results for the JPEG2000 project are shown below in Table 1. These results are compared against the performance of the handcrafted VHDL.

To achieve the results, the DWT portion of the JPEG2000 design was synthesized directly from Handel-C to FPGA hardware. The design was optimized to achieve maximum system speed while maintaining or reducing the hardware utilization area. To validate the methodology, the results were compared against the same DWT functionality as originally hand-coded in VHDL. Using a C based methodology and C logic synthesis the designer was able to achieve a comparable size design with faster performance results in less time.

Table 1: JPEG2000 DWT function implementation metrics

<table>
<thead>
<tr>
<th>JPEG2000 Original VHDL</th>
<th>Celoxica SCSD</th>
<th>Final Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>800</td>
<td>646</td>
</tr>
<tr>
<td>Device Util.</td>
<td>7%</td>
<td>6%</td>
</tr>
<tr>
<td>Speed (MHz)</td>
<td>128</td>
<td>110</td>
</tr>
<tr>
<td>Lines of code</td>
<td>435</td>
<td>386</td>
</tr>
<tr>
<td>Design time (days)</td>
<td>20*</td>
<td>6</td>
</tr>
</tbody>
</table>

*VHDL design time does not include partitioning
**Final SCSD result includes HDL IP block

6. Case Study II: High Speed Process FA Controller

As a leader in electronics, Omron Corporation wished to evaluate the benefits of C based design and synthesis, and it therefore selected the Handel-C language and DK Design Suite to help in this. The design example chosen by Omron was a Factory Automation (FA) controller used in high-speed processes. The objectives of the design work were to:

1. Assess C based design methodologies and synthesis; and,
2. Shorten the processing of the controller, from 330 microseconds to 100 microseconds, to give better control.

First of all, the FA controller was investigated and reviewed from an architectural and behavioral perspective. The controller system architecture was analysed and a development section identified (Figure 9). It was decided that using FPGA technology could help to speed up processing time and introduce efficiencies and flexibility into the system (Figure 10). This could be achieved by implementing functionality, which was currently described in C and executed sequentially on a microprocessor, into an FPGA (Figure 11) and executing the program in parallel. Moreover using a C based design methodology the current firmware code (described in C) could be ported to hardware in the FPGA. This would avoid time consuming and error prone re-write of the C code into an HDL.

Omron engineers carefully partitioned the system software into hardware and software system components. They analysed resources such as the access status of the RAM and ROM and Bus latency and bandwidth. They then began to port the hardware...
component to Handel-C for implementation in the selected FPGA (Virtex600E, E = extended memory). The `par` construct defined in Handel-C was used to execute the sequential C code in parallel and DP-RAM (Dual Port RAM) interfaces were also used. DP-RAM access was easy to define and implement using Handel-C; it looks like a C structure and accesses like a C structure and provided extra bandwidth and flexibility for the design. The following is a declaration with `mpram` in Handel-C:

```c
Mpram DualPortRAMName
{
    ram unsigned 16 outside[256];
    ram unsigned 16 inside[256];
}
```

Its instance is created as follows

```c
mpram DualPortRAMName Taro with
{block=1};
```

and can be written to, as follows

```c
Taro,outside[n]=6;
```

To simulate the debug Omron used the DK Design Suite prototyped the system by using a development board, running the FPGA and system software in parallel. The hardware description described in Handel-C was synthesized to the FPGA and the design was optimized for the FPGA device using the DK compiler. (Figure 12)

Figure 12. DK Compiler using direct EDIF to optimally synthesize the design into the FPGA

As everything that can be described in Handel-C translates to hardware the synthesis was efficient and it meant the designer did not have to refine the design down to a synthesizable subset.

Omron designers paid particular attention to Bus access analysis and were able to reduce bus latency and the number of accesses by making efficient use of the on-board FPGA memory and DualPort RAM.

Table 2 show the analysis results for initial system and Table 3 shows the results for the second-generation system using FPGA. DP-RAM was used as an interface between the C language and FPGA and a common global variable was set up in the DP-RAM. To reduce the FPGA accessing external memory and utilising bus bandwidth, memory was moved inside the FPGA at system boot-up.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM</td>
<td>140 times</td>
</tr>
<tr>
<td>RAM</td>
<td>172 times</td>
</tr>
<tr>
<td>WORK</td>
<td>240 times</td>
</tr>
</tbody>
</table>

Table 3 Bus analysis next generation FPGA system

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM</td>
<td>10 times</td>
</tr>
<tr>
<td>RAM</td>
<td>76 times</td>
</tr>
<tr>
<td>WORK</td>
<td>92 times</td>
</tr>
</tbody>
</table>

Overall, Omron achieved a 43% speed-up in the performance of the FA controller. Using a C based design methodology the engineers were able to verify the process speed-up whilst maintaining the current software features. They found it easy to build an environment for accurately assessing the cost/benefits of hardware implementation and found that C based design can be used for production and prototyping.

7. The Compilation Process

In both these case study designs he advanced DK compilation technology synthesized the description to hardware by parsing the user’s program by recursive descent of the abstract syntax tree representation. At this time it performs width inference on variables and operators, and various checks to ensure that design violations are not present in the circuit.

There are back-end netlist transformation routines that distil the netlist so that it is optimized and suitable for different FPGAs. These transformations might impose a ceiling on the number of inputs to any gate, for example, or might implement one of the compiler’s abstract flip-flops with one of a small number of physical circuits depending on the target technology.

The resulting netlist could be implemented by any available means. In the case of Programmable SoC or FPGAs, place & route software is used to produce a configuration bitmap from the netlist for download to the target architecture at system initiation time. The output of the place and route software also gives an estimate for worst-case delay through the combinational circuitry. This can be used to set the FPGA clock speed.

As we are compiling from software to hardware we can benefit from both software compiler and logic optimization techniques.

The input code is first transformed to an abstract syntax tree. During this transformation the compiler performs width inferencing, constant folding and other syntactic expansions (such as macro expressions, loop unfolding etc.). The abstract syntax tree is then compiled to a high-level netlist containing coarse blocks of functionality. Optimizations are applied to this high-
level netlist before it is expanded to a technology specific lower-level netlist (Figure 13).

Figure 13. Compilation flow after parsing

The DK design suite also has the capability to apply advanced optimization techniques to Handel-C based designs. These include Technology Mapping\(^{xxii}\) and Retiming.

7.1. Retiming (Figures 14\(^{xiii}\)) is a method for transforming a digital circuit to minimize its clock period and the number of registers required in its implementation. Retiming allows the designer to concentrate on the functionality of an algorithm whilst having to pay less attention to the complex task of spreading logic evenly between registers. This allows designers to achieve timing closure quicker and more easily than when having to balance the logic themselves.

Figure 14a. A simple circuit

Figure 14b. Retiming for minimum registers

Figure 14c. Retiming for a minimum period

8. Conclusions

C based logic synthesis is gaining momentum and acceptance within the design community. Its higher level of design abstraction shields designers from lower-level detail, yet delivers very significant productivity gains that do not compromise area or performance\(^{xxiv}\).

Customers using the technology have realized design time reductions of some 80\(^%\)\(^{2}\) whilst still meeting area and performance requirements. Others\(^{3}\) have undertaken independent studies that demonstrate QoR equivalent to, or over-and-above RTL synthesis. By coupling C based logic synthesis with system verification and IP reuse designers have the opportunity to rapidly develop custom IP, critical in product differentiation and value-add, enhance current design methodologies and complement existing RTL design. Handel-C’s compilation technology provides for the rapid synthesis, optimization and device specific targeting of C based algorithmic implementations are produced. Optimization stages and technology specific mapping phases ensure that the output hardware is both speed and area efficient. The success achieved with Handel-C and its compilation technology serves as reference for other developers, vendors and languages.

---

\(^{1}\) International Technology Roadmap for Semiconductors, http://public.itrs.net/
\(^{2}\) G.E. Moore, Cramming more components onto integrated circuits, Electronics, Volume 38, Number 8, April 19, 1965
\(^{3}\) L. Rizzatti, IEEE Tech Forum, May 21, 2002
\(^{vi}\) VSIA, http://www.vsi.org/aboutVSIA/index.htm
\(^{vii}\) T. Zhang, L. Benini, G De Micheli, Component Selection and Matching for IP-Based Design, DATE, 2001
\(^{viii}\) K. Wakabayashi, C-based Synthesis Experiences with a Behavior Synthesizer, "Cyber", DATE, 1999
\(^{ix}\) R. Passerone, L. Lavagno, E. Sentovich, ECL: the Esterel-C language, EECS/ Cadence Berkeley Labs, 1999
\(^{xi}\) D. Gajski, Z. Jianwen et al, Spec C: Specification Language and Design Methodology,
\(^{xii}\) OSCL, www.systems.org
\(^{xiii}\) D.Gajski, R.Kuhn, Guest editors introduction: New VLSI tools, In IEEE Computer, December 1983
\(^{xiv}\) G. Jones, Programming in Occam, Prentice-Hall, 1987
\(^{xx}\) www.xilinx.com
\(^{xix}\) http://www.jpeg.org/JPEG2000.html
\(^{xx}\) http://www.windriver.com/products/family/ide.html
\(^{xxi}\) http://www.omron.com/index2.html
\(^{xxii}\) DK Design Suite, http://www.celoxica.com/dk
\(^{xxiii}\) J. Cong, Y. Ding, FlowMap: An optimal technology mapping algorithm for delay optimization in Lookup-table based FPGA designs, UCLA, 1994
\(^{xxiv}\) N. Shenoy, R.Rudell, Efficient implementation of retiming, Synopsys Inc., ACM, 1994
\(^{xxv}\) C. Sullivan, M. Saini, Software-compiled system design optimizes Xilinx Programmable Systems, Xilinx Xcell, Summer 2003

\(^{2}\) Non-disclosed North American Defense contractor
\(^{3}\) Non-disclosed North American National Scientific Laboratory