MedCrave
Step into the World of Research

International Robotics & Automation Journal

Research Article

Open Access

CrossMark

Seamless signal processing block implementation using the cubed-c design environment

Abstract

Design environments and automated CAD systems are proliferated nowadays with various preferences and restrictions in their work environments. One serious problem of automated high-level synthesis tools is their inability of at least difficulty to use for low, bit level functions such as signal processing blocks. Here the Cubed-C environment is used for the rapid implementation of a number of low level functions and blocks such as UARTs without difficulty. Cubed-C is a full-strength high-level synthesis CAD system; nevertheless, its structure and properties make it particularly suitable for this type of fine-grained applications. The experiments in this paper prove that the Cubed-C synthesis tools are particularly suitable for both complex and for low, bit level signal coding functions.

Keywords: high-level synthesis, rapid prototyping, low level signal processing, serial communications

Volume 2 Issue 4 - 2017

Michael Dossis

Department of Informatics Engineering, TEI of Western Macedonia, Greece

Correspondence: Michael Dossis, Department of Informatics Engineering, TEI of Western Macedonia, Greece, Email mdossis@yahoo.gr

Received: October 31, 2016 | **Published:** June 21, 2017

Introduction

The proliferation and complexity of current integrated circuits (ICs) both as FPGA and as Application-Specific Integrated Circuits (ASICs) and System-on-Chips (SoCs) determine that a high proportion of the custom, specialized logic on chip is designed and verified with High-level Synthesis/Verification techniques. However, low-level functions such as bit-level processing or signal coding are implemented manually, destroying the advantage of having a unified design flow for all parts of the chip. The main contribution of this work is to demonstrate that our Cubed-C High-level Synthesis (HLS) tool and method is able to deal with both complex and low-level blocks with ease and integrate the verification of both parts in one formal step, such as the cycle-accurate simulators which are produced at the output of the tool. The tools is flexible and can be used to design loop-based signal coding response blocks for both NRZ and RTZ algorithms, as shown further down in this paper. All of this of course are simulated and shown both at the RTL and the cycle-accurate simulator accuracies. Next session discusses existing work for HLS. Session III outlines the Cubed-C tools and methodology. Session IV discusses signal-coding algorithms. Session V presents experimental results. The last session concludes upon this work and draws future work extensions.

Related work

Improved methodologies and tools started appearing as early as the late 90s and continue with enhanced input programming code sets as well as scheduling and other optimization algorithms. Furthermore, system level synthesis matured in the last decade by using more (application-wise) specialized and platform-oriented methodologies. The CoWare hardware-software co-design environment[1] employs a data model that allows the user to specify, simulate and produce heterogeneous implementations from heterogeneous specification source models. The specific synchronous dataflow (SDF) type of DSP applications is implemented into hardware using languages such as SILAGE,[2] DFL,[3] and LUSTRE,[4,5] The advantage of this type of designs is that they can be scheduled at compile time and the execution of the compiled code can be two orders of magnitude faster than event-driven VHDL (e.g. RTL) simulations. In contrast to this, dynamic dataflow (DDF) algorithms consume and produce tokens that are data-dependent, and thus they allow for complex if-then-else and while loop control constructs. This is the largest part of real applications and it is dealt with by the Cubed-C synthesizer. CAD systems that allow for specifying both SDF and DDF algorithms and perform as much as possible static scheduling are the DSP-station from Mentor Graphics,[3] PTOLEMY,[6] GRAPE-II,[7] COSSAP from Synopsys and SPW from the Alta group.[8]

C programs that include dynamic memory allocation, pointers and the functions malloc and free are mapped onto specific hardware.[9] The SpC tool[9] takes a C function with complex data structures and generates a Verlog model. The different techniques and optimizations described above have been implemented using the SUIF compiler environment.[10] The memory model consists of distinct location sets, and it is used to map memory locations onto variables and arrays in Verlog. A heuristic for scheduling behavioral code with complex conditional control flow is discussed.[11] This heuristic is based on a specific intermediate design representation which apart from established techniques such as chaining and multi cycling, it enables more advanced techniques, such as conditional resource sharing and speculative execution, which are suitable for scheduling conditional behaviors. The developed tool can generate VHDL or C code from "Hierarchical Control and Data Flow Graphs", but no reports about translating a standard programming language into HCDG are known so far. The synthesis approach in[12] utilizes a coordinated set of coarse-grain and fine-grain parallelizing transformations on the input design model. These transformations are executed in order to deliver synthesis results that don't suffer from the negative effects of complex control constructs in the specification code. The synthesis techniques were implemented in the SPARK HLS tool, which transforms specifications in a small subset of C into RTL VHDL hardware models. A resource-constrained scheduler is used in SPARK and it is essentially a priority-based global list scheduling heuristic. Nevertheless, there are serious restrictions on the subset of the C language that SPARK accepts as input, and limitations such as inability to accept design hierarchy modules (e.g. subprograms) and of "while" type of loops.

Typical HLS tasks such as scheduling, resource allocation, module binding, module selection, register binding and clock selection are executed simultaneously in[13] so as to achieve better optimization in design energy, power and area. The scheduling algorithm utilized

in[13] applies concurrent loop optimization and multi cycling and it is driven by resource constraints. The tool generates RTL Verlog implementations. The developed HLS system is targeted at control-intensive applications but it is also applicable to dataflow dominated designs. An incremental floor planner is discussed in[14] which combine an incremental behavioral and physical optimization into HLS. These techniques were integrated into an existing interconnect-aware HLS tool called ISCALP.[14] The average improvements of IFP-HLS over ISCALP, for implementations with unity aspect ratio functional units, are 12% in area, 7% in power consumption, 100% in reduction in the number of merge operations, and for some benchmarks the IFP-HLS CPU run time was 6 times less than that of the ISCALP method[15,16] introduces a synthesis methodology which is suitable for the design of distributed logic and memory architectures. Beginning with a behavioral description of the system in C, the methodology starts with behavioral profiling in order to extract simulation statistics of computations and references of array data. This allows the generation of footprints which contain the accessed array locations and the frequency of their occurrence. This synthesis approach is implemented into an industrial tool called Cyber.[17]

Communicating processes which are part of a system specification are implemented.[18] In contrast to the conventional HLS approach which synthesizes each concurrent process of the system individually, the impact of the operation scheduling is considered globally in,[18] in the system critical path (as opposed to the individual process critical path). The authors in[18] claim that their methodology allocates the resources where they are mostly needed in the system, which is in the critical paths, and in this way it improves the overall multi-process designed system performance. In Gal[19] memory access management is integrated within a HLS design flow. It mainly targets digital signal processing (DSP) applications but also more general streaming systems can be included along with specific performance constraints. Mutually exclusive scheduling methods[19–21] are implemented with the "Extended Data-flow Graph". This is achieved because EDFG allows for data and conditional semantics to be handled in the same way, and thus the exploitation of potential design parallelism can be maximized. The processed graph is then given to the GAUT HLS tool[22] to perform operator selection and allocation, scheduling and binding. This methodology is rather more suitable for dataflow dominated systems such as video streaming and linear DSP algorithms.

A combined execution of decomposition and pattern-matching techniques is applied on HLS problems, in order to reduce the total circuit area in.[23] The data path area is reduced by decomposing multi cycle operations, so that they are executed on monocycle functional units (FUs that take one clock cycle to execute and deliver their results). A simple formal model that relies on a FSM-based formalism for describing and synthesizing on-chip communication protocols and protocol converters between different bus-based protocols is discussed in.[24] The work in[24] contributes towards three aspects of protocol converter synthesis: a formal, FSM-based model for protocol definition, a precise definition of protocol compatibility and a definition of converters and converter correctness (for a given pair of existing and known protocols). Protocol converter test cases that were used to evaluate the work in[24] included an ASB to APB converter and a set of converters between the Open Core Protocol (OCP) and the AMBA family of bus protocols. The existing synthesis framework is limited to protocols that can be defined by a single FSM, and in the future more than one FSM per protocol description capabilities are envisaged by the authors.

The methodology of System Co Designer[25] uses an actor-oriented approach so as to integrate HLS into electronic system level (ESL) design space exploration tools. Its main aim is to automate the design and building of correct-by-construction System on a chip (SoC) implementations from a behavioral model. The design starts with an executable System C model. Then, commercial synthesizers such as Forte's Synthesizer are used in order to generate hardware implementations of actors from the behavioral model. Modules or processes are modeled in[25] as an actor's which communicate with other actors via a number of communication channels. This is the starting point for modeling a system in.[25] The specification language of an actor is a subset of System C which is defined in System oC library.[25] The final FPGA bit stream is generated in[25] using the Xilinx EDK (Embedded Development Kit) tools. A motion-JPEG test was used to validate the proposed methodology in.[25] A formal approach is followed in[26] so as to prove that every HLS translation of a source code model produces a RTL model that is functionally-equivalent to the one in the behavioral input to the HLS tools. The validating system in[26] is called SURYA and it is using the Simplify theorem proved to implement the validation algorithms. SURYA was used to validate the SPARK HLS tool,[11] and consequently SURYA managed to find two bugs in the SPARK compilations, which were unknown before.

An assumption is made by the formal model of refinement in,[27] that the specification and the implementation are single entry and single exit programs. A transition diagram represents every process in these programs. This diagram uses generalized program locations and program transitions. A program location represents a point in the control flow of the program and it is either a node identifier, or a pair of two locations which refer to the state of two processes that are running in parallel. The replacement of flip-flop registers with latches is proposed in[28] in order to yield better timing in the implemented designs. The justification for this is that latches are inherently more tolerant to process variations than flip-flops. The latch replacement in[28] is executed not only during the register allocation task, but in all steps of HLS, including scheduling, allocation and control synthesis. This method assumes that the delay of the controller is negligible, as compared to the transparent and non-transparent phase times. Nevertheless, implementing registers with latches instead of edge-triggered flip-flops is generally considered to be cumbersome due to the complicated timing behavior of latches.

The cubed-c design environment

The Cubed-C HLS design environment consists of the frontend and the backend compilers. The frontend compilers process programs in ADA and C and produce an intermediate format named ITF. More on the description of this format can be found in.[29] The frontend compilers extract all the information from the source code which is required to transform the input subprograms into functionally-equivalent RTL modules in hardware. The backend compiler is built with Prolog predicates and therefore its HLS transformations are formal and the produced hardware implementations are provably-correct. Most of these transformations and optimizations are captured in an aggressive scheduler called PARCS. PARCS will always try to bring the best result (most compressed schedule) however obeying to the data and control dependencies as well as any existing module-locked or global resource constraints.

Figure 1 depicts the design and verification flow within the Cubed-C environment. The internal structure of the Cubed-C

Citation: Dossis M. Seamless signal processing block implementation using the cubed-c design environment. *Int Rob Auto J.* 2017;2(4):147–152.
DOI: 10.15406/iratj.2017.02.00029

hardware compilation system is obscured in this figure. However, the main features of unified synthesis/verification strategy are apparent in Figure 1. PARCS is based on formal techniques and it is a resource-constrained scheduler. It can be driven with local (module-wise) or global resource constraints. The backend compiler can be driven by options about the architectural template, the HDL language, the location of large multi-dimensional data objects, the use of custom blocks, etc. Apart from the RTL code, the backend compiler uses the same internal formal model for the hardware FSMs to extract one cycle-accurate simulator in ANSI-C for each module in the input code hierarchy. By compiling and executing these simulators, the user can go through the FSM states and observe the changes in various storage elements inputs and outputs of the design. Thus, this verification is based on formal means of verifying the same model used by synthesis, and in all of our test cases the functionality of the generated hardware structure coincided with that of the input code, which was expected due to the formal nature of the hardware synthesis transformations.

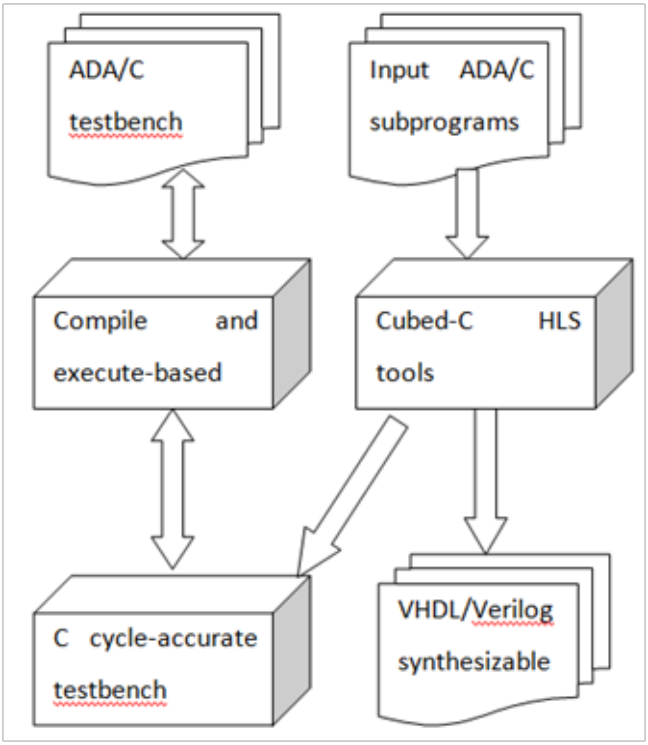

Figure 1 The Cubed-C synthesis/verification flow.

Signal coding algorithms

The 0 and 1 value of digital signals are sometimes transmitted as are with low and high voltage corresponding values. Often, and in order to enable clock recovery from the signal waveform, as well as information compression, they are modulated or mapped onto signal voltage level change or stable value as well. In general there are four categories of signals:

a) Non-return to zero (NRZ) signals.

b) Return to zero (RZ) signals.

c) Phase Encoded (PE) or phase split signals.

d) Multiple level signals – Multi-level binary (MLB) signals.

NRZ-level signals map the 1 value to high voltage and 0 to low, or the other way round. It is the simplest type of NRZ signal. NRZ-mark signals map the change in voltage to level 1 and the absence of change to level 0. NRZ-space coding is the opposite of NRZ-mark. These types of signal coding don't offer error correction or clock recovery and they have a constant component which makes it impossible to transmit it with capacitive or inductive links. Unipolar RZ is the simplest coding of including the clock information. Unipolar RZ is the logical AND between the signal and the clock, therefore when value 1 then there is a 1 pulse for the first half of the period, and 0 otherwise. In Unipolar PPM coding value 0 is coded as a short pulse (1) at the beginning of the clock period, and value 1 is coded as a short pulse (1) somewhere at the middle of the clock cycle. In Unipolar PDM value 0 is coded as a short pulse (1) at the beginning of the clock cycle and value 1 as a long pulse (1) at the beginning of the clock cycle. In this way this group of coding schemes carries in its data the clock information, which can be recovered at the receiver.

Another group of signal coding is the Phase coding group. The first coding is called the Polar Biphasic level, and it is otherwise known as Manchester code. In Manchester code value 0 is coded as 0-to-1 transition at the middle of the cycle and value 1 is coded as a 1-to-0 transition at the middle of the clock cycle. Polar Biphasic Mark and Polar Biphasic Space are essentially Frequency Key Shifting. In Biphasic M. value 0 is coded with stable and alternating (with continuous 0) levels, and value 1 is coded with double frequency clock. Polar Biphasic S. is the opposite coding of Biphasic M. In delay modulation which is also known as Miller code value 0 is coded with alternating level (at the end of the cycle) if it is followed by 0, or same level if it is followed by 1. In Miller code, value 1 is represented by a change of the level in the middle of the clock cycle. An interesting group of signal coding is referred to as multi-level coding. This type of coding avoids the constant component and offers excellent synchronization of the receiver. The Polar RZ represents values 1 and 0 with positive and negative pulse of duration half of the cycle, respectively. Bipolar or alternate mark inversion features half-duration alternating positive/negative pulse for value 1 and level 0 for value 0. Decode coding represents 0-to-1 change with a positive pulse and 1-to-0 change with a negative pulse, and stable level otherwise. In pair selected ternary pairs of bits are encoded depending on an encoding table. In Duo binary coding, when value changes there is a change in level but only to half height, e.g. from 1 to 0, from 0 to -1, from -1 to 0 and from 0 to 1, and no change when values remain the same. For this group of coding it is necessary to encode at least 3 levels of signal, e.g. 1, 0 and -1. For this we have selected encoding with two bits that will drive a DAC at the output (not shown in our experiments).

Design experiments

Although most of the signal coding algorithms that were reported above, were coded in high-level ADA and implemented in the Cubed-C framework, for the sake of economy here we describe two representative ones the Manchester code and the Polar RZ. The algorithms were coded and debugged in executable ADA programs according to the verification scheme of Figure 1. Then the code was ported to the input of the Cubed-C compiler that synthesized the RTL VHDL code. The RTL models of the code/decode processors were simulated and verified in a RTL simulator. Then the RTL output (schematics) of the Cubed-C backend compiler we simulated and the simulations were compared to the behavior of the input ADA code. In all cases the behavior matched that of the source code programs.

Citation: Dossis M. Seamless signal processing block implementation using the cubed-c design environment. *Int Rob Auto J.* 2017;2(4):147–152.
DOI: 10.15406/iratj.2017.02.00029

Figure 2, Figure 3 depicts snapshots of the RTL simulations for the Manchester modulator/demodulator.

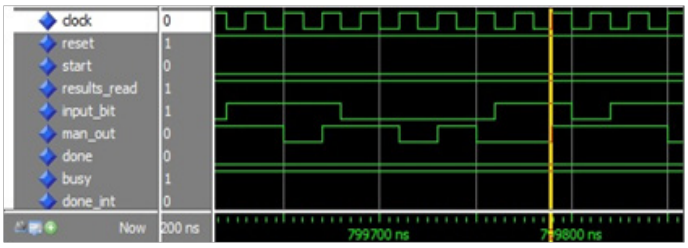

Figure 2 RTL simulation of the Manchester encoder.

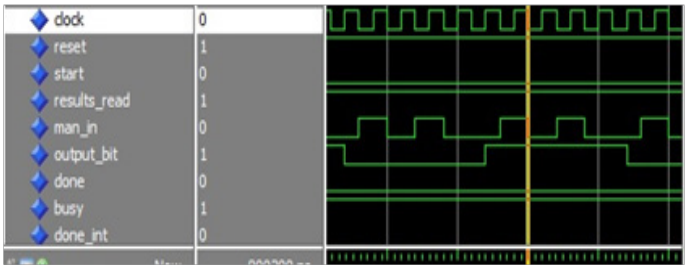

Figure 3 RTL simulation of the Manchester decoder.

Moreover, the RTZ Polar RZ algorithm was implemented with the same flow and the automatically generated RTL code of the Polar RZ processors was simulated to verify the expected correctness of the synthesis process. Figure 4, Figure 5 contains snapshots of the Polar RZ modulator/demodulator respectively. As shown in the above figures, in all synthesis experiments with Cubed-C the behavior of the synthesized code matches the intended one of the ADA source code.

It is worthy to mention that most of the experiments took less than an hour each, which reinforces our contribution towards a sizable increase of the designer's productivity. The Polar RZ experiments utilized the custom blocks option of the Cubed-C compiler to design the custom Boolean functions required for the signal value translations.

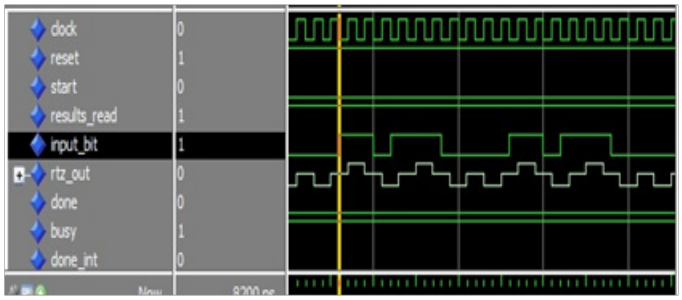

Figure 4 RTL simulation of the Polar RZ encoder.

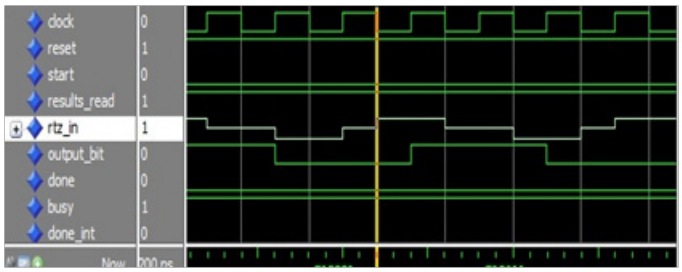

Figure 5 RTL simulator of the Polar RZ decoder.

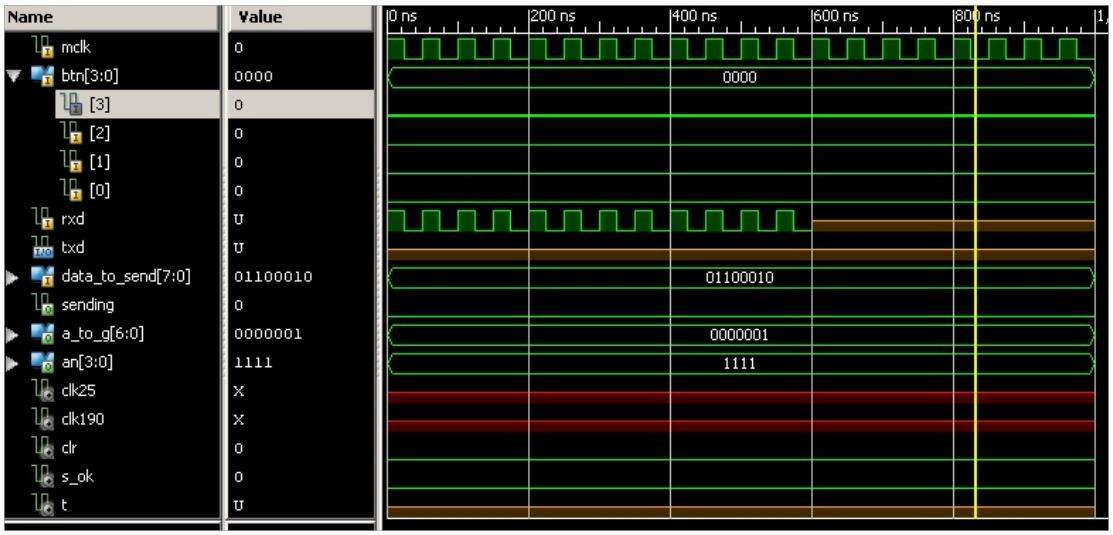

Figure 6 RTL VHDL simulation snapshot of the UART design.

A small UART serial communications processor

A UART design was verified and synthesized using the Cubed-C synthesizer. Table 1 shows the Xilinx Spartan 3 FPGA statistics of its implementation. The worse-case delay was around 8 ns, allowing an implementation speed of clock up to 100MHz. The UART was verified at the ADA level, at the cycle-accurate simulator level (produced by Cubed-C) and at the RTL VHDL simulation. The latter is showed in a snapshot in Figure 6. It must be noted that in all cases the RTL and test benches as well as the cycle-accurate C models were produced in an automatic and formal way directly from the internal formal FSM model that is kept in the memory of the Cubed-C compiler. Therefore, the verification flow is formal and automatic which save the engineer from a lot of week's manual work trying to remove functional and timing bugs.

Citation: Dossis M. Seamless signal processing block implementation using the cubed-c design environment. *Int Rob Auto J.* 2017;2(4):147–152.
DOI: 10.15406/iratj.2017.02.00029

Table 1 UART design implementation statistics

Object	SPARTAN-3 XILINX FPGA Stats		
	Used	**Total**	**Utilization**
Slice Flip Flops	84	7168	1%
4 Input LUTs	122	7168	1%
Occupied Slices	93	3584	2%
Bonded IOBs	27	173	15%
BUFGMUXs	2	8	255

Conclusion and Future work

Formal and rapid automated synthesis and automated formal verification methods flows are used in the Cubed-C tools. The major contribution of this work is that low level, bit-wise; detailed hardware signal coding algorithms are rapidly and formally implemented with the Cubed-C framework, although Cubed-C is a full-blown HLS system. In all cases, RTL/gate level verification (simulations) showed that the behavior of the generated hardware processors matches the behavior of the input specification code and model. Future work includes more low level modulation algorithm implementation, more experiments with Cubed-C and the embedded Cycle-accurate simulator.

Acknowledgments

None.

Conflict of interest

Author declares that there are none of the conflicts.

References

1. Bolsens I, De Man HJ, Lin B, et al. Hardware/software co–designof digital telecommunication systems. *Proc of the IEEE*. 1997;85(3):391–418.
2. Hilfinger PN, Rabaey J, Genin D, et al. DSP specification using the SILAGE language. *Proc Int Conf on Acoust Speech Signal Process*. 1990. p. 1057–1060.
3. Willekens P. Algorithm specification in DSP station using data flow language. *DSP Applicat*. 1994;3(1):8–16.
4. Halbwachs N, Caspi P, Raymond P, et al. The synchronous dataflow programming language Lustre. *Proc IEEE*. 1991;79(9):1305–1320.
5. Van Canneyt M. Specification, simulation and implementation of a GSM speech codec with DSP station. *DSP and Multimedia Technol*. 1994;3(5):6–15.
6. Buck JT, Soonhoi Ha, Edward A, et al. PTOLEMY: A framework for simulating and prototyping heterogeneous systems. *Int J Computer Simulation*. 1992. p. 527–543.
7. Lauwereins R, Engels M, Ade M, et al. GRAPE–II: A system level prototyping environment for DSP applications. *IEEE Computer*. 1995;28(2):35–43.
8. Rafie MS. Rapid design and prototyping of a direct sequence spread–spectrum ASIC over a wireless link. *DSP and Multimedia Technol*. 1994;3(6):6–12.
9. Semeria L, Sato K, De Micheli G. Synthesis of hardware models in C with pointers and complex data structures. *IEEE Trans VLSI Systems*. 2001;9(6):743–756.
10. Wilson RP, Robert S, Christopher, et al. Suif: An infrastructure for research on parallelizing and optimizing compilers. *ACM SIPLAN Notices*. 1994;28(9):67–70.
11. Kountouris A, Wolinski C. Efficient Scheduling of Conditional Behaviors for High–Level Synthesis. *ACM Trans on Design Aut of Electr Sys*. 2002;7(3):380–412.
12. Gupta S, Gupta RK, Dutt ND, et al. Coordinated Parallelizing Compiler Optimizations and High–Level Synthesis. *ACM Trans on Des Aut of Electr Sys*. 2004;9(4):441–470.
13. Wang W, Tan TK, Luo, et al. A comprehensive high–level synthesis system for control–flow intensive behaviors. *Proc 13th ACM Great Lakes symp on VLSI*. 2003. p. 11–14.
14. Gu ZP, Wang J, Dick RP, et al. Incremental exploration of the combined physical and behavioral design space. *Proc of the 42nd annual conf on des aut DAC*. 2005. p. 208–213.
15. Zhong L, Jha NK. Interconnect–aware high–level synthesis for low power. *Proc IEEE/ACM Int Conf Comp–Aided Des*. 2002. p. 110–117.
16. Huang C, Ravi S, Raghunathan A, et al. Generation of Heterogeneous Distributed Architectures for Memory–Intensive Applications Through High–Level Synthesis. *IEEE Trans on Very Large Scale Integr (VLSI) Sys*. 2007;15(11):1191–1204.
17. Wakabayashi K. C–based synthesis experiences with a behavior synthesizer, "Cyber". *Proc Des Autom and Test in Eur Conf*. 1999. p. 390–393.
18. Wang W, Raghunathan A, Jha NK, et al. High–level Synthesis of Multi–process Behavioral Descriptions. *IEEE International Conference on VLSI Design*. 2003. p. 467–473.
19. Gal BL, Casseau E, Huet S. Dynamic Memory Access Management for High–Performance DSP Applications Using High–Level Synthesis. *IEEE Trans Comput–Aided Des Integ Circuits Syst*. 2008;16(11):1454–1464.
20. Wakabayashi K, Tanaka H. Global scheduling independent of control dependencies based on condition vectors. *IEEE Conf Des Autom*. 1992. p. 112–115.
21. Gupta S, Gupta R, Dutt N, et al. Dynamically increasing the scope of code motions during the high–level synthesis of digital circuits. *Proc IEEE Conf Comput Digit Techn*. 2003;150(5):330–337.
22. Martin E, Santieys O, Philippe J. GAUT, an architecture synthesis tool for dedicated signal processors. *Proc IEEE Int Eur Des Autom Conf*. 1993. p. 14–19.
23. Molina MC, Ruiz–Sautua R, Garcia–Repetto P, et al. Frequent–Pattern–Guided Multilevel Decomposition of Behavioral Specifications. *IEEE Trans Comput–Aided Des Integ Circuits Syst*. 2009;28(1):60–73.
24. Avnit K, D'silva V, Sowmya A, et al. Provably correct on–chip communication: A formal approach to automatic protocol converter synthesis. *ACM Trans on Des Autom of Electr Sys (TODAES)*. 2009;14(2):19.
25. Keinert J, Streubuhr M, Schlichter T, et al. SystemCoDesigner–an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. *ACM Trans on Des Autom of Electr Sys (TODAES)*. 2009;14(1):1.
26. Kundu S, Lerner S, Gupta RK. Translation Validation of High–

Level Synthesis. *IEEE Trans Comput–Aided Des Integ Circuits Syst*. 2010;29(4):566–579.

27. Falk J, Haubelt C, Teich J. Efficient representation and simulation of model–based designs in SystemC. *Proc of the Forum of Des Lang Darmstadt*. 2006. p. 129–134.

28. Paik S, Shin I, Kim T, et al. HLS–l: A High–Level Synthesis framework for latch–based architectures. *IEEE Trans Comput–Aided Des Integ Circuits Syst*. 2010;29(5):657–670.

29. Michael Dossis. Intermediate Predicate Format for Design Automation Tools. *Journal of Next Generation Information Technology (JNIT)*. 2010;1(1):100–117.

Citation: Dossis M. Seamless signal processing block implementation using the cubed-c design environment. *Int Rob Auto J.* 2017;2(4):147–152.
DOI: 10.15406/iratj.2017.02.00029