Middle-East Journal of Scientific Research 22 (2): 241-254, 2014
ISSN 1990-9233

DOI: 10.5829/idosi.mejsr.2014.22.02.21852

High Level Synthesis Tools-an Overview from Model to Implementation

[1]M. Chinnadurai and [2]M. Joseph

[1]Department of Information Technology,
E.G.S. Pillay Engineering College, Nagapattinam, Tamilnadu, India
[2]Department of Computer and Science Engineering,
St.Joseph's College of Engineering and Technology, Thanjavur, Tamilnadu. India

Abstract: The design community has to switch over from Register Transfer Level (RTL) to higher abstraction level to design a digital circuit since the design complexity of digital circuits and System on Chip (SoC) are swelling. This increasing complexity and recent advancement in on-chip circuit has paved the way for High Level Synthesis (HLS) improvement (precisely for FPGA). The commercial HLS in earlier stage were unsuccessful due to may reason, however the recent HLS tools are developed to support in various means, such as, covering maximum language, modeling the design based on the platform, an enhancement in the algorithm, targeting domain-specific and technology-based implementation. The approach to select the HLS tool among the various tools available based on result quality (i.e. QoR), capabilities and usability are assessed in this paper. The industries are accepting the HLS tools in their design flow since the tools are progressing steadily.

Key words: Register Transfer Level (RTL) · System on Chip (SoC) · High Level Synthesis (HLS) · Quality of Result (QoR) · FPGA

INTRODUCTION

The design abstraction with superior productivity than RTL is in need, as the hindrance in System on Chip (SoC) increase rapidly. The semiconductor industry depends on ESL (Electronic System Level) design that has been extensively recognized as a boost for superior yield. High Level Synthesis (HLS) performs a core responsibility in ESL by automating the process of synthesis (high level specification) to RTL (low level specification) for proficient realization in FPGA or ASICs. The design flow of High Level Synthesis (HLS) and Register Transfer Level (RTL) can be realized using Gajski and Kuhn's Y-chart [1]. The geometry, behaviour and structure of the design are represented in three axes as shown in Figure 1. The five levels, namely; system, algorithmic, register transfer, logical, circuit and are represented by the five initial behavioural system to final RTL. This design flow is completed after the completion of RTL synthesis followed by the place and Route. The verification needs at various stages of the design as the tools and code (RTL design) are developed by humans. The initial behavioural system to final RTL. This design flow is completed after the completion of RTL synthesis followed by the place and Route. The verification needs at various stages of the design as the tools and code (RTL design) are developed by humans. The concentric circles. This flow which has been followed for the past two decades requires the manual interpretation for the re-fine of the system from the discrepancies are removed based on the result of verification. The challenge in the design flow increases nowadays, as single chip is loaded with more number of functionality stated by Moore's law [2]. This hard-hitting criterion led to the reduction in the design yield. Since the overall functionality cannot be implemented by the humans. The automatic re-finement to RTL from algorithmic level is employed by High Level Synthesis (HLS) to enhance the design productivity. In HLS, the transition becomes smaller between design flow (automated) and specification as shown in Figure 2. The programming language like Matlab, 'C' or 'C++' can be used to write the functionality and inject as input to the HLS tool which in turn generates the RTL design. The designer handover the number of tasks to HLS tool,

Corresponding Author: M. Chinnadurai, Department of Information Technology, E.G.S. Pillay Engineering College, Nagapattinam, Tamilnadu, India.

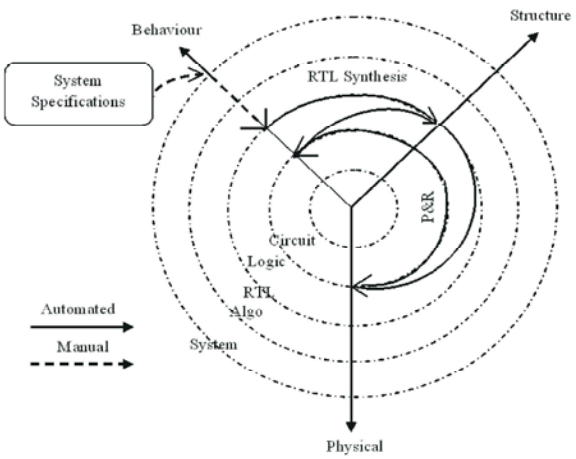

Fig. 1: Gasjki-kuhn Y-Chart for RTL Design Flow

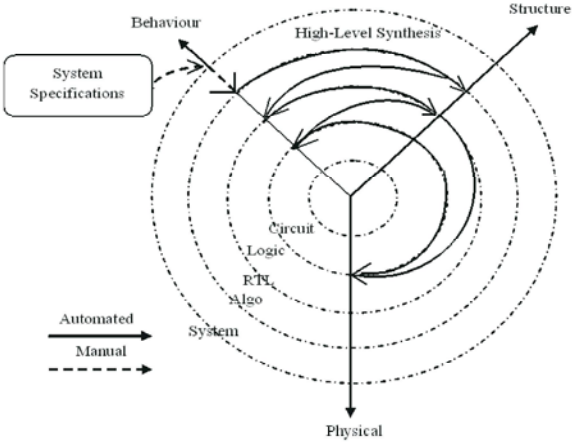

Fig. 2: Gasjki-kuhn Y-Chart for HLS Design Flow

as the number of needed memory elements and the operator types (i.e. resource allocation) are determined after the source code is analyzed. Scheduling is followed by resource allocation in which the time slot for source code with various operations are allotted.

The memory elements and operators are assigned with data and operation (i.e. binding) from the source code. It also takes care of synthesis containing interfaces. Many more sophistication can be made available by employing HLS in the design flow. The oversight made by humans during the design and time is reduced initially by reducing the number of lines of code. In addition, the validation time (require more time than design time) is reduced as the test benches are generated by the tool itself.

The complexity in coding the hardware architecture manually can be reduced to a large extent by touching the feat of higher abstraction level. Also, it reduces the complexity in the integrating number of functionality in a single chip. The milestone of redefining the FPGA prototyping and time to market are achieved by HLS. In this paper, a survey on various HLS tool has been made. Since the HLS is the sizzling research area, the tools for HLS are developing day-to-day. The future dependant of this paper should bear in mind that this assessment was made on the tools that are currently available.

This paper provides a motivation to the developer of the HLS tool, as it point out, both the shortcoming and pros of the available HLS tools. The organization of this paper is as follows, need for High Level Synthesis (HLS) is briefed in section 2, section 3 explains the evolution of High Level Synthesis (HLS) in various years, criteria to examine the tools are detailed in section 4, the survey of tools based on the Sobel edge detector is discussed in section 5, the tools assessment based on result and spider web is furnished in section 6 followed by conclusion in section 7.

Need for HLS: In 1990s, the HLS tool was declared as failure model for various issues; the HLS with high-quality and performance are in need for the following reason.

SoC with Embedded Processor: The modern embedded system involves number of software elements along with custom logic and memories on a single chip, micro-processor and digital signal processor. The C/C++ programming of the customized hardware logic and embedded software possessed by automated flow of the HLS tool made them a perfect match for System on Chip (SoC). Likewise, the programming for different boundaries of software and hardware is evaluated in-terms of power, performance and area without any tradeoffs.

Enhancing the Design Yield by reusing the Behavioral IP: The reuse of behavioral IP in behavioral synthesis is the added advantage apart from the reduction in line-count. This IP can be realized in different technologies and re-targeted to diverse realization, whereas the IP in RTL have the constraints of fixed micro-architecture.

Heterogeneous SoCs and Accelerator Development: The processor with multi programmable ability has the constraints of power which is overcome by accelerators based on custom architectures. And nowadays this accelerator was incorporated in many System on Chip (SoC) and Chip Multi-Processor (CMPs). By 2024, the accelerators present on-chip will reach 3000 as predicted by ITRS [3].

The performance of the design is increased by using custom architecture in FPGA whereas the degradation in performance was the result of using soft processor. Again, this custom logic is more suitable to High Level Synthesis.

High Level of Abstraction Rises Due to Massive Silicon Capacity: The improvement in design productivity and control over complexity is achieved by using design abstraction. As an example from NEC [4], RTL code with 300K line is needed for 1M-gate design, which is hard hitting for the manual designer. The reduction rate of 7X - 10X is achieved by utilizing the SystemC, 'C' or 'C++' as a programming language for specification of high level design.

The code complexity in the design is reduced by reducing the line of code to 30K - 40K with the help of high level synthesis[4].

Demand of System Level Verification: The system level verification [5] is approached by the method of Transaction–level modeling with a programming language called as System C [6]. The TLM based on System C is commonly used by designers which facilitate the designer with verification based on functionality, development of embedded software and finally the modeling of architecture. This would lead to the HLS solution based on System C. In today's industry the recoding of RTL manually is replaced by RTL code generated automatically by HLS tools.

Reactive Time-to-Market: The inadequacy in full custom ASIC (i.e. long time for designing chips and manufacturing cycle) is overcome by the FPGA. Faster time-to-market is achieved by the FPGA.

The tradeoffs in power, cost and performance is over-rated by reducing the time to design. This choice of tradeoffs between performance and design time is handed over to the designer in modern HLS tools.

Inefficiency of Verilog and VHDL: Application in High Performance Computing (HPC) such as bioinformatics, video and image processing, scientific computing application and financial analytics are made familiar with advancement in the reconfigurable computing of the FPGA. The application developer finds difficult in coding the HPC in VHDL or Verilog. On the other hand, the HLS tools provide the feasibility of programming the application using 'C', Matlab or 'C++'.

Inexpensive Formal Verification: The success in the first tape-out is the vital problem for the ASIC designer. From [3], over $1M is utilized to manufacture the IC in nanometer technologies. Till now mature HLS tools for formal verification of ASIC are not evolved properly. Moreover the verification is limited to SoC having multi-million gates. On the other hand the simulation coverage for FPGA is possible with higher degree. The manufacturing cost can be reduced o larger extent for more number of design Iterations.

Ultimate for Synthesis Based on Platform: The HLS tools can support a quality of result (QoR) and design methodology based on platform [7]. The above said feature is possible because, the predefined IP like embedded processors, embedded memories, embedded system bus and arithmetic units are entrenched in the modern FPGAs. In FPGA platform the modeling of the predefined logic is planned ahead, as a result of it the design can be done efficiently. The research interest in the field of HLS tools made themes available for various application domains like image processing [8], data analysis in cosmology [9], application in aerospace [10] and wireless system (3G/4G) [11]. Also, Xilinx inc. featured their user by embedding the solution of HLS in their DSP Development kit [12] and Video Development kit [13].

Early Evolution HLS: In earlier stages, the HLS tools are developed to target the ASIC rather than an FPGA. The seed was laid by the researchers at the University of Carnegie Mellon in 1970; they develop an HLS tool called CMU-DA [14, 15]. This tool uses the Instruction Set Processor Specification (ISPS) language [16] to specify the design in behavioral level. The behavioral design is converted to intermediate data called as value trace [17] afterwards converting it to RTL design. Many techniques in code-transformation such as elimination redundant sub-expression, extraction of common sub-expression, code motion, propagation of constant and elimination of dead-code made added advantages to the compiler. On the other hand, the features like controller generation, allocation of the datapath, selection module and hierarchical design support were offered at the synthesis engine.

These initial over-rated feature of the tool made the researchers turn quickly towards the tool of High Level Synthesis. As a result, the various high level synthesis tools were developed in the 1980s and 1990s for the purpose of research. The attempt of academician has led

to the development of ADAM at Southern California University [18, 19]. The interest carried-out at the Bell-Northern Research has led to the development of HAL [20]. The work at Kiel University, Germany invented the tool called as MIMOLA [21]. Stanford University developed a tool called as Hebe/Hercules [22, 23] at their university research lab. In similar, the Hyper-LP/Hyper tool [24, 25] was developed at California University, Berkeley. In contrast to this, the effort from the industry developed the tools such as IMEC [26] with their predecessor Cathedral/Cathedral II, a GM BSSC system [27] and silicon compiler [28] for IBM. Likewise, in CMU-DA, these tools also support the code-transformation and synthesis engine feature like generation of controller, resource binding, scheduling and allocation of the datapath. Apart from this, the problems are addressed individually in various tools, such as, the performance of the Silicon compiler was optimized by scheduling algorithm based on the path as well as conditional branches [29].

The HAL tool with force directed scheduling algorithm is implemented in [30] has control over the requirements of resources. The pipelined realization [31] of the design is possible by utilizing the ADAM's Schwa tools. The sharing of resources in the datapath is made possible with the techniques of conflict graph coloring techniques which are incorporated into many systems. The unbounded delay [32] can be handled using the relative scheduling algorithm which is implemented in HEBE tool. The custom languages are used to develop the design in the early stage of high level synthesis, like the ISPS language engaged in CMD-DA. Other than this, the Hercules system uses the HardwareC [33] to design the model. This language is based on 'C' programming and supports various features such as interface specification, declarative and procedural semantics and design constraints. The synthesis of hardware based on DSP [26], uniquely performed in Silage language is incorporated in Cathedral/Cathedral II. The Silage will support the easy transformation and customized data types [24, 25]. The way to domain specific approach is led by combining the Cathedral-II along with Silage Language. Many innovations were evolved as a result of earlier research based on the algorithmic synthesis. The production of the real chip based on these research are also possible but not in large. The reason behind this is that the RTL synthesis tool was not mature among the designer at that time. This lead to the failure of the HLS erect over the RTL synthesis. In addition to these drawbacks the tools like Olympus generate a sub-optimal result since the algorithms were made on the assumption (in earlier), moreover they did not depend on the technology (Technology-Independent). In the middle and later part of 1990s, major semiconductor industries had lend hand to make HLS tools practically possible one. The tools were developed by Motorola [34], Siemens [35], IBM [36] and Philips [37]. This triggering point also made several EDA vendors turn to HLS tool. In [38], the RTL implementation was generated from behavioral code by the Behavioral Compiler introduced by Synopsys in 1995. The competitor also developed the tool called as a Cadence Visual Architect [39] and Mentor Graphic's 'Monet' [40]. Although the EDA vendors and semiconductor industries developed the HLS tools, these tools did not have enough stuff to restore the RTL tools, since the design based on the behavioral model programming are not popular among the application developers and the designers.

HLS Evolution after 2000: The next generations of high-level synthesis tools were developed in 2000 by both industry and academia. The design was made possible with many tools by using the programming language like C/C++. The application developer and system designer are more convenient with tools, operating with 'C' like languages rather than HDL languages previously used. The vital character of synthesis tools such as optimization and parallelization are made easy for the designer, using the software compiler made by 'C'- like programming languages. However, the programming language like 'C' has the shortcomings that it will suitable only for the microprocessor that runs on sequential software. Indeed [41],[42], explains the ongoing contest of choosing the 'C' or HDL language for HLS. The contesters mainly focus on the HDL language, since the 'C' language has following constraints such as specification of concurrency, timing, synchronization and accuracy of bits, which is decisive to hardware design. Other than this, the language construction is complex since it involves polymorphism, pointers, recursion and managing the memory dynamically, which would lead to the synthesis complication. The constraints of 'C' language are overcome in modern HLS tool with extended 'C' language, such as SpecC [43], HardwareC [33] and Handel-C [44]. These languages make the tools, feasible to the user to specify the compiler directives and libraries free from timing, concurrency and other constraints. The two major advantages of these types of approach is that, the C/C++

compiler is enough to compile the input without change in the compiler and the co-verification/co-design of the software and hardware are not in need of re-writing the code. Dissimilar to the design based on 'C' as input, the other tools which accept the inputs other than the 'C' languages are as follows Matlab [45], BlueSpec [46] and Esterel [47]. And in nowadays, the HLS tools are varies from their ancestor. Since, the HLS tools are developed by keeping the target platform of FPGA in mind.

The reconfiguration and re-programming feature of the FPGA made them a striking candidate for many application likes video and image processing, signal processing, communication and in HPC. So the modern HLS tools are developed to target the FPGA, such as, Impulse C [48], Streams-C compiler [49], SPARK [50], Trident [51], Nallatech developed DIME-C [52], Altera developed C2H [53], CASH [54], ASC [55], GAUT [56], Mentor Graphics developed Handel-C [44] and ROCCC [57]. For example, the floating point in the FPGA can be effectively realized using Trident compiler.

The applications based on Digital Signal Processing (DSP) are implemented effectively using tools such as ASC, Streams-C, ROCCC, Impulse C and GAUT. Finally, the pplication that has resources constraint (i.e. need of optimized hardware) can utilize the work of Technoloy Driven High Level Synthesis (THLS) [58]. As of 2012, the HLS tools based on 'C' language and also the other language are as follows NEC's Cyber Workbench [59], Catapult C [60] developed by Mentor Graphics, Icarus Verilog [61], C-to-Silicon compiler [62] developed by Cadence, Synphony C [63] developed by Synopsys, Autopilot [64] from AutoESL's.

Tools Assessment Criteria Based on Lesson Learned: The various HLS tools based on many constraints such as scheduling algorithm, technology, programming language, code re-writing, dead-lock avoidance, power, performance, application, target FPGA, parallel processing, pipeline methodology, design optimization, Hardware and Software co-design/co-verification, silicon capacity, number of code lines etc., have been surveyed to the best of author knowledge till now, the remaining work will point out the needs and criteria for the assessment /evaluation of the HLS tools based on the lesson learned. The diverse criteria for the HLS tools is as follows,

Consideration on Synthesis Based on Datapath: Even though the many works has been contributed towards the HLS tool based on datapath synthesis, there are some attention need in the issues related to interfacing (i.e. the other hardware/software module interface should be feasible). In similar, the integration platform should be configurable across boundaries for ease of implementations. By doing this, the vital bottleneck of system integration can be overcome.

Documentation and Complexity in Tool: The tools should have clear documents, so that the beginners can use the tools easily, the documents should not only fulfil the initial need of the user, but also able to fine-tune and re-write the source code. The tools should posses the approach of flat learning curve for wide acceptance among the designer/application developer. The tools should provide the complex interfacing of hardware for the expert; at the same time it should be as easy as possible for the designer with less experience.

Portability and Reusability of the Design Need Attention: Many HLS tools keep the designer busy by re-tuning or modifying the source code based on information of the interface, timing constraints and synthesis constraints. This led to the bottleneck of specifying design functionality. Since the functionality became highly dependent on the target-platform, application, tool and implementation, the porting of the design for varying platform is complex.

Platform to Sustain Different Data Types: The instruction processor support two common data types in software, namely, integer and float, whereas in hardware the single bit is the only data type. The most complex data type supported by RTL synthesis tools is an integer. The freedom to vary this number of bits is given to the hardware designer. Hence the HLS tool is more optimize to the designer, if the tool supports the designer with various data types. The implementation of various data-types should be made possible at the source code level.

Deficient in Support of Design Language: High-level programming language was not supported by the HLS tools developed initially. Hence the partially/un timed behavioral model is used to specify the functionality of the design. This made the learning curve complex for both the hardware and software developers. In the next generation of HLS tools the language support was extended to programming languages such as 'C'/'C++' or Matlab, but the 'C' language has the drawback of sequential processing, concurrency, polymorphisms, pointers and dynamic allocation of memory. Although the

HLS tools with other programming languages are evolved the tool selection should be made by examining the extended support of programming languages.

Need for Verification along with the Design: The verification complexity increases as the functional block in the design increases. In other words, the verification time is more than that of design time. The HLS tools provide much advantage for the verification engineer, but the only need is the interface should be maintained properly throughout the design. This is would help the tools to generate the testbench along with the design. The speed of verification is increased by generating the design and testbench in parallel. Further the testbench used to validate the design can be reused for source code also. The HLS tools can be suggested if it provides the feature of integrating the design and source code under one testbench.

Quality of Result Not Satisfies the Designer: The degradation in the QoR in HLS tools evolved during the 1990s and after a decade is mainly due to the lack of correlation between physical design and functional logic. Also the HLS tools lack the enhancement, measure and track since there is no platform for RTL to GDS II support for HLS. For instance, the latency and functional unit count can be reduced by the common algorithm, but when it comes to real time the correlations deviate from the power, area of silicon and performances. And so the designer are not willing gain in design productivity without accomplishing the Quality of Result (QoR).

Design and Tool Correctness: The design generated by the tool was correct is the major claim of some vendors of HLS. The correctness of the tool plays a vital role in the correctness of the design. The correctness of the tools should also be considered based on the validation of the design. In precise, if the tool is correct the generated design will be correct and if the design is correct the design will pass the verification and again concludes that the tools is correct. Hence the designer should adopt the correctness criteria for the tool selection.

Design Considerations after Synthesis: The consideration of the design after synthesis depends mainly on the factor such as latency, size and resource usage. The scheduling algorithm provides latency in terms of the clock cycle which is fixed. The clock rate achieved at this time and also at the final design cannot be predicted. In similar, the size in terms of FPGA (logic elements) and ASIC (sq.mm) cannot be determined accurately. Hence the tools are selected based on the resources, size and overall latency of the design after synthesis.

Suggested Reason by the Designer: The first generation of the HLS tools is not familiar with the design community because the complexity in RTL synthesis is manageable in the late 1990s. And in the second generation, the HLS tools produce even more performance improvement than the first generation, but the researchers were panic to adopt new technology rather than the known RTL technology. Hence, the designer needs some vital "tripping-point "or some promising reason for the technology revolution (i.e. to accept/adopt an HLS design methodology). Along with this the following are some valuable points to design the HLS tools for future generation,

- The design accuracy is checked in terms of bit and algorithm (based on 'C' code). The algorithm specialists should be easily able to read the source code.
- The implementation of the parallelizable algorithm is made possible by modifying the 'C' code to a smaller extent.
- The generated design should compete with the RTL implementation where the verification is done after the manual and automatic optimization.
- The re-modifying/restoring of the source code is made possible for the implementation process of the design. Further the designer should have the feasibility of performing the optimized-design process.

The constraints and the proceeding of the HLS tools are discussed by both the academicians and industry peoples and we believe that HLS tools are evolving effectively and rapidly now (i.e. the "tripping-point"). From our previous discussion and conclusion from [65, 66], we are eager and delight to see the latest generation of high level synthesis tools. And the feature expects from the tools are HLS algorithm based on core, coverage of wide language, modeling based on platform, technology independent. In the next section, the application base tools survey is made and the results are furnished in terms of the spider web diagram.

```
for   (r  :  1 . . rows )
for   (c  :  1 . . cols )
if  (pixel_in [c,r]  on image border)
pixel_out [c,r] = White
else
gradX  = pixel_in [c-1,r-1] +  2*pixel_in[c-
1,r] + pixel_in[c-1,r+1]   \\
    -  Pixel_in[c+1,r-1] – 2*pixel_in [c+1,r]
       – pixel_in[c+1,r+1]
gradY  = pixel_in [c-1,r-1]  +
2*pixel_in[c,r-1]  +  pixel_in [c+1,r-1] \\
    -  Pixel_in [c-1,r+1]  -  2*pixel_in[c,r+1]
       - pixel_in [c+1,r+1]
grad   = abs (gradX)  + abs (gradY)
if  (grad>255) grad = 255
pixel_out [c,r] = 255 – grad
end  if
```

Fig. 3: Algorithm for Sobel Edge detector

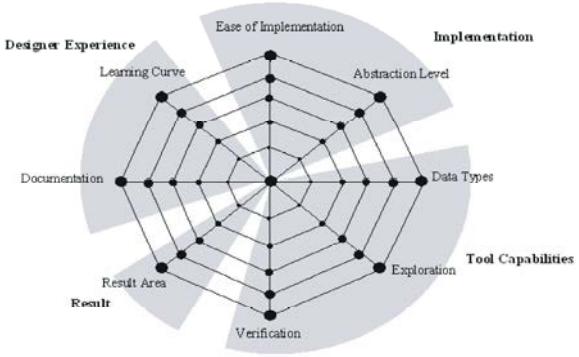

Fig. 4: Spider Web

HLS Tools Survey Based on Test Application: As mentioned earlier, the HLS tools available as of 2012 are surveyed based on test application. The application should be trivial, but at the same time, the implementation effort should be reduced. Here, Sobel edge detector [67] is a perfect candidate to match the balance. The Sobel edge detector is an image processing algorithm, where the pseudo code is shown in Figure 3. The edges in the bitmap image are detected with the help of the Sobel edge detector algorithm by determining the vertical and horizontal color gradient of each pixel in 3x3 windows. The generated image has the white border pixel. The output image with darker pixel and input with sharper transition will be the result of higher values. Even though the algorithm is simple, the HLS tools can be effectively examined by this algorithm.

And the iteration of the two loops is free from data dependencies. As a result, the (any number) execution of the iteration in the algorithm is possible. The test application in various HLS tools is evaluated based on the criteria such as verification, area, level of abstraction, documentation, exploration, learning curve, data types and implementation complexity. This evaluation is presented in graphical view with the help of spider web as shown in Figure 4.

The ability of the design implementation of the tools is present in top-right, the QoR in terms of area is shown at bottom-left of the spider web, the experience of the user in HLS tools is shown in top-left axes of spider web finally the capability of the tools are shown in the bottom-right of the spider web. From the experience and knowledge as of now, we are conscious that many HLS tools are available, but to reduce the complexity, the tools are selected on account of software license available.

The tool quality will increase, as the distance between the spider web and outer axis decreases.

Autopilot: The Xilinx acquired the Autopilot recently which was developed by AUTOESL. The HDL developed from the various input languages (C-based) can be compiled by Autopilot. The RTL design generation did not need greater effort to modify the source code. The tools can support optimal in terms of latency, power and area. The tool is renamed as Vivado ESL after the incorporation of the tool by Xilinx.

Arbitrary precision data type's conversion should be made for each data types. Then the design should perform the interface and algorithm synthesis. The exploration of the architecture is the first step of synthesis in this tool, followed by the interface and actual implementations. The behaviour of the tool varies from datapath and control; loop unrolling is used to evaluate the datapath and FSM will take care of the control over the loop. The binding and scheduling algorithm is determined based on the user constraints.

The experience reveals that only minimal modification efforts are in need to generate the RTL. 43 lines of code are utilized from the reference design. The resultant VHDL code has the total of 2100 lines. The readability of the code over-rides the comprehensibility of the generated code. The function and loops are provided with pipeline proficiency which enables the code concurrency also the tools provide different types of interface (with/without handshake). The exploration of the design is possible by the process of loop unrolling. And at last, the generation of design report reveals the information on latency, resource estimation and total clock periods. The tool will generate a wrapper automatically, which is used to communicate between C- testbench and generated RTL design. This enables the reuse of testbench written in 'C'

code for simulation and verification. The verification time of the RTL simulation is reduced to a large extent by using the 'C'-based test bench.

Synphony C Compiler: The PICO tool was developed by Synfora and Synopsys inc. acquires the Synfora in 2010. Having this platform Synopsys developed the HLS tool, namely Synphony C Complier. The C++ and ANSI C programming language are supported by the tool. The generated architecture based the specific architecture is known as Pipeline of Processing Arrays (PPA). TCAB (Tightly Coupled Accelerator Blocks) is individually compiled which is present in PPA. The wide range of programming language constructs such as C++ and C are accepted by the Synphony C Compiler. Hence the implementation and developing the code for the Sobel edge detector is simple and moreover the manual guide you to develop the code easily. Pragmas are added in the source code so that the designers are available with more optimization options. Further, the GUI is used to have the exclude features like latency constraints, clock-rate and target-library. The tool can support and generate interface based on streaming and memory. Effective design is not possible by using the default options. The design in symphony leads to additional output/input buffer as result huge area is consumed; later it is overcome by setting the correct options.

The feasibility of automatic verification is provided by the Synphony, the verification is carried out in different level, namely, after scheduling, after preprocessing, source-code, after synthesis and on overall generated design. The HDL simulator from other vendors can also be called from the tool. The golden reference is used to cross check the simulation result. The result is stored in the form of text and also be compared with the other testbench. The output bitmap of the Sobel edge detector can be verified by this method also. There is possibility of interactive simulation while the default simulation running in the background. The RTL synthesis script was generated by Synphony C compiler and after the synthesis of RTL; the competition tool finds average results in terms of area.

Agility Compiler: Agility Design Solutions (ADS) developed an HLS tool called as Agility compiler. The ADS were formed in Jan '2008 as the Catalytic joined hands with Celoxica. The product of SystemC and Handel-C are maintained by Mentor Graphics after it acquires the ADS in Jan '2009. SystemC is synthesized by the agility compiler effectively. The agility hardware support and OSCI SystemC are the support given completely by the agility compiler. This compiler support automatic code-generation for FPGA from different vendors such as Altera, Xilinx and Actel (now Microsemi). The constraint in this tool is that the sensitivity list and hardware process should be written by the user manually. The description of the hardware can be made by transaction level modeling which is the vital advantage over the other HDL entry. The approach behind this tool is to form a channel by separating the communication between modules implementation and modules. The module's interface function will handle the process of transactions. The function will handle the details of data exchange at low level. Another major drawback is the automatic generation of testbench is not available. The testbench should be manually developed using the SystemC.

The possible feature is provided by the tool during synthesis process is re-timing, code-modification, automatic tree-balancing and fine grained logic sharing. The tool uses the event-driven kernel along with C++ class library for simulation. The IP which already exists in black box can be called by the feature of additional hardware present additionally. The synthesis optimization and area estimation after each step is generated by extensive XML. The automatic creation of control graph and data flow graph are then extended by the tools. The use of channels is the critical advantage of the tools as far as our experience. And the operator overloading of the C++ language can be used precisely by the user at expert level and conclude this tool as it is best suited for the application where the description of the hardware is at low level.

Impulse CoDeveloper: Impulse accelerated technologies developed this tool (Impulse CoDeveloper) which generate the RTL based on 'C' programming. The supported FPGA platforms are Pico, Nallatech, Xilinx and Altera. The applications based on data-flow are supported by Impulse CoDeveloper. The availability of documentation is more and it is easy for the beginner.

Processor Acceleration and Module generation are the two ways the tool can facilitate the designer. The hardware accelerators can be produced by Processor Accelerators that can further connect to embed FPGA processor. On the other hand, RTL module that can be created by module generation, then it can be embed with various IP to form a huge application. ANSI C is extended to form an ImpulseC and this language is not more

complex in forming the design (algorithmic and behavioral level). Hence this tool needs minor changes in the original 'C-code' to implement the Sobel Edge Detector. CSP (Communication Sequential Processor) is the tool depend for the programming model. The tool lacks the customized in-build IP, however the readability of the code is high. Shared-memories, register, streams and signals are used for communication among the process. The tool support various bus-interface and the interface between the embedded processor and the targeted FPGA is established by including several libraries. The design optimization of the tool is as follows, the parallelism can be monitored by using the Stage Master Explorer. The stream visualization is made possible by Application Monitor. The pipeline graph is used to find the delay in optimal-stage and pipelines are observed by data-flow graph. The Impulse CoDeveloper includes various tools for the process of verification, debugging and simulation. The prototype is compiled and validated by Desktop Simulation Model and also the ModelSim is interfaced with the tool. The inclusion of cycle-based hardware simulator and Stage Maser Debugger depends on the license. The synthesis result reveals that the code-line is increased due to the inclusion of interface and as a result there is degradation in readability of the code.

C-to-Silicon: The cadence developed a recent HLS tool, namely, C-to-Silicon. The SystemC is used as a design language in this tool. The SystemC wrapper will be generated by the CtoS tool, if the design code was written in TLM 10, 'C' or C++. This tool cannot support the OSCI SystemC standard since the SystemC based wrapper was generated by CtoS. This tool is mainly developed by targeting the ASIC design. And to target the FPGA flow, this tool should turn off some optimization techniques. The writing and implementing the source code for the Sobel edge detector is easy, since the C++ and ANSI C is accepted by CtoS. The documentation as well starting the C-to-S GUI is somewhat difficult. Default settings are often missing make the user un-comfortable and moreover the non availability of the documentation support made the tool curve very complex or steep. The SystemC data types support fixed point and bit width data types. The users are facilitated with memory mapping and loop optimization based on many constraints. The exploration of the design has been too complex since the option avail by the tool the user is hard-hitting. By using the RTL synthesis tool, the CtoS create the technology library while the design is in process. The RTL model generated along with verification wrapper is cycle-accurate. Exclusive verification should be made using the Cadence IUS simulator. Cycle-by-Cycle basis comparison is made between the reference and generated output. This statement proves that, the number of clock-cycle modified in scheduling process or generating a design based on source code (untimed) cannot be verified automatically.

DK Design Suite: Mentor Graphics acquired DK Design Suite in 2009. The sign is made on the basis that DK customer will be supported by Mentor Graphics till the Catapult-C transition occurs. Then the major feature enhancement is made by Mentor Graphics and intimated in websites. The interface definition is made available in the language that is a subset of 'C' language and Catapult-C is one among them. This tool uses the Catapult-C for many extended advantages. Apart from this, Catapult-C can support interfacing such as memories and FIFO and can handle parallelism and precision data types. The competitive construct of the language consume the designer time by specifying the parallelism explicitly. This process makes the job of the designer (even experienced) to develop the code for Sobel edge detector a complicated one. The GUI of DK can handle easily and it is straightforward. The changes can be determined by looking into software IDE. In alternate to this interface can be made available from the command line. The learning curve is simple, since the manual and online supports for documentation are ease of access. This tool is developed vitally to support the FPGA platform and hence the source code is modified to pin mapping and component selection. The demerits of the tool arise here since it cannot import to different platforms. Exploration of the design is simple by modifying the source-code in terms of macros, parallelism, nested loops. The simulation is needed to carry out the verification of the modified optimization.

The verification of the source code is done by the source level simulator present in GUI. The interface is used to write or read from the simulators. The verification of the generated design is harder since the porting of data-port from DK is difficult. In addition the automatic testbench generation is not available. The behavioral Verilog or VHDL design is generated by DK design suite. Finally a lot of manual efforts are needed to fine-tune the throughput and latency.

Icarus Verilog: Stephen Williams developed the Icarus Verilog which is open source. It uses the Verilog to develop the source code for the design. The developed

design can be targeted to FPGA and modifying the source code is possible. Icarus Verilog V 0.8 supports the synthesis and the release after this version (i.e. V0.9) is only intended for simulation process. There are many reasons for the halt of synthesis in the newer version, since the advance feature in V 0.9 has modified the source code of the synthesis leading to the obstruction of the synthesis process. Apart from that the existing FPGA vendors provides the synthesis tools that are more efficient than the Icarus Verilog. In addition the V 0.8 facilitates the synthesis process with some limitations [65].

On the other hand, Verilog Virtual Processor (VVP) and the IVerilog compiler are the vital part of the Icarus Verilog. This tool will generate an intermediate code for the design called as VVP assembly code which will act as bridge between IVerilog compiler and VVP. The verilog code, developed for the Sobel edge detector is translated into VVP assembly code by the IVerilog compiler (also a translator). The interrupt based on the events are supported by the VVP simulator. The event can be processed by the VVP simulator by interrupting the VVP assembly code.The verilog language is used to develop the design inspite of 'C'-programming language, since the restriction in 'C' is overcome by using the modules to specify the design or applications. Moreover, the simulations with hierarchical module are not appropriate and produce far-optimal results. The five phases that are narrowing down by the IVerilog compiler is as follows Preprocessor, Parser, Elaboration, Optimizer, Code generator.The define-directive's macro substitution and include directive's file inclusion process is performed by the preprocessor. The internal representation of the source file is generated by parser block, followed by the syntax and semantics checking. The elaboration phase will generate flattened netlist based on hierarchy, before the generation the module instantiation is expanded and root module is located. The efficiency of the simulation is improved in the optimization stage by transforming the internal netlist. The entire design information is accumulated in netlist optimized internally and the code is generated in the form of VVP assembly code. The VVP simulator will act as interpreter for the design generated (VVP assembly code). The VVP assembly codes are parsed by VVP simulator and expose the structural part of the netlist. Then, various inputs from the testbench are inserted into the design to check the functionality. In JAVA the compiler and interpreter are separated, likewise the Icarus Verilog separates the VVP simulator and IVerilog compiler. The byte-code in JAVA is the counterpart of VVP assembly code in Icarus Verilog. Although the documentation support is not feasible for the user, the tool developer provides lots of online support. The software engineers are still working on the tool to support both the simulation and synthesis for the targeted FPGA. The development area varies as maintaining the perfect parallelism, validation accuracy, application-driven interrupt, large design space exploration documentation support [66].

Xilinx AccelDSP: AccelChip DSP technology develops the HLS tool, namely, AccelDSP. Later in 2006, Xilinx inc. acquired the AccelChip DSP technology. Matlab description is used in AccelDSP and works on own GUI. The code should be developed explicitly by the designer for streaming function, since the tool is available only with streaming function. Signal and image processing application can be targeted in this tool. The tool support only the limited part of the standard M-code. The basic building blocks are instantiated and Matlab-code should be scanned to generate the HDL-code automatically. The Golden reference model is used by the designer to write the floating point operation. The design of Sobel edge detector is made easy by using the M-code. There is no need for manual conversion from fixed point to floating point or vice versa, since the bit-width is estimated dynamically. 53-bit is assigned to the fixed point and the extension is should be reported to the toolbox. The over-flow in the operation needs some manual interpretation, which is exposed by our experiments. The design exploration in AccelDSP is more when compared to a Xilinx system generator. The possible supports available are memory mapping, matrix and loop manipulation unrolling and pipelining. The Sobel edge filter is used to test the AccelDSP design exploration. 61MHz clock speed is needed initially and 158 MHz clock speed is achieved with a 4% increase in the resource when the pipeline option is enabled. The feature of fixed point probes is provided by AccelDSP, the internal signal can be plotted at a specific time by inserting the command line manually. The testbench can be generated automatically, so that the validation complexity of the design is reduced. The design can be targeted to FPGA by FIL (FPGA-in-loop) simulation. The three types of generation available after the design process are creating the Xilinx system generator block, simulation and generating the RTL-code. The handshake-interface is used to communicate between the generate RTL-code to other process. The Synchronous control signal is used by the module to produce and request data. The tool

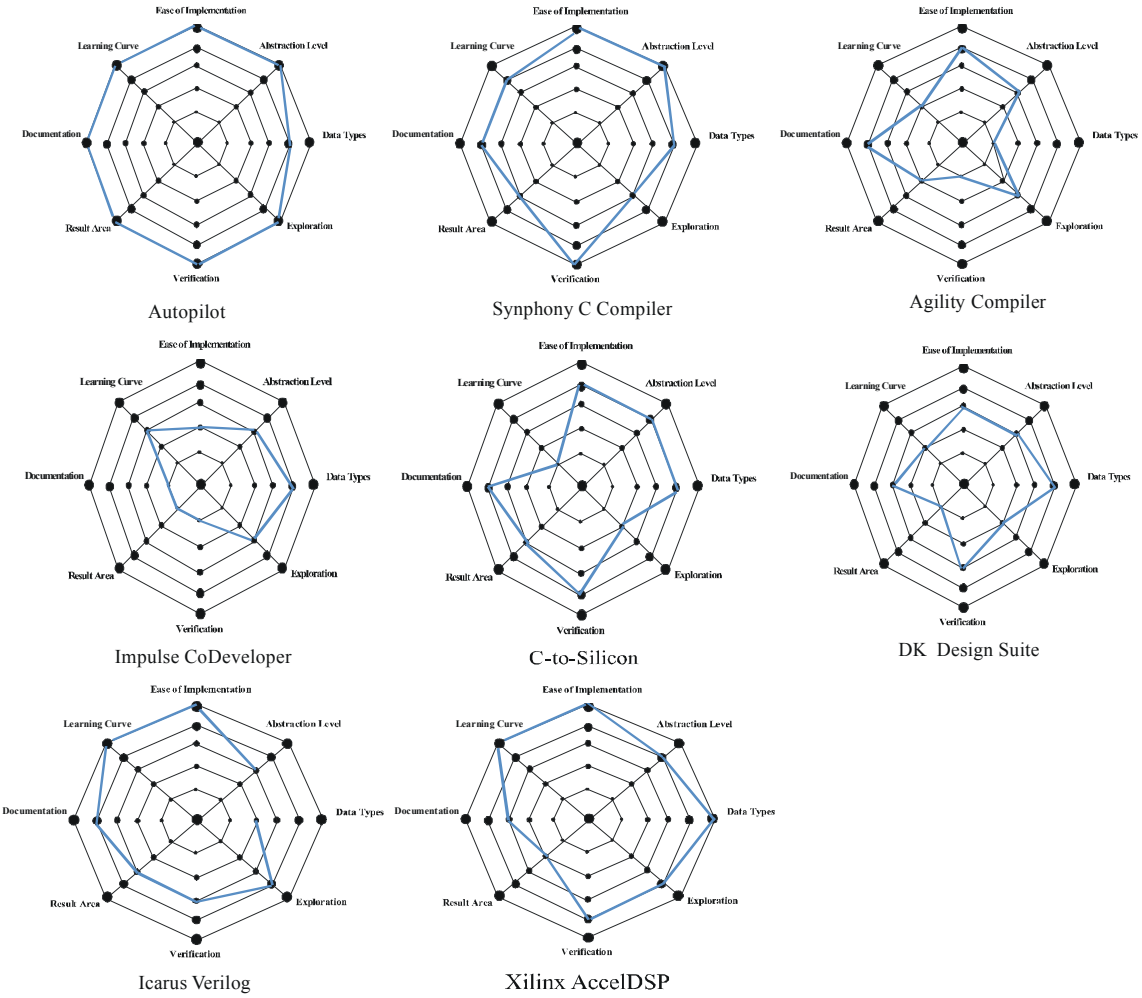

Fig. 5: Graphical Diagram (Spider Web) for Tools in Section

provides a powerful support limiting to applications based on the streaming data. The streaming loop need modified Matlab-code. The direction from the designer is still needed apart from the conversion of fixed point automatically. Generation of the Xilinx system generator block is added feature along with RTL generation [67].

Evaluation of the Tools: To develop the good HLS tools several features should be incorporated as per the designer needs. The source language should support the following features such as cycle-accurate and untimed abstraction level for control application and data-flow respectively. The learning curve for creating the design and using the tool should be flat. In similar, the extension of the design and exploring the design need smaller modification. The tool should support the interfacing with RTL synthesis and verification. In addition the design quality should meet at least the near optimal QoR of the RTL design.The various commercial HLS tools are presented. The Table 1 shows the various vital-characteristic of the tools and the graphical view of the same data is represented using the spider web (Figure 5). The tools quality is better, if the formed spider-web is closer to the outside circle. As per our knowledge and experience we have scored the tools with different criteria. However, the result and evaluation presented in spider web and Table 1 is a subject always under debate. And this survey result gives a preference for the designer to start-up with the HLS tools in a more precise manner. The scoring value varies from 1-5, the exploration of the design need little modification if the value is 5 and more modification is expected if the value is 1. In similar, the area consumed by the design is less if it is 5 and varies down to 1. Developing the design and modifying the source code is easy, if the learning curve is flattened (i.e.5). The implementation complexity is increased if

Table 1: Review of HLS Tool

Tools/Features	Autopilot	Synphony C Compiler	Agility Compiler	Impulse CoDeveloper	C-to-Silicon	DK Design Suite	Icarus Verilog	Xilinx AccelDSP
Source	C, C++, SystemC	C,C++	SystemC	ImpulseC	C, SystemC	HandelC	Verilog	Matlab
Floating/FixedPoint	Fixed point	Fixed point	Fixed point	Fixed point	Fixed point	Fixed point	Floating point	Auto conversion
Abstraction Level	Untimed & Cycle	Untimed	Cycle accurate	Cycle & Untimed	Cycle & Untimed	Cycle accurate	Cycle & Untimed	Untimed
Testbench generation	Y	Y	N	N	Y	N	N	Y
Design Exploration	5	3	3	3	2	2	4	4
Implementation	5	5	4	2	4	3	5	5
Documentation	5	4	4	1	4	3	4	9
Learning Curve	5	4	2	3	1	2	5	5
Verification	5	5	1	1	4	3	3	4
FPGA Slices	232	1047	107	4611	1776	311	1312	2411

the value is at 1. Likewise, the feasibility and complexity vary for different features based on the value scored. To analyze the result, we have implemented the design in Xilinx Virtex5/Virtex2Pro FPGA and in restricted time the achieved result is reasonable. In contrast to this the resource consumed are varies based on the constraints and tools.

CONCLUSION

The various High Level Synthesis (HLS) tools are presented in this paper. The digital circuits are designed and developed using various abstraction levels. The complexity in handling the embedded system can be made easier with these evolved tools. The verification of the design is made faster with the help HLS, which take more tasks from the designer. To accomplish the task of system implementation (a high level application) and time to market, the perfect counterparts of the FPGA and HLS are utilized. The design productivity can be still increase by finding solutions to the challenges, which we came across during our work. At first, the development of specific tools needs to be trained to the designer (i.e. design entry should standardizes). The optimization and exploration options of all the tools should be augmented even with better tools. For example, reducing the memory utilization can be included in optimization stage. The architecture and functionality of the design varies by modifying source-code. An application based on control logic and data flow needs some tools targeting the application. The tools that have a single tool flow and language for developing the design will not fit for all the applications. The designer can have a view on HLS tools at present, by the efficient comparison given in this paper. The designer can select a tool among various tools available by sort out the vital metrics in the spider web diagram. The tools can be further enhanced by the developers and vendors by the reviewing this paper and thus everyone will be benefitted.

REFERENCES

1. Gajski, D.D. and R.H. Kuhn, 1983. New VLSI tools. Computer, pp: 11-14.
2. Moore, G.E., 1965. Cramming more components onto integrated circuits, Electronics, pp: 116-144.
3. International Technology Roadmap for Semiconductors (ITRS), 2009edition. http:// www.itrs.net/ links/2009ITRS/Home2009.htm
4. Wakabayashi, K., 2004. C-based behavioral synthesis and verification analysis on industrial design examples ASPDAC, 04: 344-348.
5. Ghenassia, F., 2005. Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems, Springer.
6. IEEE and OCSI, 2005. IEEE 1666TM-2005 Standard for SystemC. http://www.systemc.org.
7. Keutzer, K., S. Malik, A.R. Newton, J. Rabaey and A. Sangiovanni-Vincentelli, 2000. System level design: orthogonalization of concerns and platform-based design, IEEE Trans. CAD, 19: 1523-1543.
8. Denolf, K., S. Neuendorffer and K. Vissers. 2009. Using C-to-gates to program streaming image processing kernels efficiently on FPGAs, FPL, 09: 626-630.
9. Kindratenko, V. and R. Brunner, 2009. Accelerating cosmological data analysis with FPGAs. FCCM, 09: 11-18.
10. Pingree, P.J., L.J. Scharenbroich, T.A. Werne and C.M. Hartzell, 2008. Implementing legacy-C algorithms in FPGA co-processors for performance accelerated smart payloads, IEEE Aerospace Conference, pp: 1-8.
11. Guo, Y., D. McCain, J.R. Cavallaro and A. Takach, 2006. Rapid industrial prototyping and SoC design of 3G/4G wireless systems using an HLS methodology, EURASIP Journal on Embedded Systems, pp: 1.
12. Avnet Spartan-6 FPGA DSP Kit. http://www.xilinx.com/ products/devkits/ AES-S6DSP-LX150T-G.htm.

13. Xilinx Spartan-6 FPGA Consumer Video Kit. www.xilinx.com/ products/ devkits/TB-6S-CVK.htm.
14. Director, S., A. Parker, D. Siewiorek and D. Thomas Jr, 1982. A design methodology and computer aids for digital VLSI, IEEE Trans. Circuits and Systems, 28: 634-645.
15. Parker, A., D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Leive and J. Kim, XXXX. The CMU design automation system: an example of automated data path design. DAC, 79: 73-80.
16. Barbacci, M., G. Barnes, R. Cattell and D. Siewiorek, 1978. The symbolic manipulation of computer descriptions: the ISPS computer description language, Dep. Comput. Sci, Carnegie-Mellon Univ., Pittsburgh, PA,Tech. Rep.
17. Snow, E., D. Siewiorek and D. Thomas, XXXX. A technology-relative computer aided design system: abstract representations, transformations and design tradeoffs. DAC, 78: 220-226.
18. Granacki, J., D. Knapp and A. Parker, XXXX. The ADAM advanced design automation system: overview, planner and natural language interface, DAC, 85: 727-730.
19. Jain, R., K. Kucukcakar, M.J. Mlinar and A.C. Parker, XXXX. Experience with ADAM synthesis system, DAC, 89: 56-61.
20. Paulin, P.G., J.P. Knight and E.F. Girczyc, XXXX. HAL. A multi-paradigm approach to automatic data path synthesis, DAC, 86: 263-270.
21. Marwedel, P., XXXX. The MIMOLA design system: tools for the design of digital processors. DAC, 84: 587-593.
22. De Micheli, G., D. Ku, F. Mailhot and T. Truong, 1990. The Olympus synthesis system, IEEE Design & Test of Computers, 7: 37-53.
23. Micheli G. De and D. Ku, XXXX. HERCULES. A system for high-level synthesis, DAC, 88: 483-488.
24. Chandrakasan, A., M. Potkonjak, J. Rabaey and R. Brodersen, XXXX. HYPER-LP. a system for power minimization using architectural transformations, ICCAD, 92: 300-303.
25. Rabaey, J., C. Chu, P. Hoang and M. Potkonjak, 1991. Fast prototyping of datapath-intensive architectures, IEEE Design & Test, 8: 40-51.
26. Man H. De, J. Rabaey, J. Vanhoof, P. Six and L: Claesen, 1986. Cathedral-II—a silicon compiler for digital signal processing, IEEE Design &Test of Computers, 3: 13-25.
27. Yassa, F.F., J.R. Jasica, R.I. Hartley and S.E. Noujaim, XXXX. A silicon compiler for digital signal processing: methodology, implementation and applications. in Proc. IEEE, 7: 1272-1282.
28. Composano, R., XXXX. Design process model in the Yorktown Silicon Compiler, DAC, 88: 489-494.
29. Composano, R., 1991. Path-based scheduling for synthesis, IEEE Trans. CAD, 10: 85-93.
30. Paulin, P.G. and J.P. Knight, 1989. Force-directed scheduling for the behavioral synthesis of ASIC's, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 8: 661-678.
31. Park, N. and A. Parker, XXXX. Sehwa. a program for synthesis of pipelines, DAC, 86: 595-601.
32. Ku, D. and G. De Micheli, XXXX. Relative scheduling under timing constraints, DAC, 91: 59-64.
33. Ku, D. and G. De Micheli, 1990. A language for hardware design (version 2.0). Technical Report, UMI Order Jumber: CSL-TR-90-419, Stanford University.
34. Küçükçakar, K., C.T. Chen, J. Gong, W. Philipsen and T.E. Tkacik, 1998. An architectural design tool for commodity ICs, IEEE Design and Test of Computers, 15: 22-33.
35. Biesenack, J., M. Koster, A. Langmaier, S. Ledeux, S. Marz, M. Payer, M. Pilsl, S. Rumler, H. Soukup, N. Wehn and P. Duzy, 1993. The Siemens high-level synthesis system CALLAS, IEEE Trans. VLSI Systems, 1: 244-253.
36. Bergamaschi, R.A., R.A. O'Connor, L. Stok, M.Z. Moricz, S. Prakash, A. Kuehlmann and D.S. Rao, 1995. High-level synthesis in an industrial environment, IBM Journal of Research and Development, 39: 131-148.
37. Lippens P.E.R., J.L. van Meerbergen, A. van der Werf, W.F.J. Verhaegh, B.T. McSweeney, J.O. Huisken and O.P. McArdle, XXXX. PHIDEO. A silicon compiler for high speed algorithms, EDAC, 91: 436-441.
38. Knapp, D.W., 1996. Behavioral synthesis: digital system design using the Synopsys Behavioral Compiler, Prentice-Hall.
39. Hemani, A., B. Karlsson, M. Fredriksson, K. Nordqvist and B. Fjellborg, XXXX. Application of high-level synthesis in an industrial project, VLSI Design, 94: 5-10.
40. Elliott, J.P., 1999. Understanding behavioral synthesis: a practical guide to high-level design, Springer.

41. Edwards, S.A, 2006. The challenges of synthesizing hardware from C-like Languages, IEEE Design & Test of Computers, 23: 375-386.
42. Sanguinetti, J., 2006. A different view: hardware synthesis from SystemC is a maturing technology, IEEE Design & Test of Computers, 23: 387-387.
43. Gajski, D., J. Zhu, R. Dömer, A. Gerstlauer and S. Zhao, SpecC. 2000. specification language and methodology, Kluwer Academic Publishers.
44. Agility Design Solutions. 2007. Handel-C language reference manual.
45. Haldar, M., A. Nayak, A. Choudhary and P. Banerjee, XXXX. A system for synthesizing optimized FPGA hardware from MATLAB, ICCAD, 01: 314-319.
46. BlueSpec, Inc. http://www.bluespec.com.
47. Edwards, S.A., XXXX. High-level synthesis from the synchronous language Esterel. IWLS, 02.
48. Pellerin, D. and S. Thibault, 2005. Practical FPGA programming in C, Prentice Hall Professional Technical Reference.
49. Gokhale, M., J. Stone, J. Arnold and M. Kalinowski, XXXX. Stream-oriented FPGA computing in the Streams-C high level language, FCCM, pp: 49-56.
50. Gupta, S., R. Gupta, N. Dutt and A. Nicolau, SPARK, 2004. A parallelizing approach to the high-level synthesis of digital circuits, Springer.
51. Tripp, J.L., M.B. Gokhale and K.D. Peterson, 2007. Trident: from high-level language to hardware circuitry, IEEE Computer, 40: 28-37.
52. Nallatech, Inc., DIME-C user guide.
53. Altera Corporation. 2009. Jios II C2H compiler user guide, version, 9: 1.
54. Budiu, M., G. Venkataramani, T. Chelcea and S. Goldstein, XXXX. Spatial computation, ASPLOS, 04: 14-26.
55. Mencer, O., XXXX. ASC. a stream compiler for computing with FPGAs, IEEE Trans. CAD, 25: 1603-1617.
56. Coussy, P., C. Chavet, P. Bomel, D. Heller, E. Senn and E. Martin, GAUT, 2008. A High-Level Synthesis Tool for DSP Applications, in P. Coussy and A. Morawiec Eds. High-Level Synthesis: From Algorithm to Digital Circuit, Springer.
57. Villarreal, J., A. Park, W. Najjar and R. Halstead, XXXX. Designing modular hardware accelerators in C with ROCCC 2.0, FCCM, 10: 127-134.
58. Joseph, M., Bhat Narasimha, B. Sekaran and K. Chandra, 2007. Technology driven High-Level Synthesis, International Conference on ADCOM, pp: 485-490.
59. Wakabayashi, K., B. Schafer, P. Coussy and A. Morawiec Eds, 2008. High-Level Synthesis: From Algorithm to Digital Circuit, Springer.
60. Bollaert, T., 2008. Catapult synthesis: a practical introduction to interactive C synthesis, in P. Coussy and A. Morawiec Eds. High-Level Synthesis: From Algorithm to Digital Circuit, Springer.
61. http://iverilog.icarus.com.
62. Bailey, B., F. Balarin, M. McNamara, G. Mosenson, M. Stellfox and Y. Watanabe, 2010. TLM-Driven Design and Verification Methodology, Cadence Design Systems.
63. http://www.synopsys.com/Systems/BlockDesign/HLS/Pages/default.aspx.
64. Zhang, Z., Y. Fan, W. Jiang, G. Han, C. Yang and J. Cong, 2008. AutoPilot: a platform-based ESL synthesis system, High-Level Synthesis: From Algorithm to Digital Circuit, ed. P. Coussy and A. Morawiec, Springer Publishers.
65. Cong J. and W. Rosenstiel, 2009. The last byte: the HLS tipping point, IEEE Design & Test of Computers, 26(4): 104.
66. Urard, P., J. Yi, H. Kwon and A. Gouraud, 2008. User needs, in P. Coussy and A: Morawiec Eds. 2008. High-Level Synthesis: From Algorithm to Digital Circuit, Springer.
67. Green, B., 2011. Edge detection tutorial. http:// www.pages.drexel.edu/ ~weg22/edge.html. Accessed 30 March 2011.