International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

COPY RIGHT

IJIEMR Transactions, online available on 25th Oct 2017. Link :http://www.ijiemr.org/downloads.php?vol=Volume-6&issue=ISSUE-9

Title: **HIGH-LEVEL SYNTHESIS FUNCTIONAL VERIFICATION WITH SOURCE CODE ERROR DETECTION**

Volume 06, Issue 09, Pages: 276 – 281.

Paper Authors

CHITTIPROLU.ANUSHA, KONDA KIRAN KUMAR
Nalgonda institute of technology and science, nalgonda
.

USE THIS BARCODE TO ACCESS YOUR ONLINE PAPER

To Secure Your Paper As Per UGC Guidelines We Are Providing A Electronic Bar Code

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

HIGH-LEVEL SYNTHESIS FUNCTIO

NAL VERIFICATION WITH SOURCE CODE ERROR DETECTION

[1]CHITTIPROLU.ANUSHA, [2]KONDA KIRAN KUMAR

[1]M.Tech Scahloar, Dept of E.C.E(VLSI & ES), Nalgonda institute of technology and science, nalgonda.

[2]Assistant Professor, Dept of E.C.E, Nalgonda institute of technology and science, nalgonda.

ABSTRACT—A dynamic functional verification method that compares untimed simulations versus timed simulations for synthesizable [high-level synthesis (HLS)] behavioral descriptions (ANSI-C) is presented in this paper. This paper proposes a method that automatically inserts a set of probes into the untimed behavioral description. These probes record the status of internal signals of the behavioral description during an initial untimed simulation. These simulation results are subsequently used as golden outputs for the verification of the internal signals during a timed simulation once the behavioral description has been synthesized using HLS. Our proposed method reports any simulation mismatches and accurately pinpoints any discrepancies between the functional Software (SW) simulation and the timed simulation at the original behavioral description (source code). Our method does not only determine where to place the probes, but is also able to insert different type of probes based on the specified HLS synthesis options in order not to interfere with the HLS process, minimizing the total number of probes and the size of the data to be stored in the trace file in order to minimize the running time. Results show that our proposed method is very effective and extremely simple to use as it is fully automated.

I. INTRODUCTION

Raising the level of abstract some distinct advantages over traditional register-transfer action in VLSI design has level (RTL) design methods. First, most of the designs start with a high-level model in order to validate the application to be implemented. High-level synthesis (HLS) provides a direct path between these models and RTL. It has been shown [1] that one line of C-code translates into 7–10× more gates than RTL. This further implies that behavioral descriptions are easier to maintain and debug, and that fewer bugs will be introduced by designers. Second, in many cases, the design specifications are unstable and any changes in them can lead to major architectural changes (e.g., the use of on-die memory or external memory). At the RTL, this requires major redesigns, while at the behavioral level, these changes can be tackled easier. Third, raising the level of abstraction allows software and hardware designers to *speak* the same language. Applications to be implemented in custom

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

hardware are getting extremely complex and are based on complex mathematical models that in many cases are difficult to understand by the hardware designer. Using the same behavioral description language allows both hardware and software designers to communicate at the same level of abstraction using the same language. Some examples of complex applications include dedicated hardware security engines based on complex encryption and decryption algorithms [2]. In this paper, we address the issue of how to efficiently verify the functionality between an original untimed SW behavioral description and the synthesized design after behavioral synthesis. One of the main advantages of HLS is that it allows, in combination with model generators at different stages of the HLS flow, the reuse of the untimed test vectors used during the functional SW verification. Moreover, these model generators also allow using the SW simulation outputs as golden reference outputs for the timed Hardware (HW) verification. The main problem is that these test vectors only cover the inputs and outputs of the behavioral description. In case of mismatches, the user needs to start the debugging process. This is normally done by dumping the internal signals onto a Value Change Dump (VCD) file and manually verifying the waveform for any discrepancies. This poses serious problems to the designers. First, in case of largzr designs, the waveform file size can be significant if all the internal signals are dumped onto the VCD file (often impractical). Second, the user needs to fully understand the timing of the circuit and how the synthesizer has scheduled the behavioral description. Often the error is introduced in the system at a given cycle, but will not become visible at the outputs after many cycles. Finally, the designer needs to trace the error back to the original source code, which is sometimes not easy in HLS, because the synthesizers might have renamed the signals, duplicated these or optimized them away.

II. PROPOSED WORK

Our proposed method applies to the first category for HLS (presilicon), but uses concepts used in the second category (postsilicon). The notion of probes has been taken from typical VLSI postsilicon verification flows. For example, commercial Field Programmable Gate Array tool vendors provide on-chip support to allow the observability of internal signals (e.g., Chipscope in Xilinx [12] and SignalTap in Altera [13]). These tools insert probes to signals in the design to be tested and capture them using a sampling clock, while storing them in a buffer. The buffer content is transmitted to a PC and displayed graphically, once the buffer is full or certain number of samples taken. The designer can then manually verify the correctness of the design. ARM does also provide a similar technology to debug ARM-based systems-on-a-chip with the Advanced High-performance Bus (AHB) trace macrocell, which gives visibility on Advanced Microcontroller Bus Architecture AHB busses, offering visibility of accesses to memory areas [14]. To be able to root-cause design bugs, postsilicon validation requires

to have full controllability and observability of the circuit under debug's (CUD) internal behavior. This can currently not be achieved due to the extremely larger number of signals that would need to be traced. A more effective debug technique is to selectively monitor some of the internal signals. Designers typically select to tap a number of signals in the CUD, but only a subset of the tapped signals are traced concurrently during debug phase due to trace bandwidth limitation. This is achieved by inserting a mux tree that links the tapped signals to trace buffers or trace ports. These systems also include trigger units, which are used to determine when to start and stop signal tracing in order to further reduce trace bandwidth requirement [18]. The effectiveness of these trace-based debug systems, hence, rely considerably on the signals being traced. In current postsilicon validation flows designers usually manually select those signals that are important for analysis to trace, based on their own design experience. This ad hoc method, however, cannot guarantee the quality of debug process [17]. More importantly, bugs often occur in unexpected scenarios and it is very difficult, if not impossible, to predict which signals will be related to them during the design phase. Ko and Nicolici [19] first introduced an automated method identifying a small set of trace signals from which a large number of states can be restored using a compute-efficient algorithm. This enlarged set of data can then be used to aid the search of functional bugs in the fabricated circuit. Liu and Xu [21] expanded this paper conducting circuit-level propagation of risibilities from traced signals to untraced ones achieving a more accurate visibility estimation. Although our work applies to a completely different VLSI design stage, its main objective is similar to the postsilicon validation techniques. This paper targets the verification at the synthesis level. A classification of synthesis verification is given in [25]. This paper classifies the synthesis verification into presynthesis verification of algorithm(s) to be synthesized typically using software verification methods, formal methods using theorem provers and postsynthesis verification, where the synthesized results are verified against the input behavioral descriptions. This last category is the most widely used today, to which this paper also belongs. This last category can be further classified into simulation based and formal based. Formal methods have been applied to verify the HLS process usng translation validation. For example, Ashar et al. [27] focused on the valid binding stage of HLS, while recently [26] focused on the scheduling and concurrent systems modeling communicating sequential processes. Formal methods have gained popularity because RTL simulations for larger designs, simulations are too slow and cannot detect corner cases. Other formal verification approaches include [30], where a fully automatic equivalence verification of a design before and after the scheduling step of HLS is presented. This paper was extended in [31] by mapping the designs into virtual controllers and virtual datapaths. A more recent work [32] uses a finite-state machine (FSM) with datapath models to

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

represent both behaviors (untimed and timed). Our work is fundamentally different from this previous works as it is simulation based. An early work on simulation-based HLS verification is presented in [28]. The advantages of simulation based methods are that simulations are always needed for overall functional verification. Moreover, we apply our method to compare pure software (untimed) simulations and use cycleaccurate model simulations as the timed model instead of the synthesizable RTL generated by HLS. Cycle-accurate models have been reported to be 10–100× faster than RTL simulations [3], making our method fast enough to work for larger designs. We can define the problem to be solved as follows. Problem Definition: Find the signals to be traced and probe insertion points in a behavioral description for HLS in order to locate the operation in the source code where the error is first introduced, minimizing the simulation running time and trace file sizes, without distorting the intended HLS result. To the best of our knowledge, this is the first work investigating functional verification methods comparing untime versus timed simulations at the behavioral level. Section II-A describes a typical HLS verification flow, indicating where our proposed method fits in the overall VLSI verification flow, followed by a detailed description of our proposedmethod.

A. High-Level Synthesis Verification Flow HLS takes as inputs a behavioral description, e.g., C, C++, or SystemC, and generates synthesizable RTL(Verilog or VHDL) by creating a control unit in the form of an FSM and a data path unit. The datapath unit mainly consists of a number of functional units (FUs) combined with registers and multiplexers. Most commercial HLS tools provide tools to verify and debug the design at the highest possible level of abstraction in order to facilitate the verification process. For this purpose, they normally include model generators that create different types of simulation models depending on the design stage. Fig. 1 shows the different verification stages, including the different model generators.

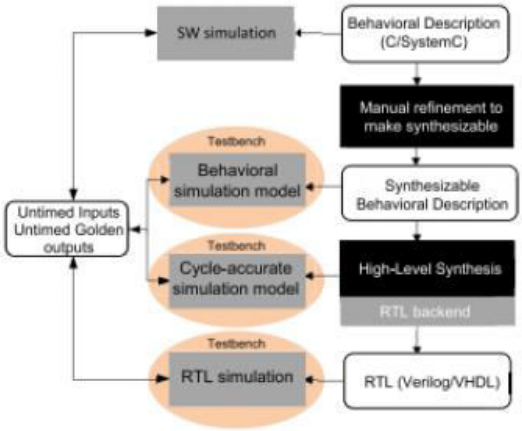

Fig. 1. HLS design flow and verification overview.

When using HLS, he first step designers need to take is to manually refine the original SW description in order to make it synthesizable. Some of the typical constructs that are not supported in HLS are dynamic memory allocation and recursion. At this stage, the designer also refines the data types in order to obtain the smallest possible and most efficient HW design. For this purpose, most HLS vendors extend the C syntax providing their own data types (e.g., CatapultC uses ac_types [5] and

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

CyberWorkBench var_types [6]). In the case of SystemC, the sc_types are used. For the verification at this state, some of the vendors include behavioral model generators. These model generators create a behavioral program that models the original behavioral description including its custom data types. After HLS, a cycle-accurate simulation can be performed for timing verification. These cycle-accurate models again generate a behavioral description in any high-level language, i.e., ANSI-C or C++/SystemC, and mimic the behavior of the RTL cycle accurately. The input to these cycle-accurate model generators are normally the result of the HLS scheduling phase (a byproduct of HLS). These models have been proven to be consistently faster than RTL by a factor of 100–1000× for the behavioral model and 10–100× faster for the cycle-accurate model [3]. A testbench generator that allows the reuse of the untimed SW inputs and outputs is typically part of these model generators.

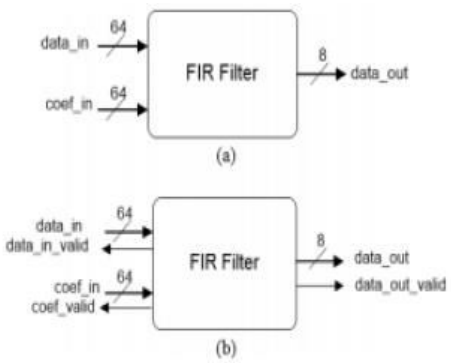

Fig. 2. (a) Regular FIR filter. (b) FIR filter with valid signals for DEC.

III. CONCLUSION

In this paper, we have presented a complete automated verification flow for synthesizable behavioral descriptions in order to detect where in the source code mismatches between the original untimed simulation and the timed synthesized design occur. Our proposed verification flow leverages the latest verification features of commercial HLS tools, which allow the reuse of transaction level test vectors for timed simulations. By automatically inserting a set of internal probes our method can efficiently detect mismatches between the untimed behavioral simulation and the synthesized circuit and locates where the error is introduced directly at the source code based on the distances between probes. This paper introduces the term SCED to determine the quality of our verification environment. The proposed method inserts different types of probes based on the synthesis directives for arrays and loops and makes use of synthetic operators in probes for arrays to avoid the probes interfering with the HLS results. Three different probe insertions methods are presented each with unique tradeoffs (SCED versus simulation runtime versus VCD file size). A set of experiments were conducted and an error was found in one of the designs that would have taken much longer time to find using a manual approach, further validating our verification methodology. The probe library is currently being extended to include probes, e.g., partial loop unrolling.

REFERENCES

[1] H.R.Rategh et al., "A CMOS frequency synthesizer with an ijectedlocked frequency divider for 5-GHz wirless LAN receiver," IEEE J Soli-State Circuits, vol. 35, no. 5, pp. 780-787, May 2000.

[2] P. Y. Deg et al., "A 5 GHz frequency synthesizer with an injection locked frequency divider and differential switched capacitors," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 56, no. 2, pp. 320-326, Feb. 2009.

[3] L. Lai Kan Leung et al., "A I-V 9.7-mW CMOS frequecy synthesizer for IEE 802.lIa transceivers," IEEE Trans. Microw. Theor Tech., vol. 56, no. I, pp. 39-48, Jan. 2008.

[4] M. Alioto and G. Palumbo, Model and Design of Bipolar and MOS Current-Mode Logic Digital Circuits. New York: Springer, 2005.

[5] Y. Ji-ren et al., "Atue single-phae-clock dynamicCMOScircuit technique," IEEE J Solid-Statem Circuits, vol. 24, no. 2, pp. 62-70, Feb. 1989.

[6] S Pellerno et aI., "A J3.5-mW 5 GHz frequenc synthesizer with dyamic-logic frequency divider, " IEE J. Solid-State Circuits, vol. 39, no. 2, pp. 378-383, Feb. 2004

[7] V. K. Manthena et aI., "A low power fully programmable J MHz resoluti on 2.4 GHz CMOS PLL frequenc synthesizer, " in Proc. IEE Biomed. Circuits Syst. Conf, Nov. 2007.pp 187-19.

GUIDE DETAILS

Konda Kiran Kumar Assistant Prof.
M.Tech (Vlsi Sd),Kiran3712@Gmail.Com
Dept.Of Ece (Vlsi System Design)

STUDENT DETAILS:

Chittiprolu.anusha,Dept of ECE : vlsi & ES Chittianusha363gmail.com
Nalgonda institute of technology and science (nalgonda)