A Formal Equivalence Checking Methodology for Simulink and Register Transfer Level Designs

Muharrem Orkun Saglamdemir*†, Alper Sen‡, Gunhan Dündar*
*Department of Electrical and Electronics Engineering, Bogazici University, Istanbul, Turkey
‡Department of Computer Engineering, Bogazici University, Istanbul, Turkey
†Ericsson Microelectronics Design Center, Istanbul, Turkey
Emails: orkun.saglamdemir@boun.edu.tr, alper.sen@boun.edu.tr, dundar@boun.edu.tr

Abstract—**Driven by the increase in complexity of design, time-to-market pressure and the need for a high level of collaboration between multiple discipline teams in a project, model based design has become the inevitable choice for IC Design projects. High-level models are being substantially used as the reference for implementation of the Register Transfer Level (RTL) counterpart of the designs. In that respect, Matlab/Simulink is one of the adopted high level modeling platforms in the IC design industry. However, checking the formal equivalence of the models with their RTL counterparts is still an area of interest to be investigated. In this study, a methodology addressing that matter is proposed. Simulink models of interest in this paper comprise built-in Simulink blocks, Stateflow blocks modeling the state machines, and user-defined blocks. Proposed methodology utilizes Simulink's Hardware Design Language (HDL) Coder and Real Time Workshop (RTW) tools, Mentor Graphics' Catapult, and Synopsys' Formality in the flow. Building of the methodology is explained with a simple example. Then the methodology is applied to multiple designs, including Advanced Encryption Standard (AES) to verify its applicability.**

I. Introduction

Model Based Design is a natural response to today's IC Design industry trends, since complexity of designs increase exponentially, the intention to offer new products in a fast pace increases time-to-market pressure and implemented designs require collaboration of teams from multiple disciplines like Analog/Mixed-Signal/Digital or Hardware/Software [1]. By the help of model based design, verification, which consumes most of the time in an IC project, becomes faster and architectural optimizations become feasible in early design phase.

In model based design, an executable model description is the starting point to define the system at the highest level of abstraction. Languages like C/C++, SystemC are widely used in model based design [2]. Matlab/Simulink is also commonly used as a platform for modeling the algorithm specification and simulation, so that it is adopted for merging design, optimization, and ASIC hardware verification steps as in [3], and defined to be the starting point of model based System-on-Chip (SOC) design at the system level [4].

Automatic generation of low-level codes is another motivating factor of model based design, since manual implementation of the system level model is error prone and time consuming. Automatically generated codes can be in the form of RTL (Verilog, VHDL) or C/C++. There are various High Level Synthesis (HLS) tools generating RTL codes from C/C++ and SystemC. Simulink has tools, namely Hardware Design Language (HDL) Coder and Real Time Workshop (RTW), for generating RTL and C codes, respectively. In [5], Simulink and HDL Coder is used for High Level Modeling and code generation of Low Density Parity Check Decoders. In [4], RTW is utilized for generating C code from the Simulink model as a part of their SOC design flow in system level.

As it is the case for other HLS processes, correctness of the automatic code generation needs to be checked. In [6], formal validation of the translation from Simulink to C by means of RTW is studied for a subset of built-in Simulink blocks. However, to the best of our knowledge, formal equivalency of the HDL code generated by Simulink HDL Coder is not investigated in a study.

In the IC design industry, it is a common practice that Simulink model is designed by the system designers and the RTL counterpart is written by the design engineers. This process may be error prone, even if the same designer writes both code. So the equivalence between the Simulink model and its RTL counterpart should be guaranteed in order not to have misalignments. It is possible to use simulations on system level and RTL designs for checking equivalence. However, this suffers from coverage, whereas formal equivalence checking gives 100% coverage. In that respect, a methodology for checking the equivalence between Simulink models and their RTL implementation is required. In this paper, a formal equivalence checking methodology for Simulink and RTL is proposed.

A typical Simulink model comprises blocks from three different domains: Built-in Simulink blocks, Stateflow charts, and User-defined blocks. For a Simulink model comprising blocks of all those three types, our methodology utilizes Matlab's HDL Coder and RTW tools, Mentor Graphics' Catapult C, and Synopsys' Formality in the flow. To summarize the proposed methodology, HDL files for built-in Simulink blocks and Stateflow charts are generated by HDL Coder. Then, for user defined blocks, which are in S-functions in our case, C files are generated by RTW. Since HDL Coder is not able produce HDL files for those blocks, the generated C files are further arranged in a form to be fed to Catapult-C which generates the RTL files. Finally, the integration of the RTL files generated by both Simulink HDL Coder and Catapult-C forms the complete RTL implementation of the design. At this point, it may be questioned why automatically generated codes are not used as RTL counterpart, but metrics like performance, area and power are still better for manual implementation, so it is the preferred implementation. In the last stage, both the manual code and the automatically generated RTL designs are loaded into Formality for formal equivalence checking. Figure 1 illustrates our methodology, details of which will be explained in the following sections.

II. Methodology

A. Simulink Background

Simulink is a software for modeling, simulating, and analyzing real time systems [7]. Simulink models consist of a set of blocks that are connected by signals specifying the flow of data. Models can be structured hierarchically with the help of subsystems. Simulink provides predefined blocks, which are organized under several block libraries, like math operations and lookup tables.

In addition to predefined blocks, Simulink models can include state machines designed by Stateflow. Stateflow extends Simulink with a design environment for developing state charts and flow graphs. Furthermore, Simulink user defined blocks can be added to Simulink by means of Matlab functions or System-functions (S-functions). S-functions are written in C in a pre-defined format just like function calls used for built-in Simulink blocks. After compilation of those files to Matlab executable mex files, S-functions can be integrated to Simulink. This predefined integration process is automated by means of S-function builder functionality of Simulink, which generates files in the predefined format as C files.

Simulink provides two tools for automatic code generation: (i) HDL Coder generates portable, synthesizable VHDL or Verilog code from user defined functions, Simulink blocks, and Stateflow charts, (ii) RTW generates C and C++ code from Simulink diagrams, Stateflow charts, and user defined functions.

B. Building of the Methodology

In a typical digital IC Design flow, in which Simulink is utilized for system level design, it is a common practice that Simulink model and its RTL design counterpart are verified separately in their own domains. This simulation-based practice suffers from incompleteness of equivalence checking. Nonexistence of a methodology for formal equivalence checking of Simulink models is mostly due to the fact that Simulink lacks formal semantics. That has motivated several studies working on translating Simulink models to a semantically equivalent formal model. In [8], formal model is HyVisual, whereas in [9], it is Hybrid Automata, and in [10], it is Input/Output Extended Finite Automata.

For our specific case of digital IC design, counterparts of the system level models are mostly implemented in Verilog/VHDL. Those two languages are both standardized by IEEE, and they can constitute a suitable domain with formal semantics into which the Simulink level models can be translated. For our case, we opted for Verilog. As the formal equivalence checking part is considered, there already exist commercial formal equivalence checking tools like Formality and Conformal. Since Verilog is decided to be the common formal representation domain for both the system level and HDL level designs, a flow for translating a Simulink model into a Verilog implementation is needed first.

In order for a flow to be qualified for conversion from Simulink models into HDL, it should be able to handle all three types of blocks used in Simulink modeling. For a large number of built-in Simulink blocks referred in the distribution document and Stateflow charts, HDL Coder can generate HDL codes. For user-defined blocks, which are defined as S-function blocks, HDL Coder is not able to generate HDL. In order to convert those blocks into HDL, a different approach should be applied. RTW can be employed to generate C files from S-functions. Those C files are in the form of functions and the operation relaized by an S-function is described by the function called ”outputs_wrapper”. For the final phase of HDL generation from C files, Catapult C Synthesis tool is utilized [11]. As C files from RTW are input to Catapult C and outputs_wrapper function is set as top level, HDL files for S-functions are generated. By integration of these HDL code for the S-functions to the code generated by the HDL Coder, the complete Verilog representation for the Simulink model is obtained. The final phase of our methodology is to check for formal equivalence of automatically generated HDL code and its manually implemented counterpart. Although this phase could be completed by any available formal checker, in our case we deployed Formality [12]. A visual representation of our methodology is depicted in Figure 1.

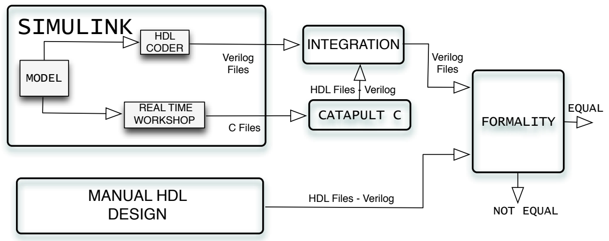

Fig. 1. Formal Equivalence Checking Methodology.

C. A Simple Simulink Example

In order to demonstrate the operation of the methodology, we use a simple design in Figure 2 including built-in Simulink blocks, a Stateflow chart and a user-defined S-function block as a case study. Add1 and UnitDelay1 blocks form the built-in

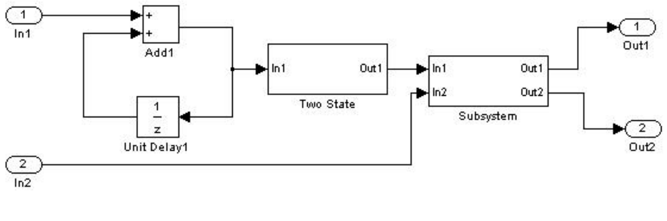

Fig. 2. Case study with built-in Simulink, Stateflow and S-function blocks.

Simulink block part of the design, realizing a counter, which counts in steps input by In1. Following the built-in blocks, Two State subsystem is a Stateflow chart shown in Figure 3. Stateflow charts are put under subsystems as a requirement of HDL Coder. The last block named Subsystem is a user-defined S-function, which outputs the difference of its inputs and the addition of inputs from Out1 and Out2, respectively. I/O's, their datatypes and functionality realized by the S-function are all entered by the S-function builder block of Simulink via GUI.

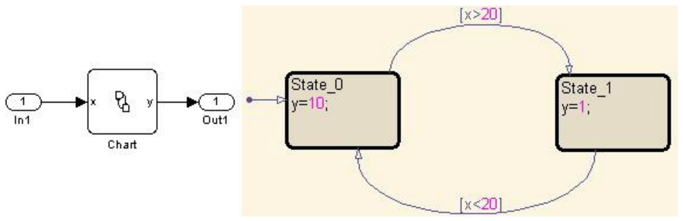

Fig. 3. Two State Subsystem and its state diagram.

An HDL Coder run on the design generates HDL for built-in Simulink blocks and Stateflow chart, but reports failure for the S-function block. Hence, as mentioned earlier, C files are generated for the S-function block by means of RTW and those files are input to Catapult C. Setting outputs_wrapper function as top level, Catapult generates the HDL files corresponding to the S-function. After the integration of the code for S-function, automatically generated HDL and manually written code are input to Formality, where formal equivalence is checked and successfully verified.

III. Experiments

We now demonstrate the applicability of our methodology on complex designs. These designs are Configurable Finite Impulse Response (FIR) and Least Mean Squares (LMS) filters, being distributed in the MATLAB package and AES, being a design of a well-known and published encryption standard.

1) Configurable FIR Filter: This design is a configurable pipelined symmetric 32 tap FIR filter. It is designed in a 4 stage pipelined structure. It has 4 adders/multipliers/accumulators in each stage and there are 16 coefficient addressable registers in the design. The top level of the Simulink model is as shown in Figure 4.

The model consists of two main parts: a finite state machine, which is modeled using Stateflow, and a datapath, which is implemented using generic Simulink blocks. The Stateflow part comprises 5 states and is implemented using functions and truth-table structures. The datapath is designed in pipeline form including 3 subsystems, pre_add (4 adders), mult (4 multipliers) and acc (4 adders/delay elements) in addition to 8 delay elements with 3 adders. The proposed methodology was followed and the formal equivalence of the manually written HDL code was verified after fixing the errors on the manual implementation side.

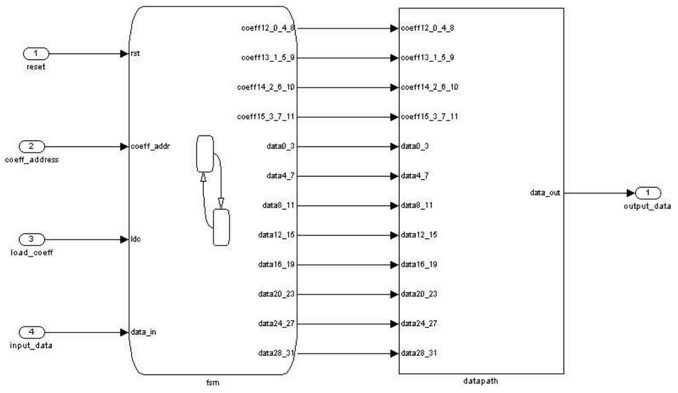

Fig. 4. The top level of Configurable FIR Filter Simulink Model.

2) LMS Filter: This design takes input signal, desired signal, Step_Size and Reset_Weights as inputs and outputs the error. Input signal is filtered and the difference between filtered signal and desired signal is fed back, as can be seen in Figure 5. Each of the 4 blocks on the figure comprises 10 LMS Taps and 10 adders. Each tap is designed using 2 multipliers, 2 delay elements, one switch and one adder.

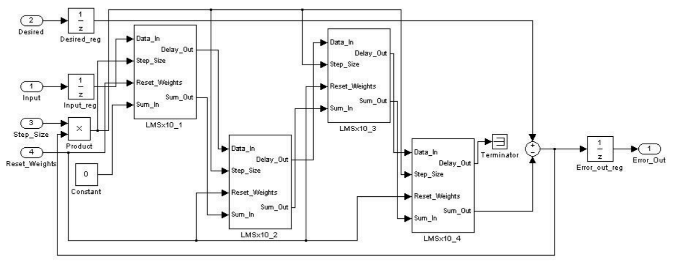

Fig. 5. The system model of LMS filter.

Following the methodology, the HDL code was generated and verified to be formally equivalent to the manually written one.

3) AES: AES is an encryption standard, specifications of which have been published by The National Institute of Standards and Technology (NIST) [13]. The AES cipher operates on 128 bits and uses cipher keys with 128, 192 or 256 bits in length. All the AES cipher transformations are performed on a two dimensional 4x4 array of bytes. AES cipher performs a set of four transformations (SubBytes, ShiftRows, MixColumns, and AddRoundKey) on the array for certain number of rounds depending on the cipher key length. For our case, we chose cipher key of 128 bits, which requires 10 rounds of transformation. First AddRoundKey transformation is applied, then in the first 9 rounds, SubBytes, ShiftRows, MixColumns, and AddRoundKey transformations are applied respectively. In the last round SubBytes, ShiftRows, and AddRoundKey transformations are applied.

The manual HDL implementation of AES was obtained from [14] by rewriting SystemVerilog implementation as Verilog. The hierarchy of the implementation is shown in Figure 6. In that design AES128_Key_Expand module, which generates the keys, is pipelined with the AES128_cipher_top module. While the AES128_cipher_top module performs an iteration of the encryption transformations on the array using the previously generated keys, the AES128_Key_Expand produces the next round's keys. AES128_Rcon calculates a parameter for AES128_Key_Expand for key generation process.

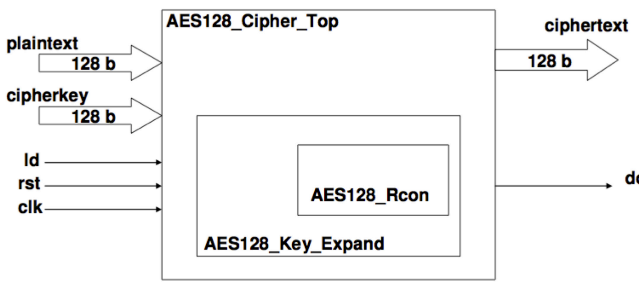

Fig. 6. The hierarchy of the manual implementation for AES 128 [14].

For the Simulink model design, built-in Simulink models and S-functions were used. MixColumn transformation, which is a matrix multiplication operation, was implemented utilizing S-function builder, since it can be expressed more easily in software language. The remaining parts of the model were realized using built-in Simulink blocks.

By following our proposed methodology, the HDL code for MixColumn transformation was generated by the application of RTW and Catapult C in succession. For Simulink design except MixColumn transformation, HDL codes were produced by means of HDL Coder. Then Formality confirmed the formal equivalence of the Simulink model and the manual implementation. It should be noted that by means of our methodology, we found that AES128_Rcon was not implemented correctly in our manual RTL, while we were running Formality.

Some metrics for the discussed experiments are given in Table 1. For HDL generation, we utilized a dual-core computer with 2.4 GHz clock speed, and 4 GB RAM. For equivalency check, we used an 8-CPU workstation with 16 GB RAM. As it can be seen from the table, our flow executes in a reasonable time. Considering the total durations needed for executing flow, which are the sums of HDL generation times and equivalency check times for each experiment, it can be said that those durations correlate with the number of registers. Furthermore, for each experiment, it is seen that the number of RTL lines generated automatically is greater than the number of RTL lines written manually, which is an expected result.

Table I. Some metrics for the experiments.

Parameter	Conf FIR	LMS Filter	AES
# registers	1155	1312	408
HDL gen time (sec)	13.1	36.1	27.5
# RTL lines (Auto)	1424	6611	12138
# RTL lines (Manual)	882	552	1103
Eqv check time (sec)	100.3	180.9	47.4

IV. Conclusion

Simulink is a widely used system level design platform in the IC Design industry. Traditional workflow in those projects is that the system is designed in Simulink and verified by Simulink simulations; then, RTL implementation is done and verified. In this workflow, a notion of formal equivalence between Simulink model and its HDL counterpart was missing. In this paper, we proposed a methodology addressing that problem. Our methodology was exercised by standardized, publicly distributed complex designs and proven to be working. As our methodology is dependent on off-the-shelf tools, we are limited by the capabilities of such tools. Our methodology can potentially be adoptedin IC Design Projects as we further investigate and improve our methodology.

As Simulink is also used in modeling Analog/Mixed Signal designs, extension of our methodology to check for equivalence of Analog/Mixed Signal models and their Verilog AMS counterpart will be our future work.

Acknowledgment

Muharrem Orkun Saglamdemir was partially supported by Ericsson Microelectronics Design Center. Alper Sen was supported in part by Marie Curie European Reintegration Grant within the 7th European Community Framework Programme, and the Turkish Academy of Sciences.

References

[1] D. Auger, *Programmable hardware systems using model-based design*, Proceedings of IET and Electronics Weekly Conference on Programmable Hardware Systems, London, 2008.
[2] P. Urard, A. Maalej, R. Guizzetti, N. Chawla, V. Krishnaswamy, *Leveraging sequential equivalence checking to enable system-level to RTL flows*, Proceedings of 45th ACM/IEEE Design Automation Conference, 2008.
[3] D. Markovic, C. Chang, B. Richards, H. So, B. Nikolic, R.W. Brodersen, *ASIC Design and Verification in an FPGA Environment*, Proceedings of IEEE Custom Integrated Circuits Conference, 2007.
[4] D. Araki, A. Nakamura, M. Miyama, *Model-based SoC design using ESL environment*, Proceedings of International System on Chip Design Conference, 2010.
[5] S.M. Aziz and S. Sharma, *New Methodologies for High Level Modeling and Synthesis of Low Density Parity Check Decoders*, Proceedings of 11th International Conference on Computer and Information Technology, Bangladesh, 2008.
[6] M. Ryabtsev, O. Strichman, *Translation validation: From Simulink to C*, Computer Aided Verication, Vol. 5643, Springer, 2009.
[7] http://www.mathworks.com/products/simulink/.
[8] R. Ray, *Automated Translation of MATLAB Simulink/Stateflow Models to an Intermediate Format in HyVisual*, Master of Science Project Report, Chennai Mathematical Institute, 2007.
[9] A. Agrawal, G. Simon and G. Karsai, *Semantic Translation of Simulink/Stateflow models to Hybrid Automata*, Electronic Notes in Theoretical Computer Science, Vol. 109, December, 2004.
[10] C. Zhou, R. Kumar, *Modeling Simulink Diagrams using Input/Output Extended Finite Automata*, Proceedings of 33rd Annual IEEE International Computer Software and Applications Conference, 2009.
[11] http://www.mentor.com/esl/catapult/overview.
[12] http://www.synopsys.com/tools/verification/formalequivalence/pages/formality.aspx.
[13] National Institute of Standards and Technology, Federal Information Processing Standards Publication 197, 2001.
[14] B. Hakhamaneshi, *A Hardware Implementation of the Advanced Encryption Standard Using SystemVerilog*, Master of Science Project Report, California State University Computer Engineering Department, 2009.