SystemC design flow for a DES/AES CryptoProcessor

J. CASTILLO, P. HUERTA, J. I. MARTINEZ
Grupo de diseño HW/SW, DIET-ESCET
Universidad Rey Juan Carlos
C\ Tulipan, Mostoles, Madrid
SPAIN

Abstract: - As the complexity of electronic systems increases every day, new ways for describing these systems also appear. One interesting way consists of capturing the whole system's functionality using a system level executable specification language. This high level specification is the entry point to a top-down design flow that results in the final implementation of the system. In this context synthesizable IP cores can be designed ad-hoc or obtained from third parties, with open source cores being an especially interesting option. A complete implementation of an AES/DES cryptoprocessor, from SystemC[1] specification to FPGA implementation - comparing an ad-hoc solution to one with open source IP cores - is fully described in this paper. Verification of the design in various levels using Transaction Level Modelling Style is presented. This methodology is extended in order to verificate the physical implementation presenting the concept of Physical Transactors.

Key-Words: - SystemC, DES, AES, Cryptography, Reusability, Open source.

1 Introduction

The goal of this paper is to describe the complete design flow of an AES/DES cryptoprocessor, from the System Level specification of the design to the final implementation on a prototype board. The verification methodology using Transaction Level Modelling style along all the design flow is also described. The design was done using SystemC 2.0, a modeling language based on C/C++.

In sections 1.1 and 1.2 the main characteristics of SystemC and a brief overview of AES/DES algorithms are presented. In section 2 the executable specification of the system is described, as well as a complete verification environment from the earliest stages of the design, using the SystemC Verification Library and the Transaction Level Modeling Style. Section 3 analyzes how to get from the system level specification to the synthesizable description. Section 4 describes the integration of Open Source cores in the design to compare with an ad-hoc solution. An estimation of the time/money saved by this approach is included. Section 5 describes the environment used to verify the system. Section 6 shows the synthesis flow. The physical implementation on a development board and the verification of this implementation using Physical Transactors concept is shown in section 7. Finally, the conclusions are presented in section 8.

1.1 SystemC 2.0 overview

Nowadays the common way to describe hardware is the use of HDLs such as VHDL and Verilog. However, software and system designers program in C/C++. Hardware/software codesign becomes a very hard task, due to the problems of using different languages at each abstraction level, or even in the same level (IP exchange). In this context, a single language that can be used in all the design stages is needed.

SystemC is a library of classes for C++ and a simulation kernel that provides all the features needed to describe a system in all its abstraction levels and a reference platform for IP exchange. SystemC also provides a specification called the SystemC Verification Standard (SCV)[2]. This library provides classes and methods to build a verification methodology called Transaction Model Style (TLM) based on the use of transactors. One of the most important features of the SCV is the constrained random generator that combined with self-checking testbenches allows to detect many corner cases in the design. This verification methodology can be used with designs made with other HDLs. Many EDA vendors provide tools that allow SystemC to be mixed with other languages.

This work has been supported by the Spanish PROFIT and Medea+ programme under FIT-0700000-2003-930 contract.

1.2 AES and DES overview

Cryptography is becoming a crucial part of modern electronic systems. Applications such as data secure transmission or user autentification are being implemented in many devices. Regarding this demand for encryption/decryption systems, many algorithms have been proposed. The two most used algorithms are DES/3DES[3] and its successor AES[4]. Both are Federal Information Processing Standard (FIPS) methods for symmetric encryption/decryption.

DES takes an input and a key of 64 bits length each and generates an encrypted data block of 64 bits. To decrypt the cypher message the same key must be used. 3DES is an extension of the DES algorithm where the data is encrypted three times instead of once, using different keys. The second stage of 3DES is a decryption, but with a different key, so the result is another cypher.

In January 1997, the National Institute of Standards and Technology (NIST) started a process to select an encryption algorithm for the AES standard. NIST stated that they were looking for an algorithm *"as secure as 3DES but much more efficient"*. In October 2002, NIST announced that Rijndael was the algorithm selected to become the AES. The AES takes an input of 128 bits and a key of 128, 196 or 256 bits and generates an output of encrypted 128 bits. The decryption is made with the same key used in the encryption process with a similar process.

2 System Level Description

One of the main bottlenecks in a design flow is the verification stage[5], thus it is also a problem in IP-based design[6]. It is estimated that at least 60% of the design effort is made in the verification stage. Traditional verification schemes also have the problem that the System Level Verification is the last stage of the design, extending the critical path and making architectural redesign almost impossible.

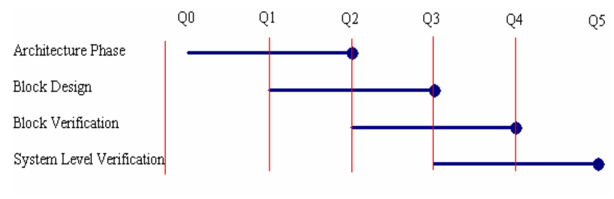

Fig. 1: Traditional design flow

New solutions to avoid this kind of problem are proposed. One of them is the Transaction Level Modeling Style described in the SystemC Verification Standard. With this verification methodology, the verification effort begins in parallel with the system level design.

The components made in the system level description can be reused in the block verification step and in the final verification step, where all the blocks are replaced with their synthesizable models. Also, new architectural optimizations can be evaluated with low effort.

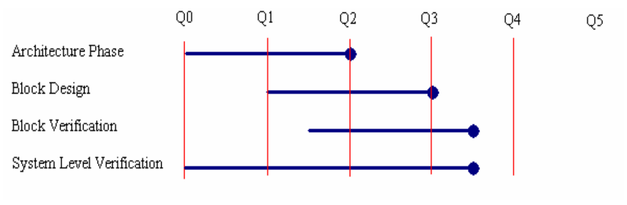

Fig. 2: TLM design flow

The model below is a behavioral description of the system, which means that no time and no information about the final implementation is included. It only reflects the required functionality of the whole system, and it will be used in later verification stages as the golden model for the designed blocks. This kind of model is called an "Untimed Functional Model" (UTF) for the reasons explained before.

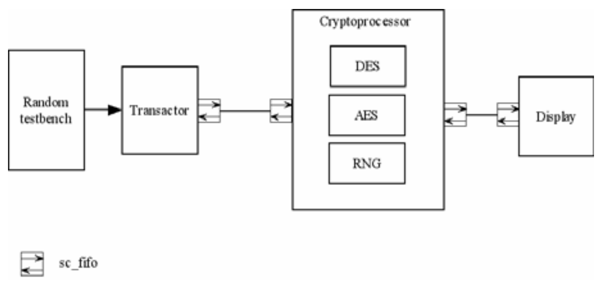

Fig. 3: Functional model

One of the most important parts of the verification methodology is the testbench generation. Using SCV features, a testbench that generates random keys and data for the AES/DES models was designed. The transactor takes the stimuli generated by the random testbench and applies them to the model. If the abstraction level of the model is changed, the same testbench can still be used by simply changing the transactor.

At this stage the cryptoproccesor model is a set of C++ functions with a SystemC wrapper and many sc_fifo channels to connect them to the testbenches and the display.

3 Refining the model

Before the module design phase can begin, it is necessary to go down in the abstraction level. In this level, information about the interfaces of the modules is added, as well as a clock. In this case no accurate time information is added to the model at this stage because no time specifications exist.

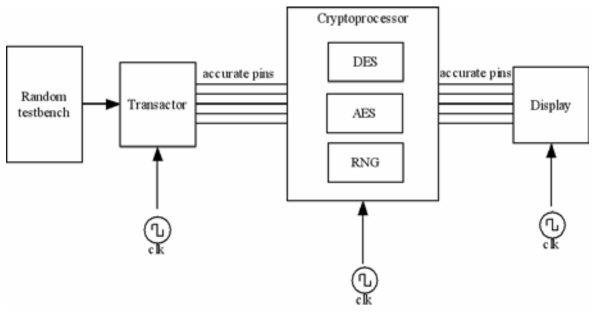

Fig. 4: Pin accurate model

The cryptoprocessor is described in four parts:

1 Bus interface
2 Controller of AES/DES module
3 Random number generator
4 DES encryption/decryption module
5 AES encryption/decryption module

The bus interface depends on the system bus. In this case, the bus is Wishbone[7] compatible. Wishbone is a bus with separate data and address lines with multiple masters and slaves whose specification is freely downloadable at www.opencores.org. Wishbone is also the standard bus for OpenCores[8] designs. OpenCores is an initiative for creating and distributing Open Source hardware designs via their webpage. In OpenCores, nearly a hundred downloadable designs from a complete RISC processor to communication controllers such as Ethernet or USB can be found.

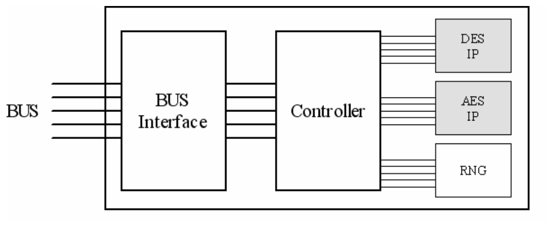

Fig. 5: Cryptoprocessor modules

The controller of AES/DES modules takes the configuration word of the cryptoprocessor and generates the signals to manage the AES/DES modules. It also takes the data and the keys from the data registers and applies them to the modules, writing back the cyphered block in the output registers.

The random number generator is based on the scheme[9] below, where an LFSR and a CASR in parallel are used to generate a random number generator with good statistical properties and a cycle length of 2^80. It is important to notice that the seed of the random number generator can be changed writing in the data register of the random generator.

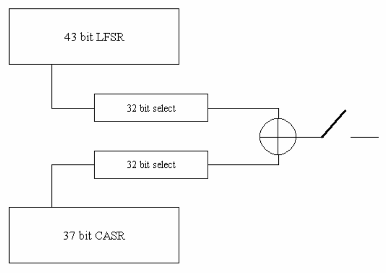

Fig. 6: RNG structure

When designing the AES and DES blocks we have the possibility of using:

1 Ad-hoc solution
2 Third party solution
   - Commercial IP
   - Open Source IP

The ad-hoc solution is expensive in time and money. However, reusability is one of the best ways to save time in the current design process. But even so, the main cost involved in reuse must be analyzed. There are three primary metrics[10] that can determine the magnitude of cost and saving via reuse: original development time, amount of design modification and verification effort.

4 Comparing solutions

As mentioned in the previous section, two approaches to the design were used: an ad-hoc solution and the use of Open Source IPs freely available on the Internet.

The first step in the design is writing the executable specification following the Transaction Level Modeling style. This specification took one week to be developed. For the ad-hoc solution, AES and DES encryptor/decryptor blocks were developed in SystemC following the FIPS standards. The goal for these modules was a very low area occupation and a low critical path, with no concern for throughput. With this objective, a multicycle architecture for both was selected. These modules could be useful for small embedded systems with low cryptographic requirements.

In the table below the time spent by one engineer working full time in the design can be seen. The development time includes the bus interface, the random generator and the controller written in SystemC. It also contains a week spent in documenting the DES and AES algorithms, and a week used in block and module verification.

Development time	Design modification	Verification effort
5 weeks	0 weeks	1 week

Table 1: Ad-hoc solution metrics

The other solution proposed was the use of Open Source cores. Both the AES and DES implementations were found on the OpenCores webpage. They are very similar to the ad-hoc ones developed, meaning a multicycle non-pipelined implementation focused on area constraints. The development effort in this case is spent on designing the bus interface, the controller and the random generator in Verilog and integrating the IPs. No modification to the IPs was needed.

Development time	Design modification	Verification effort
1 week	0 weeks	0.5 weeks

Table 2: Open Source solution metrics

Now a comparison between the two approaches can be made. The first metric is the original development time. It is important to notice that in the ad-hoc solution, knowledge of the system to be implemented is needed, where as in the Open Source based one, such knowledge is not needed. In this particular implementation this represents one week saved compared with the ad-hoc solution. Two weeks were spent on the design of the modules and one on module verification. Many of the problems arising from the use of Open Source designs come from poor verification methodology and poor documentation. This problem can appear depending on the source of the design. It is important to select this source carefully. OpenCores provides cores from some well-known IP companies that distribute some of their designs under an open license and can be used with the same guarantee as a commercial IP. In this case the cryptographic modules used are designed by asics.ws[11] a well-known IP company, so the functionality can be trusted. Finally, the design level verification step takes one week in the ad-hoc solution and only half a week in the Open Source one. In this step, the verification environment designed in the System Level specification stage is applied to the design. In the ad-hoc solution, some errors appeared in the cryptographic modules that had to be corrected. In the Open Source solution, only a few errors in the bus interface needed solving. In brief, 6 weeks are spent in the ad-hoc solution compared with 1.5 weeks in the Open Source one, which represents a saving of 75% of the time spent in the design process.

5 Verification environment

To guarantee the IP quality, a complete verification environment must be developed. Verification of the design must cover different levels, from the IP blocks created in the ad-hoc solution to whole system verification.
Three verification levels are proposed:

- Block level verification
- Module level verification
- System level verification

A block is a component of the system that must be verified before being integrated in a module. An example of block could be the key generation block of the DES and AES modules. In order to verify these blocks, classic signal-oriented testbenches were applied.

Another verification level is the module one. In this level the modules that compose the cryptoprocessor are verified using a classic testbench as in the block level, and also a random verification in the case of DES and AES blocks. This module random verification is very similar to the one used in the System level verification. In both cases the random testbench applies stimuli to the RT model to be verified and to the C code used as a golden model.

The outputs of both modules are passed to the checker that compares them. If a mismatch between the data is found, an error is reported and the simulation ends. The test was executed with several different seeds during long periods of time.

In the System Level case the testbench developed for the System Level specification is reused by simply changing the transactor functionality. At this level, the transactor applies the stimuli to the RT synthesizable design and to the C++ model of the cryptoprocessor used as a golden model.

The system level verification environment is presented in the figure below:

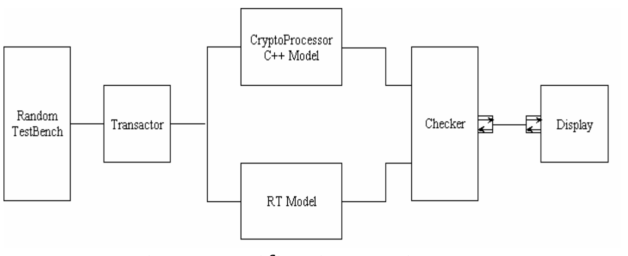

Fig. 7: Verification environment

6 Synthesizing SystemC

The last step in the design flow is the RTL synthesis.

No commercial synthesis tools support synthesizing SystemC, except Synopsys Cocentric[12] SystemC Compiler. But in fact this tool does not synthesize SystemC, it translates a SystemC description to a Verilog equivalent and then applies the Verilog synthesis flow to the design.
In order to resolve this problem, a SystemC to Verilog translator has been developed. The translator takes as input a synthesizable SystemC design and gives as output an equivalent Verilog one. The translator was written in ANSI C and uses Flex and Bison, a text scanning program generator and a parser generator respectively, distributed under the GNU GPL license. The SystemC to Verilog translator will be distributed under the same license.

The modules written in SystemC are translated to Verilog using this tool and then synthesized using FPGA vendor tools.

The results obtained for a Xilinx Virtex 800 FPGA are:

	LUTs used	Per. used	Freq. (MHz)
Ad-Hoc	799	4%	64.1
Open Source	1270	6%	97.5

Table 3: DES synthesis results

	LUTs used	Per. used	Freq. (MHz)
Ad-Hoc	687	3%	67.4
Open Source	719	3%	84.2

Table 4: AES synthesis results

In the ad-hoc solutions, the aim was to find a low area implementation, and the results obtained are better than the ones with the Open Source solutions. In the DES case, the ad-hoc solution is about 37% smaller, but on the other hand the Open Source solution is faster. In the AES implementation the same thing happens: the ad-hoc solution is smaller than the Open Source one but the second is faster. The conclusion is obvious, the ad-hoc solution fits our needs better than the reusable one, but at a cost that is not always feasible.

7 On board verification

The advantages of using a design methodology based in the use of Transaction Level Modelling Style were described earlier. One of the main advantages shown was that the verification environment could be reused in other stages by only changing the transactor functionality. Here, this concept is extended in order to verify the functionality of the physical implementation over a development board.

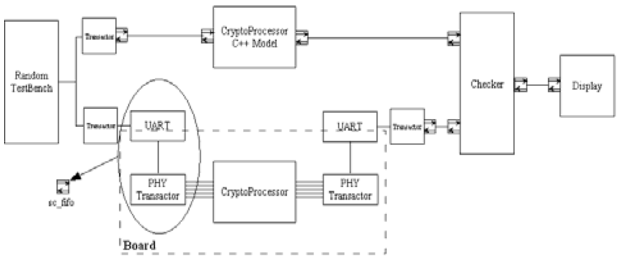

Fig. 8: On board verification

The Physical Transactor concept is the main fact introduced in this level of verification.

This kind of transactor converts the data from the UART on the board to the physical signals applied to the cryptoprocessor ports. Another Physical Transactor takes the outputs and send them back to the verification environment through the UART.

This transactor in combination with the UART works as a *sc_fifo* channel that blocks the simulation until a data from the board arrives. This kind of model is equivalent to an UTF model, where the sc_fifo channels connected to the physical implementation are exchanged by their equivalent models, made up of the UART and the physical transactor.

As shown in the System Level verification stage, the C++ model of the cryptoprocessor is connected in parallel with the design under verification. The outputs generated by the board are sent back to the verification environment via the UART on the board and compared with the ones generated by the C++ golden model by the checker.

The design was downloaded onto an XESS-800[13] development board, as shown below:

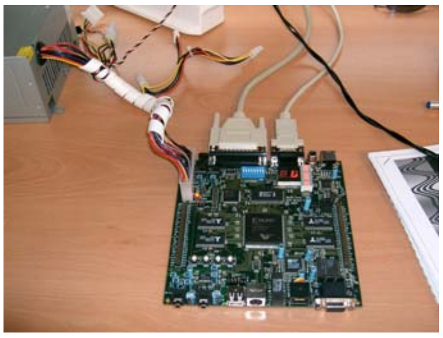
Fig. 9: XESS-800 development board

The verification environment was run and the results appeared in the host screen via the display module.

8 Conclusions

This paper presents a complete design from the System Level Specification in SystemC to the board implementation. SystemC, as a system level design was used to develop the system specification. The use of the TLM style, defined in the SystemC Verification Standard, allows the designer to begin the system verification in earlier design stages, saving a large amount of time and effort, and allowing low cost architectural exploration.

Two approaches were evaluated in order to implement the design: an ad-hoc solution and an Open Source one. The Open Source solution has many advantages compared with the ad-hoc solution: it is free, and it saves time and money. On the other hand the ad-hoc solution fits better with our specifications. However, before using the Open Source IP it is necessary to check that it is free of errors. Some IP providers (such as OpenCores) distribute designs sufficiently verified to trust them.

In order to synthesize the design, a translator from SystemC to Verilog was developed. The design was synthesized and implemented in a commercial development board successfully.

The concept of Physical Transactor, introduced in this paper, allows the designer to extend the TLM style down to the physical verification of the system, using the same verification environment designed from the beginning for the System verification.

References:

[1] OSCI, "*SystemC 2.0.1*",http://www.systemc.org
[2] OSCI, "*SystemC Verification Standard*", http://www.systemc.org
[3] FIPS, "*Data Encryption Standard*", Jan, 1977
[4] FIPS, "Avanced *Encryption Standard*", Nov, 2001
[5] *The International Technology Roadmap For Semiconductors*. 2001 Edition. http://public.itrs.net/Files/2001ITRS/home.htm
[6] R. Wilson, "*Design reuse expands across industry*", EETimes. March, 2003.
[7] OpenCores, "*WISHBONE System-on-Chip(Soc) Interconnection Architecture for Portable IP Cores*", http://www.opencores.org. Sep, 2002
[8] OpenCores, http://www.opencores.org
[9] T. Tkacik, "*A hardware random generator*", CHES 2002
[10] A. Dey, J. Moudy. *"Cost Saving via Reuse"*, Electronic Design Process Workshow(EDP), 2002
[11] asics.ws, http://www.asics.ws
[12] Cocentric System Studio, http://www.synopsys.com
[13] XESS Corporation, http://www.xess.com