Design and Verification of CoreConnect™ IP Using Esterel

Satnam Singh

Xilinx, 2100 Logic Drive, San Jose, California 95124
Satnam.Singh@xilinx.com

Abstract. This paper explores the practicality of describing and verifying both the hardware and software components of System-on-Chip (SOC) architectures using Esterel. We describe experiments to design and build working hardware based around IBM's CoreConnect™ Intellectual Property (IP) bus. The flow we analyse has been used to produce working hardware realized on Xilinx's FPGAs with soft 32-bit processors. Interesting properties about these systems have been proved by static analysis based on model checking.

1 Introduction

There has been a considerable interest in the ability to map a given function to either software or hardware in order to meet constraints for performance, area and time. Recently there has also been much interest in the use of assertions in hardware description languages to aid the verification process of complex systems. Conventional approaches for mapping a function to hardware or software typically involve performing totally separate software and hardware implementations (which is time consuming) or trying to derive one automatically from the other (which often produces poor results especially when one tries to infer efficient hardware from sequential software descriptions). The checking of assertions is often performed by a dynamic analysis i.e. simulation and this is known to have very poor coverage. This paper evaluates an approach which has some promising properties to help solve both of these problems.

We report experiments with the Esterel V7 programming language [7] which we have used for the synthesis of both hardware and software. We also report our experience of performing static analysis of hardware systems with properties expressed graphically as synchronous observers which are checked using an embedded model checker. It is our belief that such graphical descriptions of assertions may be more accessible to engineers than grammar based approaches.

2 Design and Property Specification in Esterel

Previous work in the area of using Esterel for generating efficient protocol code has already been reported as part of the HIPPCO project [4][5]. Much has been written about using Esterel for synthesizing software (specially C code) [3].

An Esterel description can be automatically synthesized either into reactive C software or into a variety of hardware descriptions languages (e.g. VHDL and Verilog) and then synthesized into hardware. The ability to synthesize either hardware or software from the same statically analysable description is one of the most appealing aspects of this technology. Although we do not present them here Esterel also contains many other language constructs for concurrent programming.

Our research investigation seeks to answer the following questions:

1. Can the intuitive graphical safe state machine notation be used effectively by engineers for specifying assertions which can be statically checked? Might this notation be more accessible than grammar based approaches?
2. Can Esterel descriptions be synthesized into efficient hardware and software (including a mixture of both) and work seamlessly with conventional vendor tool Assertions using Synchronous Observers?

Emerging techniques for specifying assertions typically involve using an extra language which has suitable operators for taking about time and logic relationships between signals. These languages are often concrete representations of formal logics and assertion languages are really temporal logics which can be statically analysed. Can the graphical safe state machine notation provide an alternative way of specifying properties about circuits which has the advantage of being cast in the same language as the specification notation? And can these circuit properties be statically analysed to formally prove properties about circuits?

To investigate these questions we performed an experiment which involved designing a peripheral for IBM's OPB bus which forms part of IBM's CoreConnect™ IP bus [6]. We chose the OPB bus because it is used by the MicroBlaze soft processor which is available on Xilinx's FPGAs.

An example of a common transaction on the OPB-bus is shown in Fig. 1. The key feature of the protocol that we will verify with an example is that a read or write transaction should be acknowledged within 16 clock ticks. Unless a control signal is asserted to allow for more time if a peripheral does not respond within 16 ticks then an error occurs on the bus and this can cause the system to crash. Not shown is the OPB_RNW signal which determines whether a transaction performs a read or a write.

Fig. 1. A sample OPB transaction

We considered the case of a memory mapped OPB slave peripheral which has two device registers that a master can write into and a third device register that a master can read from. The function performed by the peripheral is to simply add the contents of the two 'write' registers and make sure that the sum is communicated by the 'read' register. A safe state machine for such a peripheral is shown in Figure 2.

This generated VHDL for this peripheral was incorporated into Xilinx's Embedded Developer Kit and it was then used as a building block of a system which also included a soft MicroBlaze processor, an OPB system bus and various memory

resources and interfaces. We wrote test programs to check the operation of the peripheral with a 50MHz OPB system bus. The peripheral always produced the correct answer.

Fig. 2. An OPB-slave peripheral

Having successfully implemented an OPB peripheral from the Esterel specification we then attempted to prove an interesting property about this circuit. We choose to try and verify the property that this circuit will always emit an OPB transfer acknowledge signal two clock ticks after it gets either a read or write request. If we can statically prove this property we know that this peripheral can never be the cause of a transfer acknowledge timeout event.

We expressed this property as a regular Esterel safe state machine as shown in Figure 3. This synchronous observer tracks the signal emission behaviour in the implementation description and emits a signal if the system enters into a bad state i.e. a read or write request is not acknowledged in exactly two clock ticks.

One way to try and check this property is to try and use it in simulations to see if an error case can be found. Esterel Studio supports this by either simulation directly within the Esterel framework or by the automatic generation of VHDL implementation files and test benches which can check properties specified as synchronous observers.

However, the Esterel Studio system also incorporates a built-in model checker (Prover-SL from Prover Technology) which can be used to try and *prove* such properties. We use the latest V7 version of the Esterel language which allows us to reason about data as well as control which is an improvement from previous versions of the language. We configured the model check to see if the error signal corresponding to a bad state being entered is ever emitted i.e. might the circuit take

Fig. 3. An assertion expressed as a synchronous observer

longer than two clock ticks to acknowledge a transfer? It took Esterel Studio less than two seconds on a Sun Sparc Ultra-60 workstation to prove that this signal is never emitted.

```
esverify -v OPB.eid -checkis0 XFERACK_MISSING
--- esverify: Reading model from file "OPB.eid".
--- esverify: Checking if output "XFERACK_MISSING" is 0
--- esverify: Start model-checking properties
--- esverify: Verification complete for signal XFERACK_MISSING: --- esverify: --
--- esverify: Model-Checking results summary
--- esverify: --- esverify: Status of output "XFERACK_MISSING": Never emitted.
```

We then produced a deliberately broken version of the peripheral which did not acknowledge read requests. Within two seconds the software was able to prove that there is a case when the acknowledge signal is not asserted after a transaction and provided a counter-model and VCD file.

A conventional approach to catching such approach bugs involves either simulation (which has poor coverage) or the use of bus monitors which snoop the bus at execution time looking for protocol violations. A failure to acknowledge a transaction is one of the types of bugs that such systems can be configured to catch. However, it is far more desirable to catch such problems with a static analysis. We are currently trying to convert a list of around 20 such bug checks used in a commercial OPB bus monitor into a collection of Esterel synchronous observers to allow us to check peripheral protocol conformance with static analyses.

3 Future Work

Here we described our experience with just one system and method for realizing assertions which can be statically analysed. We have started the process of repeating these static property checks using Sugar (with IBM's FOCs system) and VERA with a view to writing a comparative study of the pros and cons of each approach. Sugar-based systems could be used indirectly in the Esterel flow by compiling synchronous observers into rules (which reside in a separate file from the VHDL design) and then

using the generated VHDL with vendor software for performing static and dynamic analyses of Sugar-based assertions. This would allow Esterel safe charts to act as graphical front end for some types of Sugar assertions.

For the verification of CoreConnect protocols we are developing Esterel models for the PLB (fast complex 64-bit system bus), OPB (a peripheral bus used as a 32-bit bus in Xilinx IP and used as the main bus for Xilinx's soft processor) and DCR (a simpler device control register bus) components which can then be used to help write synchronous observers for IP blocks without replicating the functionality of the system arbiters in each observer. Previous work [8] using the model checker Rulebase [1][2] for proving properties about CoreConnect arbiters suggests that this approach should be feasible.

To investigate how feasible it is to make hardware/software trade-offs using this flow we are developing implementations for network-on-chip protocols which are implemented in a combination of hardware and software. Using this flow can experiment with what portions need to be in hardware for performance and we may also be able to perform interesting static analyses of protocol behaviour, performance and correctness.

4 Conclusions

The approach of using Esterel to produce hardware and software seems to show some promise. Initial experiments show that serviceable hardware and software can be produced and implemented on real hardware and embedded processors. The possibility to enter system specifications graphically makes this method much more accessible to regular engineers than competing formalisms which uses languages which are quite different to what engineers are used to. For any realistic system the developer still has to write some portions textually and become aware of the basic underlying principles of Esterel. It remains to be seen if the cost of learning this formalism is repaid by increased productivity, better static analysis and the ability to trade off hardware and software implementations.

An appealing aspect of this flow is the ability to write assertions in the same language as the system specification. This means that engineers do not need to learn yet another language and logic. Furthermore, the formal nature of Esterel's semantics may help to make static analysis easier. Our initial experiments with using the integrated model checker are certainly encouraging. However, we need to design and verify more complex systems before we can come to a definitive conclusion about this promising technology for the design and verification of hardware and software from a single specification.

A very useful application of this technology would be to task-based dynamic reconfiguration. This method would avoid the need to duplicate implementation effort and it would also allow important properties of dynamic reconfiguration to be statically analysed to ensure that reconfiguration does not break working circuits.

There are some limitations to the technique we present here. There are some refinements that need to be made to the Esterel language to properly support hardware description. Most of these requirements are easily met without upsetting the core design of the language. Examples include a much more flexible way of converting between integers and bit-vectors and to allow arbitrary precision bit-vectors.

Currently performing an integer-based address decode for a 64-bit bus is possible in Esterel but one has to process the bus in chunks not larger than 31 bits.

“Virtex-II” is a trademark of Xilinx Inc. “CoreConnect” is a trademark of IBM. We would like to thank the staff at Esterel Technologies for their generous assistance during this project.

References

- [1] I. Beer, S. Ben-David, C. Eisner, D. Geist, L. Gluhovasky, T. Heyman, A. Landver, P. Paanah, Y. Rodeh, G. Ronin and Y. Wolfsthal. Rulebase: *Model Checking at IBM*. Conference on Computer Aided Verification. CAV’97. 1997.
- [2] I. Beer, S. Ben-David, C. Eisner and A. Landver. Rulebase: *An Industry-Oriented Formal Verification Tool*. The 33rd Design Automation Conference. 1996.
- [3] Gérard Berry. *The Foundations of Esterel*. Proof, Language and Interaction: Essays in Honour of Robin Milner, G. Plotkin, C. Stirling and M. Tofte, editors, MIT Press, 1998.
- [4] C. Castelluccia and W. Dabbous. *Modular Communication Subsystem Implementation using a Synchronous Approach*. Proceedings of the Usenix Symposium on High-Speed Networking, Oakland, California, U.S.A., August 1994.
- [5] C. Castelluccia, Walid Dabbous and Sean O’Malley. *Generating Efficient Protocol Code from an Abstract Specification*. ACM SIGCOMM, Stanford 96.
- [6] IBM, “The CoreConnect™ Bus Architecture”, http://www.chips.ibm.com/product/coreconnect/docsrcon_wp.pdf, 1999.
- [7] Gérard Berry and Ellen Sentovich. *Multiclock Esterel*. Correct Hardware Design and Verification Methods. CHARME 2001.
- [8] Amit Goel and William R. Lee. *Formal Verification of an IBM CoreConnect Processor™. Local Bus Arbiter Core*. Design Automation Conference. 2000.