Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

A uniform ADL for embedded processor

(2013)

Abstract

instructions with lower operand widths. The reusage of opcodes requires only one additional distinguishing bit. This process is repeated to obtain variable length opcodes for all the instructions.

In the context of LISA, the instructions are arranged hierarchically in the form of LISA operations. The opcode synthesis algorithm is applied to different hierarchies to obtain the local group's opcode. However, it is observed that the local encoding is suboptimal in comparison with the global context. On the other hand, applying the synthesis algorithm in a global context can produce better results, but is confronted with the issue of high runtime complexity. This issue is dealt with by identifying (manually or heuristically) a set of key operations, which unambiguously represents a single instruction of the processor. After identifying all key operations in the whole architecture description, the operand lengths are determined by accumulating all operand lengths of the associated nonterminal operations on the path to the key operation. Subsequently, the set of key operations are encoded.

5.3 AUTOMATIC OPTIMIZED PROCESSOR IMPLEMENTATION

A processor description in LISA can be subjected to an HDL generator, which is capable of producing optimized RTL implementation of the target processor. Unlike traditional processor modeling, where the software toolsuite and the RTL implementation were developed separately, this approach provides important advantages. First, the single LISA specification eliminates the problem of discrepancy in different processor tools and suboptimal processor performance because of that. Second, the quick generation of RTL description allows a more accurate performance estimation in the early design space exploration phase. Third, for manually written RTL descriptions, this automatically generated RTL description can serve as a reference description for verification. Finally, the RTL description can be further subjected to commercial physical synthesis flow to manufacture the processor. To qualify the automatically generated RTL for all these, an important prerequisite is that it should be highly optimized in terms of area-timing and power. The HDL generator tool of LISA utilizes high-level structural information, embedded in the LISA description, to make the quality of RTL description equivalent or even better than a manually described RTL model. In the following subsections, the central principle of LISA-based HDL generation and its inherent optimization techniques are discussed.

5.3.1 Automatic Generation of RTL Processor Description

The HDL generation process from LISA [5, 15] is graphically depicted in Fig. 5.9. The LISA description is initially parsed and represented internally in a representation termed as Unified Description Layer (UDL). In the UDL, the structural as well as behavioral details of the processor are stored in an easy-to-use format. At

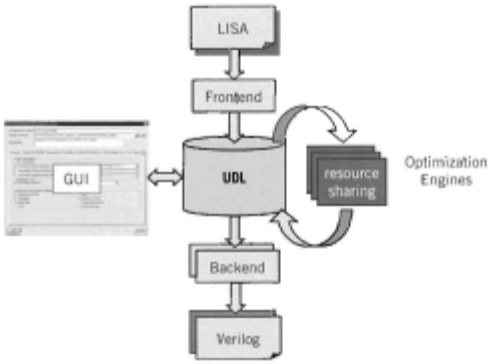

FIGURE 5.9

Automatic RTL description generation from LISA.

this level, a GUI-based user interface allows the designer to navigate the synthesis flow. The designer may opt for various optimization engines, as well as set up definite optimization constraints. The user may also generate gate-level synthesis and RTL simulation scripts tuned to commercial tools. Depending on the user options, the optimization engines are triggered. From the optimized UDL representation, language-specific back-ends are called to generate the RTL description in synthesizable form of VHDL, Verilog, or SystemC. The entire HDL generation process is modular and extensible. For example, new optimization engines or new language back-ends can be conveniently plugged in.

5.3.2 Area and Timing-driven Optimizations

In order to generate an RTL description for the purpose of implementation, it is imperative to match the quality of it to the manually optimized implementation. A definite step for that is to plug in optimizations driven by area and timing. Though the commercial gate-level synthesis tools perform such optimizations, several key structural information available in the high-level LISA description (and preserved in the UDL) are difficult to determine from the RTL description. Furthermore, given the complexity of a modern embedded processor, manually performing optimizations all over the design is a tedious job. The area and timing-driven optimizations performed during LISA-based HDL generations are proposed to exactly solve this problem. Surely, an experienced designer can reap further benefits of this model by doing manual RTL modifications.

A key information preserved in the UDL format for optimizations is the exclusiveness information available in the LISA description. In LISA, the processor instructions are hierarchically arranged in groups of *LISA operations*, among which a detailed exclusiveness relation exists. These exclusiveness information is stored in the UDL. The datapaths of the basic behavioral blocks (corresponding to each LISA operation) are also stored in the UDL in Control Data Flow Graph (CDFG) and Data Flow Graph (DFG) format. Based on these information, the following major optimizations are applied:

- *Structural simplifications*: LISA designer usually models the processor without delving into structural fine-tuning. This leaves a scope of several structural simplifications [16]. One such example is *decision minimization*. This is essentially a variant of condition-independent code motion. By decision minimization, condition-independent hardware operations are moved outside the conditional blocks. Another structural simplification is called *signal scope localization*. In this optimization, the signals declared all over the LISA model are examined and if a signal is accessed within the scope of a single simulation cycle without the necessity of storing its value, it is converted to a local variable. The same optimization can also be performed for registers, where instead of grouping all the register files together in a dedicated hardware block, locally used register resources are moved close to the corresponding functional units.
- *Path sharing*: To allow ease of programming at the high-level abstraction of LISA, register resources can be read and written multiple times from a single LISA operation. This produces multiple register access paths, which in turn results in huge multiplexers in the register file. By analyzing the exclusiveness of all the basic blocks in the processor, the read and write paths to the same register resources are shared.
- *Resource sharing*: In a LISA model, costly hardware operators like multiplication and addition can be extensively used all over. This grants an ease-of-programming in high-level abstraction at the expense of heavy area consumption. The designer can turn on *resource sharing* optimization to share the resources among exclusive behavioral blocks. The sharing is performed iteratively. At each step of sharing, a set of shareable resources is formed and the effect of sharing on the neighboring logic is estimated. The estimation is done on the basis of an abstract cost model. Depending on the estimation and the designer-specified area, and also timing constraints, the resource sharing is performed with little effect on the critical path.

5.3.3 Energy-driven Optimizations

Due to the stagnant battery capacity, the power budget of modern embedded processors is tight. This makes the energy-driven optimization an important part of processor implementation flow. In the HDL generation from LISA, several energy-driven optimizations are available, as discussed here.

- *Clock gating*: To reduce the dynamic power of a processor, it is important to gate the storage elements when no new data is being read. Due to its overwhelming effect on overall power consumption, commercial gate-level synthesis tools are currently supporting automatic clock gating. However, to determine the gating signal, that is, the signal to indicate when a new data is not available, is an important research problem. In the perspective of LISA, it is identified by tracing the register write operations to LISA operations [17]. The clock gating for registers as well as pipeline registers is performed by inserting dedicated gating blocks during HDL generation.
- *Operand isolation*: In operand isolation, the redundant switching activity in the functional blocks of a circuit is reduced by blocking the inputs during idle state. Similar to the clock gating, this requires an understanding of the processor's data flow and determination of the idleness condition on the basis of *Observability Don't Care* of the functional blocks' outputs. Given the access to the structural organization of the LISA processor's functional blocks, this is obtained in a straightforward manner [7].

Apart from these energy-driven optimizations during HDL generation, energy can be saved by minimizing the consumption of power in the instruction and data buses in a processor. In the context of LISA, this is also performed by altering the instruction opcode and modifying the register allocation [18].

5.3.4 Automatic Generation of JTAG Interface and Debug Mechanism

The LISA-based HDL generation framework allows automatic generation of processor features such as hardware debug mechanism [19]. This feature is important for the designers willing to access and modify the processor state via an additional interface, commonly referred to as JTAG interface. Such an interface and debug mechanism enables the designer to debug the processor after fabrication. Implementing this feature manually in RTL is clearly a prohibitive task with the trade-off between debugging capability and performance overhead, for example, area and timing, caused by this. In LISA, the JTAG interface is debug mechanism settings can be performed via the GUI during HDL generation. The area impact of various choices is obtained immediately. With this, the trade-off between debugging flexibility and performance can be done early in the design phase. Depending on the configuration, the complete interface and resources necessary for debug mechanism are automatically instantiated in the generated RTL.

5.4 PROCESSOR VERIFICATION

Validation and verification are the two essential phases of a processor design. While validation ensures that the design-intent is correctly reflected in the design specification, the task of verification is to check that the design matches with the

specification. In the context of application-specific processor design, both these include the check if the target application(s) are correctly executed. In the LISA-based flow, this can be done across multiple abstraction levels down to RTL via simulation.

5.4.1 Equivalence Check via Simulation

Simulation of target application(s) and corresponding equivalence check across various levels of LISA abstraction is graphically displayed in Fig. 5.10. Using an instruction-accurate LISA model, a designer can easily verify the correct application execution using the LISA simulation capabilities. The simulator is aided with wide-ranging debugging features, for example, memory tracing, LISA-level behavior stepping, and setting breakpoints. With the high-level LISA description acting as a golden model, the processor states (registers, memory, status flags) can be dumped at the end of each cycle in a widely used Value Change Dump (VCD) format. The VCD file can be regenerated by running the application with a changed LISA model. Obviously, while executing the application for a cycle-accurate LISA model, adjustments in the application are necessary compared to the one running for an instruction-accurate LISA model. The VCD file from the original golden model and the new LISA model can be directly compared via perl script to locate mismatches. Further down the implementation track, the RTL implementation of the processor can be subjected to the same application. The VCD file generated from RTL simulation can again be compared with the VCD file created during LISA simulation.

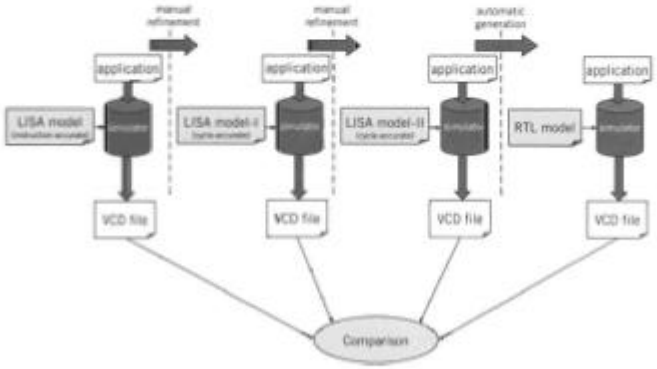

FIGURE 5.10
LISA-based verification via simulation.

5.4.2 Generation of Test Vectors from LISA Descriptions for Instruction-set Verification

Considering a broader perspective, the designed processor needs to be tested for applications yet unknown. This can be guaranteed by formally verifying that each individual instruction performs as expected for all possible processor states. Given the enormity of this verification space, current formal verification tools fall short of this task. Consequently, major ADLs focus on uncovering hidden bugs in the design by verifying the design with various test vectors. The test vectors are designed with two definite goals: first, to improve the coverage of the design, that is, to test as many likely scenarios as possible. Second, to feed the design with corner-case test vectors. In both the cases, the expertize of designer improves the test vector generation by reducing the redundancy, thereby saving costly verification time. Therefore, the LISA-based test vector generation focuses on a semiautomatic approach, where the designer/verification engineer can guide the process of test vector generation from LISA. In the following paragraphs, two different test vector generation approaches based on LISA are presented. While the first one is more focussed on creating corner-case test suite, the second approach is motivated toward achieving full coverage test vectors.

Integration with Genesys

The test generation environment from IBM, termed Genesys, is integrated with LISA to derive intelligent test cases. Genesys has a wide range of functionalities to target the verification of cache protocols, multi-processor configurations, etc. Though it was originally conceived to deliver test cases for large-scale processors [20], Genesys has been successfully applied to the verification of DSPs and ASIPs [21].

The building blocks of Genesys test generation environment are shown in Fig. 5.11. The architecture-specific details, for example, the testing knowledge, simulator, and instruction-set are fed into the test generator. The simulator is automatically generated from a LISA description. Besides the regular simulation

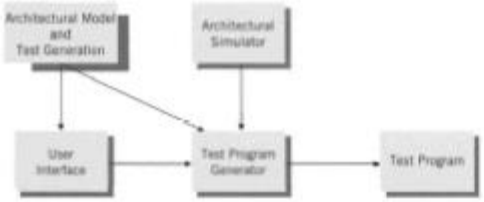

FIGURE 5.11

Genesys test generation environment.

REFERENCES

[1] J. Campbell, "Speaker recognition: a tutorial," *Proc. IEEE*, vol. 85, pp. 1437–1462, Sept. 1997.

[2] D. A. Reynolds, T. Quatieri, and R. Dunn, "Speaker verification using adapted Gaussian mixture models," *Digital Signal Processing*, vol. 10, no. 1–3, pp. 19–41, 2000.

[3] D. A. Reynolds, "Comparison of background normalization methods for text-independent speaker verification," in *Proc. Eurospeech*, 1997.

[4] D. A. Reynolds and R. C. Rose, "Robust text-independent speaker identification using Gaussian mixture speaker models," *IEEE Trans. Speech Audio Processing*, vol. 3, no. 1, pp. 72–83, 1995.

[5] J. L. Gauvain and C.-H. Lee, "Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains," *IEEE Trans. Speech Audio Processing*, vol. 2, pp. 291–298, Apr. 1994.

[6] E. Bocchieri, "Vector quantization for the efficient computation of continuous density likelihoods," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1993, pp. 692–695.

[7] K. M. Knill, M. J. F. Gales, and S. J. Young, "Use of Gaussian selection in large vocabulary continuous speech recognition using HMMs," in *Proc. Int. Conf. Spoken Language Processing*, 1996.

[8] D. B. Paul, "An investigation of Gaussian shortlists," in *Proc. Automatic Speech Recognition and Understanding Workshop*, 1999.

[9] T. Watanabe, K. Shinoda, K. Takagi, and K.-I. Iso, "High speed speech recognition using tree-structured probability density function," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1995.

[10] J. Simonin, L. Delphin-Poulat, and G. Damnati, "Gaussian density tree structure in a multi-Gaussian HMM-based speech recognition system,"

[11] T. J. Hanzen and A. K. Halberstadt, "Using aggregation to improve the performance of mixture Gaussian acoustic models," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1998.

[12] M. Padmanabhan, L. R. ahl, and D. Nahamoo, "Partitioning the feature space of a classifier with linear hyperplanes," *IEEE Trans. Speech Audio Processing*, vol. 7, no. 3, pp. 282–288, 1999.
in *Proc. Int. Conf. Spoken Language Processing*, 1998.

[13] R. Auckenthaler and J. Mason, "Gaussian selection applied to text-independent speaker verification," in *Proc. A Speaker Odyssey—Speaker Recognition Workshop*, 2001.

[14] J. McLaughlin, D. Reynolds, and T. Gleason, "A study of computation speed-ups of the GMM-UBM speaker recognition system," in *Proc. Eurospeech*, 1999.

[15] S. van Vuuren and H. Hermansky, "On the importance of components of the modulation spectrum of speaker verification," in *Proc. Int. Conf. Spoken Language Processing*, 1998.

[16] B. L. Pellom and J. H. L. Hansen, "An efficient scoring algorithm for Gaussian mixture model based speaker identification," *IEEE Signal Processing Lett.*, vol. 5, no. 11, pp. 281–284, 1998.

[17] J. Oglesby and J. S. Mason, "Optimization of neural models for speaker identification," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1990, pp. 261–264.

[18] Y. Bengio, R. De Mori, G. Flammia, and R. Kompe, "Global optimization of a neural network—hidden Markov model hybrid," *IEEE Trans. Neural Networks*, vol. 3, no. 2, pp. 252–259, 1992.

[19] H. Bourlard and C. J. Wellekins, "Links between Markov models and multilayer perceptrons," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, pp. 1167–1178, Dec. 1990.

[20] J. Navrátil, U. V. Chaudhari, and G. N. Ramaswamy, "Speaker verification using target and background dependent linear transforms and multi-system fusion," in *Proc. Eurospeech*, 2001.

[21] L. P. Heck, Y. Konig, M. K. Sonmez, and M. Weintraub, "Robustness to telephone handset distortion in speaker recognition by discriminative feature design," *Speech Commun.*, vol. 31, pp. 181–192, 2000.

[22] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. R. Statist. Soc.*, vol. 39, pp. 1–38, 1977.

[23] K. Shinoda and C. H. Lee, "A structural Bayes approach to speaker adaptation," *IEEE Trans. Speech Audio Processing*, vol. 9, no. 3, pp. 276–287, 2001.

[24] K. Fukunaga, *Introduction to Statistical Pattern Recognition.* New York: Academic, 1990.

[25] J. C. Junqua, *Robust Speech Recogntion in Embedded Systems and PC*

[26] U. V. Chaudhari, J. Navrátil, S. H. Maes, and R. A. Gopinath, "Transformation enhanced multi-grained modeling for text-independent speaker recognition," in *Proc. Int. Conf. Spoken Language Processing*, 2000.

[27] Q. Lin, E.-E. Jan, C. W. Che, D.-S. Yuk, and J. Flanagan, "Selective use of the speech spectrum and a VQGMM method for speaker identification," in *Proc. Int. Conf. Spoken Language Processing*, 1996.

[28] S. Raudys, *Statistical and Neural Classifiers: An Integrated Approach to Design.* New York: Springer, 2001.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986, pp. 318–364.

[30] [Online] Available: http://www.nist.gov/speech/tests/spk/index.htm.

[31] J. Pelecanos and S. Sridharan, "Feature warping for robust speaker verification," in *Proc. A Speaker Odyssey—Speaker Recognition Workshop*, 2001.

[32] B. Xiang, U. V. Chaudhari, J. Navrátil, N. Ramaswamy, and R. A. Gopinath, "Short-time Gaussianization for robust speaker verification," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 2002.

[33] G. R. Doddington, M. A. Przybocki, A. F. Martin, and D. A. Reynolds, "The NIST speaker recognition evaluation—overview, methodology, systems, results, perspective," *Speech Communication*, vol. 31, pp. 225–254, 2000.

Bing Xiang (M'03) was born in 1973 in China. He received the B.S. degree in radio and electronics and M.E. degree in signal and information processing from Peking University in 1995 and 1998, respectively. In January, 2003, he received the Ph.D. degree in electrical engineering from Cornell University, Ithaca, NY.

From 1995 to 1998, he worked on speaker recognition and auditory modeling in National Laboratory on Machine Perception, Peking University. Then he entered Cornell University and worked on speaker recognition and speech recognition in DISCOVER Lab as a Research Assistant. He also worked in the Human Language Technology Department of IBM Thomas J. Watson Research Center as a summer intern in both 2000 and 2001. He was a selected remote member of the SuperSID Group in the 2002 Johns Hopkins CLSP summer workshop in which he worked on speaker verification with high-lelvel information. In January, 2003, he joined the Speech and Language Processing Department of BBN Technologies where he is presently a Senior Staff Consultant-Technology. His research interests include large vocabulary speech recognition, speaker recognition, speech synthesis, keyword spotting, neural networks and statistical pattern recognition.

Toby Berger (S'60–M'66–SM'74–F'78) was born in New York, NY, on September 4, 1940. He received the B.E. degree in electrical engineering from Yale University, New Haven, CT in 1962, and the M.S. and Ph.D. degrees in applied mathematics from Harvard University, Cambridge, MA in 1964 and 1966, respectively.

From 1962 to 1968 he was a Senior Scientist at Raytheon Company, Wayland, MA, specializing in communication theory, information theory, and coherent signal processing. In 1968 he joined the faculty of Cornell University, Ithaca, NY where he is presently the Irwin and Joan Jacobs Professor of Engineering. His research interests include information theory, random fields, communication networks, wireless communications, video compression, voice and signature compression and verification, neuroinformation theory, quantum information theory, and coherent signal processing. He is the author/co-author of Rate Distortion Theory: A Mathematical Basis for Data Compression, Digital Compression for Multimedia: Principles and Standards, and Information Measures for Discrete Random Fields.

Dr. Berger has served as editor-in-chief of the IEEE TRANSACTIONS ON INFORMATION THEORY and as president of the IEEE Information Theory