View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Loughborough University Institutional Repository

This item was submitted to Loughborough's Institutional Repository (https://dspace.lboro.ac.uk/) by the author and is made available under the following Creative Commons Licence conditions.

creative commons

COMMONS DEED

Attribution-NonCommercial-NoDerivs 2.5

You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.

No Derivative Works. You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Legal Code (the full license).

Disclaimer

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/

A System-on-Chip Vector Multiprocessor for Transmission Line Modelling acceleration

Vassilios A. Chouliaras, James A. Flint, Yibin Li
Department of Electronic and Electrical Engineering
Loughborough University
Loughborough, Leicestershire, UK
{v.a.chouliaras,J.A.Flint,Y.Li2}@lboro.ac.uk

Jose L. Nunez-Yanez
Department of Electronic Engineering
University of Bristol
Bristol, UK
J.L.Nunez-yanez@bristol.ac.uk

Abstract— **We discuss a configurable, System-on-Chip vector multiprocessor for accelerating the Transmission Line Modeling (TLM) algorithm with an architecture capable of exploiting the two primary forms of parallelism in the code, thread and data level parallelism. Theoretical results demonstrate an order of magnitude reduction in the dynamic instruction count for a scalar-processor/vector-coprocessor configuration at a vector length of sixteen 32-bit single-precision elements. Furthermore, a multi-vector SoC architecture consisting of ten such vector accelerators provides a near-linear theoretical performance benefit of the order of 88% in three out of four benchmark configurations which is orthogonal to the benefit realized by vectorization alone. We discuss in detail this potent architecture and present implementation data for the 2-way multi-processor VLSI macrocell.**

I. INTRODUCTION

Prior attempts to parallelizing the TLM algorithm [1] on general-purpose programmable architectures targeted either shared memory, cache coherent multi-processors [2, 3] or distributed processors [4] with shared- memory machines typically demonstrating better performance. In addition, custom architectures for accelerating TLM codes have been proposed in the past by Stothard and Pomeroy [5].

The TLM is a highly-parallel three-dimensional numerical algorithm (kernel) which has the potential for being accelerated along its innermost loop, via vectorization thus tapping parallelism at the data level (DLP). Furthermore, the algorithm can be statically 'sliced' (threaded) along the second outer loop, and be executed on the previously mentioned platforms via different processors executing different iterations. Such parallelism is known as thread-level-parallelism (TLP) [6] and is currently being pursued by all major microprocessor vendors.

Successful acceleration of such parallel codes depends very much on the algorithmic communication pattern which dictates the level of data sharing across the multiple processors. In the case of the TLM, data transfer between individual nodes is very high and in extreme cases the data transfer during the *connect* part of the algorithm can be much more expensive, in terms of CPU cycles, than the numerical calculations during *scattering*. The performance differential between shared memory and distributed machines is often attributed to such data sharing issues.

The contribution of this work in the area of thread and data parallel TLM codes is five-fold: a) a three-dimensional transmission line modeling software kernel (3D-TLM) was developed in vector form, statically threaded (with shared-memory semantics) and it's data and thread parallel performance evaluated for a number of mesh configurations on a proprietary exclusive-read, exclusive-write parallel RAM; b) The architectural model of an open-source CPU [7] was extended with a context ID register, to allow for the identification of the CPU context a software application thread executes on. In addition, a novel, lightweight, hardware-based synchronization mechanism was introduced to dispense with the need for atomic-instruction-based synchronization primitives; c) the single-processor open-source system was extended to allow for multiple such modified processors to be instantiated in a bus-based, symmetric, cache-coherent configuration; d) The basecase floating-point functionality of an open-source FPU [8] was extracted and encapsulated in a custom, RISC-like, vector coprocessor wrapper. That coprocessor was subsequently introduced, in a tightly-coupled configuration, within each of the modified processors in the multiprocessor system resulting in a SoC-based multi-vector architecture capable of exploiting, per processor, the DLP of the 3D TLM code and across the multiple-processors the TLP of that workload, quite unlike earlier work; e) finally, the architecture and microarchitecture of the developed hardware platform are highly parameterized via compile-time constants enabling the system to be utilized not only in the case of the 3D TLM code but on applications requiring strong floating-point performance such as molecular simulations and 3D geometry processing.

II. BENCHMARK

We have utilized a basic implementation of the SCN TLM algorithm [1] in which no external boundary conditions were used. In the particular case, a single output node was used as a diagnostic aid to verify correct operation. We used the accelerated scatter method of Naylor and Ait-Sadi as proposed in [9].

The non-vectorized (scalar) algorithm was profiled both in native mode (X86 Linux) as well as on our simulated processor for consistency of results. Scalar code profiling revealed a *scatter:connect* dynamic instruction count ratio of 63:37, averaging over all the studied configurations.

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on May 05,2010 at 15:41:06 UTC from IEEE Xplore. Restrictions apply.

Our simulation infrastructure is based around the simplescalar toolset [10] which provides a complete computer architecture modeling and performance evaluation environment. The compiler used was GCC 2.7.3 with optimizations (-O2). The multi-threaded results were collected on a proprietary Exclusive-Read, Exclusive-Write (EREW) Parallel-RAM (PRAM) simulator, originally based on the simplescalar Instruction Set Simulator. Sim-system as it is known, was developed from the simplescalar code base and implements a parametric EREW PRAM machine with parameters being the number of CPU contexts that participate in a given simulation run. Sim-system produces dynamic execution traces, per CPU context, which are used to drive a currently cycle-accurate back end which models arbitrary, shared-memory, multi-context architectures including multi-threaded processors, multiprocessors or multithreaded multiprocessors, as well as arbitrary interconnect. The simulation infrastructure is elaborated in [11].

III. VECTOR COPROCESSOR PROGRAMMERS MODEL

The programmer's model specifies a parametric number of vector registers (VRMAX), each consisting of a parametric number of 32-bit IEEE 754 single-precision elements (VLMAX).

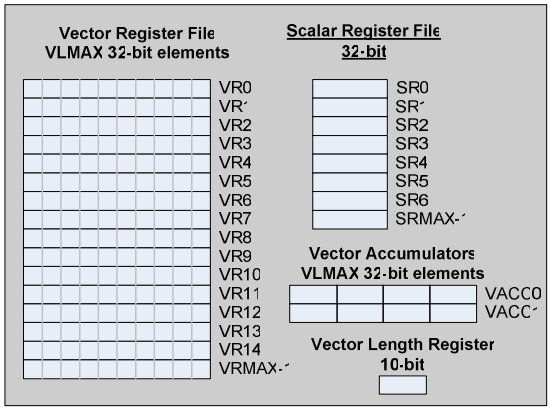

Figure 1. Vector Coprocessor programmer's model

TABLE I. TABLE 1 : VECTOR ACCELERATOR ISA

Instruction	Description
MVSR2VLEN	Transfer scalar register to vector length register (VLEN)
MVSR2CSR	Transfer RISC scalar register to coprocessor scalar register
MVCSR2R	Transfer coprocessor scalar register to RISC register
MVSR2CVEL	Move RISC scalar register to coprocessor vector element
MVCVEL2R	Move coprocessor vector element to RISC scalar register
VLDU	Load vector register unaligned under VLEN
VSTU	Store vector register unaligned under VLEN
VPERM	Three-operand byte-wise vector permute
VSPLAT	Splat coprocessor scalar register to coprocessor vector register
VFPADD.S	Vector floating-point add (single precision) under VLEN
VFPSUB.S	Vector floating-point sub (single precision) under VLEN
VFPMUL.S	Vector floating-point mult (single precision) under VLEN
VFPMAC.S	Vector floating-point multiply-accumulate

There is a scalar register file consisting of a parametric number of scalar 32-bit elements (SRMAX), used for virtual address computation, immediate passing and vector splat operations. Additionally, there are two vector accumulators each holding VLMAX single-precision elements and finally, the vector length register (VLEN) which specifies the number of bytes that will be affected by the currently executing vector opcode. The ISA of the accelerator includes standard vector floating point operations except division, vector load/stores, and a generalized permute instruction. The programmer's model and ISA are summarized in Fig. 1 and Table I respectively.

IV. VECTOR COPROCESSOR MICROARCHITECTURE

The vector coprocessor is tightly-coupled to an open-source, configurable, extensible, Sparc V8-compliant RISC CPU. The processor/coprocessor combination communicates via the AHB On-Chip Bus [12] to the SDRAM controller which controls the off-chip SDRAM part. A high level schematic of the scalar processor and vector accelerator is depicted in Fig. 2:

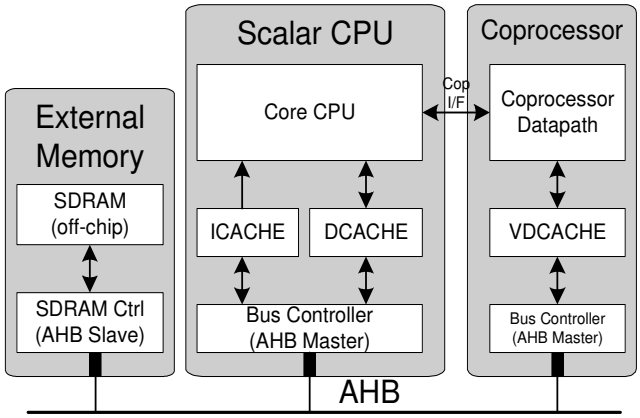

Figure 2. Single Processor/Coprocessor architecture

As shown in the figure, there exists a bidirectional communication channel across the scalar processor and the vector accelerator. Though the open source CPU provides a coprocessor interface, initial experimentation could not establish its ability to operate in a pipelined fashion. It was therefore decided to implement a custom channel in order to ensure pipelined, lockstep operation of the coprocessor and CPU and the timely transfer of data across them.

The diagram of Fig. 3 shows a coprocessor data operation on cycle 1 followed by a host-to-coprocessor register transfer on cycle 2. In cycle 3, a coprocessor register is requested by the RISC processor but due to internal stall conditions, data are made available one cycle later than the expected time (cycle 5 instead of cycle 4). During that time, the main processor is held with the *holdn* signal. Finally, a second read operation, this time directed to Coprocessor 1, is initiated in cycle 6. Results are made available to the main pipeline in cycle 7.

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on May 05,2010 at 15:41:06 UTC from IEEE Xplore. Restrictions apply.

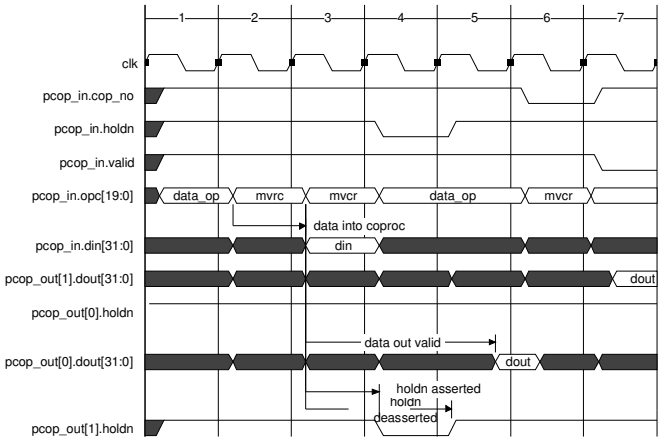

Figure 3. Processor/Coprocessor I/F Transactions

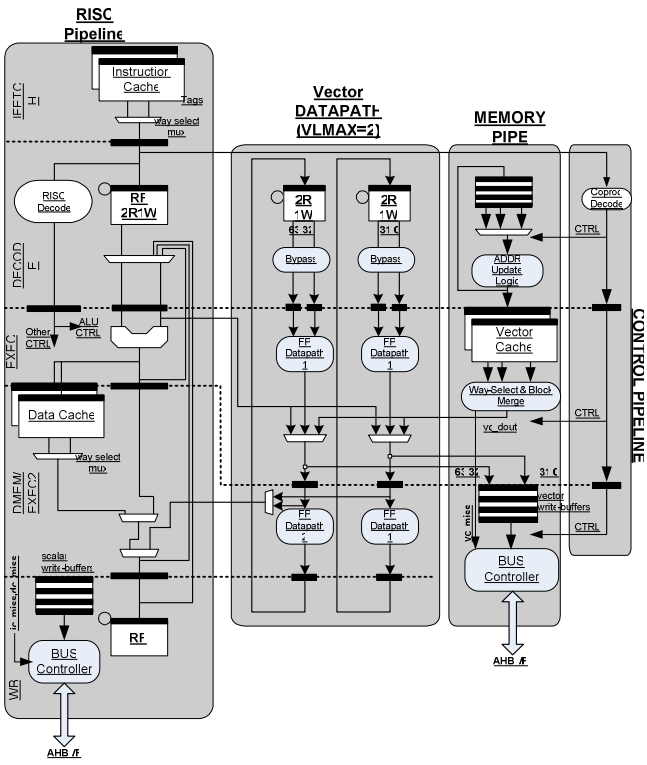

Figure 4. Detailed scalar and vector core microarchitecture

Fig. 4 depicts the detailed microarchitecture of the processor-coprocessor combination: Instructions are fetched from the multi-way set-associative instruction cache and stored in a single 32-bit register. Typically, high-performance RISC processors of equal pipeline depth would extract the source operand fields right after instruction cache access and set up the synchronous register file. Unfortunately, this is not the case in the particular processor which, due to the windowing scheme of the Sparc V8 architecture, requires access to the current-window-pointer (CWP) register in order to compute a physical register file address. As a result, source operand addresses are set-up on the falling edge of the clock in the DECODE stage. During this stage, the register file is accessed and the two source registers are retrieved. Operand bypassing takes then place and the resolved operands are clocked into the ALU input registers, ready for execution. It is during this stage that the vector opcodes are identified and dispatched to the tightly-coupled vector accelerator. Decoding logic in the later produces a number of control fields which are pipelined down the control pipeline. Vector operands accesses are triggered by the falling edge of the clock during decode, for reasons of symmetry to the scalar pipeline.

During the EXEC stage, the RISC CPU executes the scalar instruction or computes the virtual address of a Load/Store operation. In the same stage, the vector accelerator performs the first stage of the pipelined floating point computations. In the next stage, scalar data return to the main processor via the data cache return path whereas the vector accelerator performs the last stage of execution. Due to the very tight timing constraints, floating point results are stored in an intermediate register prior to committing to the vector register file.

V. VECTOR MULTIPROCESSOR MICROARCHITECTURE

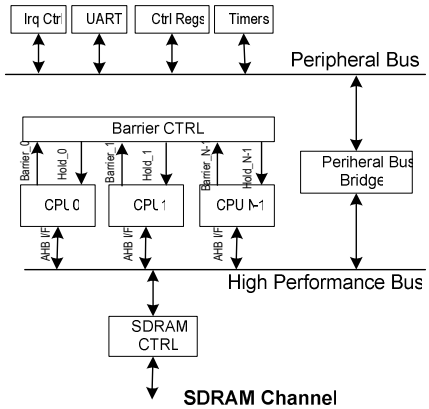

Figure 5. N-way parametric SoC Multiprocessor architecture:

The multi-processor architecture consists of a configurable number of such processor-coprocessor pairs residing on the high performance AHB bus as depicted in Fig. 5. There is an AHB-to-APB (on-chip high-performance to on-chip peripheral bus) bridge connecting the streaming processing subsystem to a number of peripherals. In this case, CPUs 1 through N-1 do not service interrupts neither can access the onboard peripherals. CPU 0 is the main controlling processor executing all I/O. The number of CPUs and the geometry of the instruction and data caches are specified via static configuration switches in the RTL source code.

VI. RESULTS

The reference problem chosen for benchmarking was a fixed mesh of 1000,000 nodes. This number is convenient as it gives a prime factorization of $2^6 \times 5^6$, which allows for the

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on May 05,2010 at 15:41:06 UTC from IEEE Xplore. Restrictions apply.

aspect ratio of the problem space to be varied over a reasonable range whilst maintaining the same number of nodes.

We measured the dynamic instruction count of the scalar code for all configurations of interest. Then, the vectorized code was run and its instruction count recorded for a maximum vector length of up to 16 single-precision elements. Figs. 6 and 7 depict the normalized dynamic instruction count of the vectorized algorithm over maximum vector length for various configurations of the 10^6 nodes problem space.

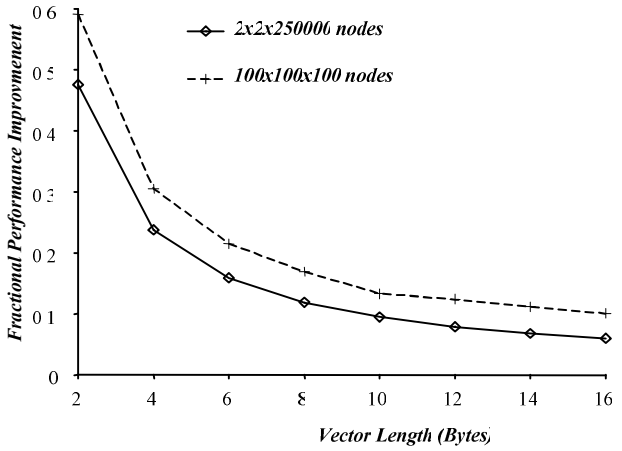

Figure 6. Vector performance for a Thin-cubic problem

Fig. 6 suggests that the optimal (less computationally expensive) configuration is where the problem space is thin, i.e. where the vector length is maximized.

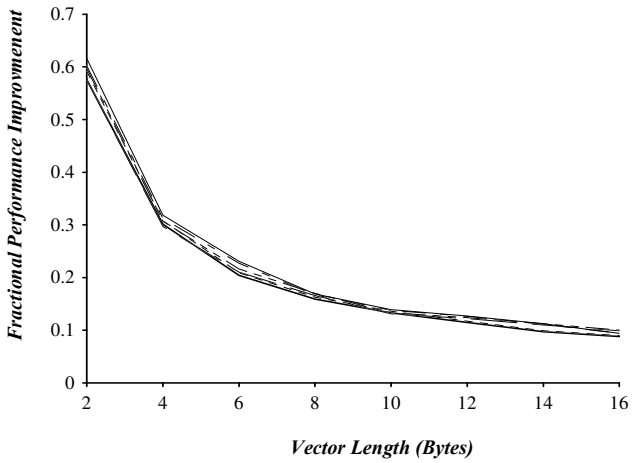

Figure 7. Vector performance for 80 x 100 x 125 node mesh with differing alignment relative to the vector direction

Fig. 7 depicts an *80×100×125* configuration compared with a mesh of *100×125×80*, *80×125×100*, etc. These mesh dimensions were chosen as being typical of realistic model of an electromagnetic scattering situation. Results demonstrate that vectorization alignment changes only slightly the complexity (and hence run time) in all configurations. A vector length of 16 IEEE 754 single-precision elements showed a speedup of approximately an order of magnitude thus clearly demonstrating the benefit of using parallelism at the data level.

We performed a second experiment for the quantifying the TLP of the application. Fig. 8 demonstrates the relative dynamic instruction count for meshes of *22 x 10 x 10*, *22 x 10 x 50*, *22 x 10 x 250* and *22 x 10 x 500* respectively. These preliminary results clearly show that there is a significant acceleration potential from exploiting TLP. A very important observation is that the TLP benefit is orthogonal to that of DLP. This potentially translates to substantial dynamic instruction count reduction, per processor-coprocessor combination, and associated runtime benefits.

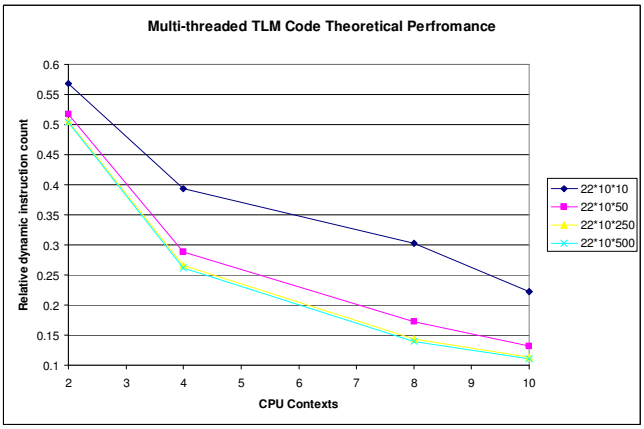

Figure 8. TLP Performance for 22x10x10, 22x10x50, 22x10x250 and 22x10x500

VII. VLSI MACROCELL

We implemented the N=2 configuration of the system depicted in Fig. 5 in a high performance 0.13 um CMOS process. The Sparc CPU processors have an 8-window scalar register file, and 4-way set-associative instruction and data caches of 8KB and 16KB respectively. The vector accelerator includes a 16x32 scalar register file and a 8X128, 3R1W vector register file.

The design was synthesized for maximum performance initially on Synopsys *Design Compiler* and then, read into Cadence *SoC Encounter* where floorplanning and power routing took place. The clusters were exported to Synopsys *Physical Compiler* for placement optimization and imported again into SoC encounter for detailed routing. Figs. 9 and 10 depict the floorplan and final layout of the N=2 configuration. The macrocell implementation data are tabulated in Table II.

VIII. CONCLUSION

We have developed a two-tier, configurable, extensible SoC architecture based on open-source intellectual property cores, for exploiting the major forms of parallelism in the SCM TLM code. Tier-1 consists of a parametric vector accelerator exploiting the DLP and demonstrates a ten-fold performance improvement for a vector register length of sixteen 32-bit elements. Tier-2 instantiates multiple such

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on May 05,2010 at 15:41:06 UTC from IEEE Xplore. Restrictions apply.

processor-coprocessor pairs in a symmetric, shared memory configuration and exhibits near-linear theoretical performance improvement. Both results are orthogonal to one another clearly showing the very good potential of this combined DLP-TLP architecture for highly-parallel applications. Subsequently, we implemented a 2-way vector multiprocessor architecture utilizing 8-wide vector floating point coprocessors and residing on the industry standard AHB bus.

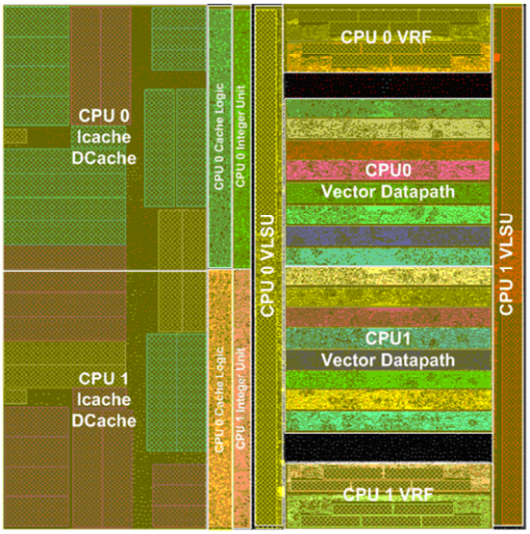

Figure 9. Figure 1: 2-way SMP floorplan

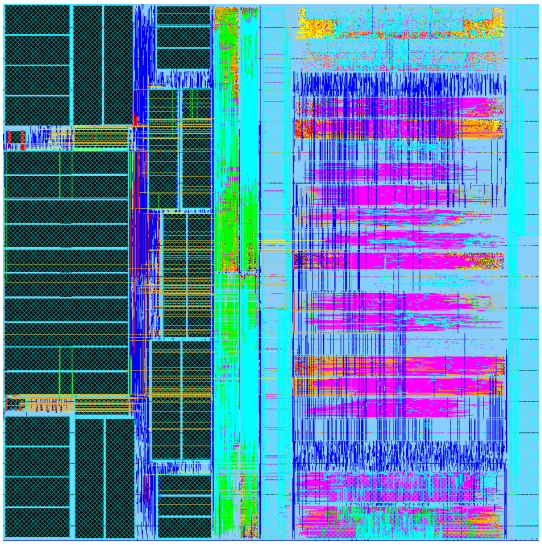

Figure 10. Figure 8: 2-way SoC Multiprocessor layout

TABLE II. TABLE 2: VLSI MACROCELL DATA

Parameter	Value
Std cells	110099
RAMs	62
Fmax	158.5 MHz
Size	3424x3426 μm^2 (11733569 μm^2)
Utilization	83.8% (Top-level)

IX. ACKNOWLEDGMENTS

The authors would like to acknowledge the support of the Department of Electronic and Electrical Engineering of Loughborough University and the anonymous referees for their very constructive comments.

X. REFERENCES

1. P. B. Johns, "A symmetrical condensed node for the TLM method", IEEE Trans. Microw. Theory Tech., vol. 35, no. 4, pp. 370–377, 1987.
2. J. L. Dubard, O. Benevello, D. Pompei, J. Le Roux, P. P. M. So, and W. J. R. Hoefer, "Acceleration of TLM through signal processing and parallel computing", in Computation in Electromagnetics, pp. 71–74, IEE, 25–27 November 1991.
3. C. C. Tan and V. F. Fusco, "TLM modelling using an SIMD computer", Int. J. Numerical Modelling: Electronic Networks, Devices and Fields, vol. 6, pp. 299–304, 1993.
4. P. J. Parsons, S. R. Jaques, S. H. Pulko, and F. A. Rabhi, "TLM modeling using distributed computing", IEEE Microw. and Guided Wave Lett., vol. 6, no. 3, pp. 141–142, 1996.
5. D. Stothard and S. C. Pomeroy, "Dedicated TLM array processor", Applied Computational Electromagn. Soc. J., vol. 13, no. 2, pp. 188–196, 1998.
6. J. Henessy, D. A. Patterson "Computer architecture: A quantitative approach", Morgan Kaufmann publishers, ISBN 1-55860-329-8
7. "The Leon-2 processor User's manual, XST edition, ver. 1.0.14", http://www.gaisler.com
8. http://www.opencores.org/projects.cgi/web/fpu/overview
9. P. Naylor and R. Ait-Sadi, "Simple method for determining 3-D TLM nodal scattering in nonscalar problems", Electron. Lett., vol. 28, no. 25, pp. 2353–2354, 1992.
10. D. Burger, T. Austin, 'Evaluating Future Microprocessors: The Simplescalar Tool Set', http://www.simplescalar.com
11. Ashwin K. Kumaraswamy, V. A. Chouliaras , T. R. Jacobs, and J. L. Nunez-Yanez, "System-on-Chip Design Framework (SDF) unifying Specification Capture and Design Modeling", Proceedings of the 2005 Electronic Design Processes (EDP) Workshop, April 6-8, Monterey Beach Hotel, Monterey, California, USA
12. "AMBA Specification (Rev 2.0)", www.arm.com

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on May 05,2010 at 15:41:06 UTC from IEEE Xplore. Restrictions apply.