CACTI BATCH GENERATOR

Noha Abuaesh

noha.abuaesh@gmail.com

This readme file presents the memory simulation tool(CACTI 6.5) that can be used to obtain energy, performance and area measurements in research concerning various types of memory. I will also explain in detail the interface developed for the tool to achieve the experimental results. The interface implemented in this repository comes along with an offline version of CACTI simulator.

USING THIS PROJECT TO COLLECT SIMULATION DATA

Below is a thorough overview of the simulation tool that can be used to obtain memory data concerning power, access times and area for research. The used tool is CACTI 6.5 [36]. An interface to CACTI 6.5 has been implemented in a previous research and was used to get the data displayed in Appendix A – Simulation Data. I also present a guide on how to use the interface I developed to get batch simulation data and discuss possible sources of inaccuracy initiated by the tool and inherited in my work.

1 ABOUT CACTI SIMULATOR

In this section, I display a quick review of CACTI in general; what it is, its origin, its forms and how to use it.

1.1 Definition

CACTI is a cache, scratchpad and RAM memory simulation tool [24]. It integrates models based on HSPICE[1] simulations[9] for access time, cycle time,

[1] HSPICE is an optimizing analog circuit simulator produced by Synopsys. It is used to simulate electrical circuits in steady-state, transient, and frequency domains. It provides fast, accurate circuit and behavioral simulation, facilitating circuit-level analysis of performance and yield, by using Monte Carlo, worst-case, parametric sweep, and data-table sweep analyses [74].

area, leakage, and dynamic power together and grants the confidence that tradeoffs between time, power, and area are all based on the same assumptions. Hence, the performance, energy and area results given by CACTI for different memory parameters may be considered mutually consistent [5]. According to CACTI developers, the tool is intended for the use of computer architects who need to better understand the performance tradeoffs inherent in memory system organizations.

Please note that the tool meant in my context is different from another well-known Cacti software; a network graphing tool topping Google's hits for "cacti". Probably it is more common than this simulator since it is a commercial tool, while our simulator is a research project.

1.2 Quick History and Origin

CACTI was originally developed as an analytical model for calculating the access and cycle times of direct-mapped and set-associative caches only in 1994 by Steve Wilton and Norm Jouppi [9], from The Western Research Laboratory (WRL); a computer systems research group founded by Digital Equipment Corporation in 1982 and focusing on computer science research relevant to the design and application of high performance scientific computers. It was built as a research prototype developed and tested by designing, building, and using real systems.

Power analysis was later integrated in the tool by Reinman and Jouppi in year 2000 [5]. In 2001, Shirakumar and Jouppi integrated area analysis [7]. The tool did not include simulation models for memory types other than cache until 2007 when Manulimanohar, Balasubramonia and Jouppi added scratchpad and DRAM to the supported types of memory [24]. The tool was later acquired by HP Labs in 2009[8], while still remaining as an open-source research project. Until the time of writing this thesis, CACTI is not a commercial product, and is not intended to become one[36].

1.3 CACTI Forms

HP Labs made available two forms of CACTI [36]: a web-based version and a C++ source code version.

The web interface version is frequently updated with the latest versions and bug fixes, and allows CACTI to be easily accessible to a larger user community. The web interface can be accessed online from [47].

In addition, CACTI source code is still available in C++ language for modification and integration into other tools especially for research support.

2 THE USED VERSION IS CACTI 6.5 (OFFLINE FORM)

Because the team at HP Labs announced that previous web versions of CACTI will not remain available online once newer versions are released [36], and because it is much more convenient for modification, integration and research support, it is always a better idea to use the offline source of CACTI for obtaining simulation data in your research.

For consistency issues, it is recommended by CACTI's team in HP Labs (HPL) to use a single version of the source code since results for specific memory configurations and technologies change as new versions of CACTI are released and more bugs are fixed due to continuous upgrading [36]. The latest version of CACTI as of the start of this work was 6.5. Hence, I used CACTI 6.5 in my study and continued using it even though newer versions may have become available by the end of this research.

CACTI 6.5 is a significantly enhanced version that includes major extensions over its predecessor--release notes can be found in [24]. CACTI 6.5 supports 32, 45, 68, and 90 nm technologies.

3 USING CACTI'S OFFLINE VERSION

To run the tool, you need to install it successfully on your machine. The only system requirement for running CACTI is a Linux system with a C compiler installed. The size of the tool is trivially small and it does not require a significant amount of memory for installation neither while running.

The tool takes as input a configuration file containing the parameters of the memory module to be simulated, like its type, size, technology, associativity—if applicable, ports and a lot of other attributes. The generated output is a file

containing simulation data about the specified memory module; like access time, access energy(dynamic and static), area and other data; see Figure 3.1.

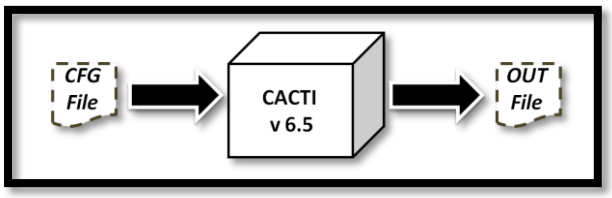

Figure 3.1 A diagram showing the inputs and outputs of CACTI.

The configuration file must be in the same format of the given sample *.cfg* file provided with the tool. A miss-formatted configuration will generate an error and prevent the tool from correct execution. The command used for invoking CACTI with a given configuration file and saving the results in *res.out* is: *./cacti in.cfg –o res.out*. The output file is generated in the same directory of CACTI or in the desired directory, if specified.

3.1 Setting up CACTI 6.5 from Github

This repository contains an offline version of CACTI simulator—version 6.5(The latest as of the writing of this document. As mentioned above, you can get CACTI from its original creators from here:[36]. Or, if for some reason CACTI version 6.5 is not available online, and you are interested in using it, you can download from this repository: https://github.com/abuaesh/CACTI-Batch-Generator. Then follow these simple steps:

1. Download the reposiroty.
2. In a terminal, change directory to the *cacti65* folder.
3. Type make, and allow to compile.
4. After compilation, you get the resulting executable file cacti.
5. You can now use the simulator with the memory configurations of your choice. Just modify the parameters in the *.cfg* file to specify the parameters of the memory to be simulated. Note: You can find examples of *.cfg* files in the same directory, like *cache.cfg*.
6. After you have configured the sepcifications in the *.cfg* file, type in your terminal:

   ./cacti –infile [file.cfg]

Your input file can have any name, not just cache, as long as it has *.cfg* extension and is of the correct format. The results will be displayed in the console.

4 USING CACTI TO GENERATE BATCH SIMULATION DATA

In our research, we are going to need simulation data for different SPM sizes—at least the sizes from 0 to 8K bytes must be available. Running CACTI individually for each and every configuration then extracting the resulting 8192 output files to get the needed data is a process that will obviously exhaust our time and effort before we can even start tackling our main research problem. Thus, we had to think of a way to automate the gruesome process of collecting the simulation data needed for our research. Consequently, we implemented the batch generator program that aided us in completing the task.

It is true that we consider only scratchpad memories and assume a technology of 32 nm in this research; however, the work may be extended for other types and technologies as well in the future. The batch generator program is therefore highly re-usable for generating simulation data for any type of memory with any size and under any technology, under the condition that they are supported by CACTI.

To run the batch generator program implemented on top of CACTI, you need to:

1. Install any version of CACTI on your machine. Make sure it works correctly by testing it with any configuration file. See the previous section (Using CACTI's Offline Version) for more details about using CACTI for a single file.
2. Save the interface file: *cacti_if.cpp* and the configuration file: *temp.cfg* together in the same directory.
3. Update the path to the installed version of CACTI in the file: *cacti_if.cpp*, line 178.
4. Compile *cacti_if.cpp* using the g++ command, as follows:
   g++ cacti_if.cpp -o cacti_if
5. Run the compiled file with the following parameters:
   ./cacti_if [start] [end] [step] [technology] [type]
   where *start*, *end* and *step* represent memory sizes in bytes. The available technologies for the *technology* field are 32(default), 45, 68 and 90 nm. And,

type can have any value from 1 to 3; such that 1 stands for cache, 2 for scratchpad (default) and 3 for main memory (or RAM).

The program gives the time elapsed for running. The final results are saved in a file named after the type, technology and sizes. For example, to simulate a cache memory module --Type 1 memory- with sizes of 512 bytes, 528 B, 544 B, up to 1024 bytes--that is a step of 16 bytes- in the 32 nm technology, run the command:

./cacti_if 512 1024 16 32 1

which generates the file: *cache32nm512:1024:16.txt* containing the desired data shown in Figure 4.1. Upon wrong invocation, an error message that explains the command in detail appears. [2]

[2] If you notice any anomalies in the batch generator or face any difficulties using it, please send to *noha_abuaesh@aucegypt.edu*.

Technology: 32 nm
Memory Type: Cache

Size(B)	Dyn. Energy(nJ)	Stat. Power(mW)	Perf.(ns)	Area(mm2)
512	0.00411748	0.0291453	0.00213663	0.695145
528	0.00414604	0.0294377	0.00213936	0.698734
544	0.00414604	0.0294377	0.00213936	0.698734
560	0.0041746	0.0297301	0.00214209	0.70229
576	0.0041746	0.0297301	0.00214209	0.70229
592	0.00420316	0.0300224	0.00214482	0.705813
608	0.00420316	0.0300224	0.00214482	0.705813
624	0.00423172	0.0303145	0.00214755	0.709304
640	0.00423172	0.0303145	0.00214755	0.709304
656	0.00426028	0.0306065	0.00215027	0.712765
672	0.00426028	0.0306065	0.00215027	0.712765
688	0.00428884	0.0308985	0.002153	0.716196
704	0.00428884	0.0308985	0.002153	0.716196
720	0.0043174	0.0311903	0.00215573	0.719598
736	0.0043174	0.0311903	0.00215573	0.719598
752	0.00434596	0.0314821	0.00215846	0.722972
768	0.00434596	0.0314821	0.00215846	0.722972
784	0.00437452	0.0317737	0.00216118	0.726319
800	0.00437452	0.0317737	0.00216118	0.726319
816	0.00440308	0.0320653	0.00216391	0.729644
832	0.00440308	0.0320653	0.00216391	0.729644
848	0.00443164	0.0323568	0.00216664	0.732967
864	0.00443164	0.0323568	0.00216664	0.732967
880	0.0044602	0.0326482	0.00216936	0.736294
896	0.0044602	0.0326482	0.00216936	0.736294
912	0.00448876	0.0329395	0.00217209	0.739623
928	0.00448876	0.0329395	0.00217209	0.739623
944	0.00451732	0.0332308	0.00217481	0.742955
960	0.00451732	0.0332308	0.00217481	0.742955
976	0.00454588	0.033522	0.00217754	0.74629
992	0.00454588	0.033522	0.00217754	0.74629
1008	0.004578	0.0346136	0.00219006	0.759355
1024	0.004578	0.0346136	0.00219006	0.75935

Figure 4.1 Snapshot of a sample output file (cache32nm512:1024:16.txt) generated by the batch generator.

What actually takes place behind the scenes in the generator program is shown in Figure 4.2. First, all the configuration files of the memory modules in the specified range are generated using the same format as in the temp file provided with CACTI in step 2 above. These configuration files are saved in a folder named *cacti_cfgs*. Then, CACTI is invoked with each and every configuration file in *cacti_cfgs* and the resulting output is saved to another folder called *cacti_outs*. After CACTI is done processing all the configuration files and all output files are generated and saved in their corresponding folder, the program then extracts the performance, dynamic and static energy as well as area information from all the output files in the *cacti_outs* folder and stores them to the corresponding columns in the results file(*cache32nm512:1024:16.txt*).

Finally, the elapsed time taken to find the required results since the command was initiated is displayed in the command window.

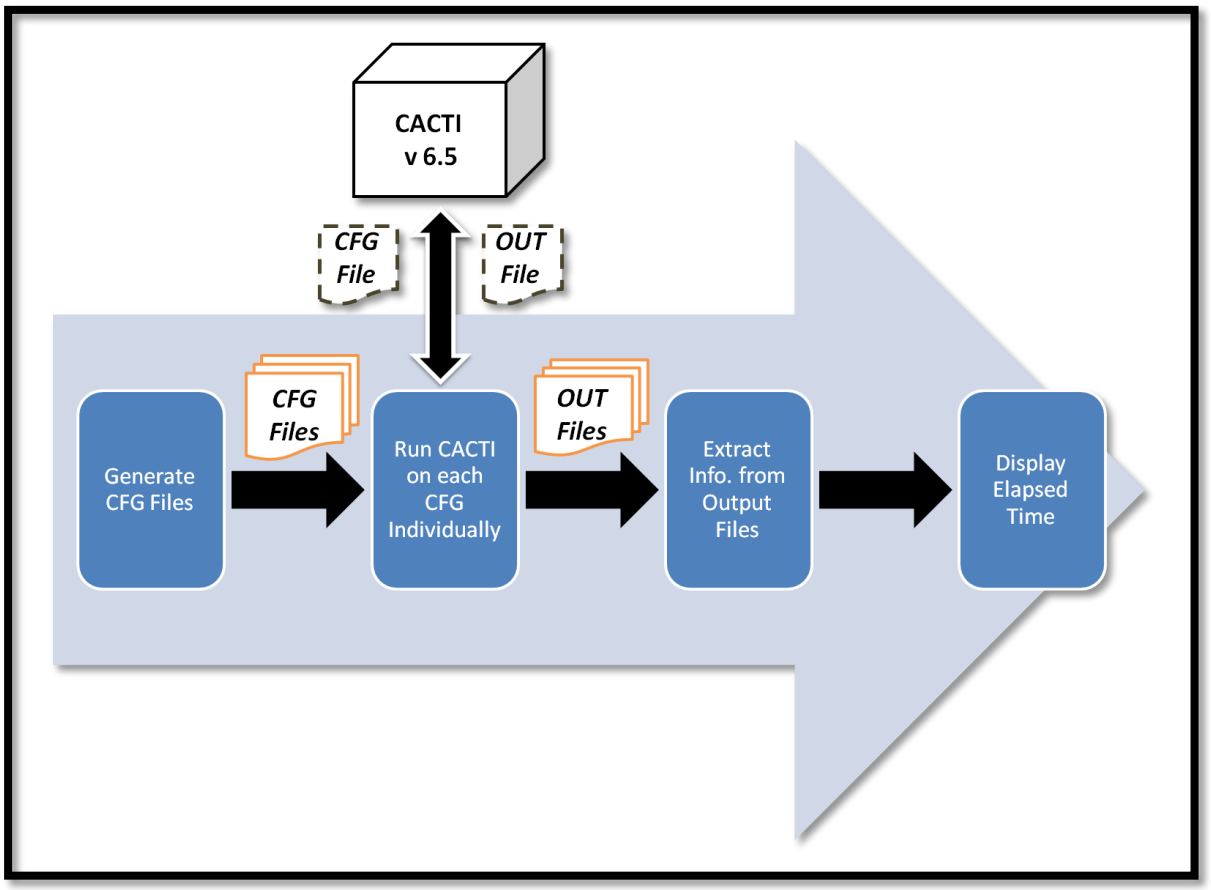

Figure 4.2 Workflow of the batch generator.

It may take considerably long time to generate large batches of simulation data. For example, generating the data for scratchpad sizes up to 8K bytes with a step of 4 bytes--that is, running CACTI for 1920 different memory configurations and extracting the results into one file- took more than an hour to complete. However, the program not only saves a much-much-longer time that would have elapsed if the process was otherwise done manually, but also eliminates a tedious amount of file processing that the user would have went through in the other case.

Nevertheless, the efficiency of the simulation data batch generator is not really important since it will be run only once in a lifetime and once the tables in Appendix A – Simulation Data are generated with the needed memory settings; the generator will not be needed any more.

5 LIMITATIONS AND SHORTCOMINGS

A major limitation that we faced while using CACTI for generating simulation data is that it did not retrieve data for memory modules of sizes less than 512 bytes. As a solution, we used extrapolated values for smaller memory sizes.

In addition to not being able to retrieve reliable simulation data for memory sizes below 0.5 KB, another difficulty is that CACTI as a tool is not very user friendly. Perhaps the reason is that it is basically intended for research purposes and not for commercial distribution. Even the batch generator we developed lacks a user-friendly interface. Nevertheless, we did not wish to put needless effort in developing a GUI since the command-based interface is not overly complicated and because once the tables in Appendix A – Simulation Data are generated, the tool will be needed no more.

As mentioned earlier, CACTI is based on HSPICE models. According to CACTI's technical report, the generated estimates are within 10% of HSPICE results for selected circuits used in testing CACTI's accuracy [9]. This inaccuracy will consequently be inherited in our work since CACTI's simulation data are the base for our experiments and results.

As announced in HP Labs official website [36], the systems built are research prototypes; they are not intended to become products; which means that they are continuously under development and are vulnerable to major changes. The continuous upgrading of CACTI implies variable simulation results for specific cache configurations and technologies as new versions are released and bugs are fixed. Therefore, the work depending on simulation data obtained from CACTI needs to be periodically updated with data attained by running the latest version of CACTI.

REFERENCES

[1] G. Grewal, S. Coros, D. Banerji and A. Morton "Assigning data to dual memory banks in DSPs with a genetic algorithm using a repair heuristic," in Springer Science and Business Media, LLC 2006.

[2] L. Benini, A. Macii, and M. Poncino, A recursive algorithm for low-power memory patitioning, in *Proc. ACM/IEEE Int. Symp. Low Power Electronics and Design*, Rapallo, Italy, July 2000, pp. 78-83.

[3] F. Angiolini, L. Benini, and A. Caprara, An efficient profile-based algorithm for scratchpad memory partitioning, *IEEE Trans. CAD*, vol. 24, no. 11, pp. 1660-1676, Nov. 2005.

[4] O. Golubeva, M. Loghi, M. Poncino, and E. Macii, Architectural leakage-aware management of partitioned scratchpad memories, in *Proc. ACM/IEEE Design Automation and Test in Europe*, Nice, France, Apr. 2007, pp. 1665-1670.

[5] M. Kandemir, M.J. Irwin, G. Chen, and I. Kolcu, Banked scratch-pad memory management for reducing leakage energy consumption, in Proc. *IEEE/ACM Int. Conf. on Computer-Aided Design*, San Jose CA, Nov. 2004, pp. 120-124.

[6] G. Reinman and N. Jouppi, Cacti 2.0: an integrated cache timing and power model, *COMPAQ Western Research Lab*, Palo Alto, 1999.

[7] P. Shivakumar and N. P. Jouppi, Cacti 3.0: an integrated cache timing, power and area model, *COMPAQ Western Research Lab*, Palo Alto, Aug. 2001.

[8] N. Muralimanohar , R. Balasubramonian, Cacti 6.0: a tool to model large caches, *Internatinal Symposium on Microarchitecture*, Chicago, Dec 2007.

[9] S.J. E. Wilton and N. Jouppi, Cacti: an enhanced cache access and cycle time, *IEEE Journal of Solid State Circuits* , vol.31, no.5, May 1996.

[10] M. Kandemir, M.J. Irwin, G. Chen, and I. Kolcu, Compiler-guided leakage optimization for banked scratch-pad memories, *IEEE Trans. VLSI Systems*, vol. 13, no. 10, pp. 1136-1146, Oct. 2005.

[11] F. Balasa, H. Zhu, and I.I. Luican, Computation of storage requirements for multi-dimensional signal processing applications, *IEEE Trans. VLSI Systems*, vol. 14, no. 4, pp. 447-460, Apr. 2007.

[12] P.R. Panda, F. Catthoor, N. Dutt, K. Dankaert, E. Brockmeyer, C. Kulkarni, and P.G. Kjeldsberg, Data and memory optimization techniques for embedded systems,

ACM Trans. Design Automation of Electronic Syst., vol. 6, no. 2, pp. 149-206, Apr. 2001.

[13]F. Balasa, C. V. Gingu, I. I. Luican and H. Zhu, Design space exploration for low power memory systems in embedded signal processing applications, *The 19th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2013)*, Aug. 2013.

[14]F. Balasa, I. I. Luican, C. V. Gingu, Energy-aware banking of on chip memory for embedded signal processing systems, unpublished.

[15]U. Ko, P.T. Balsara, and A.K. Nanda, Energy optimization of multilevel cache architectures for RISC and CISC processors, *IEEE Trans. VLSI Syst.*, vol. 6, no. 2, pp. 299-308, June 1998.

[16]H. Zhu, I.I. Luican, F. Balasa, Memory size computation for multimedia processing applications, *Proc. Asia & South-Pacific Design Automation Conf. (ASP-DAC 2006)*, pp. 802-807, Yokohama, Japan, Jan. 2006.

[17]H. Zhu, I.I. Luican, F. Balasa, and D.K. Pradhan, Formal model for the reduction of the dynamic energy consumption in multi-layer memory subsystems, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sc.*, vol. E91-A, no. 12, pp. 3559-3567, Dec. 2008.

[18]L. Benini, L. Macchiarulo, A. Macii, and M. Poncino, Layout-driven memory synthesis for embedded Systems-on-Chip, *IEEE Trans. VLSI Systems*, vol. 10, no. 2, pp. 96-105, Apr. 2002.

[19]Zhu, H.; Luican, I.I.; Balasa, F., Mapping Multi-Dimensional Signals into Hierarchical Memory Organizations, *Design, Automation & Test in Europe Conference & Exhibition*, Apr. 2007.

[20]W. Shiue and C. Chakrabarti, Memory design and exploration for low power embedded systems, in *VLSI Signal Processing*, vol. 29, pp. 167–178, 2001.

[21]W. Shiue and C. Chakrabarti, Memory exploration for low power embedded systems, in *Proc. 36th ACM/IEEE Design Automation Conf.*, pp. 140-145, New Orleans, June 1999.

[22]S. Coumeri and D.E. Thomas, Memory modeling for system synthesis, *IEEE Trans. VLSI Systems*, vol. 8, no. 3, pp. 327-334, June 2000.

[23]P. R. Panda, N. D. Dutt and A. Nicolau, On-chip versus off-chip memory, the data partitioning problem in embedded processor based systems, *ACM Transactions on*

Design Automation of Electronic Systems (TODAES), vol. 5 Issue 3, pp 682 – 704, July 2000.

[24]N. Muralimanohar, R. Balasubramonian, N. Jouppi, Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0, *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 3-14, Washington, 2002.

[25]F. Angiolini, L. Benini, and A. Caprara, Polynomial time algorithm for on-chip scratchpad memory partitioning, *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES 2003,* San Jose, California, Nov., 2003.

[26]V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computing Surveys*, vol. 37, no. 3, Sept. 2005, pp. 195–237

[27]R. Banakar, S. Steinke, B.S. Lee, M. Balakrishnan, and P. Marwedel, Scratchpad memory : A design alternative for cache on-chip memory in embedded systems, in *Proc. 10th Int. Workshop on Hardware/Software Codesign*, Estes Park CO, May 2002, pp. 73-78.

[28]F. Balasa, I. Luican, H. Zhu and D. Nasui, *Signal assignment model for the memory management of multi-dimensional signal processing applications*, Springer, 2009.

[29] F. Balasa, P. G. Kjeldsberg, A. Vandecappelle, M. Palkovic, Q. Hus and F. Catthoor, Storage estimation and design space exploration methodologies for the memory management of signal processing applications, *Journal of Signal Processing Systems*, vol.53, no.1-2, pp. 51-71, Nov. 2008.

[30]A. Farrahi and M. Sarrafzadeh, System partitioning to maximize sleep time, *in Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, San Jose CA, Nov. 1995, pp. 452-455.

[31]A. Macii, L. Benini, and M. Poncino, *Memory Design Techniques for Low Energy Embedded Systems*, Boston: Kluwer Academic Publishers, 2002.

[32]M. Verma and P. Marwedel, *Advanced Memory Optimization Techniques for Low-Power Embedded Processors*, Springer, 2007.

[33]P. Grun, N. Dutt, and A. Nicolau, Access pattern based local memory customization for low-power embedded systems, in *Proc. ACM/IEEE Design, Automation and Test in Europe*, Munich, Germany, Mar. 2001, pp. 778-784.

[34]M. Kandemir, J. Ramanujam, M.J. Irwin, N. Vijaykrishnan, I. Kadayif, and A. Parikh, Dynamic management of scratch-pad memory space, in *Proc. ACM/IEEE Design Automation Conf.*, Las Vegas NV, June 2001, pp. 690-695.

[35]V. Delaluz, M. Kandemir, N. Vijaykrishnan, M.J. Irwin, A. Sivasubramaniam, and I. Kolcu, Compiler-directed array interleaving for reducing energy in multi-bank memories, in *Proc. Asia & South Pacific Design Automation Conf.*, Bangalore, India, Jan. 2002, pp. 288-293.

[36]CACTI 6.5 [Online] http://www.hpl.hp.com/research/cacti/

[37]F. Balasa, H. Zhu, and I.I. Luican, Signal assignment to hierarchical memory organizations for embedded multidimensional signal processing systems, *IEEE Trans. on VLSI Systems*, vol. 17, no. 9, pp. 1304-1317, Sept. 2009.

[38] Darte, R. Schreiber, and G. Villard, Lattice-based memory allocation, *IEEE Trans. Computers*, vol. 54, pp. 1242-1257, Oct. 2005.

[39]P. Clauss and V. Loechner, Parametric analysis of polyhedral iteration spaces, *VLSI Signal Processing*, vol. 19, no. 2, pp. 179-194, 1998.

[40]Balasa, F.; Zhu, H.; Luican, I.I., Signal Assignment to Hierarchical Memory Organizations for Embedded Multidimensional Signal Processing Systems, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.17, no.9, pp.1304,1317, Sept.

[41]F. Balasa, I.I. Luican, C.V. Gingu, Scratch-pad memory banking for energy reduction in embedded signal processing systems, *Proc. of the 56th IEEE Int. Midwest Symposium on Circuits and Systems*, pp. 844-847, Columbus OH, Aug. 2013.

[42]G. Talavera, M. Jayapala, J. Carrabina, and F. Catthoor, Address generation optimization for embedded high-performance processors: A survey, *J. Signal Processing Systems*, Springer, vol. 53, no. 3, pp. 271- 284, Dec. 2008.

[43]E. De Greef, F. Catthoor, H. De Man, Memory size reduction through storage order optimization for embedded parallel multimedia applications, special issue on: Parallel Processing and Multimedia (ed. A. Krikelis), in *Parallel Computing*, Elsevier, vol. 23, no. 12, pp. 1811-1837, Dec. 1997.

[44] R. Tronc¸on, M. Bruynooghe, G. Janssens, F. Catthoor, Storage size reduction by in-place mapping of arrays, *Verification, Model Checking and Abstract Interpretation*, pp. 167-181, 2002.

[45]V. Lefebvre, P. Feautrier, Automatic storage management for parallel programs, *Parallel Computing*, vol. 24, pp. 649-671, 1998.

[46]A. Darte, R. Schreiber, G. Villard, Lattice-based memory allocation, *IEEE Trans. Computers*, vol. 54, pp. 1242-1257, Oct. 2005.

[47]CACTI Latest Version [Online]http://quid.hpl.hp.com:9081/cacti/

[48]Lattice Power Calculator [Online] www.latticesemi.com/products/designsoftware/diamond/index.cfm

[49]T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "Dynamic Programming," in Introduction to Algorithms, 2nd ed, Cambridge, MA: MIT Press, 2001, ch. 15, sec. 3, pp. 339–345.

[50]F. Balasa, I.I. Luican, N. Abuaesh, and C.V. Gingu, "Compiler-directed memory hierarchy design for low-energy embedded systems," *Proc. of the 11th ACM/IEEE Int. Conf. on Formal Methods and Models for Codesign*, Portland OR, Oct. 2013, pp. 147-156.

[51]S. Heath, *Embedded Systems Design*, EDN series for design engineers (2nd ed.), 2003.

[52]F. Catthoor, S. Wuytack, E. De Greef, F.Balasa, L. Nachtergaele, and A. Vandecappelle, *Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design*, Boston: Kluwer Academic Publishers, 1998.

[53]T. C. Lee, S. Malik, V. Tiwari, and M. Fujita, "Power analysis and minimization techniques
for embedded DSP software," *IEEE Trans. VLSI Systems*, vol. 5, no. 1, pp. 123-135, March 1997.

[54]M.E. Wolf and M.S. Lam, "A data locality optimization algorithm," *Proc. ACM SIGPLAN'91 Conf.*, Toronto, Canada, June 1991, pp. 30-44.

[55]A. Darte, "On the complexity of loop fusion," *Parallel Computing*, vol. 26, no. 9, pp. 1175-1193, 2000.

[56]Q. Hu, A. Vandecapelle, M. Palkovic, P.G. Kjeldsberg, E. Brockmeyer, and F. Catthoor, "Hierarchical memory size estimation for loop fusion and loop shifting in data-dominated applications," in *Proc. Asia & South-Pacific Design Automation Conf.*, Yokohama, Japan, Jan. 2006, pp. 606-611.

[57] F. Balasa, P.G. Kjeldsberg, M. Palkovic, A. Vandecappelle, F. Catthoor, "Loop transformation methodologies for array-oriented memory management," *Proc. IEEE*

Int. Conf. on Application-Specific Systems, Architectures, and Processors, Steamboat Springs CO, Sept. 2006, pp. 205-212.

[58]G. Chen, M. Kandemir, N. Vijaykrishnan, M.J. Irwin, and W. Wolf, "Energy savings through compression in embedded Java environments," *Proc. Int. Conf. Hardware-Software Codesign and System Synthesis*, 2002.

[59]K. Itoh, K. Sasaki, and Y. Nakagome, "Trends in low-power RAM circuit technologies," *Proc. of the IEEE*, vol. 83, no. 4, pp. 524-543, April 1995.

[60]S. Bhattacharjee and D.K. Pradhan, "LPRAM: A Novel Low-Power RAM Design with Testability," *IEEE Trans. Computer-Aided Design on IC's and Systems*, vol.\ 23, no.\ 5, pp.\ 637-651, May 2004.

[61]R. Banakar, S. Steinke, B.S. Lee, M. Balakrishnan, and P. Marwedel, "Comparison of cache and scratch-pad based memory systems with respect to performance, area and energy consumption," *Technical Report #762*, University of Dortmund, Sept. 2001.

[62]M. Takahashi, T. Nishikawa, M. Hamada, T. Takayanagi, H. Arakida, N. Machida, H. Yamamoto, T. Fujiyoshi, Y. Ohashi, O. Yamagishi, T. Samata, A. Asano, T. Terazawa, K. Ohmori, Y. Watanabe, H. Nakamura, S. Minami, T. Kuroda, and T. Furuyama, "A 60MHz 240mW MPEG-4 videophone LSI with 16Mb embedded DRAM," *IEEE J.Solid-State Circuits*, vol. 35, no. 11, pp. 1713-1721, Nov. 2000.

[63]F. Balasa, I.I. Luican, D.V. Nasui, "High-quality data assignment to hierarchical memory organizations for multidimensional signal processing," *Proc. 5th IEEE Asia Symposium on Quality Electronic Design*, pp. 89-96, Penang, Malaysia, Aug. 2013

[64]A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley, New York, 1986.

[65]S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power,"in *Proc. Symp. Computer Arch.*, June 2001, pp. 240-251.

[66]K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: Simple techniques for reducing leakage power," in *Proc. Symp. Computer Architecture*, May 2002, pp. 148-157.

[67]M. Loghi, O. Golubeva, E. Macii, and M. Poncino, "Architectural leakage power minimization of scratchpad memories by application-driven subbanking," *IEEE Trans. Computers*, vol. 59, no. 7, pp. 891-904, July 2010.

[68]R. Allen and K. Kennedy, *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*, Morgan Kaufmann Publ., 2001.

[69]M.Moonen, P.V. Dooren, and J.Vandewalle, "An SVD updating algorithm for subspace tracking, " *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 4, pp. 1015-1038, 1992.

[70]E. Chan, S. Panchanathan, "Motion estimation architecture for video compression," *IEEE Trans. Consumer Electronics*, vol. 39, pp. 292-297, Aug. 1993.

[71]T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "Linear Programming," in Introduction to Algorithms, 2nd ed, Cambridge, MA: MIT Press, 2001, ch. 29, pp. 843.

[72]P. Huybers, "Backtracking Algorithms", in *Pascal Huybers Homepage* [Online]: http://www.huybers.net/backtrack/backe.html

[73]D. Matuszek, "Backtracking", in *Pen Computer and Information Science Website* [Online]: http://www.cis.upenn.edu/~matuszek/cit594-2012/Pages/backtracking.html

[74]Synopsis, HSPICE User Guide: Simulation and Analysis, version B-2008.09, 2008, ch 1, pp 3.

[75]ARM. Advanced RISC Machines Ltd, ARM7TDMI Reference Manual. http://www.arm.com/pdfs/DDI0210B_7TDMI_R4.pdf

[76]ATMEL. Atmel Corporation. http://www.atmel.com

APPENDIX A – SIMULATION DATA

This appendix lists memory data concerning power, access times and area used in this research and obtained using an interface to CACTI 6.5.

The data can be found at the following folder:

Appendices\CACTI Results\CACTI Simulation Data-SPM.xlsx

The Excel file contains four tabs for data obtained with different technologies: 32nm, 45nm, 68nm and 90 nm.

APPENDIX B – TESTING RESULTS

The optimal partitioning cut sets found by the algorithm in chapter **Error! Reference source not found.** are shown in this appendix along with data captured for each generated solution; like the total energy consumption, the total time cost of the partitioning and the total area needed for the partitioning. The memory simulation data were acquired for the 32 nm^2 technology, scratchpad memory type.

The results in this appendix were obtained by running a C++ implementation of the algorithm on a PC with an i5 core at a 2.5 GHz processor. The program was tested using the same benchmark used to test the previous approaches with different values of Φ and granularity.

The data can be found at the following folder:

Appendices\DP Algorithm 5.1 Results\Algorithm 5.1 Results.xlsx

The Excel file contains three tabs for data obtained with each optimization target: energy, performance and area.