Synopsys' Educational Generic Memory Compiler

Conference Paper · May 2014

DOI: 10.1109/EWME.2014.6877402

CITATIONS
5

READS
3,511

5 authors, including:

Ronald N. Goldman
Rice University
235 PUBLICATIONS **3,328** CITATIONS
SEE PROFILE

T. Wood
Synopsys
11 PUBLICATIONS **71** CITATIONS
SEE PROFILE

Vazgen Melikyan
Synopsys Armenia CJSC
109 PUBLICATIONS **165** CITATIONS
SEE PROFILE

Synopsys' Educational Generic Memory Compiler

R. Goldman[1], K. Bartleson[1], T. Wood[1], V. Melikyan[2], E. Babayan[2]
[1] *Synopsys, Inc., CA, USA*
[2] *Synopsys Armenia CJSC, Yerevan, Armenia*
vazgenm@synopsys.com

Abstract – A software tool Synopsys' Educational Generic Memory Compiler (GMC) that enables automatic generation of static RAM cells (SRAMs) based on the parameters supplied by the user is presented. The software and the generated SRAMs are made to be free from intellectual property restrictions and can be easily integrated into educational designs. GMC deployment proved its effectiveness in educational process.

I. Introduction

An important component of successful microelectronics education is implementation of real-world complexity designs at universities [1]. This enables students to get practical experience in addition to theoretical education. Modern state-of-the-art designs are in the area of highly configurable embedded processors used widely in the today's emerging mobile ecosystem [2] mostly optimized for low power consumption [3]. Implementation of complex designs in universities is usually a challenge from viewpoint of unavailability of designs and semiconductor foundry information. There are solutions available that provide these components through Synopsys University Programs [4-6].

However, modern configurable embedded processors require built-in static random access memories (SRAMs) for implementation of cache storage, etc. Different configurations of processor require different architectures and sizes of SRAMs. In the industry this requirement is solved by foundries or EDA vendors providing designers with Memory Compilers [7-10], which are software tools that could generate memories of different types and configurations based on user-defined parameters for specific technology node.

Such compilers are thus required in university environment to bring university designs closer to industrial ones. However industrial memory compilers are specific to foundry technology nodes and have the same intellectual property restrictions as technological data. This poses issue for using such compilers by universities. As embedded processors become an everyday design projects, there is a requirement from universities to have similar software for customizable generation of memory designs for technology nodes available to universities.

II. Memory Compiler Principles

Memory compilers are based on the fact that memory circuits, and SRAMs in particular, have flexible architecture and once designed can be easily extended to have larger storage. This section presents the flexible nature of SRAM architectures as well as different known industrial implementations of memory compilers.

A. Flexibility of memory architectures

In SRAM design bits are stored in bitcells[7] each of which holds single bit. Bitcells are arranged in two-dimensional array structures (fig. 1). Each row in the array is a single word and each column corresponds to bit position. The rest of the circuitry is for control, there are tree control units.

- decoder, taking word address and activates specific row which is addressed
- read/write circuitry, which then writes or reads to or from the activated row
- a timing controller, ensuring flawless sequence of these operations.

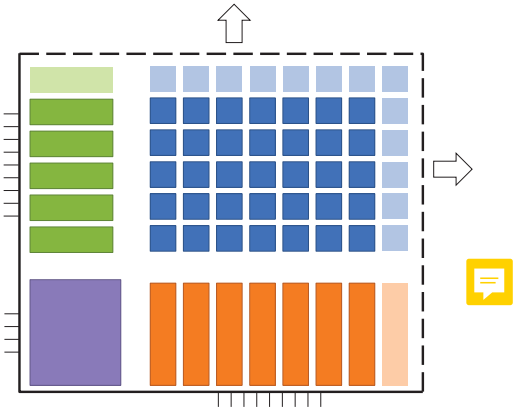

Fig. 1. Memory architecture

The first thing that is noticeable here is the easy scalability of the structure. If larger size memory is needed it is achieved by creating larger array. With new columns periphery is added, for new row - decoder is expanded. While expanding the cell increases SRAM size and storage the components used remain the same. This means that designing new memories of different sizes from existing components can be automated.

Other approaches to memory scaling are also highly automatable. The first approach [7] is called multiplexing it using for preventing increase of decoder size, and making more uniformly sized structures saving area. For multiplexing the bitcell array is divided to a few parts and these are put near each other and selected using multiplexor. Using multiplexors enables usage of smaller number of read/write blocks by selective and partial to the array (fig. 2). Another approach [8] is banking, when memory array is divided into separate banks which are connected to the same decoder through buffers amplifying decoder signals. As with the multiplexing, banking helps saving memory space by reusing the same decoder for larger array. Again no new components are added to the circuit though its storage can be

978-1-4799-4016-5/14/$31.00 ©2014 IEEE

increased by few times.

These examples show that scaling of memory is possible to automate and there are various variables that can be provided to users to configure: memory array size, number of banks, size of multiplexors, etc.

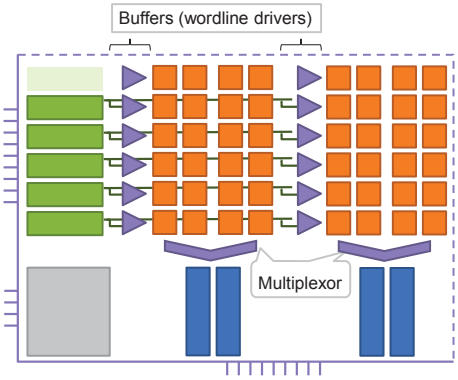

Fig. 2. Memory banking and multiplexing

B. Existing memory compilers

There are few implementations of industrial memory compilers. For most of the compilers the output formats resemble the list of file types usually created by normal design process: layout and netlist views, Verilog/VHDL models, timing and power models and datasheet [7-10]. Common user defined variables are: word size, defining how many bits one record contains, address width, defining number of words, multiplexing ratio (none, 2, 4, 8, etc.), banking mode (none, 2, 4, 8, etc.) [8,9]. As for the structure of the compiler, separate generator (assembler) structures are used (fig. 3) which take as input architecture templates together with memory design components and generate respective views [7,10].

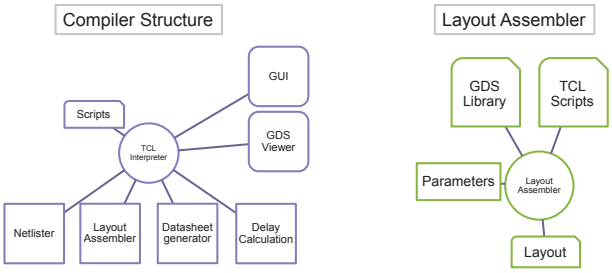

Fig. 3. Memory compiler structure example

In most of the implementations [8, 10] scripting languages such as TCL are used for software control and architecture template creation.

II. GENERIC MEMORY COMPILER

The main goal of GMC development was providing software capable of compiling different types of SRAMs with different parameters, being free from intellectual property restrictions. Also, as both the tool and memories should be free from IP restrictions it was targeted on 90nm and 32/28nm abstract technologies available to universities through Synopsys University Programs [5,6]. These technologies are already supported by respective iPDKs and EDKs. Thus another goal of GMC implementation was flexible and extensile software design which will enable implementation of different memory types and technology nodes with no or little modification of program itself.

A. Capabilities and user interface

For each supported memory type GMC generates

- Open Access[6] layout view,
- GDSII layout data,
- SPICE netlist,
- parasitic extracted SPF netlist,
- LEF views,
- Verilog and VHDL model,
- NLDM and CCS timing and power libraries,
- DRC/LVS verification reports.

The GMC is implemented as a command line tool which takes as argument user configuration file (fig. 4) based on which compilation is performed.

SRAM4x16.config

```
mem_type=dual_32
word_count=16
word_bits=4
instance_name=SRAM4x16d
do_layout=1
do_spice=1
do_gds=1
do_logic=1
do_lib_ccs=1
do_lib_nldm=1
do_drc=1
do_lvs=1
do_lef=1
```

Fig. 4. Memory configuration example

Each line in configuration file defines user specified variable value, variables starting with do_* control generation of separate views.

B. Components

As with known implementations, GMC consists of several components generating different views of SRAMs. These components are: layout and netlist generators, logic model generator and characterization, physical verification, parasitic extraction and LEF generation modules. Layout and netlist generation components are described below as examples of GMC implementation.

Layout generation is the process that generates layout of SRAM out of a set of leaf cells. Leaf cells are: bitcell, read/write circuits, decoders, timing controller, etc.

Layout generation was created around tiling placement concept [8]. Thus most of the leaf cells are designed to connect by abutment. Generation (or tiling) process consist of the following steps:

- Placement of cells,
- L-Shape routing of not-abutted connections,
- Power network synthesis,
- Generation of I/O pins.

Generation (or tiling) process is based on relations defined between different leaf cells that the layout is constructed of

(fig. 5). Some of the leaf cells are fixed to absolute coordinates while location of others is defined relative to others, and depending on some parameters, for example, memory size NxM, final layout could be generated.

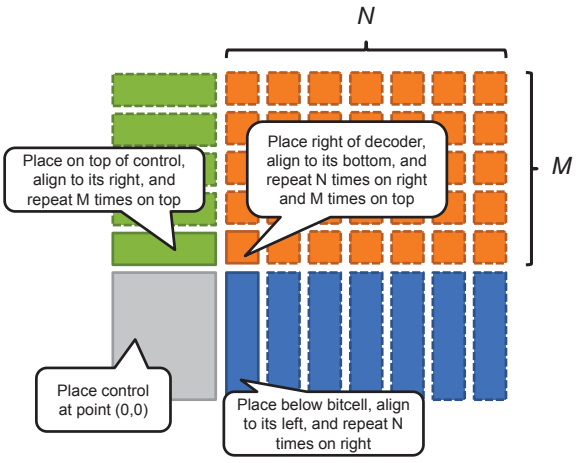

Fig. 5. Layout tiling example

In the implementation of GMC such relationship is defined in terms of layout template script in TCL language (fig. 6).

```
create_cell my_cell

place_condition   my_cell i1 box_a absolute 0 0
set_attribute     my_cell.i1 mirrorx
set_attribute     my_cell.i1 columns 1
set_attribute     my_cell.i1 rows  1
set_attribute     my_cell.i1 rotation 90

place_condition   my_cell i2 box_b relative top-right i1
set_attribute     my_cell.i2 offset-left 20

place_condition   my_cell i3 box_c overlay right-top i2
set_attribute     my_cell.i3 offset-top 20
set_attribute     my_cell.i4 mirrory
```

Fig. 6. Layout tiling template example

Another key component of circuit generation is netlist generator. Netlist generation outputs SPICE netlist based on the netlists of each of leaf cells and SRAM architecture provide in form of template. As a template language for netlister an extension of Verilog language was used (fig. 7).

```
`define words              ${word_count}
`define bits               ${word_bits}
`define bitcell_rows       ${word_count}/2
`define bitcell_columns    ${bitcell_rows}*2

module leaf_cell( a, b, c);
    input   a;
    input   [0:`word_count`] b;
endmodule

module some_top_`name`();
   wire [0:7] multibit_a;
   wire singlebit_b;
   leaf_cell #8 (.a(multibit_a), .b(singlebit_b) );
endmodule
```

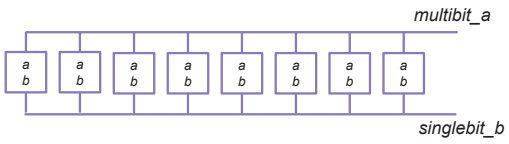

Fig. 7. Netlist template example and generated netlist structure

The extension to Verilog makes few changes to syntax enabling creation of flexible templates for SRAM architecture. The first extension is possibility to use user-defined SRAM parameters from configuration file inside template using `${variable_name}` syntax. In addition, arithmetic operations on these and Verilog preprocessor variables are allowed inside "`define" statements. The variables defined in this way can be used to replace any text inside the file when written surrounded by ticks "``". Thus names of modules could also be modified according to requirements (fig. 7). Another important extension is ability to do multiple instantiations on single line using instance multiplication factors "#<number of instances>" syntax. Such statement places multiple instances of the same type when Verilog template is expanded to SPICE. In addition based on the width of the wires connected to the pins of the instance they could connect to multi-bit buses one by one, or be shortened to a single bit wire. An example of this feature is shown in the fig. 7., where pins "a" of instances of "leaf_cell" are shortened to connect to a single-bit wire "singlebit_b, while pins "b" are connected to separate bits of bus "multibit_a".

B. Extensibility

GMC was designed to be easily extensible which means new memory types and technology nodes could be added to the system. This is required as the goal was to support 90nm and 32/28nm existing abstract technologies [4-6] as well as other technologies that will be used in the future. Similarly, support for different types of SRAMs can be added.

Extensibility of the GMC was achieved by implementation of plugin-based design, where technology node and memory architecture related data are separated into independent plugins and are included together with GMC distribution (fig. 8). While GMC is a standalone generating designs of available memory plugins based on technology plugins.

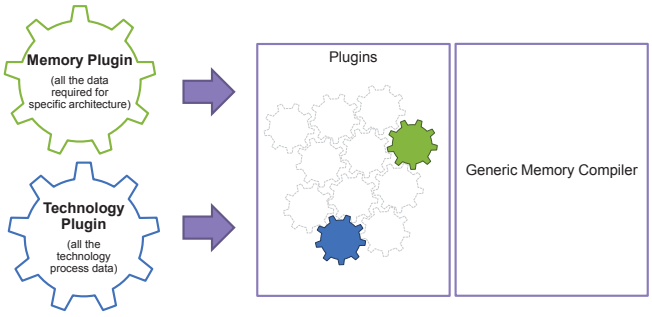

Fig. 8. Generic Memory Compiler plugin system

A plugin is implemented as a directory with the required set of files and a configuration file listing paths of all the components. For example memory plugin configuration file (fig. 9) lists paths to SPICE and GDS views of leaf cells, Verilog template discussed above, and other data required in the compilation process. Each memory plugin is designed for specific target technology for which technology plugin should be available in the systems. Similar to memory plugins, technology plugins are directories with set of required files for technology node (physical verification runsets, Open Access technology file, parasitic models etc.) and configuration file listing the content; these are normally available in foundry process design kits.

Users-defined technology and memory plugins are supported. While technology plugins contain mostly unmodified technology related files otherwise available,

memory plugin design requires parameterization of existing memory designs and their representation in the formats required by the GMC.

saed32nm.cfg

```
target_tech       = saed32
arch_verilog      = ./d_dual_32/cells.v
lib_spice         = ./d_dual_32/cells.sp
lib_lef           = ./d_dual_32/cells_dual.lef
power_pins        = VDD,VSS
gds_path          = ./d_dual_32/dual.gds
ref_lib_path      = ./d_dual_32/sram_dual_ref_oa
ref_lib           = 32nm_lib

#timing
curve_path        = ./common_32/curve.txt
libtemp_dir       = ./common_32
libdata           = ./common_32
libdata_bot_path  = ./common_32
libdata_head_path= ./common_32
libdata_cases     = max_0d95_125,typ_1d05_m40
```

Fig. 9. Generic Memory Compiler plugin configuration example

E. Current version

Synopsys GMC was built to be used with University Tool Bundle that includes licenses for a comprehensive suite of Synopsys EDA tools. The University Tool Bundle is provided to the universities through Synopsys University Programs. GMC works closely with Synopsys custom design flow tools to generate output data, for example Custom Designer layout capture tool is used for generation of Open Access layout view of the SRAM. Some of the data, such as Verilog and VHDL models are generated directly by GMC (fig. 10).

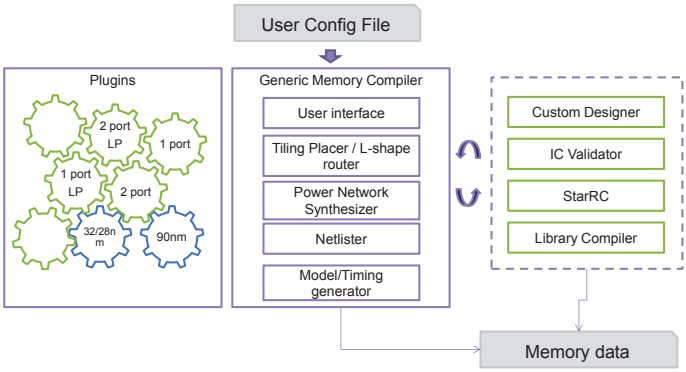

Fig. 10. Generic Memory Compiler system

GMC contains plugins with the following capabilities.

– Memories are generated for SAED 32/28nm and SAED 90nm abstract technologies.

– 4 types of memories are supported
  - Dual port SRAMs
  - Single port SRAMs
  - Low power dual port SRAMs
  - Low power single port SRAMs

– Number of words for memory is limited to be one of the following values:
  - 16, 32, 64, 128, 256 for dual port memories
  - 16, 32, 64, 128, for single port memories

D. Supporting documentation

In addition to the software Synopsys also provides user guide and step-by step tutorial to be used as a starting point for university users. Also software distribution includes demo examples of memory configurations that could be compiled out-of-the-box to test GMC operation and prepare custom memory configurations.

IV. Availability and Deployment

The Synopsys GMC is readily available through Synopsys University Programs for all eligible universities and customers.

GMC was used in the design of Synopsys ARC 600 embedded processor for 32/28nm abstract technology[4] at Synopsys Armenia Educational Department[4-6]. Memories generated with GMC complemented the set of memories included in the SAED 32/28nm EDK[4] and were used together with it. Also, the tool has been used in more than 25 universities around the world with positive feedback.

V. Conclusion

Educational software for automatic generation of SRAMs based on user-supplied configuration was developed. The software is developed to be free from intellectual property restrictions and is provided to universities include in Synopsys University Programs. GMC is developed to be highly extensible to include new memory types and technology nodes in the future. Memories generated with GMC were successfully used in complex configurable designs together with SAED EDKs and proved help in educational designs.

References

[1] D. Flynn, T. Wood, Ph. Dworsky, V. Melikyan, E. Babayan. Teaching IC Design with the ARM Cortex-M0 DesignStart Processor and Synopsys 90nm Educational Design Kit // Proceedings of the Interdisciplinary Engineering Design Education (IEDEC) Conference, Santa Clara, CA, USA, March 4-5, 2013.-P.36-38

[2] C. Valderrama, J. Laurent, P. Possa. "Convergence in reconfigurable embedded systems.", 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), IEEE, 2010, pp. 1144-1147.

[3] K. Lampaert. "Implementing high-performance, low-power embedded processors: Challenges and solutions: Designer track." Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on. IEEE, 2012.

[4] R. Goldman, K. Bartleson, T. Wood, V. Melikyan, E. Babayan, Synopsys' Low Power Design Educational Platform // Proceedings of the 9th European Workshop on Microelectronics Education (EWME 2012), Grenoble, France, 2012.-P.23-26

[5] R. Goldman, K. Bartleson, T. Wood, K. Kranen, C. Cao, V. Melikyan, G. Markosyan, "Synopsys' open educational design kit: capabilities, deployment and future," IEEE International Conference on Microelectronic Systems Education, 2009, pp. 20-24

[6] R. Goldman, K. Bartleson, T. Wood, K. Kranen, C. Cao, V. Melikyan, "Synopsys' Interoperable Process Design Kit," European Workshop on Microelectronics Education, 2010

[7] H. Homayoun, S. Avesta, J. Gaudiot, A. Veidenbaum. "Reducing Power in All Major CAM and SRAM-Based Processor Units via Centralized, Dynamic Resource Size Management." Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 19, no. 11, pp. 2081-2094, 2011

[8] G. Harling. "A DRAM Compiler for fully optimized memory instances." IEEE International Workshop on Memory Technology, Design and Testing, IEEE, 2001, pp. 3-8

[9] Y. Xu, G. Zhiqiang, H. Xiangqing. "A flexible embedded SRAM IP compiler." IEEE International Symposium on Circuits and Systems, 2007, pp. 785-787.

[10] Zy. Zhang, Ch. Chia-Cheng, and Jian-Bin Zheng. "A 90-nm CMOS embedded low power SRAM compiler." IEEE 8th International Conference on, IEEE, 2009. Pp. 625 - 628

[11] Wu, Sheng, et al. "A 65nm embedded low power SRAM compiler." IEEE 13th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010. IEEE, 2010. pp. 123-124