OpenRAM: An Open-Source Memory Compiler

Invited Paper

Matthew R. Guthaus[1], James E. Stine[2], Samira Ataei[2],
Brian Chen[1], Bin Wu[1], Mehedi Sarwar[2]
[1] Department of Computer Engineering, University of California Santa Cruz, Santa Cruz, CA 95064
{mrg, bchen12, bwu8}@ucsc.edu
[2] Electrical and Computer Engineering Department, Oklahoma State University, Stillwater, OK 74078
{james.stine, ataei, mehedis}@okstate.edu

ABSTRACT

Computer systems research is often inhibited by the availability of memory designs. Existing Process Design Kits (PDKs) frequently lack memory compilers, while expensive commercial solutions only provide memory models with immutable cells, limited configurations, and restrictive licenses. Manually creating memories can be time consuming and tedious and the designs are usually inflexible. This paper introduces OpenRAM, an open-source memory compiler, that provides a platform for the generation, characterization, and verification of fabricable memory designs across various technologies, sizes, and configurations. It enables research in computer architecture, system-on-chip design, memory circuit and device research, and computer-aided design.

1. INTRODUCTION

Static Random Access Memories (SRAMs) have become a standard component embedded in all System-on-Chip (SoC), Application-Specific Integrated Circuit (ASIC), and microprocessor designs. Their wide application leads to a variety of requirements in circuit design and memory configuration. However, manual design is too time consuming. The regular structure of memories leads well to automation that produces size and configuration variations quickly, but developing this with multiple technologies and tool methodologies is challenging. In addition, memory designs play a significant role in overall system performance and costs, so optimization is important. Thus, a memory compiler is a critical tool.

Most academic ICs design methodologies are limited by the availability of memories. Many standard-cell Process Design Kits (PDKs) are available from foundries and vendors, but these PDKs frequently do not come with memory arrays or memory compilers. If a memory compiler is freely available, it often only supports a generic process technology that is not fabricable. Due to academic funding restrictions, commercial industry solutions are often not feasible for researchers. In addition, these commercial solutions are limited in customization of the memory sizes and specific components of the memory. PDKs may have the options to request "black box" memory models, but these are also not modifiable and have limited available configurations. These restrictions and licensing issues make comparison and experimentation with real world memories impossible.

Academic researchers are able to design their own custom memories, but this can be a tedious and time-consuming task and may not be the intended purpose of the research. Frequently, the memory design is the bare minimum that the research project requires, and, because of this, the memory designs are often inferior and are not optimized. In memory research, peripheral circuits are often not considered when comparing memory performance and density. The lack of a customizable compiler makes it difficult for researchers to prototype and verify circuits and methodologies beyond a single row or column of memory cells.

The OpenRAM project aims to provide an open-source memory compiler development framework for memories. It provides reference circuit and physical implementations in a generic 45nm technology and fabricable Scalable CMOS (SCMOS), but it has also been ported to several commercial technology nodes using a simple technology file. OpenRAM also includes a characterization methodology so that it can generate the timing and power characterization results in addition to circuits and layout while remaining independent of specific commercial tools. Most importantly, OpenRAM is completely user-modifiable since all source code is open source at:

https://openram.soe.ucsc.edu/

The remainder of this paper is organized as follows: Section 2 provides a background on previous memory compilers. Section 3 presents the reference memory architecture in OpenRAM. Section 4 specifically introduces the implementation and main features of the OpenRAM memory compiler. In Section 5, an analysis of the area, timing and power is shown for different sizes and technologies of memory. Finally, the paper is summarized in Section 6.

2. BACKGROUND

Memory compilers have been used in Electronic Design Automation (EDA) design flows to reduce the design time long before contemporary compilers [2, 9]. However, these compilers were generally not portable as they were nothing more than quick scripts to aid designers. Porting to a new technology essentially required rewriting the scripts. However, the increase in design productivity when porting

ICCAD '16, November 07 - 10, 2016, Austin, TX, USA

ISBN 978-1-4503-4466-1/16/11…$15.00

DOI: http://dx.doi.org/10.1145/2966986.2980098

designs between technologies has led to more research on memory array compilers [3, 8, 14, 22].

As technology entered the Deep Sub-Micron (DSM) era, memory designs became one of the most challenging parts of circuit design due to decreasing static noise margins (SNM), increasing fabrication variability, and increasing leakage power consumption. This increased the complexity of memory compilers dramatically as they had to adapt to the ever-changing technologies. Simultaneously, design methodologies shifted from silicon compilers to standard cell place and route methods which required large optimized libraries. During this time, industry began using third-party suppliers of standard cell libraries and memory compilers that allowed their reuse to amortize development costs. These next-generation memory compilers provided silicon-verification that allowed designers to focus on their new design contribution rather than time-consuming tasks such as memory generation.

Contemporary memory compilers have been widely used by industry, but the internal operation is typically hidden. Several prominent companies and foundries have provided memory compilers to their customers. These memory compilers usually allow customers to view front-end simulation, timing/power values, and pin locations after a license agreement is signed. Back-end features such as layout are normally supplied directly to the fab and are only given to the user for a licensing fee.

Specifically, Global Foundries offers front-end PDKs for free, but not back-end detailed views [6]. Faraday Technologies provides a "black box" design kit where users do not know the details of the internal memory design [5]. Dolphin Technology offers closed-source compilers which can create RAMs, ROMs, and CAMs for a variety of technologies [4]. The majority of these commercial compilers do not allow the customer to alter the base design, are restricted by the company's license, and usually require a fee. This makes them virtually unavailable and not useful for many academic research projects.

In addition to memory compilers provided by industry, various research groups have released scripts to generate memories. However, these designs are not silicon verified and are usually only composed of simple structures. For example, FabMem is able to create small arrays, but it is highly dependent on the Cadence design tools [15]. The scripts do not provide any characterization capability and cannot easily integrate with commercial place and route tools.

Another recent, promising solution for academia is the Synopsys Generic Memory Compiler (GMC) [7]. The software is provided with sample generic libraries such as Synopsys' 32/28nm and 90nm abstract technologies and can generate the entire SRAM for these technologies. The GMC generates GDSII layout data, SPICE netlists, Verilog and VHDL models, timing/power libraries, and DRC/LVS verification reports. GMC, however, is not recommended for fabrication since the technologies it supports are not real. Its sole purpose is to aid students in VLSI courses to learn about using memories in design flows.

There have been multiple attempts by academia to implement a memory compiler that is not restricted: the Institute of Microelectronics' SRAM IP Compiler [22], School of Electronic Science and Engineering at Southeast University's Memory IP Compiler [11], and Tsinghua University's Low Power SRAM Compiler [21]. These are all methodologies and design flows for a memory compiler, but there are no public releases.

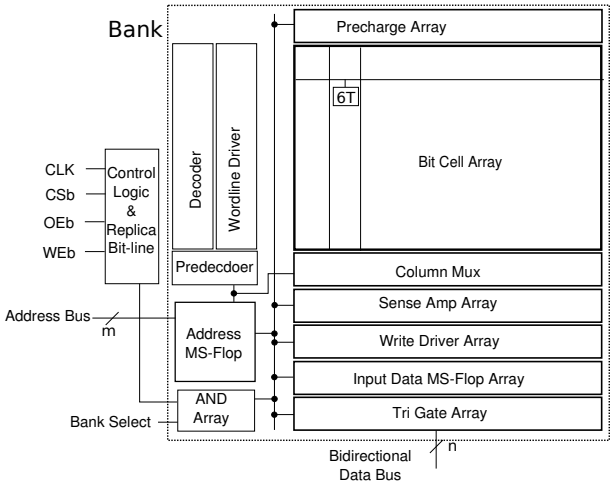


Figure 1: An OpenRAM SRAM consists of a bit-cell array along with decoder, reading and writing circuitry and control logic timed with a replica bit-line.

3. ARCHITECTURE

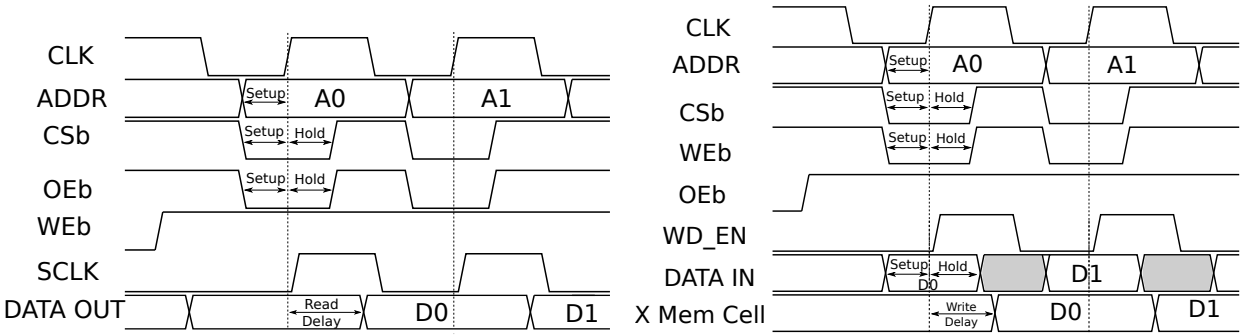
The OpenRAM SRAM architecture is based on a bank of memory cells with peripheral circuits and control logic as illustrated in Figure 1. These are further refined into eight major blocks: the bit-cell array, the address decoder, the word-line drivers, the column multiplexer, the pre-charge circuitry, the sense amplifier, the write drivers, and the control logic.

Bit-cell Array: In the initial release of OpenRAM, the 6T cell is the default memory cell because it is the most commonly used cell in SRAM devices. 6T cells are tiled together with abutting word- and bit-lines to make up the memory array. The bit-cell array's aspect ratio is made as square as possible using multiple columns of data words. The memory cell is a custom designed library cell for each technology. Other types of memory cells, such as 7T, 8T, and 10T cells, can be used as alternatives to the 6T cell.

Address Decoder: The address decoder takes the row address bits as inputs and asserts the appropriate word-line so that the correct memory cells can be read from or written to. The address decoder is placed to the left of the memory array and spans the array's vertical length. Different types of decoders can be used such as an included dynamic NAND decoder, but OpenRAM's default option is a hierarchical CMOS decoder.

Word-Line Driver: Word-line drivers are inserted between the address decoder and the memory array as buffers. The word-line drivers are sized based on the width of the memory array so that they can drive the row select signal across the bit-cell array.

Column Multiplexer: The column multiplexer is an optional block that uses the lower address bits to select the associated word in a row. The column mux is dynamically generated and can be omitted or can have 2 or 4 inputs. Larger column muxes are possible, but are not frequently used in memories. There are options for a multi-level tree



(a) Read operation timing

(b) Write operation timing

Figure 2: OpenRAM uses a synchronous SRAM interface using a system clock (clk) along with control signals: output enable (OEb), chip select (CSb) and write enable (WEb).

mux as well.

Bit-line Precharge: This circuitry pre-charges the bit-lines during the first phase of the clock for read operations. The precharge circuit is placed on top of every column in the memory array and equalizes the bit-line voltages so that the sense amplifier can sense the voltage difference between the two bit-lines.

Sense Amplifier: A differential sense amplifier is used to sense the voltage difference between the bit-lines of a memory cell while a read operation is performed. The sense amplifier uses a bit-line isolation technique to increase performance. The sense amplifier circuitry is placed below the column multiplexer or the memory array if no column multiplexer is used. There is one sense amplifier for each output bit.

Write Driver: The write drivers send the input data signals onto the bit-lines for a write operation. The write drivers are tri-stated so that they can be placed between the column multiplexer/memory array and the sense amplifiers. There is one write driver for each input data bit.

Control Logic: The OpenRAM SRAM architecture incorporates a standard synchronous memory interface using a system clock (clk). The control logic uses an externally provided, active-low output enable (OEb), chip select (CSb), and write enable (WEb) to combine multiple SRAMs into a larger structure. Internally, the OpenRAM compiler can have 1, 2, or 4 memory banks to amortize the area/power cost of control logic and peripheral circuitry.

All of the input control signals are stored using master-slave (MS) flip-flops (FF) to ensure that the signals are valid for the entire clock cycle. During a read operation, data is available after the negative clock edge (second half of cycle) as shown in Figure 2(a). To avoid dead cycles which degrade performance, a Zero Bus Turn-around (ZBT) technique is used in OpenRAM timing. The ZBT enables higher memory throughput since there are no wait states. During ZBT writes, data is set up before the negative clock edge and is captured on the negative edge. Figure 2(b) shows the timing for input signals during the write operation.

The internal control signals are generated using a replica bit-line (RBL) structure for the timing of the sense amplifier enable and output data storage [1]. The RBL turns on the sense amplifiers at the exact time in presence of process variability in sub-100nm technologies.

4. IMPLEMENTATION

OpenRAM's methodology is implemented using an object-oriented approach in the Python programming language. Python is a simple, yet powerful language that is easy to learn and very human-readable. Moreover, Python enables portability to most operating systems. OpenRAM has no additional dependencies except a DRC/LVS tool, but that is disabled with a warning if the tools are unavailable.

In addition to system portability, OpenRAM is also translatable across numerous process technologies. This is accomplished by using generalized routines to generate the memory based on common features across all technologies. To facilitate user modification and technology interoperability, OpenRAM provides a reference implementation in 45nm FreePDK45 [17] and a fabricable option using the MOSIS Scalable CMOS (SCMOS) design rules [13]. FreePDK45 uses many design rules found in modern technologies, but is non-fabricable, while SCMOS enables fabrication of designs using the MOSIS foundry services. SCMOS is not confidential and an implementation using it is included, however, it does not include many advanced DSM design rules. OpenRAM has also been ported to other commercial technologies, but these are not directly included due to licensing issues.

OpenRAM's framework is divided into "front-end" and "back-end" methodologies as shown in Figure 3. The front-end has the compiler and the characterizer. The compiler generates SPICE models and its GDSII layouts based on user inputs. The characterizer calls a SPICE simulator to produce timing and power results. The back-end uses a spice netlist extracted from the GDSII layout using to generate annotated timing and power models.

4.1 Base Data Structures

The design modules in OpenRAM are derived from the *design* class (design.py). The design class has a name, a SPICE model (netlist), and a layout. Both the SPICE model and the layout inherit their capabilities from a hierarchical class. The design class also provides inherited functions to perform DRC and LVS verification of any sub-design for hierarchical debugging.

The design class derives from the *spice* class (hierarchy_spice.py) which has a data structure to maintain the circuit hierarchy. This class maintains the design instances, their pins, and their connections as well as helper functions to maintain the structure and connectivity of the circuit hier-

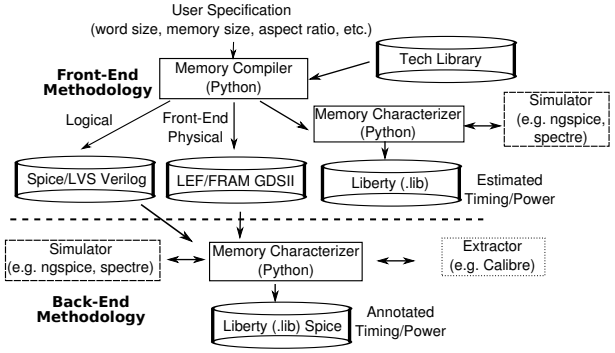


Figure 3: Overall Compilation and Characterization Methodology

archy.

The design class also derives from a *layout* class (hierarchy_layout.py). This class has a list of physical instances of sub-modules in the layout and a structure for simple objects such as shapes and labels in the current hierarchy level. In addition, there are helper functions that maintain the physical layout structures.

OpenRAM has an integrated, custom GDSII library to read, write, and manipulate GDSII files. The library, originally called GdsMill [20], has been modified, debugged, and extended for OpenRAM. Full rights were given to include the GdsMill source with OpenRAM, but to make the interfacing easier and porting to other physical layout databases possible, OpenRAM implements a *geometry* wrapper class (geometry.py) that abstracts the GdsMill library.

4.2 Technology and Tool Portability

OpenRAM is technology-independent by using a technology directory that includes the technology's specific information, rules, and library cells. Technology parameters such as the design rule check (DRC) rules and the GDS layer map are required to ensure that the dynamically generated designs are DRC clean. Custom designed library cells such as the memory cell and the sense amplifier are also placed in this directory. A very simple design rule parameter file has the most important design rules for constructing basic interconnect and transistor devices. FreePDK45 and SCMOS reference technologies are provided.

OpenRAM uses some custom-designed library primitives as technology input. Since density is extremely important, the following cells are pre-designed in each technology: 6T cell, sense amplifier, master-slave flip-flop, tri-state gate, and write driver. All other cells are generated on-the-fly using parameterizable transistor and gate primitives.

OpenRAM can be used for various technologies since it creates the basic components of memory designs that are common over these technologies. For technologies that have specific design requirements, such as specialized well contacts, the user can include call-back helper functions in the technology directory. This is done so that the main compiler remains free of dependencies to specific technologies.

OpenRAM has two functions that provide a wrapper interface with DRC and LVS tools. These two functions perform DRC and LVS using the GDSII layout and SPICE netlist files. Since each DRC and LVS tool has different output, this routine is customized per tool to parse DRC/LVS reports and return the number of errors while also outputting debug information. These routines allow flexibility of any DRC/LVS tool, but the default implementation calls Calibre nmDRC and nmLVS. In OpenRAM, both DRC and LVS are performed at all levels of the design hierarchy to enhance bug tracking. DRC and LVS can be disabled for improved run-time or if tool licenses are not available.

Table 1: Dependencies required for sub-modules

Variable	Equation
`Total Bits`	$word\_size * num\_words$
`Words Per Row`	$\sqrt{(num\_words)/word\_size}$
`Num of Rows`	$num\_words/words\_per\_row$
`Num of Cols`	$words\_per\_row * word\_size$
`Col Addr Size`	$\log_2(words\_per\_row)$
`Row Addr Size`	$\log_2(num\_of\_rows)$
`Total Addr Size`	$row\_addr\_size + col\_addr\_size$
`Data Size`	$word\_size$
`Num of Bank`	num_banks

4.3 Class Hierarchy

4.3.1 High-Level Classes

The *openram* class (openram.py) organizes execution and instantiates a single memory design using the *sram* class. It accepts user-provided parameters to generate the design, performs the optional extraction, performs characterization, and saves the resulting design files.

The *sram* class (sram.py) decides the appropriate internal parameter dependencies shown in Table 1. They are dependent on the user-desired data word size, number of words, and number of banks. It is responsible for instantiation of the single control logic module which controls the SRAM banks. The control logic ensures that only one bank is active in a given address range.

The *bank* class (bank.py) does the bulk of the non-control memory layout. It instantiates 1, 2, or 4 bit-cell arrays and coordinates the row and column address decoders along with their pre-charge, sense amplifiers, and input/output data flops.

4.3.2 Block Classes

Every other block in the memory design has a class for its base cell (e.g., sense_amplifier.py) and an array class (e.g., sense_amplifier_array.py) that is responsible for tiling the base cell. Each class is responsible for physically placing and logically connecting its own sub-circuits while passing its dimensions and port locations up to higher-level modules.

4.3.3 Low-Level Classes

OpenRAM provides parameterized transistor and logic gate classes that help with technology portability. These classes generate a technology-specific transistor and simple logic gate layouts so that many modules do not rely on library cells. It is also used when a module such as the write driver needs transistor sizing to optimize performance. The parameterized transistor (ptx.py) generates a basic transistor of specified type and size. The parameterized transistor class is used to provide several parameterized gates including pinv.py, nand2.py, nand3.py, and nor2.py.

4.4 Characterization

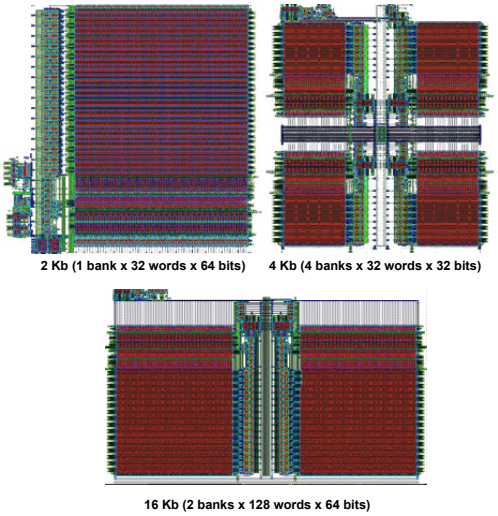


Figure 4: Single bank and multi-bank SRAMs (not to scale) use symmetrical bank placement to share peripheral circuitry and equalize signal delays.

OpenRAM includes a memory characterizer that measures the timing and power characteristics through SPICE simulation. The characterizer has four main stages: generating the SPICE stimulus, running the circuit simulations, parsing the simulator's output, and producing the characteristics in a Liberty (.lib) file.

The stimulus is written in standard SPICE format and can be used with any simulator that supports this. The stimulus only uses the interface of the memory (e.g., bi-directional data bus, address bus, and control signals) to perform "black box" timing measurements.

Results from simulations are used to produce the average power, setup/hold times, and timing delay of the memory design. Setup and hold times are obtained by analyzing the flip-flop library cell because OpenRAM uses a completely synchronous input interface. The setup time, hold time, and delay are found using a fast bisection search.

4.5 Unit Tests

Probably the most important feature of OpenRAM is the set of thorough regression tests implemented with the Python unit test framework. These unit tests allow users to add features and easily verifying if functionality is broken. The tests also work in multiple technologies so they can guide users when porting to new technologies. Every module has its own regression test and there are also regression tests for memory functionality, verifying library cells, timing characterization, and technology verification.

5. RESULTS

Figure 4 shows several different SRAM layouts generated by OpenRAM in FreePDK45. OpenRAM can generate single bank and multi-bank SRAM arrays. Banks are symmetrically placed to have the same delay for data and address while sharing peripheral blocks such as decoders.

Figure 5 shows the memory area of different total size and data word width memories in both FreePDK45 and SCMOS. As expected, the smaller process technology (45nm)

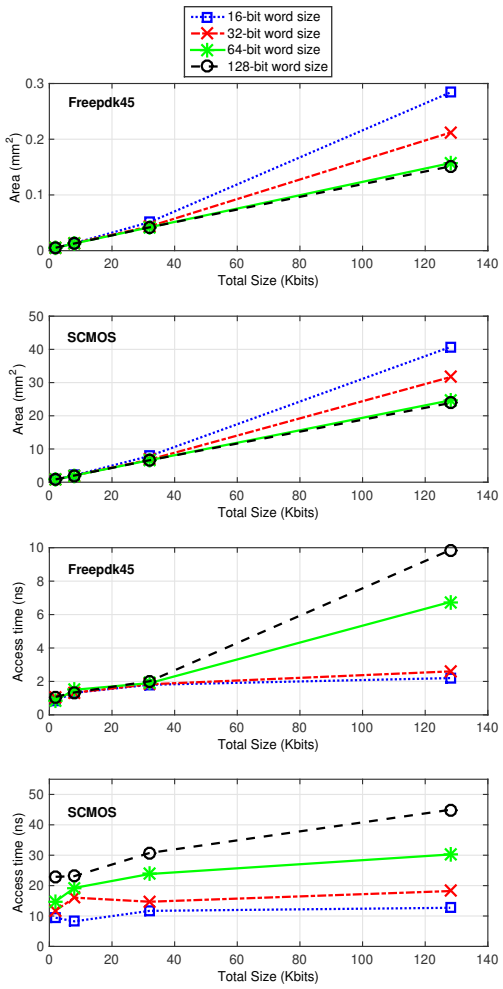


Figure 5: OpenRAM provides high-density memories in multiple technologies and sizes with corresponding characterized delays.

has lower total area overall but the trends are similar in both technologies.

Figure 5 also shows the access time of different size and data word width in FreePDK45 and SCMOS. Increasing the memory size generally increases the access time; long bit-lines and word-lines increase the access time by adding more parasitic capacitance and resistance. Since OpenRAM uses multiple banks and column muxing, it is possible to have a smaller access time for larger memory designs, but this will sacrifice density.

Comparison of power consumption and read access time of different memories is a bit more complicated to make a conclusion, because there are many trade-offs. Power and performance are highly dependent on circuit style (CMOS, ECL, etc.), memory organization (more banks is faster but sacrifices density), and the optimization goal: low-power or high-performance. In general, OpenRAM has reasonable trade-off between the two and can be customized by using an alternate sense amplifiers, decoders, or overall dimensional organization. Table 2 compares the bit-density

Table 2: OpenRAM has high density compared to other published memories in similar technologies.

Ref.	Feature Size	Tech.	Density [Mb/mm^2]
[10]	65 nm	CMOS	0.7700
[19]	45 nm	CMOS	0.3300
[12]	40 nm	CMOS	0.9400
OpenRAM	45 nm	FreePDK45	0.8260
[23]	0.5 um	CMOS	0.0036
[18]	0.5 um	BiCMOS	0.0020
[16]	0.5 um	CMOS	0.0050
OpenRAM	0.5 um	SCMOS	0.0050

of OpenRAM against published designs using similar technology nodes. The results show the benefit of technology scaling and that OpenRAM has very good density in both technologies. As a comparison, a 76ns SRAM consumes 3.9mW [16] while OpenRAM is much faster at 44.9ns but consumes 115mW for the same size.

6. CONCLUSIONS

This paper introduced OpenRAM, an open-source and portable memory compiler. OpenRAM generates the circuit, functional model, and layout of variable-sized SRAMs. In addition, a memory characterizer provides synthesis timing/power models.

The main motivation behind OpenRAM is to promote and simplify memory-related research in academia. Since OpenRAM is open-sourced, flexible, and portable, this memory compiler can be adapted to various technologies and is easily modified to address specific design requirements. Therefore, OpenRAM provides a platform to implement and test new memory designs.

Designs are currently being fabricated to test designs using the OpenRAM framework in SCMOS. We are also continuously introducing new features, such as non-6T memories, variability characterization, word-line segmenting, characterization speed-up, and a graphical user interface (GUI). We hope to engage an active community in the future development of OpenRAM.

7. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1205685 and CNS-1205493. Many students have contributed to the project throughout their studies including Jeff Butera, Tom Golubev, Seokjoong Kim, Matthew Gaalswyk, and Son Bui.

8. REFERENCES

[1] B. S. Amrutur and M. A. Horowitz. A replica technique for wordline and sense control in low-power SRAMs. *JSSC*, 33(8):1208–1219, Aug 1998.

[2] R. Broderson. *Anatomy of a Silicon Compiler*. Springer, 1992.

[3] A. Cabe, Z. Qi, W. Huang, Y. Zhang, M. Stan, and G. Rose. A flexible, technology adaptive memory generation tool. *Cadence CDNLive*, 2006.

[4] Dolphin Technology. Memory products. http://www.dolphin-ic.com/memory-products.html, 2015.

[5] Faraday Technologies. Memory compiler architecture. http://www.faraday-tech.com/html/Product/IPProduct/LibraryMemoryCompiler/index.htm, 2015.

[6] Global Foundries. ASICs. http://www.globalfoundries.com/technology-solutions/asics, 2015.

[7] R. Goldman, K. Bartleson, T. Wood, V. Melikyan, and E. Babayan. Synopsys' educational generic memory compiler. In *EWME*, pages 89–92, May 2014.

[8] T.-H. Huang, C.-M. Liu, and C.-W. Jen. A high-level synthesizer for VLSI array architectures dedicated to digital signal processing. In *International Conference on Acoustics, Speech and Signal Processing*, pages 1221–1224, 1991.

[9] D. Jahannsen. Bristle blocks: A silicon compiler. In *DAC*, pages 195–198, 1979.

[10] K. Kushida et al. A 0.7v single-supply SRAM with 0.495 um^2 cell in 65nm technology utilizing self-write-back sense amplifier and cascaded bit line scheme. In *ISVLSI*, pages 46–47, June 2008.

[11] C. Ming and B. Na. An efficient and flexible embedded memory IP compiler. In *CyberC*, pages 268–273, Oct 2012.

[12] S. Miyano et al. Highly energy-efficient SRAM with hierarchical bit line charge-sharing method using non-selected bit line charges. *JSSC*, 48(4):924–931, Apr 2013.

[13] MOSIS. MOSIS scalable CMOS (SCMOS). https://www.mosis.com/files/scmos/scmos.pdf, 2015.

[14] P. Poechmueller, G. K. Sharma, and M. Glesner. A CAD tool for designing large, fault-tolerant VLSI arrays. In *GLSVLSI*, 1991.

[15] T. Shah. FabMem: A multiported RAM and CAM compiler for superscalar design space exploration. Master's thesis, North Carolina State University, 2010.

[16] N. Shibata, H. Morimura, and M. Watanabe. A 1-V, 10-MHz, 3.5-mw, 1-Mb MTCMOS SRAM with charge-recycling input/output buffers. *JSSC*, 34(6):866–877, Jun 1999.

[17] J. E. Stine et al. FreePDK: An open-source variation-aware design kit. In *MSE*, pages 173–174, June 2007.

[18] N. Tamba et al. A 1.5-ns 256-kb BiCMOS SRAM with 60-ps 11-k logic gates. *JSSC*, 48(11):1344–1352, Nov 1994.

[19] S. O. Toh, Z. Guo, T. K. Liu, and B. Nikolic. Characterization of dynamic SRAM stability in 45 nm CMOS. *JSSC*, 46(11):2702–2712, Nov 2011.

[20] M. Wieckowski. *GDS Mill User Manual*, 2010.

[21] S. Wu, X. Zheng, Z. Gao, and X. He. A 65nm embedded low power SRAM compiler. In *DDECS*, pages 123–124, April 2010.

[22] Y. Xu, Z. Gao, and X. He. A flexible embedded SRAM IP compiler. In *ISCAS*, pages 3756–3759, May 2007.

[23] K. Yamaguchi et al. A 1.5-ns access time, 78 um^2 memory-cell size, 64-kb ECL-CMOS SRAM. *JSSC*, 27(2):167–174, Feb 1992.