MemGen: An Open-Source Framework for Autonomous Generation of Memory Macros

Sumanth Kamineni, Shourya Gupta, Benton H. Calhoun

University of Virginia, Charlottesville, USA

2021 IEEE Custom Integrated Circuits Conference (CICC) | 978-1-7281-7581-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/CICC51472.2021.9431501

Static Random-Access Memories (SRAMs) form an integral part of System-on-Chips (SoCs), wireless sensor nodes and other Internet-of-Things (IoT) devices. SRAM has a large, multi-dimensional design space that includes different bit-cell designs, peripheral assist-circuit designs, operating voltages, and frequency targets. Custom design of memories for any application in this broad design space is a tedious, iterative, and time-consuming process. Commercial memory compilers (CMCs) [1]-[3] provide an automated alternative, but CMCs often have a limited design space usually emphasizing high performance and may not be readily available due to cost or licensing issues, especially for newer technologies. To address these issues and allow easy, autonomous, and versatile generation of optimized memory macros, we present MemGen ("Memory Macro Generator"), an open-source memory macro generation framework that creates tapeout-ready integrated memories across a broad range of voltages, frequencies, and capacities. The new framework uses a template and cell-based design methodology and leverages the conventional digital tool flow to generate optimized memories based on high-level user intent, making it highly modular, process-portable, and easily augmentable. We demonstrate the framework's capability by generating multiple memories for various use cases in a planar 65nm and in a 12nm FinFET process. MemGen is also verified by fabricating 64kbit and 128kbit 65nm auto-generated memories.

Fig. 1 shows the design and application space covered by MemGen, CMCs, and other academic reported compilers [4]-[6]. CMCs offer many features, but they have a narrower design space, thereby limiting their application especially for power constrained circuits. Other academic compilers also offer a suite of various features, but they fall short on many other features as shown in Table 1 in Fig.1. MemGen offers to generate memories in other regions of the design space where CMCs cannot be used, such as low voltage designs for ultra-low power applications. Unlike other CMCs and academic compilers, MemGen can generate memories by performing device-circuit-architecture co-design. This is enabled by a closed-loop integrated flow that involves the translation of high-level user intent (e.g., voltage, frequency, and capacity) into an optimized SRAM layout through tightly coupled design-space exploration, optimization, and layout generation. Additionally, to the best of the authors' knowledge, MemGen is the first open-source memory compiler to support advanced FinFET processes. The framework is implemented in python and the source code is available from github.com/idea-fasoc/fasoc/tree/master/generators/memory-gen

Fig. 2 shows the high-level overview of the MemGen framework and the critical steps (1, 2 and 3) involved in generating a memory. To enable MemGen to create memories in a specific technology, a one-time Process Design Kit (PDK) setup is required. The first step in the setup process is the PDK characterization, which involves running device-level simulations to extract information such as transistor behavior, bit-cell characteristics, metal parasitics, threshold voltage (V_T), and FO4 delay using a scripted template-based methodology that separates the technology dependent and independent aspects of the circuits. The second step involves the generation of aux-cells. Aux-cells are small SRAM peripheral circuits that extend the standard-cell library and provide specific analog functionality required for memory operation. These aux-cells vary in terms of drive strengths, circuit topology, and V_T; and include all files which are required as a part of a conventional synthesis and Auto-Place-Route (APR) flow. The process setup completes by generating PDK-specific SRAM Hierarchical Memory Model (HMM), which allows quick estimation of SRAM global figures of merits (FOMs) energy(E) and delay(D), thereby circumventing the need for resource intensive and time-consuming complete circuit simulations [7].

MemGen creates SRAMs optimized for energy and area by minimizing a weighted cost function $C(x)$, subject to a user desired clock period (T_{req}) constraint. Formally, this can be defined as $\min C(x)\ such\ that\ T_{op} \leq T_{req}$. $C(x)$ is calculated as

$$C(x) = W_E E(x) + W_A A(x) = W_E \sum_{i=1}^{c} e_i(x) + W_A \sum_{i=1}^{c} a_i(x) \quad (1)$$

where T_{op} is the operating time of SRAM under design, x is a vector of n optimization variables $x_1, x_2, \ldots, x_n$, $c \rightarrow$ number of sub-components, $E(x) \rightarrow$ energy-per-access and $A(x) \rightarrow$ SRAM area, $e_i(x) \rightarrow$ component's energy, $a_i(x) \rightarrow$ components' area, $W_E \rightarrow$ energy weight, $W_A \rightarrow$ area weight. The user may vary energy and area weights as per their application priorities.

MemGen considers the number of banks (B), rows (R), and columns (C) per bank at the architectural level and device sizing, device type (High-Low-Reg. V_T), and component topology at the circuit level as optimization variables to minimize the cost function $C(x)$. For a given user intent, the optimization process starts by generating a set of optimal FOM tradeoff points using the HMM and tuning architectural knobs, as shown in Fig. 3(a). These optimal points prune the design space from an architectural perspective. If any of the optimal points satisfy the user intent, the framework moves to the macro generation phase. Otherwise, with the optimal points as the starting points, the optimizer proceeds with an iterative design space exploration by assessing the components affecting the FoMs and tuning the circuit knobs to minimize $C(x)$ until T_{req} constraint is met, as shown in Fig. 3(a)-(d). It performs sensitivity analysis to calculate the optimization's profitability and selects the auxcell corresponding to the knob that achieves the highest profit. The design space can be extended by adding new architectures or new aux-cells using scripted circuit templates, making MemGen modular and easily augmentable.

The final step involves creating the SRAM macro layout using the macro generator shown in Fig.2. Macro generator streamlines and automates the layout generation flow by wrapping several sub-generators required for digital tool flow. It creates the required inputs and the tool scripts for each stage by analyzing the previous stage results. MemGen employs a hierarchical tree-based multi-bank architecture by default, as shown in Fig. 4(a), but users can define a new architecture through a scripted template. Fig 4(b) illustrates synchronous read and write memory transactions with input and output signals. The Verilog and timing-constraints generators create synthesizable bespoke structural Verilog and timing constraints scripts using parameterized templates and according to memory architecture. Synthesizer runs the synthesis to create Register Transfer Language (RTL) netlists and other files required to run the APR. In the APR, the floor planner analyzes synthesis outputs and performs the design floor-planning and power-planning; the placer and router places all the design instances, generates the clock tree, and routes the design to form a complete multi-bank memory macro. As part of the final design signoff process, MemGen performs a functional and performance check on the final output to ensure that the design operates according to the user's intent.

Fig. 5 shows various MemGen auto-generated SRAM macro layouts and the simulated energy and delay values in two different PDKs, demonstrating MemGen's versatility. To experimentally verify the framework, we taped out two different SRAM macros (64kbit and 128kbit) auto-generated using MemGen for two user intents (1: sub-V_T operation, 50KHz. 2: super-V_T operation, 50MHz) in 65nm. Fig. 6 summarizes the power and frequency chip measurement results and the step-wise framework runtime breakdown for the generation process. The first design achieved 131nW at 100KHz and 0.5V, and the second design achieved a peak performance of 65 MHz with 2.09mW power at 1.2V. In contrast to [1]-[6], MemGen created memories that exceeded the targeted specs in broad design space, with a macro generation runtime of ≤138 min in each case.

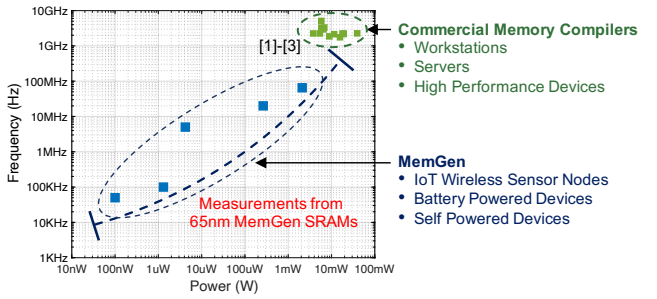
Acknowledgement:

This work was funded in part by the Defense Advanced Research Projects Agency (DARPA) under agreement no. FA8650-18-2-7844.

References:

[1]https://developer.arm.com.[2]http://www.globalfoundries.com.[3]https://www.synopsys.com [4] K. Chakraborty, et. al, IEEE TVLSI, Vol. 9, No. 2, pp. 352-364, April 2001.[5] M. Guthaus, et. al, IEEE ICCAD, Austin, TX, pp. 1-6, Nov 2016. [6] S. Ataei et. al, IEEE ASYNC, Hirosaki, Japan, pp. 1-8, 2019. [7] N. Liu et. al., ISVLSI, PA, USA, 2016, pp. 535-540.

(a)



(b)

Desired Features	This Work	CMCs [1]-[3]	Academic Reported Compilers		
			[4]	[5]	[6]
Open Source	Yes	No	No	Yes	Yes
Planar CMOS Support	Yes	Yes	Yes	Yes	Yes
FinFET Support	Yes	Yes	No	No	No
PDK Agnostic	Yes	No	No	Limited	Limited
Design Space Exploration & Optimization	Yes	No	No	No	No
Fabricable/Tapeout Ready	Yes	Yes	No	No	No
Silicon Verified	Yes	Yes	No	No	No

Fig. 1. (a) Target design space for MemGen (b) Comparison of MemGen with other compilers in terms of features and capability.

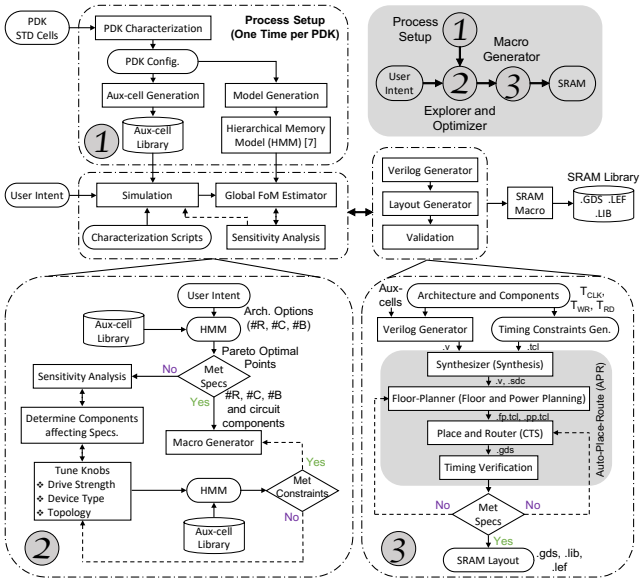


Fig. 2. MemGen Framework high-level overview and critical steps.

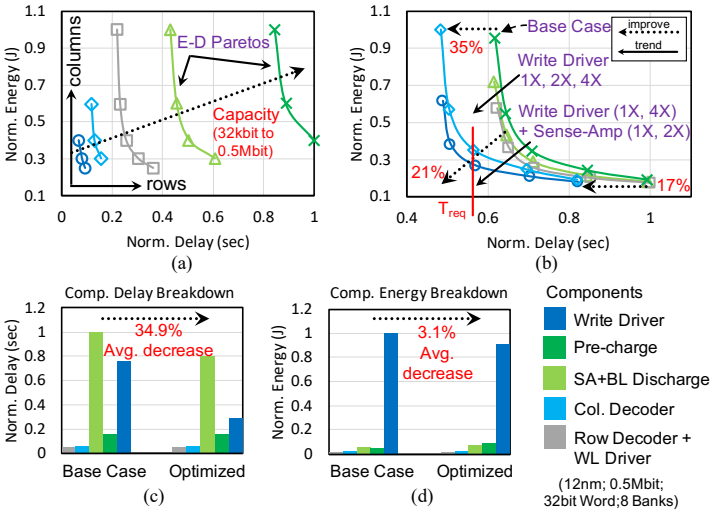


Fig. 3. (a) E and D pareto curves with varying capacity generated using HMM. (b) Pareto improvement using component level optimization. Component-wise breakdown of (c) D and (d) E.

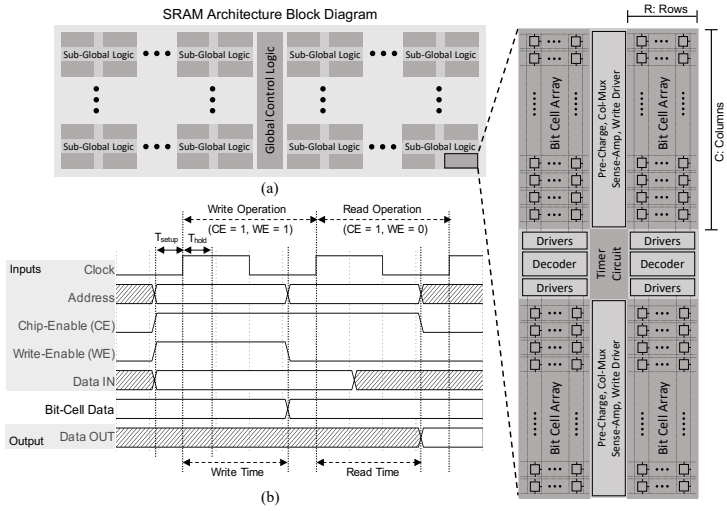
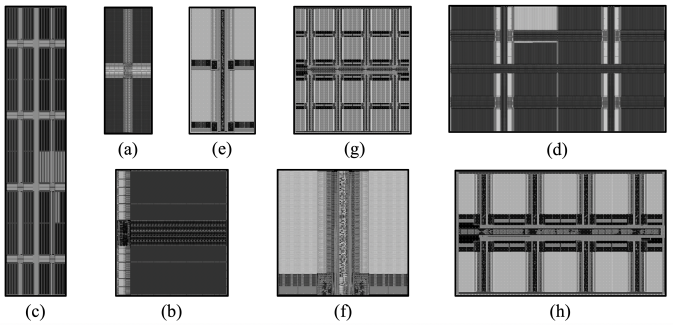


Fig. 4. (a) Block diagram of the default SRAM architecture employed in the MemGen (b) Timing diagram of the SRAM.



	65nm (Nominal V_{DD} = 1.2V)				12nm (Nominal V_{DD} = 0.8V)			
	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
Capacity (bit)	64K	64K	0.5M	0.5M	64K	64K	0.5M	0.5M
Area (mm^2)	0.1175	0.0895	1.045	0.8290	0.0109	0.00817	0.1152	0.0935
Rows	128	256	128	256	128	256	128	256
Columns	128	128	128	128	128	128	128	128
Banks	4	2	32	16	4	2	32	16
Energy (pJ)	13.1	15.6	62.4	78.3	3.49	4.64	27.89	37.13
Delay (ns)	8.3	9.7	12.2	15.3	1.31	2.48	2.40	3.57

65nm: Custom-made DRC compliant 6T bit-cell
12nm: Foundry 6T bit-cell

Fig. 5. Layouts of different SRAMs auto-generated using MemGen in planar 65nm and 12nm FinFET process.

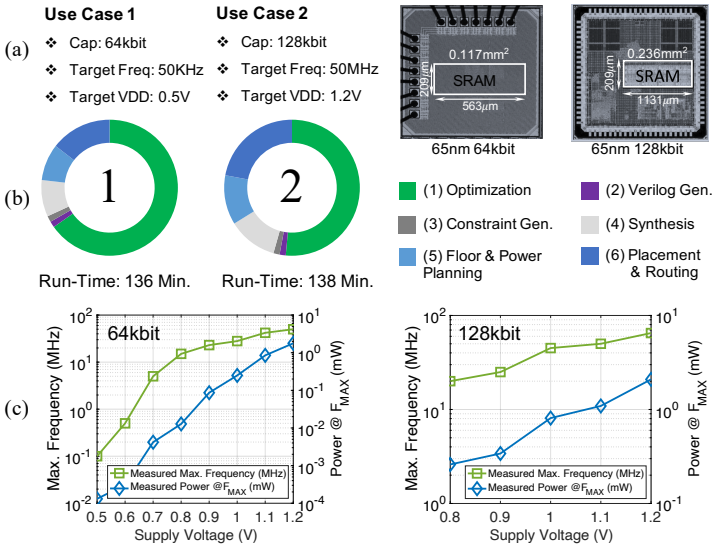


Fig. 6. (a) SRAM use cases (b) Framework runtime breakdown for use cases 1 and 2 (c) Frequency and power measurement results.