A Unified Memory Compiler for Synchronous and Asynchronous Circuits

Samira Ataei and Rajit Manohar
Computer Systems Lab, Yale University, New Haven, CT 06520
{samira.ataei, rajit.manohar}@yale.edu

Abstract—**This paper presents a unified memory compiler for synchronous and asynchronous circuits. This open-source memory compiler: (i) generates asynchronous pipelined SRAM modules with a bundled-data datapath and quasi-delay-insensitive control, (ii) provides a synchronous interface for asynchronous SRAMs to be utilized in clocked design, and (iii) generates built-in self-test engine with March C- algorithm to facilitate the embedded SRAM testing. This unified memory compiler which is based on AMC is a flexible, extensible, and portable platform that generates fabricable SRAM blocks in a broad range of sizes, configurations, and process nodes.**

I. Introduction

Embedded memory consistently plays a critical role in overall performance and power consumption of digital systems. Improving the memory access time and density, reducing its latency and power consumption and lowering its sensitivity to process variations are equally important factors and becoming more challenging to achieve in scaled technology nodes due to increased device variations.

Third party vendors and foundries provide digital designers with memory compilers to solve their on-chip memory requirement in SoC and ASIC designs. Although memory compilers greatly reduce the development costs, accessing a high quality memory compiler is still a major limitation in academia. AMC: An Asynchronous Memory Compiler [1] is the first open-source memory compiler that generates asynchronous SRAMs. SRAMs generated by AMC are fabricable and competitive with custom designed high-performance memories in different process nodes and can perform at their highest frequency and throughput [1].

AMC[1] geneartes asynchronous pipelined multi-bank SRAM modules with quasi delay-insensitive (QDI) control and bundled-data datapath. AMC was developed by starting with the OpenRAM [10], and then making major changes to all the circuits so as to support the generation of pipelined asynchronous SRAM. The description of SRAM architecture generated by AMC and the compiler usage can be find in [1], [2]. This paper introduces two new features recently added to AMC to make it a unified memory compiler for both synchronous and asynchronous circuit designs and to provide a self-test engine for evaluating SRAM macros.

First new feature is a synchronous interface that allows us to use an asynchronous SRAM in a clocked design. Asynchronous SRAMs are faster, consume less power and show less sensitivity to the process variation [1]. Using an asynchronous SRAM in a synchronous system can greatly improve the system's design metrics. AMC provides an efficient synchronous interface for its asynchronous SRAMs to be easily used in synchronous or hybrid synchronous-asynchronous designs.

Second new feature is the built-in self-test (BIST) engine that provides an automatic tester for SRAM. Embedded memory tests are the hardest type of digital circuit to test; they need enormous amount of input test patterns and readout of a huge number of cell information. Although it is possible to implement the testing functions in software, the fault coverage of software-implemented tests are low and these tests are normally slow and expensive to develop [3]. Moreover, Embedded memory tests must be quick and have very high fault coverage. Pad-limitation and at-speed testing with automatic test equipments make the memory test even more expensive and challenging. BIST functions in hardware are an attractive solution in embedded memory testing. Embedded memory with proper BIST engine is a reliable IP that saves effort and time in SoC design. BIST reduces the memory test time by an order of magnitude and allows an at-speed test for the entire memory while adds a negligible area overhead (1-3%). BIST engine of AMC uses march algorithm [4] as it is the most suitable testing algorithm for random access memories because of its higher fault coverage with shorter pattern sequence and small area overhead compared to other memory tests.

The rest of this paper is organized as follows. Section II explains the self-timed interface and architecture of asynchronous SRAMs implemented by AMC. Section III explains how synchronous wrapper converts a fully asynchronous SRAM to a clocked memory. Details of BIST core and SRAM testing algorithm are shown in Section IV. Section V concludes the paper.

II. AMC: An Asynchronous Memory Compiler

AMC is an open-source memory compiler implemented in the Python programming language [1]. AMC generates fabricable GDSII layout data, standard SPICE netlists, Verilog models, functional and physical verification reports, abstract place and route LEF file and timing and power reports of different SRAM sizes and configurations. AMC can be ported to any technology node by including technology-specific rules and few pre-made library cells. AMC has been successfully ported to 28nm, 65nm and SCN3M_SUBM 0.5μm process

[1]https://github.com/asyncvlsi/AMC

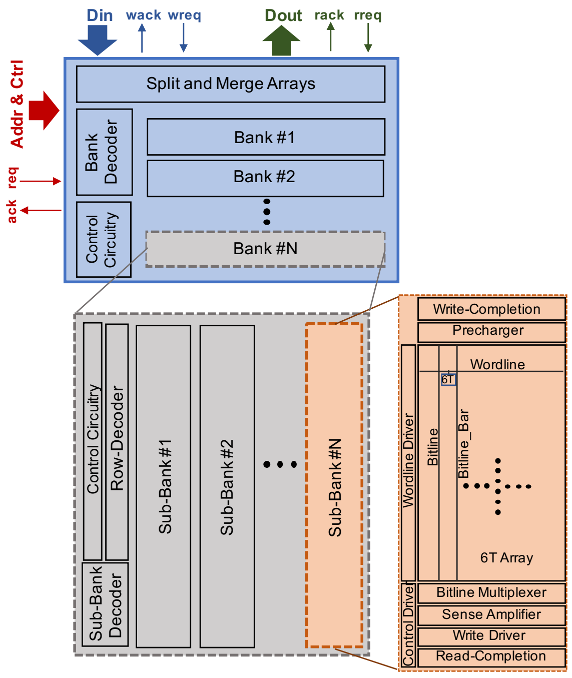

Fig. 1. An AMC multi-bank architecture with input and output channels. Each bank consists of sub-banks, decoders and control circuitry. Each sub-bank is compose of 2D bitcell array, read and write control circuits, completion detection, drivers and decoders.

technology. We are planing to port the compiler to more scaled technologies including 14nm FinFET.

By leveraging the design techniques introduced by the cache of MIPS processor [5], and other custom memory designs [6], [7], [8], [9], AMC creates pipelined SRAM macros with QDI control, bundled-data datapath and four-phase handshaking. All physical implementations and information in SCN3M_SUBM 0.5μm technology is part of the compiler as a reference implementation.

AMC's SRAM architecture is shown in an abstract view in Figure 1. In external interface of the multi-bank SRAM, *Addr*, *Ctrl*, *Din*, and *Dout* are the input address bus, input control bus, input data bus, and output data bus, respectively. The bundled-data encoding in the AMC interface uses multiple sets of independent request/acknowledge pairs.

AMC accepts user-provided parameters such as data word size, number of banks and sub-banks, and orientation of banks, physically places and logically connects all the modules to generate a multi-bank SRAM. The following are some of the supported features of AMC:

- Partitioning and floorplaning: in AMC each bank encapsulates multiple sub-banks of SRAM arrays. This design feature allows us to lower the memory access time and power consumption by breaking the SRAM array into smaller width segments. It also enables bigger SRAM banks without increasing the access time.
- Bank orientation: unlike tradition memory compilers that generate SRAM modules as hard-macros with fixed aspect ratios, AMC generates soft multi-bank layouts; banks can be placed in different orientation to get different aspect ratios for the SRAM. This feature allows designer to add the best matching SRAM layout to the rest of their design.
- Memory operations: AMC supports three operations each with different access time: read, write and read-modify-write (*rmw*). The *rmw* operation performs simultaneous read and write at a small cost compared to an individual read operation, and with less time and energy compare to two separate operations. The *rmw* operation allows memory to read a location and write it with a new value simultaneously while activating the decoder just once.
- Different memory cell layout support: AMC can generate SRAM arrays using both the foundry memory cells and hand-drawn memory cells. AMC has support for memory cells with different aspect ratios: tall-cell (conventional in older technologies) and thin-cell (needed for the design rules at 65nm and below).

III. Synchronous Interface for Asynchronous SRAM

Asynchronous circuits have the potential to work faster, consume less power and be more robust to delay and process variations. Therefore, to improve the design metrics, many modern systems contain some type of asynchronous circuits to perform part of their computation. In such hybrid systems, correctly crossing synchronous/asynchronous boundary is a challenging problem. The biggest challenge in asynchronous-synchronous interface is to minimizing the communication latency and its hardware overhead. Asynchronous circuit has to work faster than the clock rate of synchronous domain; asynchronous circuit has to be able to accept data tokens faster than the synchronous circuit produces these tokens without being stalled or entering a metastable state. Without enough care, the integration of asynchronous circuitry into a clocked system can lead to an unacceptable probability of synchronization failure.

In AMC, a single asynchronous memory bank can be described as a process that receives the address and control inputs, performs the read or write operation, and possibly transmits data to its environment. Therefore, a single asynchronous memory bank can be implemented as one process that synchronizes the handshake on the appropriate input and output channels and a number of such memory banks together makes a pipelined asynchronous multi-bank memory system. This results in asynchronous memories that can have arbitrary sizes while working at a constant peak frequency. Therefore, using the AMC's high-speed asynchronous memories can improve the overall system performance and throughput in synchronous designs.

By choosing the ”Add Synchronous Interface” option, AMC automatically adds an specialized synchronous interface with a small overhead hardware to the asynchronous SRAM module. This interface receives asynchronous data tokens, synchronizes them with a global clock and outputs valid synchronous data.

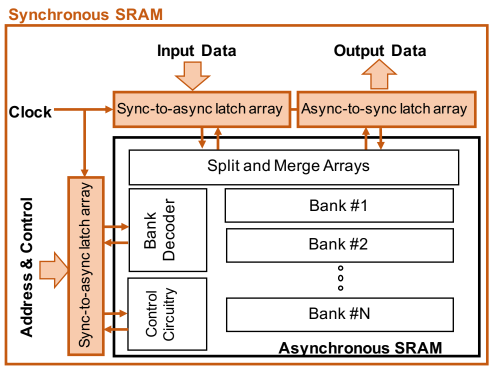

Fig. 2. Synchronous interface

This interface contains two types of boundaries: synchronous-to- asynchronous (for input data, address and controls) and asynchronous-to-synchronous (for output data). Generated synchronous SRAMs can perform at a potentially higher frequency compared to conventional clocked-SRAMs such as those generated by OpenRAM compiler [10]. In conventional synchronous SRAMs, sequence of operations needed for a memory operation (capturing the address and data bits, address decoding, bitline precharging and data driving/sensing) are bounded by clock edges. Precharging happens at first half of the clock cycle while data reading or writing happens at second half of the cycle. Using a 50% duty-cycle clock, combined with satisfying the setup-time and hold-time constraints on flip-flops doesn't allow the traditional synchronous memories to perform at their highest frequency. Generated synchronous SRAMs by AMC allows a performance gain in clocked systems without redesigning the entire chip.

The abstract view of synchronous SRAMs generated by AMC is shown in Figure 2. Top interface is similar to conventional synchronous single-port SRAMs. The inputs to this interface are asynchronous QDI signals and the outputs are data synchronized with the externally provided clock signal. This interface is independent of the width of input data. It can work with the global clock of system and gives designer the flexibility to store data in a fast and power-efficient asynchronous SRAM.

In generated synchronous SRAM, both read and write operations are self-timed with the four-phase handshake between synchronous wrapper and asynchronous SRAM. The read-modify-write operation is removed from AMC synchronous SRAMs, as such an operation would increase the clock cycle period in a synchronous memory. In the asynchronous case this operation only increases the cycle time when it is used. Therefore, AMC synchronous SRAMs support only read and write operations, each operation takes exactly one clock cycle and the clock frequency is determined by the propagation delay of critical path in a bank.

IV. Built-In Self-Test with March C- Algorithm

Almost all SoCs contain some type of embedded SRAM to increase data bandwidth and reduce hardware cost. Offering a low-cost test solution for embedded SRAMs is equally challenging and important as to create a reliable and high density SRAM array. Testing embedded SRAM is more difficult than testing commodity SRAMs because embedded SRAM array in a chip is surrounded by several logic blocks making it costly to access the SRAM from an external tester. In addition, increased speed and pad-limitation makes the SRAM testing more expensive. BIST minimizes the embedded SRAMs tester requirement and greatly reduces memory tester time throughout the test flow. A BIST engine with high fault coverage allows testing of large embedded memories on logic testers without added die area or performance-testing inaccuracies that may occur with off-chip testing methods.

A. AMC BIST architecture

The simplified architecture of the AMC BIST is shown in Figure 3. BIST engine includes the following circuit blocks:

- address-generator or address-stepper: address decoder fault-detection is covered by using linear feedback shift register (LFSR) as it is better for march test compared to a binary counters [4]. LFSRs provide an equal address-changing probability for all address bits compare to binary counters and consume less area.
- data-generator or background pattern-generator: a hardware pattern generator to algorithmically insert test-patterns into memory words during write mode of each test element.
- BIST controller: controller is the heart of BIST and is implemented as a finite state machine (FSM) which goes through all elements of march test. In a fault free SRAM, FSM inserts the *finish* signal when test is complete.
- comparator: is used for response checking and asserts the *error* signal when readout data value from memory disagrees with the input data generated by data-generator.
- multiplexers: BIST engine is connected to SRAM array by adding multiplexers to the input pins. When memory is under test all inputs are provide by BIST circuit blocks and when SRAM is in normal mode multiplexer selects the system inputs.

One of the challenges of memory BIST is that the asynchronous memory must be tested by the synchronous BIST logic. A clock generator, with the appropriate clock period based on access time, provides the clock to initialize the BIST and control the timing of all BIST modules. The FSM controls the sequence of operations in BIST engine. Test instructions and input data patterns are generated by pattern generators in sequence. Any discrepancy between the output data from the SRAM and the input data generated by the data-generator, rises the *error* signal to indicate memory is defective.

B. March C- algorithm

The sensitivity of 6T SRAM arrays to physical defects and process variations is similar to that of standard logic gates. Therefore, a set of standard address, data and control schedules is sufficient to fully test the SRAM array for all

TABLE I
MARCH C- ALGORITHM.

March Element	M_0	M_1	M_2	M_3	M_4	M_5
Sequence	$\updownarrow(W0)$	$\uparrow(R0W1)$	$\uparrow(R1W0)$	$\downarrow(R0W1)$	$\downarrow(R1W0)$	$\updownarrow(R0)$

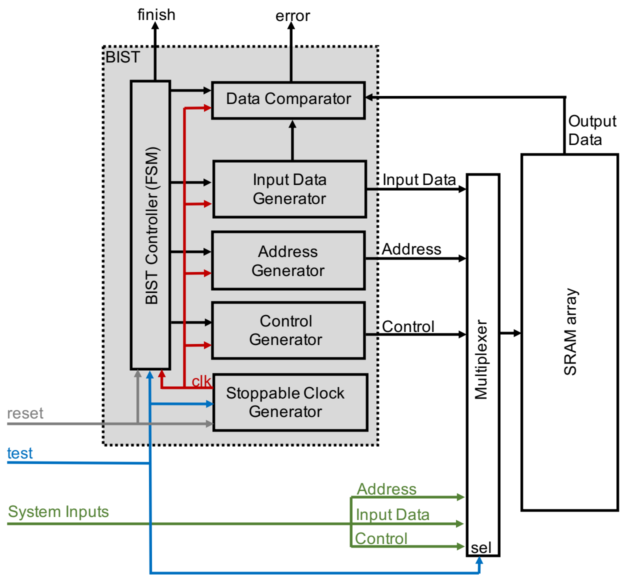

Fig. 3. BIST architecture

faults. Following are the most common type of faults that can make an SRAM array unfunctional:

- Stuck at fault: memory cell stuck at one or zero
- Address decoder fault: certain address cannot access any memory location or access multiple locations simultaneously
- Transition fault: cell fails to make a transition
- Coupling fault: transition in one memory cell inverts the content of another cell

Our BIST engine is based on the famous March C- algorithm [4] shown in Table I. M0-M5 are the six march test elements. In each march element, address sequence order is specified as ascending (↑) or descending order (↓). ↕ means either ↑ or ↓. Next memory operations which are writing to (W) and/or reading from (R) a specific location are executed. For example at M_1 element, address sequence is ascending and for each address, the algorithm performs a read operation (reading 0) followed by a write back with complementary value (writing 1) to the same location and then continues to the next address. Reading unexpected values results in an error, generated by comparing the input and output data. This algorithm requires 10N read and write operations (N is the number of memory words) and detects faults that may occur in the address decoder, read/write circuitry, and cell array.

For word-oriented (as opposed to bit-oriented) SRAMs fault coverage depends on the selected data-patterns. Determining the data-patterns that best balance the cost and fault-coverage of testing is difficult. Here we use two data-patterns, P_0, n'b0, and P_1, n'b1, (n is the size of data word), that results in a %96 fault coverage in SRAMs [3]. Adding an additional data-pattern doubles the test time and has minor improvement in fault detection for March C- algorithm [11].

V. CONCLUSION

This paper introduced an open-source unified memory compiler that can be used by both synchronous and asynchronous circuit designers and system architects. This memory compiler can generate fabricable memories in different process technologies, and enables rapid prototyping for researchers in various fields from device to system level.

SRAMs generated by AMC use techniques in modern pipelined asynchronous designs to ease the timing constraints presented in clocked memories and provide a higher throughput and best-case behavior for latency. Also, these SRAMs are pipelined and come in different orientations and aspect ratio layouts. The asynchronous SRAM generated by AMC can easily be used in synchronous designs by adding the synchronous interface to the SRAM; automatically generated and connected to SRAM by compiler. AMC also provides a BIST engine with march C- test algorithm to facilitate the at-speed testing of its SRAMs.

REFERENCES

[1] S. Ataei and R. Manohar, "AMC: An asynchronous memory compiler," in *IEEE International Symposium on Asynchronous Circuits and Systems*, May 2019.

[2] S. Ataei and R. Manohar, "AMC: Asynchronous memory compiler." http://avlsi.csl.yale.edu/act/doku.php?id=amc:start, 2019.

[3] M. L. Bushnell and V. D. Agrawal, "Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits," in *Springer*, 2000.

[4] A. J. van de Goor, "Using march tests to test srams," in *John Wiley and Sons*, 1993.

[5] A. J. Martin, A. Lines, R. Manohar, M. Nystroem, P. Penzes, R. Southworth, and U. Cummings, "The design of an asynchronous MIPS R3000 microprocessor," in *Advanced Research in VLSI*, pp. 164–181, 1997.

[6] V. N. Ekanayake and R. Manohar, "Asynchronous DRAM design and synthesis," in *Asynchronous Circuits and Systems, 2003. Proceedings. Ninth International Symposium on*, pp. 174–183, IEEE, 2003.

[7] C. Kelly IV, V. Ekanayake, and R. Manohar, "SNAP: A sensor-network asynchronous processor," in *Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems*, pp. 24–33, IEEE, 2003.

[8] J. Dama and A. Lines, "Ghz asynchronous SRAM in 65nm," in *15th IEEE Symposium on Asynchronous Circuits and Systems*, p. 8594, May 2009.

[9] C. T. O. Otero, J. Tse, R. Karmazin, B. Hill, and R. Manohar, "ULSNAP: An ultra-low power event-driven microcontroller for sensor network nodes," in *15th International Symposium on Quality Electronic Design*, pp. 667–674, IEEE, 2014.

[10] M. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, "OpenRAM: An open-source memory compiler," in *IEEE International Conference On Computer Aided Design (ICCAD)*, November 2016.

[11] C. T. Huang, J. R. Huang, C. F. Wu, C. W. Wu, and T. Chang, "A programmable BIST core for embedded DRAM," in *IEEE Design and Test of Computers*, pp. 59–70, January 1999.