MAT JOURNALS

Journal of Analog and Digital Communications
Volume 4 Issue 2

Characterization of File Memory Compiler

[1]Dr Gayathri S, [2]Sahanashree H N
Department of E & C
Sri Jayachamarajendra College Of Engineering
Mysuru, Karnataka
Email: *[1]sgmurthy_65@sjce.ac.in, [2]sahananagaraju23@gmail.com*

Abstract

As the usage of hand held devices is increasing rapidly, it is a serious requirement to design the SoC with much smaller size. Majority of the area on the SoC is occupied by the memories used like RAM, Cache etc. It is required to reduce the size of these memories on the SoC. Also as these SoC run on batteries such memories must consume very less power. So the requirement is to design high density embedded memories with less in area, with less power consumption and meeting the designer timing requirements like access time, setup and hold time constraints. The main aim of the proposed work is, once the design is done, the memory compiler need to be characterized whether it is meeting the design requirements for the given instance and given process, voltage and temperature corners and to get the memory timing and power information in datasheet and liberty formats.

Keywords: *Memory; process; Voltage ;temperature; liberty format*

INTRODUCTION

In the contemporary world, with the requirement of portable and handheld devices is increasing, designing such small SoC' s keeping the design requirements like smaller area, lesser power consumption and higher speeds of operation has become a challenging task. In addition, most of the SoC space is occupied with the memories like Caches, RAMs, TCAMs, and ROMs etc.., so designing these memories with high density is the main design challenge.

Also, characterizing the designed memory and developing view which are used in simulators is also a major challenge. With the increasing number of memory devices, the performance and power consumption of the complete SoC is mainly depends on these memory devices. So in order to design high quality SoC, accurate models of these embedded memories are required. Existing methods are either time consuming or less accurate. Hence an efficient and robust characterization method has to be developed to effectively model these embedded memories.

BACKGROUND

Embedded Memories

Depending on the requirement and usage different memories are used at different levels on a SoC. Available options are Register Files, SRAM, CAM etc.., These memories play a vital role on the overall performance of the SoC. These memories are used depending on the requirement and are divided in to three categories 1) Serial Access Memories 2) Random Access Memories 3) Content Access Memories. Random Access Memory is defined as the memory whose access time is independent of the address location. Whereas it is quite opposite in Serial Access Memory, in which the data stored is accessed in serial fashion.

In Content Access Memory, the data is accessed depending on the type of data stored in the memory. Random Access Memories are further divided into

Dynamic Random Access Memories and Static Random Access Memories. DRAMs are used as physical memories in the system level where as SRAMs are used as low level caches. This is because DRAMs design uses less transistor count when compared to the SRAM. But DRAMs use a capacitor for storing the data which need frequent refresh cycles which makes it slower than SRAM and also because of the usage of capacitors, leakage current is more in DRAMs when compared to SRAMs.

With the increase in technology nodes, the size of memories depends on their usage at different levels on the SoC. Depending on the requirement in performance and cost, different levels of memories can be incorporated in a SoC. To get the better performance, low level caches are to be used which is less in size and more costly. Higher levels are more in capacity, but far from the processor and also less in cost. Register Files and SRAMs come into Level 1, Level 2, and Level 3. For the main memory DRAMs are used and or Storage SSD or Disk storage is used.

The Lower level caches are accessed frequently and have to work at the processor frequency. So the speed of operation of these memories has to be very high and so their power consumption must be as low as possible to reduce the overall power consumption. The major difference in SRAM and Register File is the storage capacity. Usually Register Files are of very small size and are placed closer to the processor which makes them operate at high frequencies.

Compiler Flow

Memory compiler can be written using high level languages like C, C++, Perl etc.., these programming tools helps in using the designed leaf cells of schematic and layout to generate a full instance complete schematic and its layout. For getting the full instance schematic Netlister is used and for generating the full instance Layout Tiler is used. When generating memory instances using memory compiler, there are many input files which are to be given prior to memory instance generation.

Process, Voltage and Temperature Information, Memory cuts defining the size of memory, Technology Files, Control and Configuration Files, Template Files, Physical cell Schematics, Physical cell Layouts Characterized data for predefined instances, Programing tools like Tiler and Netlister. Process, Voltage and Temperature (PVT) information is used to run the simulations at the required conditions. This includes the best and worst conditions.

Technology files contain the information of the technology nodes and device models. Certain views like liberty files, datasheet files, LEF files, Verilog models etc.., When the compiler is run it uses all the predefined data, launches the simulations, runs the commands, creates the directory structure as per requirement, generates the full instance schematics and layouts using the leaf cells, and generates the Synopsys liberty files, datasheet files and Verilog views form the templates given as inputs.

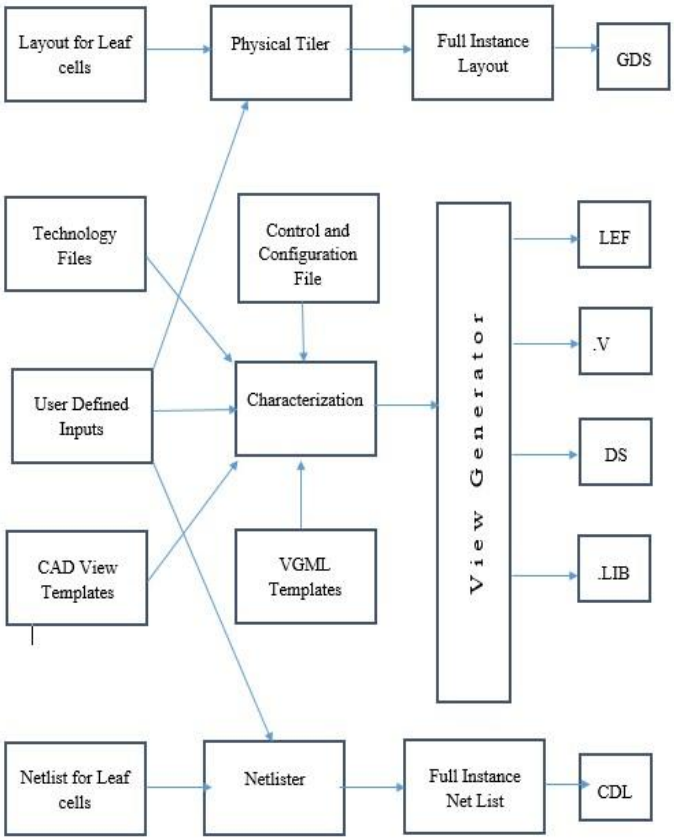

Fig1. *Compiler Design Flow*

COMPILER FLOW CHARACTERIZATION

Memory characterization is of increasing concern in SoC designing which require accurate and efficient models at all stages of design. Compounding this problem are the number and magnitude of the challenges faced. The number of memory instances per chip is increasing rapidly, with some forecasts pointing to greater than 90% of the die area being taken up by memories and other large macros within 5 years.

In addition, to support the full range of process, voltage and temperature corners (PVTs) and the sensitivity to process variation, the number of characterization runs and the number of data points per characterization run is growing exponentially. In general there are two approaches in characterizing full range of memory compiler. One is Full Instance based Characterization and the other is using the Critical Path circuit Characterization. In this full instance based approach, the entire memory is treated as black box which is simulated using high speed simulating tools like Fast Spice.

The main advantage of this approach is that we get the good power and timing models. The problem is the requirement of more simulator licenses so that simulations can be run on multiple machines. Also these Fast spice simulators are less accurate than True Spice simulators. One other drawback for this method is that it doesn't work well for generating some of the newer model formats such as noise models and cannot be scaled to generate process variation models needed for statistical static timing analysis. The main problem comes in the simulation of memory block and in extracting the timing and power values of the memory.

The main reason is, to simulate a whole memory block and get timings it takes lot of time. For example for a memory block of size 1024x256, in order to get timing values if we simulate at each corner and assuming that each corner takes 5m(best case) of simulation time on an average and 100 such simulations can be run in parallel, then it takes 9 days approximately to simulate at all corners. At the end, if there is any modification in the design then simulation has to be done again, which again takes lot of time. So this method is not used at the development phase of memory compiler.

The other approach is to do develop a critical path which is considered the worst path taken by the signals in the memory instance. Once the critical path is identified, remaining part of the circuit is replaced with load and is simulated with True Spice simulators which give accurate results. These critical paths can either be generated by automated tools or can be designed by the designers. With the latest STA tools available, it is now possible to time not only the control logic of a memory block, but also paths through the memory core array (i.e. bit-column, wordline, column mux, sense amp, and so on).

This results is an improvement in design turn-around-time, verification coverage, accuracy (within ±5%), and productivity for designs with embedded memories. It does not require netlist reduction techniques as commonly practiced in the dynamic simulation approach. A major benefit of using the STA approach is that there are no vectors needed in performing timing analysis. This alone saves tedious verification planning and processing time and eliminates the potential of human error in generating the stimulus for the dynamic simulations. But in the characterization done in this project is done by writing the stimuli and measurements manually and the critical path is designed by the designers. No tools are used in automating the flow. Below figure shows the flow which is followed in characterizing the memory compilers.

As the characterization is done for the whole compiler range, critical path modeled should be generic and should fit for any range of memory instance with in the compiler range. As the compiler has a range of minimum 16bit to maximum of 256kb, the architecture changes for memory instances depending on their size. All the possible architectures are to be modeled such that this becomes user defined parameter so that user can choose what architecture is needed based on their requirements like area, timing and power constraints.

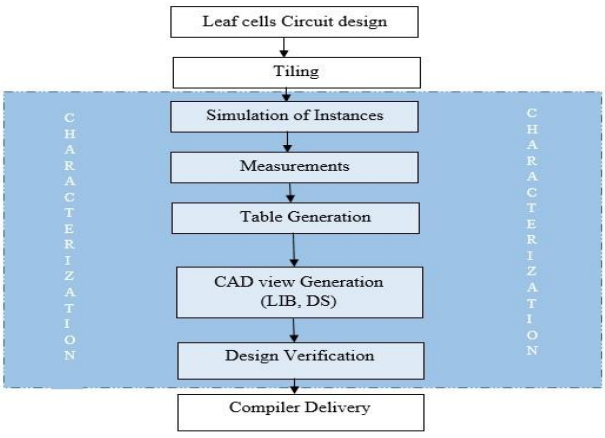

Fig 2. *Characterization Flow*

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

Characterization Of 8x8 Memory Instance

For simple understanding of how a memory instance is characterized, an 8x8 memory instance is taken and characterization is done as it takes less time for simulations. 8x8 implies 8 words and each having 8 bits all together a memory of 64bit size.

The same procedure is done for all the memory instances over the compiler range. For proper characterization, initially the functionality of the memory instance has to be verified. For that we have to create a stimuli for functionality verification such as Read Write operation, Scan chain, various power modes etc.., once the functionality is verified the same stimuli can be used to characterize the memory instance.

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

Stimuli for read/write functionalty

As the functionality and characterization can be done at the same time, single stimuli is written for both. For effective characterization of memory instance, it has to be done in minimum number of cycles and all the possible toggling has to be identified. To do this the stimuli is written in such a way that the address locations are chosen at near and far end of the memory instance. Also the data input must be in such a way that maximum number of toggling occurs at the output port and it will make sure, that all cases are verified. Since 8x8 memory instance is chosen, the address locations which are accessed are 0 and 7.

Read enable and write enable signals are also toggled each cycle. Care should be taken such that both read operation and write operation should not be done at the same location. All together we have to generate the stimuli for the following input signals for read/write operations. Read Address, Write address, read enable, Write enable, Read Clock, Write Clock. Depending on the requirement other input stimuli are applied to the memory instance.

As an example the stimuli for 2 cycles is given as below. But for verifying all the toggling cases writing 0->1, 1->0 , toggling 1-> 0->1 at locations near and far, at least we need a minimum of 8 clock cycles. The stimuli which is written is spice compatible simulated using XA simulator.

Measurements

Setup and hold measurements are taken where the data is latched by either master/slave latch. For data setup measurement we take the maximum data delay and minimum clock delay. The difference of these two gives the setup time of that particular signal. Below waveform shows the data setup measurements.

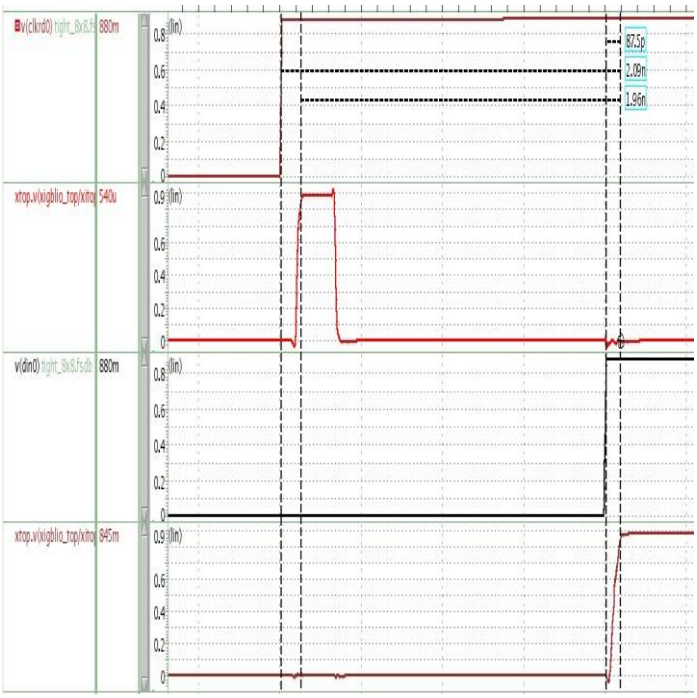

Fig 3. *Setup Time Measurement*

In the same way will write the measurements for hold time, access time, cycle time, Pinpower etc.

Tools Used to Enable Compiler

In order to create a memory compiler all the individually generated memory instances must be made generic such that depending on the user defined parameters corresponding memory instance has to be generated. Also, characterizing the whole compiler space is an impossible task because of which only certain memory instances are simulated and the whole compiler space is interpolated using these results. Finally to generate the CAD views like Datasheet view, Liberty view, Verilog views certain tools are required to perform these tasks without much human effort.

Some of the tools which are used in memory compiler characterization are mentioned below. View generator, post processing full table. This view generator helps in converting generic templates of simulation files to memory instances of user defined size and runs at the specified voltage, process and temperature. Also in creating the CAD views, initially templates for Datasheet and Liberty are written using this view generator language which is then used to generate the CAD view for any memory instance thus enabling the memory compiler. Below are some examples how view generator works is given below.

Genview –v template –o output –t char_table –D user_def_var –cc calculation_file – log log

Template file is the generic file which is parsed by the view generator tool. While parsing the template file, normal text lines are as it is printed to the output file whereas the words which are tagged with view generator identifier are replaced with user defined variables. Characterization table contains all the data of simulated timing and power information. When the particular index is pointed, it is picked up directly from the table, else an interpolation is performed based on the user option and value is returned.

The idea of memory compiler is to use the simulated data and use interpolation to cover the entire compiler space. This interpolation is done by the view generator

tool. The available data is to be written in specific format so that the tool can understand. In general it is a 2 dimensional array of data. Below figure shows a table representation where x denotes data presence and o denotes an empty data.

Word								
2048	O	O	X	X	X	X	X	
1536	O	O	X	X	X	X	X	
1032	X	X	X	X	X	X	X	
512	X	X	X	X	X	O	O	
256	X	X	X	X	X	O	O	
128	X	X	X	X	X	O	O	
72	X	X	X	X	X	O	O	
	4	16	32	56	80	104	128	Io

Fig 4. *General Matrix of Interpolation*

Since the table contains number of parameters of interest it is difficult to represent the calculated data in the above mentioned format. So instead change it to one dimension and add different parameters in the other direction. The format is as shown. TABLE tbl (i1, i2)

i1 2,	**2,**	**2,**	**2,**	**1,**	**1,**	**1,**	**1**
i2 1,	**2,**	**3,**	**4,**	**1,**	**2,**	**3,**	**4**
O1 12,	**2,**	**2,**	**10,**	**12,**	**2,**	**2,**	**10**
O2 3,	**5,**	**7,**	**10,**	**3,**	**5,**	**7,**	**10**
O3 10,	**1,**	**22,**	**9,**	**10,**	**1,**	**22,**	**9**

$$f(x) = \frac{(x-x1)(x-x2)\dots(x-xn)}{(x0-x1)(x0-x2)\dots(x0-xn)} f(x0) + \dots + \frac{(x-x0)(x-x1)\dots(x-xn\text{-}1)}{(xn-x0)(xn-x1)\dots(xn-xn\text{-}1)} f(xn)$$

$$f(x) = \frac{(x-x1)(f(x0)-f(x1))}{(x0-x1)} + f(x1)$$

For a designed memory instance, there is possibility of applying different input slopes, with different clock slews for different output loads. Each of these possibilities impact the performance of the memory instance. So the memory instance has to be analyzed with all such possible cases. Doing this for the whole compiler range is an impossible task. This impact is done by using the post processing tool.

Post processing calculates the impact of clock slew, input pin slew and load variations which are pre-defined as inputs at the beginning of the char-flow. In the post processing file, default load, slew variations, and different slews and loads for which impact has to be calculated is given. When the post processing tool is invoked back ground equation files are executed in which the impact calculations are written for the timing/power parameters of interest.

TABLE GENERATION RESULTS

After all the post processing, the available data have to be arranged in a specific format such that it is understandable by the view generator tool so that the interpolation can be done properly. This table has to be generated for each pvt corner at which the memory instances are generated. For this a fulltable generation file has to be passed to the fulltable generator tool which contains all the pvt information and memory cuts and the data units. Below shown the basic format in which table generation is to be given as input to the full table generator tool.

```
TABLE fastTallCap (PinCap)
 PinCap "min", "max"
 CLK 0.02434, 0.02543
 CSB 0.01325, 0.01575
 RWB 0.01238, 0.01249
 SLEEPB 0.006825, 0.007419
 SLOWB 0.009964, 0.01493
 ADDR 0.007985, 0.01998
 DIN 0.006879, 0.007174
 WIB 0.006847, 0.007175
 DOUT 0.006024, 0.006024
ENDTABLE
```

Fig 5. *View Generation Table Format*

The liberty and data sheet files are the ASCII representation of timing and power numbers of a designed memory instance. These timing and power values are obtained from the characterization of the memory instances. When a memory instance is obtained from the memory compiler, it generates these liberty and data sheet views along with the full tiled netlist and layout design.

The characterized data in the CAD views are either a direct output form the characterized data or the interpolated using the available data. These data has to be written in industry specific format called

Liberty format. For this internal tool like view generator and Perl are used for view generation. A generic template is created and is parsed using the view generator and executed using perl to generate the required views. For example a generic liberty file template is shown below.

CONCLUSION

The behavior of the basic SRAM cells were analyzed the latest technology nodes, and their behavior like read noise margin, hold and write noise margin were analyzed with the variation of process, voltage and temperature and their impact on the circuit behavior is identified. Further Characterization for the compiler space was done effectively. This characterization methodology reduced the number of memory instances that are to be simulated. Also the bank delta methodology effectively matched with the actual simulated results.Authors and Affiliations.

During the characterization process, it leaves with huge data base. For handling such a huge database automation is done at different levels like verification, stimuli creations and plugging in the extracted netlist files to the critical path. This reduced the validation time to a great extent. This characterization methodology can be extended even for power characterization, and also automation can be done for measurements such that it can be combined with the tight/stimuli generation tool.

So at a stretch both stimuli and measurements can be obtained within minutes and run the simulation to get the complete characteristics of the memory instance.

Further, different interpolation methods can be experimented and use the method which gives the interpolated data with minimum error so that further padding can be avoided after characterization.

REFERENCES

1. Neil H. E. Weste and David Money Harris," CMOS VLSI Design A Circuits and Systems Perspective," Edition 4, 2011.
2. Jay Abraham, "Improving SoC Design Flows with Robust and Precise Embedded Memory Models", Silicon Metrics Corp., Austin TX, USA, Unpublished.
3. Liberty User Guides and Reference Manual Suite Version 2013.03.
4. RAGHU KOPPANATHI, "8-PORT S-RAM MEMORY CELL, WITH 8 WRITES OR 16 READS SIMULTANEOUSLY", M.S Thesis, Osmania University, Andhra Pradesh, 2012.
5. Michael Cha Hamid Mahmoodi, "Hspie Quick Start", Nano-Eelectronics & Computing Research Lab school of Engg., San Fransisco State University, 2011.
6. Bhavya Daya et al, "Synchronous 16x8 SRAM Design", Electrical Engineering Department, University of Florida, unpublished.
7. Omar Shah, Shekhar Kapoor, "Extraction Techniques for High-performance, Highcapacity Simulation", Synopsys, 2009.
8. Charles Longway, YouPang Wei, Automatic Memory IP Characterization [online].
9. Chen Ming, Bai Na, "An Efficient and Flexible Embedded Memory IP Compiler," in International Conference on Cyber-Enabled Distributed Computing and Knowledge Discover, China, 2012.
10. Ken Hsieh, "The Benefits of Static Timing Analysis Based Memory Characterization," white paper, Synopsys, 2012.
11. Nai-Yin Sung and Tsung-Yi Wu, "A Method of Embedded Memory Access Time

12. Measurement," Proceeding of International Symposium on Quality Electronic Design, pp.462-465, 2001
13. Mahmut E. Sinangil, "Ultra Dynamic Voltage Scalable (U-DVS) SRAM Design Considerations", Dept. Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2008.
14. C. Yen-Yu, et al., "Rapid and Accurate Timing Modeling for SRAM Compiler," in
15. Memory Technology, Design, and Testing, 2009. MTDT '09. IEEE International Workshop on, 2009, pp 73-76
16. Gilles Gasiot et al,"Experimental characterization of process corners effect on SRAM Alpha and Neutron Soft Error Rates," an IEEE journal pp 3C4.1-3C4.5.
17. Rachana Ekbote et al," Characterization and Simulation of Different 7T SRAM Topologies," an IEEE journal on Engineering and systems, 2012, pp 1-5.
18. H. N. Mishra, P. Y. Kumar, "Design, Simulation and Characterization of Memory Cell
19. Array for Low Power SRAM using 90 nm CMOS Technology," Proceedings of IEEE
20. International Conference on Power, Control and Embedded Systems (ICPCES), Allahabad, India, November 29-December 1, 2010, pp.1-3.
21. W. Zhongyuan, et al., "A high performance embedded SRAM compiler," in ASIC, 2003.
22. Proceedings. 5th International Conference on, 2003, pp. 470-473 Vol.1
23. Zheng Guo et al, "Large-Scale SRAM Variability Characterization in 45 nm CMOS," an IEEE journal on Solid-State Circuits vol-44, pp 3174-3192, 2009.
24. Vasudha Gupta, Mohab Anis, "Statistical Design of the 6T SRAM Bit Cell", an IEEE journal on Circuits and Systems, vol-57, pp 93-104.
25. Rajasekhar Keerthi, Chein-in Henry Chen, "Stability and Static Noise Margin Analysis of Low-Power SRAM", an IEEE International Instrumentation and Measurement Technology Conference, 2008.
26. A. Prakash, "Characterization of 90 nm SOI SRAM Single Cell Failure by Nano Probing Technique and TCAD Simulation", an IEEE conference on Integrated circuits, pp 252254, 2007.