DESIGN OF LOW POWER SRAM

Mr.R.Radeep Krishna,Assistant Professor,
Department of ECE,
Kalasalingam University, Krishnan Kovil,
Virudhunagar district.

N. Suganya , P. Suganya, S. Swarnalatha,
Final year B.Tech , Department of ECE,
Kalasalingam University, Krishnan Kovil,
Virudhunagar district.

Abstract— Due to the increased demand of SRAM with large use of SRAM in System On-Chip, the oxide thickness has become a tough challenge in CMOS technology. The leakage power also affects the chip design process. The power consumption and speed of SRAMs are important issue that has lead to multiple designs with the purpose of minimizing the power consumption. Speed of SRAM and Power consumption are also taken care of for designing a chip. This paper represents the simulation of 6T SRAM; Asymmetric SRAM cells using low power reduction techniques. All the simulations have been carried out on Tanner EDA tool. In this project, SRAM cell will include one extra transistor that will control the overall capacitances during the write and read operation and will optimize the total capacitance that results in decrease in the power dissipation. This thesis focuses on the power dissipation during the Write operation in six-T CMOS SRAM as well as read operation also. The whole thesis circuit verification is done on the Tanner tool, Schematic of the SRAM cell is designed on the S-Edit and net list simulation done by using T-spice and waveforms are analyzed through the W-edit. The results are compared with Conventional 6T SRAM cell which is also being characterized in this thesis with the same technology. This low power cell consumes lesser power.

Keywords— SRAM, CMOS, 6T SRAM cell

I.Introduction

With ever increasing level of device integration and the growth in complexity of electronic circuits, increasing the demand of portable electronics devices and also dependence on the battery operated devices motivating the VLSI designers to reduce the power dissipation of the VLSI circuits so that they can be used for the long time for the given battery supply so the reduction of power is the most often used measures of the efficiency of VLSI circuits has come to the fore as a primary design goal. Also as high speed circuits dissipate large amounts of energy in a short amount of time, power minimization algorithms and techniques are strongly recommend by most of the companies in these circuits. With ever increasing operating frequency and processing capacity per chip, large currents have to be delivered and the heat due to large power consumption must be removed by proper cooling techniques. Battery life in portable electronic devices is limited. Low power design directly leads to prolonged operation time in these portable devices. Other qualities are

1. Shrinking of device size
2. Growth of Portability
3. Reliability
4. Battery size

Every digital system now a days is strongly dependent on the memory we can also say that no digital system now a days can be built without memory. It is also the heart of any microprocessor and that is why most of the research in low power design is going to design the memory.

The basic motive of this research work is to analyse the SRAM memory cell which will consume lesser power as compared to the conventional cell of the same technology. Here the modification is done on the convention SRAM cell for the reduction of the dynamic power. The new architecture will contains one extra tail transistors in the pull-down path of the respective back to back inverter to reduce overall capacitance of the structure . This one transistor will controlled by Control Signal (CS). During read or write mode at least one of the tail transistor must be turned OFF to disconnect the driving path of respective inverters which will reduce the short circuit power dissipation during the rise and fall time of the data being stored in the memory; these transistors also reduce the sub threshold current during transistor OFF condition.

A.SRAM DESIGN

The main building blocks used in implementing this project are 6T SRAM cell, row and column decoders, bit-line conditioning circuitry, read-write control circuitry, sense amplifier and clock tree buffers.

6T SRAM cell

The SRAM cell incorporates basic 6T design. The sizes used in designing 6T are 1.5:1 (W/L) for PMOS, 2.25:1 for NMOS in cross coupled inverter and 1.5:1 for access transistors. Here L=0.24um. The sizing was done keeping in mind the read and write stability and the static noise margins (SNM).For read stability this voltage should remain below the threshold of the NMOS transistors. The margin is essential to ensure the design works on silicon.

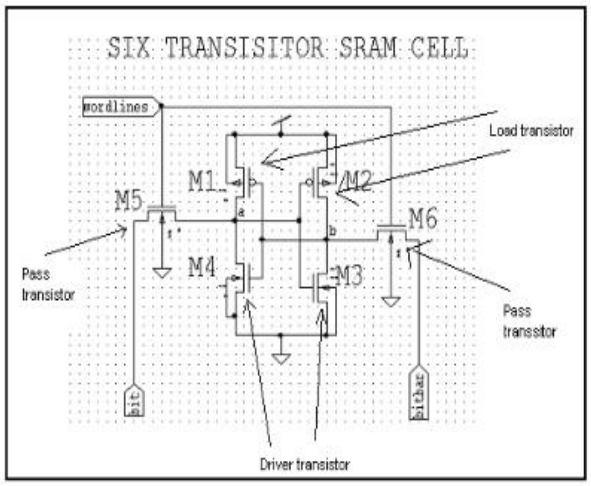


FIG 1,STRUCTURE OF 6T SRAM

B.SRAM Operation

An SRAM cell has three different states it can be in: standby where the circuit is idle, reading when the data has been requested and writing when updating the contents. The SRAM to operate in read mode and write mode should have "readability" and "write stability" respectively. The three different states work as follows:

STANDBY:

If the word line is not asserted, the access transistors M5 and M6 disconnect the cell from the bit lines. The two cross coupled inverters formed by M1-M4 will continue to reinforce each other as long as they are connected to the supply.

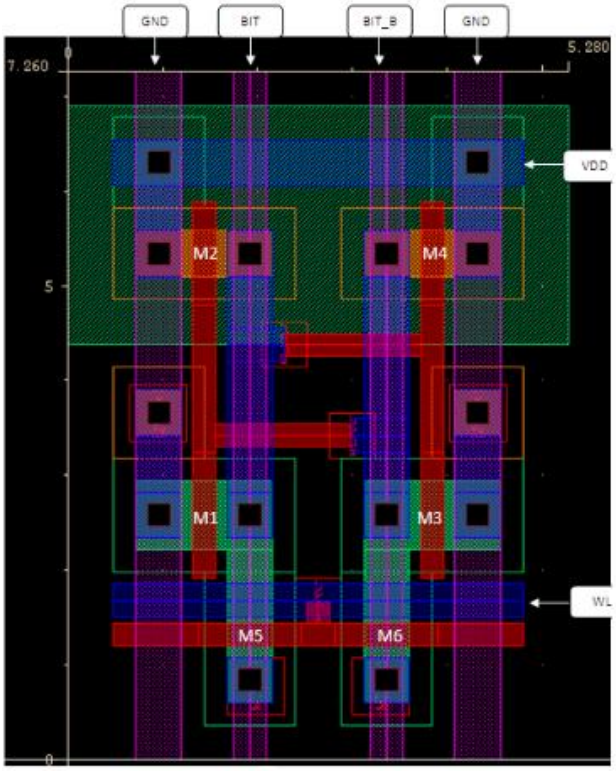
READING:

Assume that the content of the memory is a 1, stored at Q. The read operation is done by using the sense amplifiers that pull the data and produce the output. The row decoders and column decoders are used to select the appropriate cell or cells from which the data is to be read and are given to the sense amplifiers through transmission gate.

WRITING:

The start of a write cycle begins by applying the value to be written to the bit lines. If we wish to write a 0, we would apply a 0 to the bit lines, i.e. setting BL bar to 1 and BL to 0.A 1 is written by inverting the values of the bit lines. WL is then asserted and the value that is to be stored is latched in. Note that the reason this works is that the bit line input-drivers are designed to be much stronger than the relatively weak transistors in the cell itself, so that they can easily override the previous state of the cross-coupled inverters.

6T SRAM CELL LAYOUT



Cell Area = 5.28x7.26 um = 44x60.5 λ

FIG 2, SRAM LAYOUT

ROW DECODER AND WORDLINE DRIVER

For a n: 2^n decoder, 2^n n input gates need to be built. Large fan-in gates are the result of large decoders. A series stack is formed by large fan-in gates and the decoder is slower. When the number of inputs is greater than four, the decoder becomes very slow and large gate have to be broken into smaller gates. Pre-decoding causes common gates to be factored out and it saves area and is the same path effort as a decoder without the pre-decoding step. A pre-decoder was used in the design to reduce the fan-in and the size of the NAND gate needed to create the decoder. This results in a faster decoder. NAND and NOR gates were considered for the decoder design. Using delay analysis, it was found that the NAND gate is faster than the NOR gate. When designing the decoder, it was assumed that the inverters will have to be large along the entire decoder in order to drive the large word line capacitances. Inverter sizing was learned towards the end of the project time period and no time was available to make the necessary adjustments. The decoders are controlled by a clock because the SRAM is synchronous. The positive edge of a clock will allow the address to be read into the decoders, both column and row, and enable the correct word line and bit line.

A register is considered instead of a latch because the latch reacts to the input whenever the clock signal is high. The register, positive edge triggered, reacts to the input whenever the clock goes from low to high. Instead of using a register the clock can be an input into the decoder. The decoder will only decode the address when the clock input is high. The row decoder output can be connected to a driver that drives the word lines. Including the clock in the decoder is not a good approach because it can lead to a large skew. For ease of design and layout and area reduction, it is more appealing than creating registers for each input. A register is also designed and considered. Fig 3 and 4 shows the word line driver block diagram. A dynamic NAND was not used, instead an ordinary NAND was used with a large load capacitance.

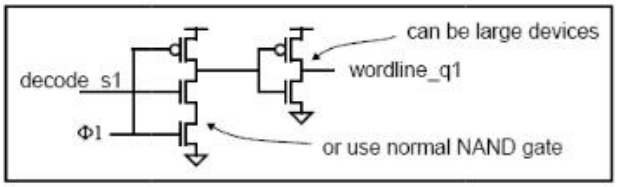


Fig 3, word line driver

After performing a simulation of the row decoder, the lowest clock frequency was found to be 400MHz.The decoder would probably be faster if the inverter sizes were changed to be incremental along the entire decoder. Polylines were used to connect the pre-decoder output to the decoder input. Meta12 could have been used to speed up the row decoder.

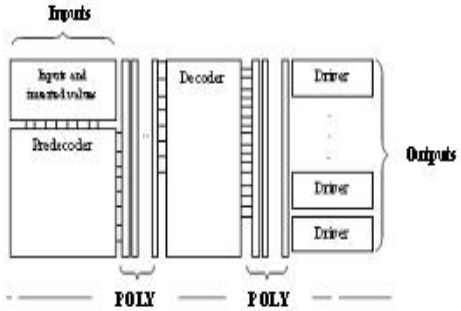


FIG 4, WORDLINE DECODER AND ROW DECODER

COLUMN DECODER AND COLUMN CIRCUITRY

A normal decoder built using logic gates has some drawbacks, the main problem is that the decoder will require a very large number of transistors leading to layout which is unnecessarily complex and so more time- consuming.

Also the capacitance associated with the long runs of wires and high gate input count will add to long delays. The address inputs will also have to be buffered to drive this huge capacitance load. Another problem is that the power consumption of such a decoder will be very high due to the large number of gates. To overcome these problems, we have used a dynamic NOR decoder. This structure reduces the number of transistors by half. It also increases the speed of the decoder and makes the layout simple and less time consuming.

The NOR decoder works like a dynamic circuit. It requires a pre-charge cycle followed by the evaluation stage. The 'pre-charge' input is asserted which turns ON the PMOS devices and all the outputs of the decoder Vdd (Logic '1').The decoder should not be read at this instant, since all the outputs will be at Vdd. Once the pre-charge is done, the PMOS device is turned OFF. The outputs will still be at Logic '1', because the charge will be stored on the capacitors. Now the inputs are applied on the three address lines. The corresponding NMOS devices will be turned ON and the charged capacitor on that line will be discharged to ground. Thus all, except one line will go to ground (Logic'0').The line, which remains high will be the decoded line and this line will drive all the NMOS transistors on that line. Thus the right data will be available on the data line. This data will be given to the output data block, which will return the correct data to the outside world OE (Output Enable) signal is asserted.

The selected line will have to drive a number of transistors and so we will need a series of buffers to drive such a heavy load. The timing is a very important parameter here because during the pre-charge cycle, all the outputs will go high. However, the necessary data is present at the line only after the pre-charge cycle is over and the pre-charge input is de-asserted (made'1'). This is done using a simple two input AND gate. Followed by a Mux circuit which interfaces this with the control circuitry as shown in fig 5.

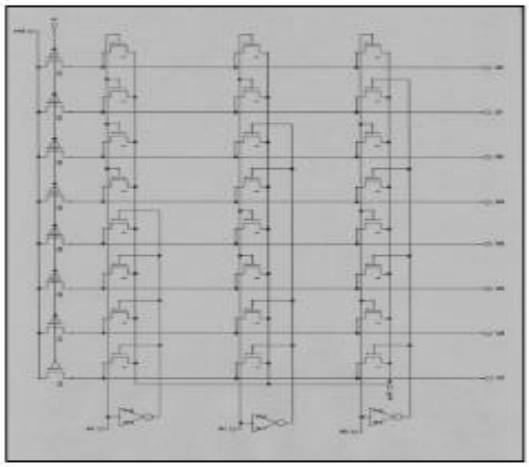


FIG 5, COLUMN DECODER

We have to make a trade-off between the decoder delay time and the size of the PMOS transistors used for pre-charging and so faster is the decoder. Here it is feasible to be fabricate large size PMOS transistors, since we have only eight of them.

II.LOW POWER SRAM

Fast low power SRAMs have become a critical component of many VLSI chips. This is especially true for microprocessors, where the on-chip cache sizes are growing with each generation to bridge the increasing divergence in the speeds of the processor and the main memory. Simultaneously, power dissipation has become an important consideration both due to the increased integration and operating speeds, as well as due to the explosive growth of battery operated appliances. This paper explores the design of SRAMs, focusing on optimizing delay and power. While process and supply scaling remain the biggest drivers of fast low power designs, this thesis investigates some circuit techniques which can be used in conjunction to scaling to achieve fast, low power operation.

TOOLS USED

I.MODELSIM

Model Sim combines simulation performance and capacity with the code coverage and debugging capabilities required to simulate multiple blocks and systems and attain ASIC gate-level sign-off. Comprehensive support of Verilog, System Verilog for Design, VHDL, and System C provide a solid foundation for single and multi-language design verification environments. Model Sim's easy to use and unified debug and simulation environment provide today's FPGA designers both the advanced capabilities that they are growing to need and the environment that makes their work productive.

A.VERILOG LANGUAGE

INTRODUCTION

Verilog HDL is one of the two most common Hardware Description Languages (HDL) used by integrated circuit (IC) designers. The other one is VHDL. HDL's allows the design to be simulated earlier in the design cycle in order to correct errors or experiment with different architectures. Designs described in HDL are technology-independent, easy to design and debug, and are usually more readable than schematics, particularly for large circuits.

Verilog can be used to describe designs at four levels of abstraction:
(i) Algorithmic level (much like c code with if, case and loop statements).
(ii) Register transfer level (RTL uses registers connected by Boolean equations).
(iii) Gate level (interconnected AND, NOR etc.).
(iv) Switch level (the switches are MOS transistors inside gates).
The language also defines constructs that can be used to control the input and output of simulation.
More recently Verilog is used as an input for synthesis programs which will generate a gate-level description (a netlist) for the circuit. Some Verilog constructs are not synthesizable. Also the way the code is written will greatly effect the size and speed of the synthesized circuit. Most readers will want to synthesize their circuits, so non synthesizable
Constructs should be used only for *test benches*. These are program modules used to generate I/O needed to simulate the rest of the design. The words "not synthesizable" will be used for examples and constructs as needed that do not synthesize. There are two types of code in most HDLs:

1.Structural code is a verbal wiring diagram without storage.
assign a=b & c | d; /* "|" is a OR */
assign d = e & (~c);
The order of the statements does not matter and is used for circuits with storage, or as a convenient way to write conditional logic.
always @(posedge clk) // Execute the next statement on every rising clock edge.
count <= count+1;

2 Procedural code is written like c code and assumes every assignment is stored in memory until over written. For synthesis, with flip-flop storage, this type of thinking generates too much storage. However people prefer procedural code because it is usually much easier to write, for example, if and case statements are only allowed in procedural code. As a result, the synthesizers have been constructed which can recognize certain styles of procedural code as actually combinational. They generate a flip-flop only for left-hand variables which truly need to be stored. However if you stray from this style, beware. Your synthesis will start to fill with superfluous latches.

3. Modules

In Verilog, circuit components are designed inside a module. Modules can contain both structural and behavioral statements. Structural statements represent circuit components like logic gates, counters, and microprocessors. Behavioral level statements are programming statements that have no direct mapping to circuit components like loops, if-then statements, and stimulus vectors which are used to exercise a circuit.

```
timescale 1ns / 1ps
//create a NAND gate out of an AND and an Invertor
module some_logic_component (c, a, b);
  // declare port signals
 output c;
 input a, b;
     // declare internal wire
 wire d;
     //instantiate structural logic gates
 and a1(d, a, b); //d is output, a and b are inputs
 not n1(c, d);    //c is output, d is input
 endmodule

//test the NAND gate
module test_bench; //module with no ports
 reg A, B;
 wire C;
  //instantiate your circuit
  some_logic_component S1(C, A, B);
```

ii .SAMPLE RAM PROGRAM

```
//Behavioral code block generates stimulus to test circuit
   initial
   begin
     A = 1'b0; B = 1'b0;
     #50 $display("A = %b, B = %b, Nand output C = %b \n",
A, B, C);
A = 1'b0; B = 1'b1;
     #50 $display("A = %b, B = %b, Nand output C = %b \n",
A, B, C);
     A = 1'b1; B = 1'b0;
     #50 $display("A = %b, B = %b, Nand output C = %b \n",
A, B, C);
     A = 1'b1; B = 1'b1;
     #50 $display("A = %b, B = %b, Nand output C = %b \n",
A, B, C);
   end
   endmodule
```

A. Assigning data out inside always block:

```
module memory
        (
           output reg [7:0] data_out,
           input [7:0] address,
           input [7:0] data_in,
           input write_enable,
           input clk
            );
           reg [7:0] memory [0:255];
         always @(posedge clk) begin
         if (write_enable) begin
         memory[address] <= data_in;
            end
         data_out <= memory[address];
             end
             endmodule
```

B.Assigning data_out using assign statement:

```
module memory
        (
         output reg [7:0] data_out,
         input [7:0] address,
         input [7:0] data_in,
         input write_enable,
         input clk
          );
         reg [7:0] memory [0:255];
         always @(posedge clk) begin
         if (write_enable) begin
         memory[address] <= data_in;
         end
         end
         assign data_out = memory[address];
          endmodule
```

C. SRAM Model

```
module sram(CSB,WRB,ABUS,DATABUS);
  input CSB;            // active low chip select
  input WRB;            // active low write control
  input [11:0] ABUS;    // 12-bit address bus
  inout [7:0] DATABUS;  // 8-bit data bus
              //** internal signals
  reg  [7:0] DATABUS_driver;
  wire [7:0] DATABUS = DATABUS_driver;
  reg [7:0] ram[0:4095];          // memory cells
  integer i;
   initial
 //initialize all RAM cells to 0 at startup
   begin
   DATABUS_driver = 8'bzzzzzzzz;
   for (i=0; i < 4095; i = i + 1)
    ram[i] = 0;
   end
always @(CSB or WRB or ABUS)
   begin
   if (CSB == 1'b0)
   begin
   if (WRB == 1'b0) //Start: latch Data on rising edge of CSB
or WRB
   begin
   DATABUS_driver <= #10 8'bzzzzzzzz;
  @(posedge CSB or posedge WRB);
   $display($time," Writing %m ABUS=%b
DATA=%b",ABUS,DATABUS);
    ram[ABUS] = DATABUS;
    end
    if (WRB == 1'b1) //Reading from sram (data becomes valid
after 10ns)
    begin
     #10 DATABUS_driver =  ram[ABUS];
     $display($time," Reading %m ABUS=%b DATA=%b",
ABUS,DATABUS_driver);
     end
endmodule
```

2. TANNER TOOL

Tanner EDA provides of a complete line of software solutions for the design, layout and verification of analog and mixed-signal (A/MS) ICs and MEMS. Customers are creating breakthrough applications in areas such as power management, displays and imaging, automotive, consumer electronics, life sciences, and RF devices. A low learning curve, high interoperability, and a powerful user interface improve design team productivity and enable a low total cost of ownership (TCO).

Tanner EDA is a suite of tools for the design of integrated circuits. These tools allow you to enter schematics, perform SPICE simulations, do physical design (i.e., chip layout), and perform design rule checks (DRC) and layout versus schematic (LVS) checks. There are 3 tools that are used for this process:

- S-edit-a schematic capture tool.
- T-SPICE-the SPICE simulation engine integrated with S-edit.
- L-edit-the physical design tool.

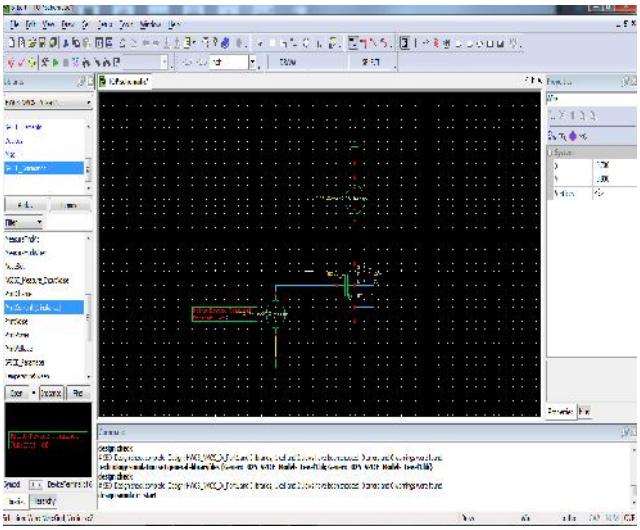
Working with S-edit

S-edit is a schematic entry tool that is used to document circuits that can be driven forward into a layout of an integrated circuit. It also provides the ability to perform SPICE simulations of the circuits using a simulation engine called T-SPICE. T-SPICE can be setup and invoked from within S-edit.

STEPS IN S-edit FOR DESIGNING NMOS AND PMOS

- Start S-Edit
- Start a New Design-File–New-New Design
- Create a new Cell
- Enter the symbol libraries
- Setup –SPICE Simulation
- Enter the Schematic to simulate
- Setup the Parameters that will be used during the DC sweep analysis
- Setup the SPICE DC Sweep Analysis

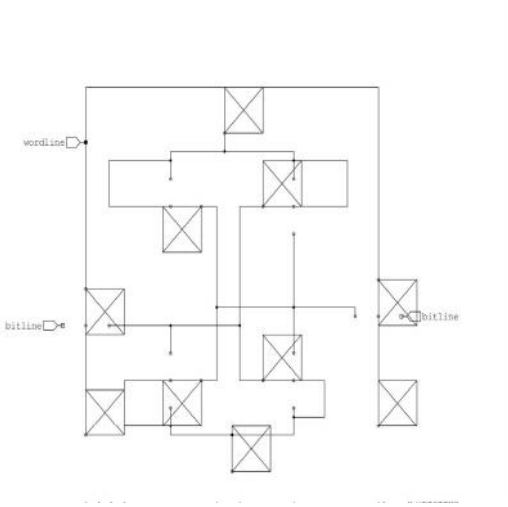
FIG 6, SCHEMATIC FOR NMOS



STEPS IN S-edit FOR DESIGNING CMOS

- Steps in s-edit for designing NMOS and PMOS
- Create the Inverter
- Export a spice net list
- Create the Inverter Symbol
- Create the TOP schematic to test the Inverter
- Simulate the Design

FIG 7,SCHEMATIC FOR CMOS



LIBRARY FILES:

.MODEL NMOS NMOS (LEVEL = 1
* Electrical Properties
+KP = 178e-6
+VTO = 0.3703728
+GAMMA = 0.029
+PHI = 0.279
+LAMBDA = 0.05
* Physical Parameters, these are overriden by the electrical parameters but are included for refernce
+UO = 284.0529492
+TOX = 5.6E-9
+NSUB = 1e15
*+LD = 0.0025e-6 * Not going to use this for now, it effects KP & COX & results in ~2-3% difference in IDS
)
.MODEL PMOS PMOS (LEVEL = 1
* Electrical Properties
+KP = 62.7e-6
+VTO = -0.4935548
+GAMMA = 0.029
+PHI = 0.279
+LAMBDA = 0.05

* Physical Parameters, these are overriden by the electrical parameters but are included for refernce
+UO = 100
+TOX = 5.6E-9
+NSUB = 1e15
*+LD = 0.0025e-6 * Not going to use this for now, it effects KP & COX & results in ~2-3% difference in IDS
)
.END

SIMULATION USING SPICE

Using the pull down menus, setup the SPICE models:

-Setup –SPICE Simulation

-High light "General" on the left on the dialog box that appears.

-On the right, click in the "Library Files" field. This is where you will specify any SPICE models you will be using in your simulations. Browse & select library files.

-On the right, click in the
"SPICE File Name" field. This is where you specify the name and location of the
SPICE Net list output.

-On the right, click in the "Simulations Results File Name" field. This is where the results of the simulation will be written. This file is what the waveform viewer will look for when you go to plot your results.

RESULTS:

FIG 8, SIMULATION OBTAINED IN MODELSIM:

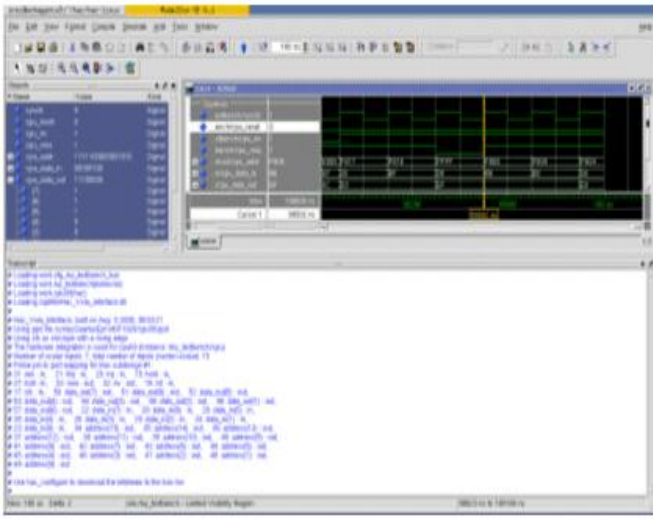


FIG 9,SIMULATION OF N-MOS

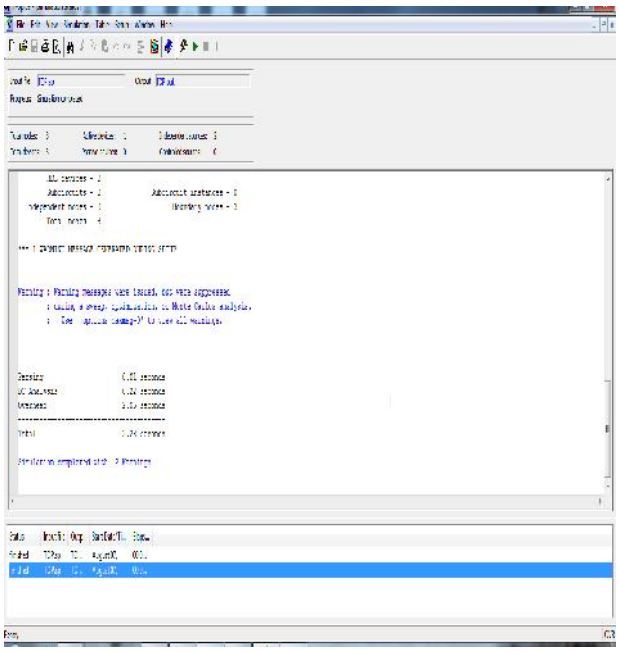
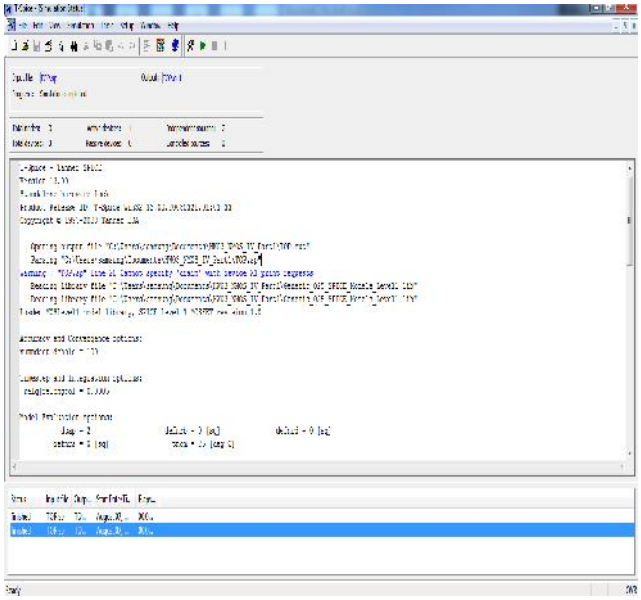


FIG 10,SIMULATION IN T-SPICE: NETLIST BUILD UP



References

[1] Prashant Upadhyay, Mr. Rajesh Mehra,"Low Power Design of 64-bits Memory by using 8-T Proposed SRAM Cell",International Journal of Research and Reviews in Computer
Science (IJRRCS), Vol. 1, No. 4, December 2010.

[2] Sreerama Reddy G.M, P. Chandrashekara Reddy, "Design and VLSI Implementation of 8 Mb Low Power SRAM in 90nm",European Journal of Scientific Research, Vol. 26, No. 2, pp. 305-314, 2009.

[3] Keejong Kim, Hamid Mahmoodi,"A Low-Power SRAM Using Bit-Line Charge Recycling", IEEE Journal Of Solid-State Circuits , Vol. 43, No. 2, 2008.

[4] K. Itoh, K. Sasaki, Y. Nakagome,"Trends in low-power RAM circuit technologies", in Dig. Tech. Papers, 1994 Symp. Low Power Electronics, 1994, pp. 84–87.

[5] Chang, Y., Lai, F., Yang, C.,"Zero-aware symmetric SRAM cell for reducing cache power in writing zero", IEEE Trans. VLSI Systems, Vol. 12, No. 8, pp. 827-36, 2004.

[6] J. L. Hennessy, D. A. Patterson,"Computer Architecture: A Quantitative Approach", 2nd ed. San Mateo, CA: Morgan Kaufmann, 1995.

[7] S. Kang, Y. Leblebici,"CMOS Digital Integrated Circuits",New York, McGraw-Hill,1999.

[8] Sreenivasa Rao Ijjada, B.Ramparamesh, Dr. V.Malleswara Rao,"Reduction of Power-Dissipation in Logic Circuits",International Journal of Computer Applications, Vol. 24, No. 6, 2011.

[9] Martin Mangla,"Low Power SRAM Circuit Design",Department of electrical and computerengineering, University of Alberta,1999 IEEE.

[10] C.M.R. Prabhu, Ajay Kumar Singh,"A proposed SRAM cell for low power consumption during write operation"Microelectronics International, Vol. 26, No. 1, pp. 37–42,2009.