ALU design, working, issues and tradeoffs

Digital VLSI Design S20

Team10: Sai Praneeth Chokkarapu (20161009), Surya Poondla (20161194), Ruchita Vucha (20161139), Sai Krishna Charan Dara (20171140)

Logic Design Techniques used in ALU

Presenter: Sai Krishna Charan Dara (20171140)

Arithmetic Logical Unit (ALU)

- ALU - digital system design. - Computer Processor
- Both - Arithmetic and Logical Operations
- ALUs required in VLSI and ASIC designs
- Smaller and more complex - Smaller computer.
- **Higher throughput** - Real-time signal and image processing applications.

Increasing Performance of ALU

- Arithmetic Part of ALU is made up of Full Adder and the logical part consists of Logical Gates such as AND,XOR,OR and XNOR.
- Improving - Adder cell and Logical Block - Improve Performance
- Various - Designs - CMOS PTL and GDI
- Trade Offs
- Choose the best logical design that suits for ALU
- Optimize power
- Optimize Delay

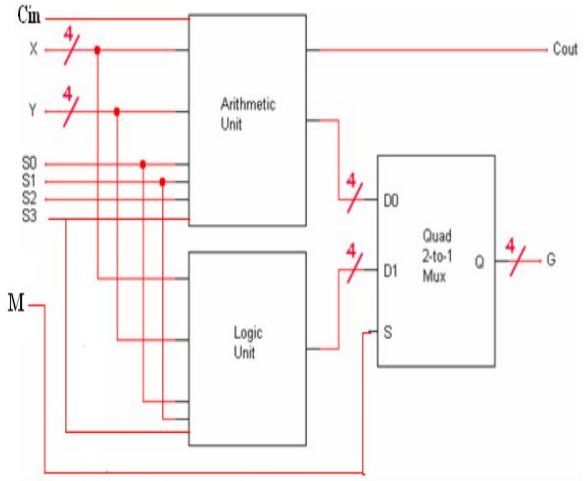
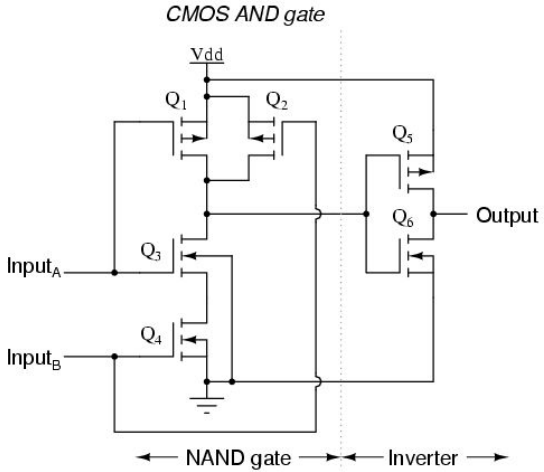
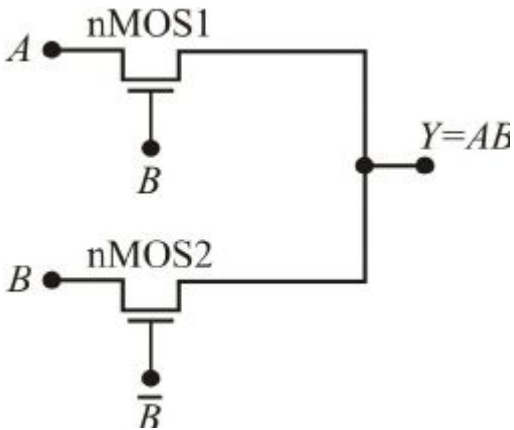


Figure 2 Block Diagram of ALU [15]

Logic Design Techniques

- Performance of logic circuits, once based on traditional CMOS technology are getting improved by developing of many logic design techniques.
- Logic that is popular in low-power digital circuits is pass-transistor logic (PTL).
- Pass transistor logic - based on NMOS transistor.



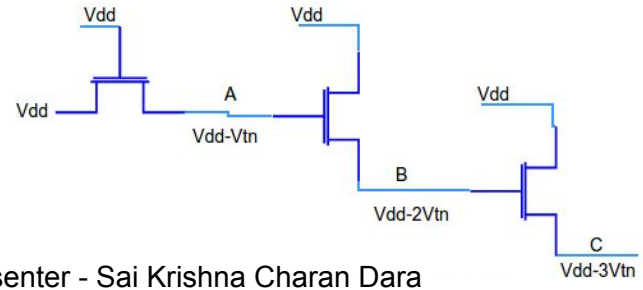


Advantages of PTL over CMOS

- Ratioless logic
- High speed
- Low power dissipation
- Lower interconnection effects
- No short circuit and static power consumption - Avoiding Sneak Path

Issues with PTL

- Threshold drop across the single-channel pass transistors
- Resulting Reduced current drive
- Slower operation at reduced supply voltages
- Operation at lower possible voltages - Not Desirable
- High Input Voltage Level - Regenerative Inverters - Not VDD
- PMOS Not fully OFF - Significant Direct Path **Static Power Consumption!.**
- No simple and universal cell library - PTL-based design.



Gate Diffusion Input (GDI)

- Solves most of the problems above
- Implementation of a wide range complex logic functions using only two transistors.
- Suitable - design of **fast, low power circuits,** using reduced number of transistors (Compared to CMOS and PTL)
- Improving **logic level swing** and **static power** characteristics and allowing simple top-down design.

GDI Cell Structure

- The GDI cell contains three inputs: G (common gate input of NMOS and PMOS)
- P (input to the source/drain of pMOS), and
- N (input to the source/drain of NMOS).
- Table-I : Generally these require 6 -12 transistors (CMOS or PTL)

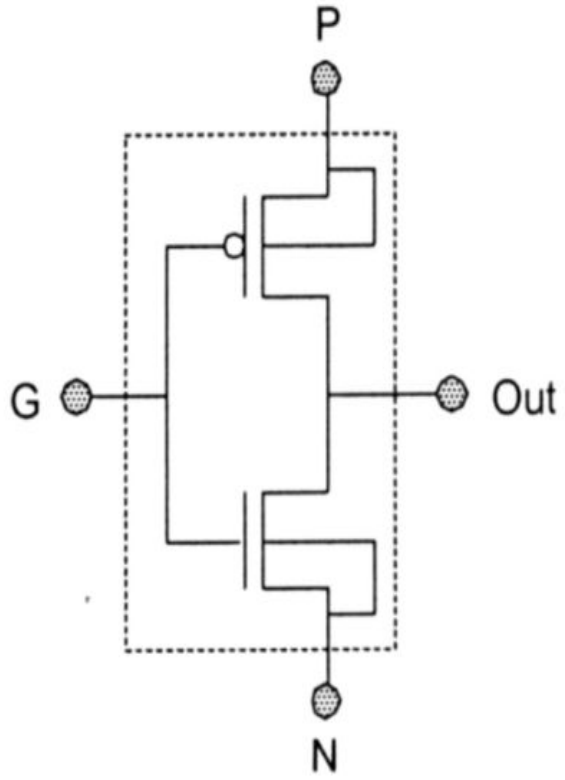
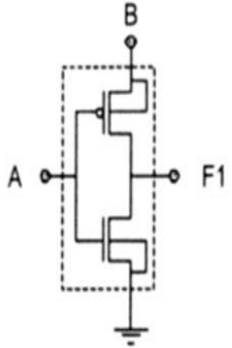


TABLE I
VARIOUS LOGIC FUNCTIONS OF GDI CELL FOR DIFFERENT INPUT CONFIGURATIONS

N	P	G	Out	Function
'0'	B	A	$\overline{A}B$	F1
B	'1'	A	$\overline{A}+B$	F2
'1'	B	A	$A+B$	OR
B	'0'	A	AB	AND
C	B	A	$\overline{A}B+AC$	MUX
'0'	'1'	A	$\overline{A}$	NOT

Swing Restoration in GDI

- Low output swing - A=0 and B=0
- Output here - V tp instead of 0 Volts.
- Only one of the four cases low output swing.
- GDI cell - Regular Inverter
- GDI cell functions - inverter buffer and recovers the voltage swing.
- Swing Restoration Logic is necessary for full output swing!



A	B	Functionality	F1
0	0	*pMOSTrans Gate*	V_{Tp}
0	1	*CMOSInverter*	1
1	0	*nMOS Trans Gate*	0
1	1	*CMOSInverter*	0

Delay of GDI compared to CMOS

- It is shown that the delay of CMOS gate compared to GDI gate with same functional complexity is given by:

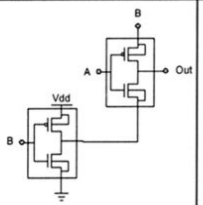
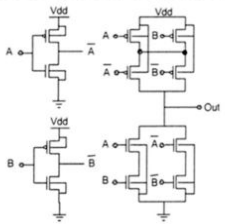
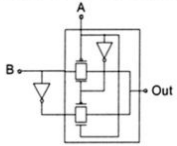
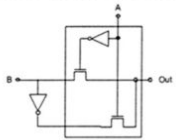
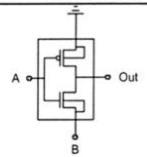
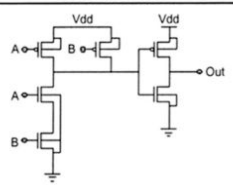
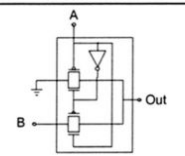
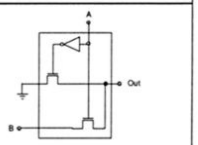
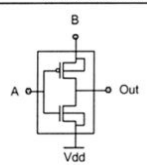
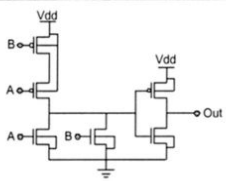
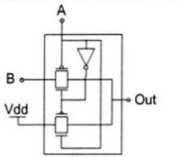
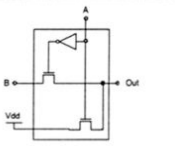
$$1.52 \leq \frac{t_{PHL(CMOS)}}{t_{PHL(GDI)}} \leq 2$$

where the high bound is for high output capacitive load and the low bound is for low output capacitance.

Performance evaluation of GDI Cell

- In ALU, we would be using different logical functions and multiplexers, so we compare GDI based implementation of delay,power and number of transistors with CMOS, Transmission gate (TG) and NMOS Pass Gate.

AND, OR, AND XOR CELLS USING GDI, CMOS, AND PTL DESIGN TECHNIQUES FOR TWIN-WELL PROCESS

	GDI	CMOS	TG	N-PG
XOR				
	4 transistors	12 transistors	8 transistors	6 transistors
AND				
	2 transistors	6 transistors	6 transistors	4 transistors
OR				
	2 transistors	6 transistors	6 transistors	4 transistors

Contd..

- To perform a fair comparison between the techniques, the measurements were carried out from cells series with buffers and not from a single cell.
- GDI and TG test circuits contain 2 basic cells with one output buffer. N-PG contains two buffers one after each cell. CMOS has no buffers in test circuit

LOGIC GATE COMPARISONS (GDI, CMOS, TRANSMISSION GATE, AND nMOS PASS GATE) USING THE CIRCUIT TOPOLOGIES FROMTABLE IV

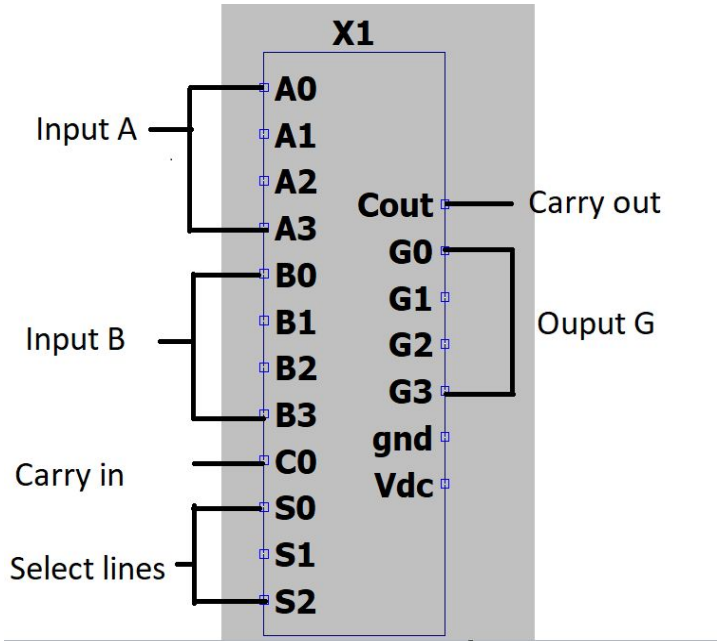
Gate type in series	Logic expression	GDI			CMOS			TG			N-PG		
		Power (μW)	Delay (nsec)	# tr.	Power (μW)	Delay (nsec)	# tr.	Power (μW)	Delay (nsec)	# tr.	Power (μW)	Delay (nsec)	# tr.
MUX	$\overline{A}B + AC$	35.7	1.1	8	49.7	2.1	24	44.9	1.0	16	47.5	3.1	16
OR	$A + B$	26.3	1.2	8	32.9	1.7	12	36.2	1.3	16	32.6	2.7	16
AND	AB	25.7	0.9	8	34.1	1.4	12	30.8	0.8	16	30.1	2.8	16
F1	$\overline{A}B$	31.2	0.8	8	45.2	1.5	12	31.8	1.1	16	31.8	2.5	16
F2	$\overline{A} + B$	32.0	1.3	8	43.1	1.9	12	33.2	1.4	16	29.6	3.5	16

Why GDI over CMOS,TG,N-PG?

- Among the presented design techniques, GDI proves to have the best performance values and lowest transistor count.
- Even in the cases, where power or delay parameters of some GDI gates are inferior, as compared to TG or N-PG, the power-delay products and transistor count of GDI are lower.
- Low Power
- Lesser Propagation Delay
- Less # of Transistors
- So **Choose GDI** !

ALU Design Type - I

- In some processors the ALU is divided into an arithmetic unit and logical unit.
- Full adder is used to perform arithmetic operations in alu. We designed a separate Logical block which takes care of the logical operations.
- Full Swing GDI technique is introduced as an alternative to CMOS logic in all the blocks.
- Proposed design of 4-bit alu consists of:
  (a) Multiplexers (b)Logical block (c)Full adder



4-bit ALU LTSpice symbol

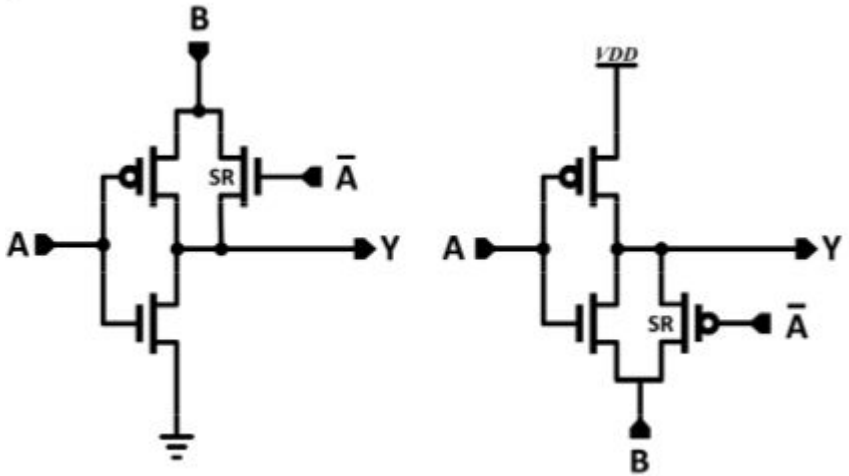
Truth Table for the 4-bit ALU

TABLE II. TRUTH TABLE OF THE PROPOSED 4-BIT ALU.

S2	S1	S0	Operations	Function
0	0	0	G = A - 1	DECREMENT
0	0	1	G = A + B	ADDITION
0	1	0	$G = A + \overline{B} + 1$	SUBTRACTION
0	1	1	G = A + 1	INCREMENT
1	0	0	$G = A \wedge B$	AND
1	0	1	$G = A \oplus B$	XOR
1	1	0	$G = \overline{A}$	XNOR
1	1	1	$G = A \vee B$	OR

Full Swing GDI

- Full-swing GDI cells proposed to improve the output swing of GDI gates as an alternative for swing restoration buffers, a swing restoration transistor used to improve the output swing of F1 and F2 gates.



Multiplexers

- 2 4X1 and 1 2X1
- 1st MUX : Arithmetic selection
- 2nd MUX: Logical selection
- 3rd MUX : Between Arithmetic and Logical

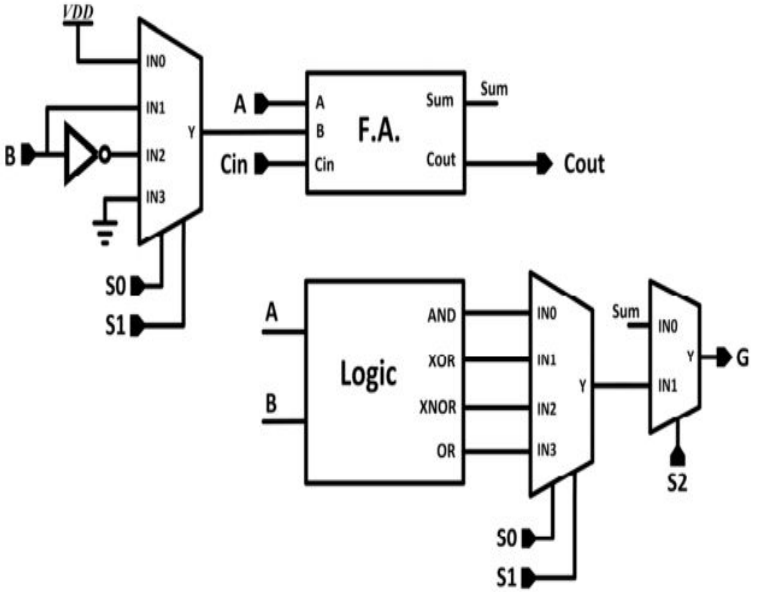
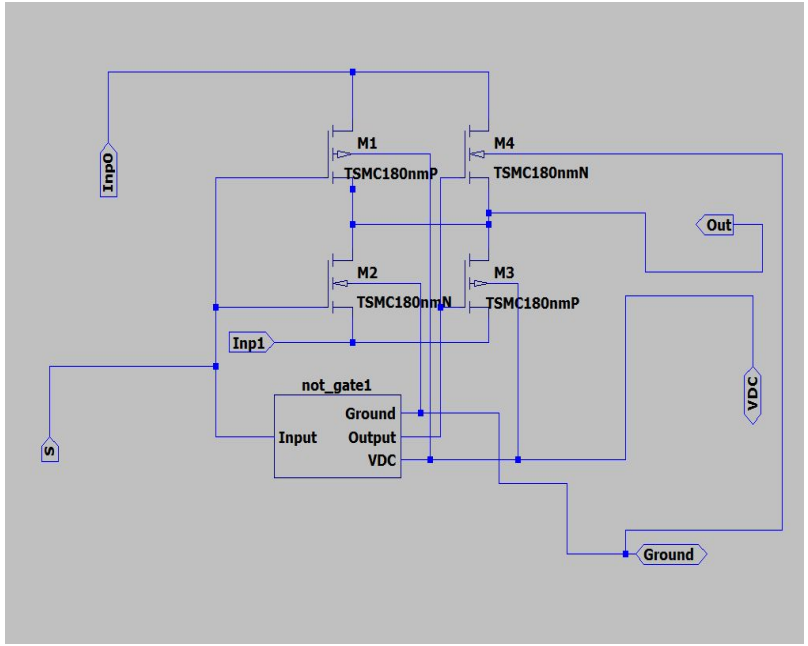
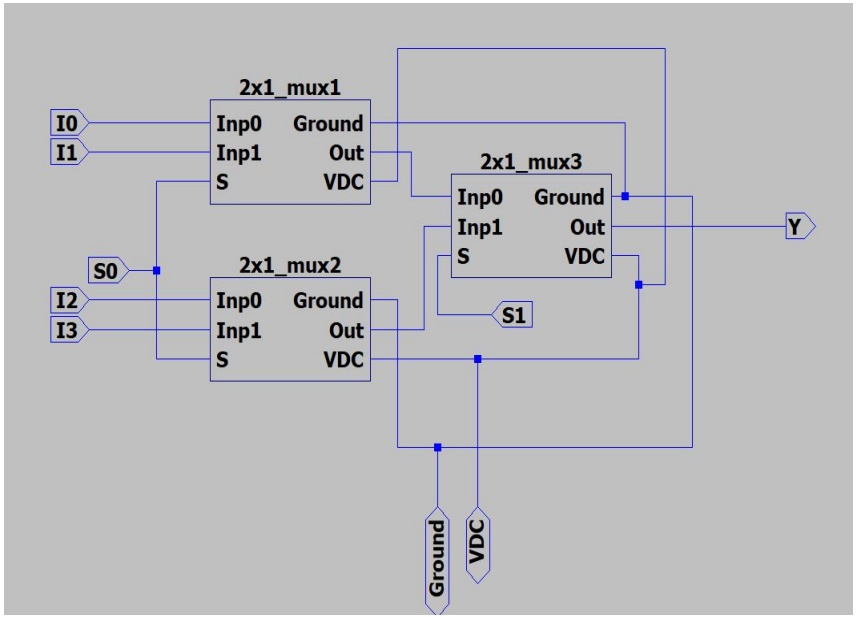


Fig. 9.Proposed 1-Bit ALU

Implementations in LTSpice



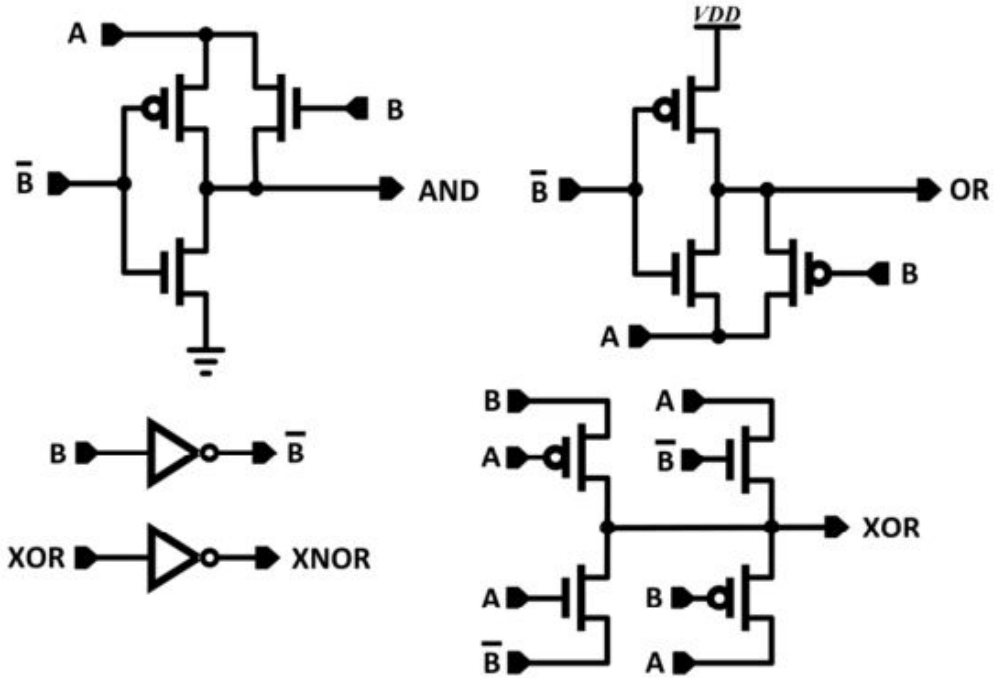
2x1 Mux - 6T



4x1 Mux - 18T

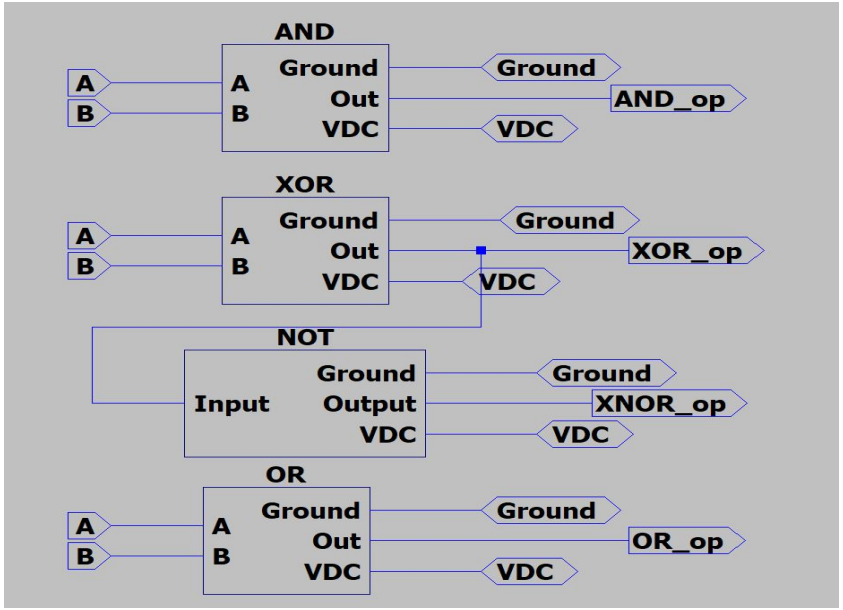
Logical Operations

- Design of logical operations
- AND XOR OR XNOR
- Using Full Swing GDI technique.
- All these are combined to form a logical block.



Logical block

- Performs the following logical functions on operands A & B: AND, XOR, XNOR & OR.
- Using a separate block for logical functions increases transistor count but decreases the delay.



Proposed ALU Designs (for Delay optimisation)

Presenter: Ruchita Vucha (20161139)

Full adder

- In the proposed design Full adder performs all the arithmetic operations.
- Design criteria - Transistor count, delay & power
- Full adder adds the 3 inputs A,B & Cin and produces 2 outputs out which can be logically expressed as follows:

$$Sum = A\mathbf{XOR}B\mathbf{XOR}C_{in}$$

$$C_{out} = A\mathbf{AND}B + B\mathbf{AND}C_{in} + A\mathbf{AND}C_{in}$$

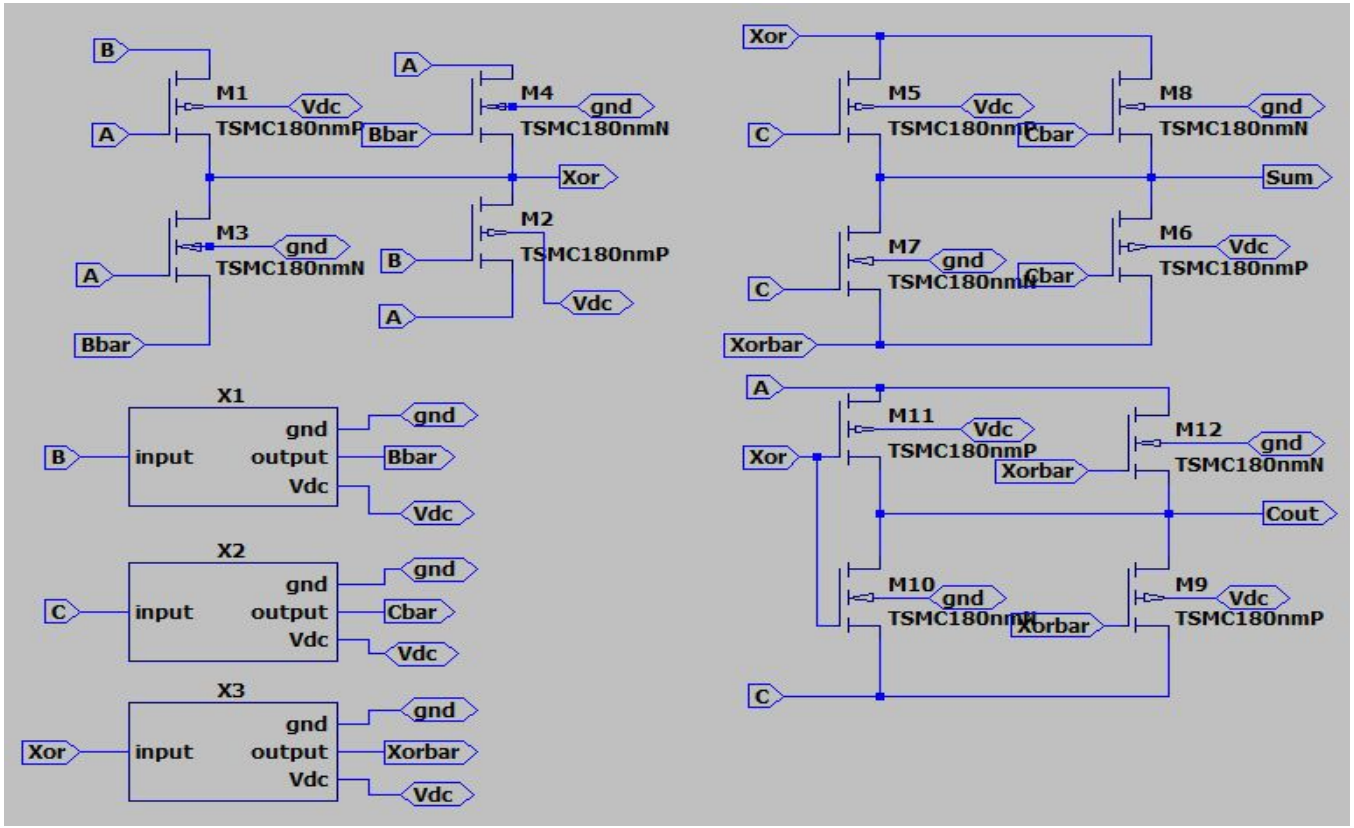
- Rewriting them :

$$Sum = \overline{C_{in}}\,(A\mathbf{XOR}B) + C_{in}\,(A\mathbf{XNOR}B)$$

$$C_{out} = (A\mathbf{XOR}B)\,C_{in} + \overline{(A\mathbf{XOR}B)}\,A$$

Presenter - Ruchitha Vucha

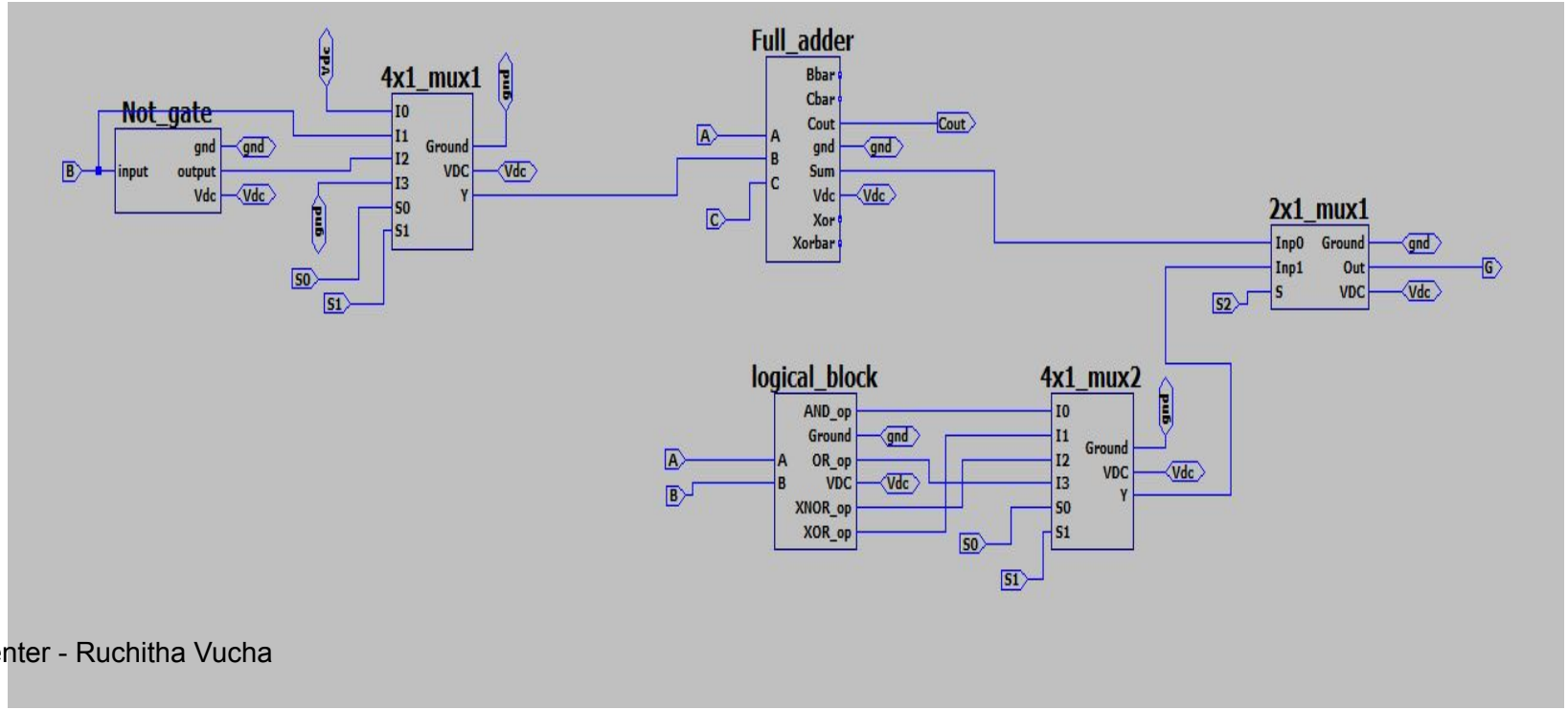
Proposed Full adder LTSpice Implementation



Presenter - Ruchitha Vucha

Integration of one-bit ALU

One-bit ALU is implemented by integrating the multiplexers, logic block and the full adder as shown below:



Presenter - Ruchitha Vucha

- Full adder performs arithmetic Operations on operand A by choosing the respective operand B among the { B, Bbar, 0 and 1 }.
- A 4x1 mux is used to choose among 4 logical operations And, Xor, Xnor & Or of the logical block using the select lines s0,s1.
- Select line s2 is used to choose between the arithmetic and logical operation.
- Using the select lines we can perform anyone of the 8 operations.

Decrement : s2 = 0 s1 =0 s0 = 0

The output of 4x1 mux is 1 so 2nd input of the full adder is 1.

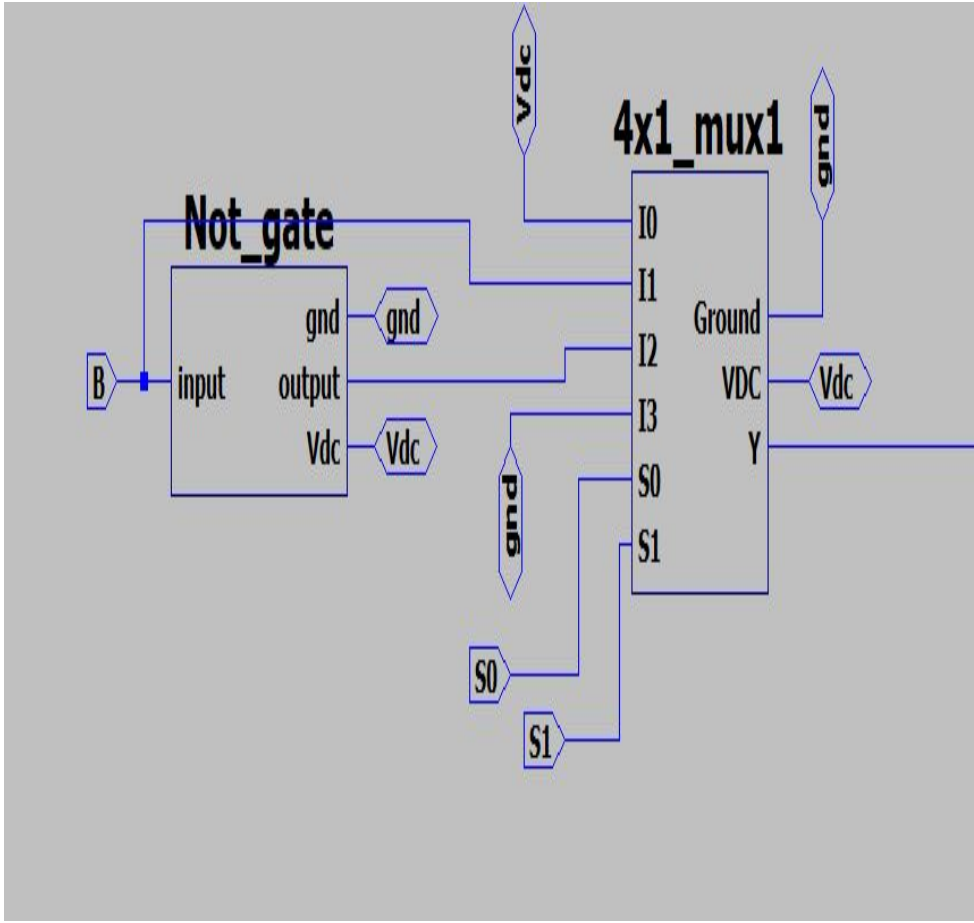
Cin = 0, A is summed with 1 (represents -1 in 2's complement)

i.e **sum = A-1**

Addition: s2 = 0 s1 =0 s0 = 1

The operand B is passed as 2nd input to the full adder and Cin = 0. A is summed up with B i.e

sum = A + B.



Subtraction: s2 = 0 s1 =1 s0 = 0

Bbar is the 2nd input of the full adder.

Performed by summing 2's complemer

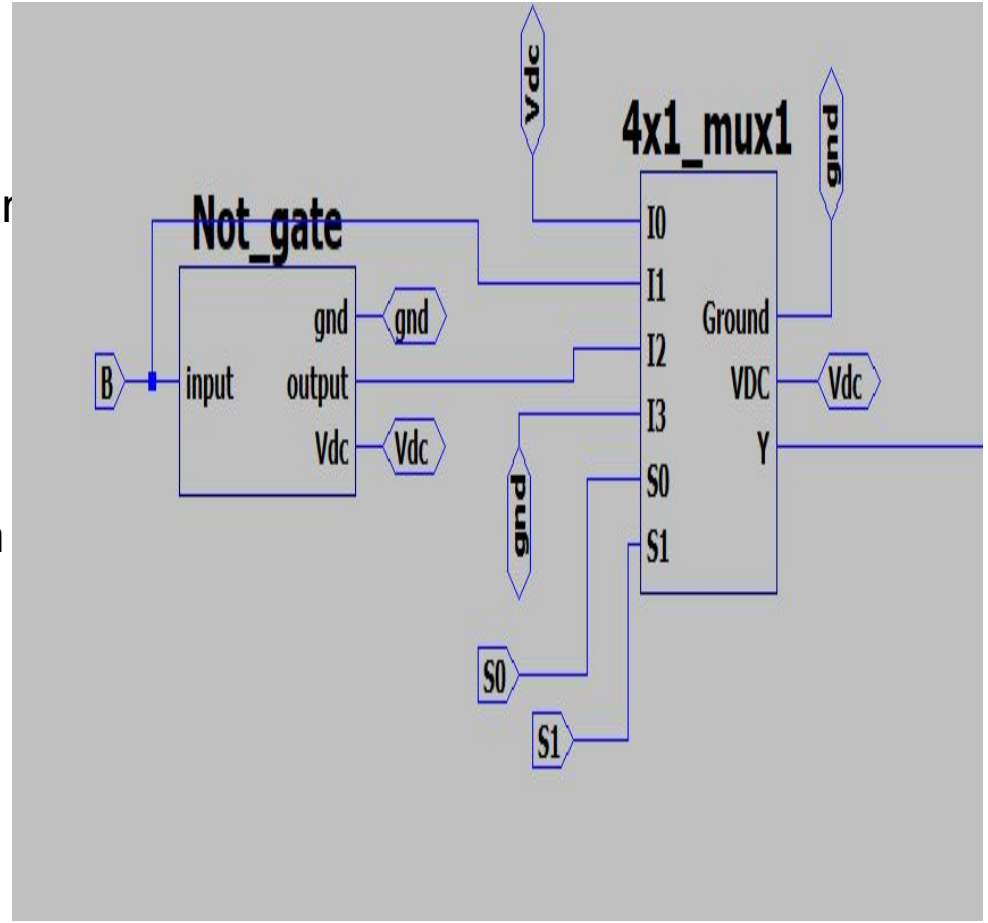
A with B' and Cin =1 i.e

Sum = A + Bbar + 1 = A - B.

Increment: s2 = 0 s1 =1 s0 = 1

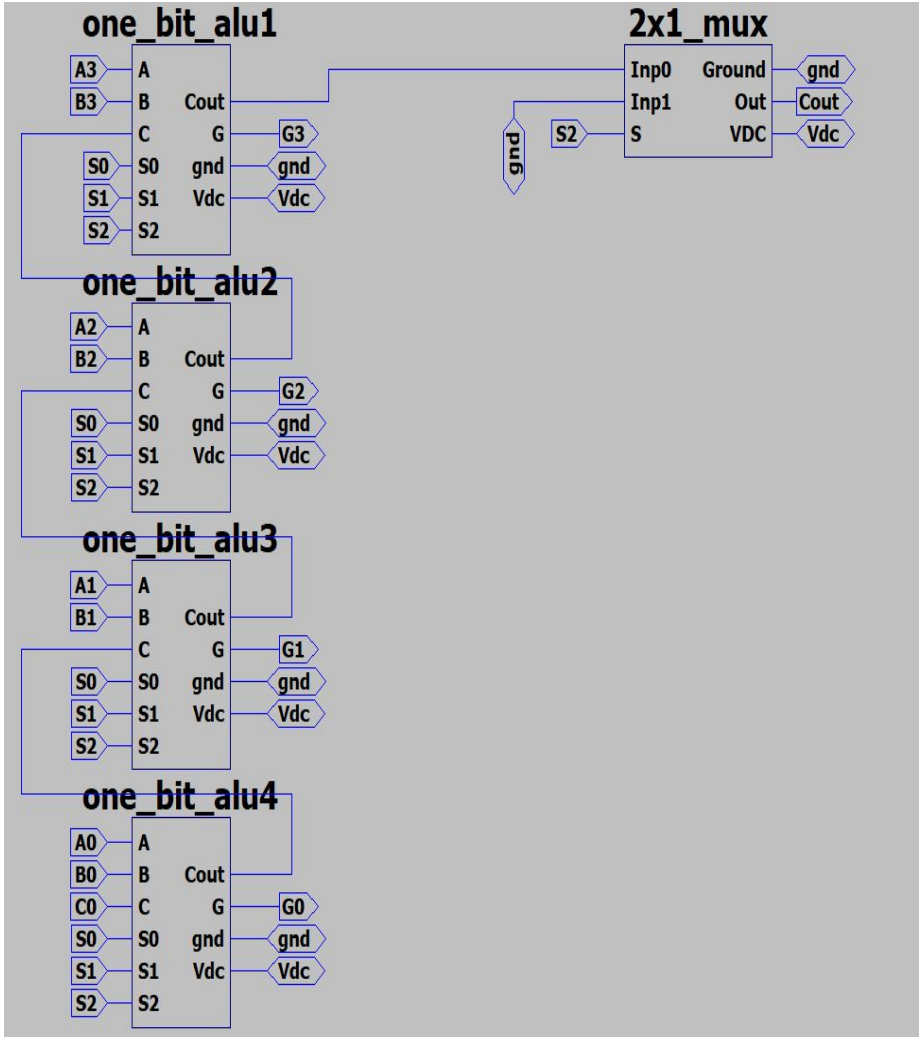
0 is the 2nd input of full adder and Cin = 1.

A is summed with 1 i.e **sum = A + 1**



Integration of four-bit ALU

Four bit alu is implemented by cascading the one-bit alus in four stages as shown below. The carryout of alu is given as carry in for the alu in the next stage.



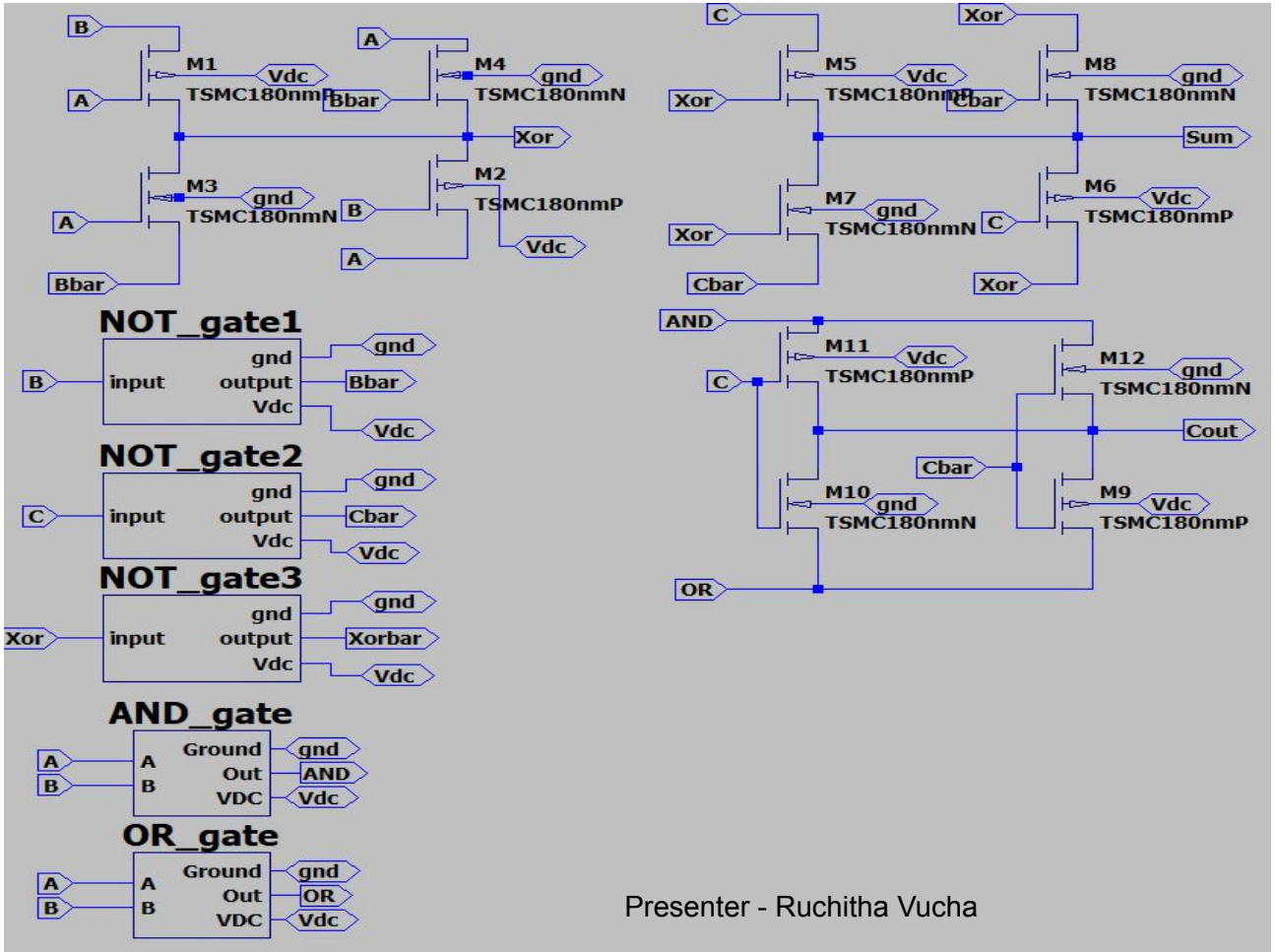
ALU Design Type - II

- Full Swing GDI technique is used. Mux implementation remains the same.
- No separate logical block as Adder performs arithmetic and logical operations as well.
- Helps maintain low power and transistor count.
- Outputs rewritten in terms of the inputs using XOR, AND & OR operations:

$$Sum = A\mathbf{XOR}B\mathbf{XOR}C_{in}$$

$$C_{out} = \overline{C_{in}}\,(A\mathbf{AND}B) + C_{in}\,(A\mathbf{OR}B)$$

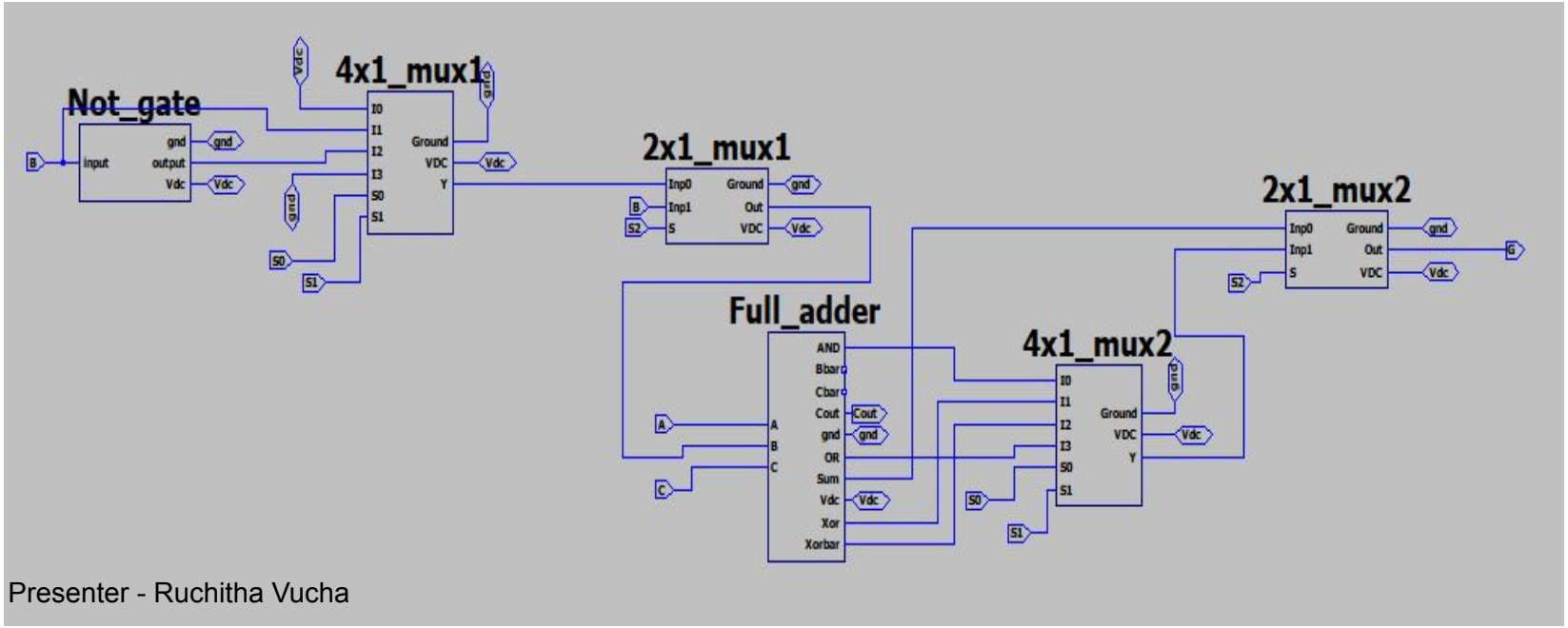
Full adder implementation in LTSpice using AND , OR & XOR gates



Presenter - Ruchitha Vucha

Integration of one-bit ALU

The full adder itself performs the 4 logical operations. 4x1 mux to choose the appropriate output. 2x1 mux to choose between the logical and arithmetic outputs. The 4-bit alu implementation remains the same.

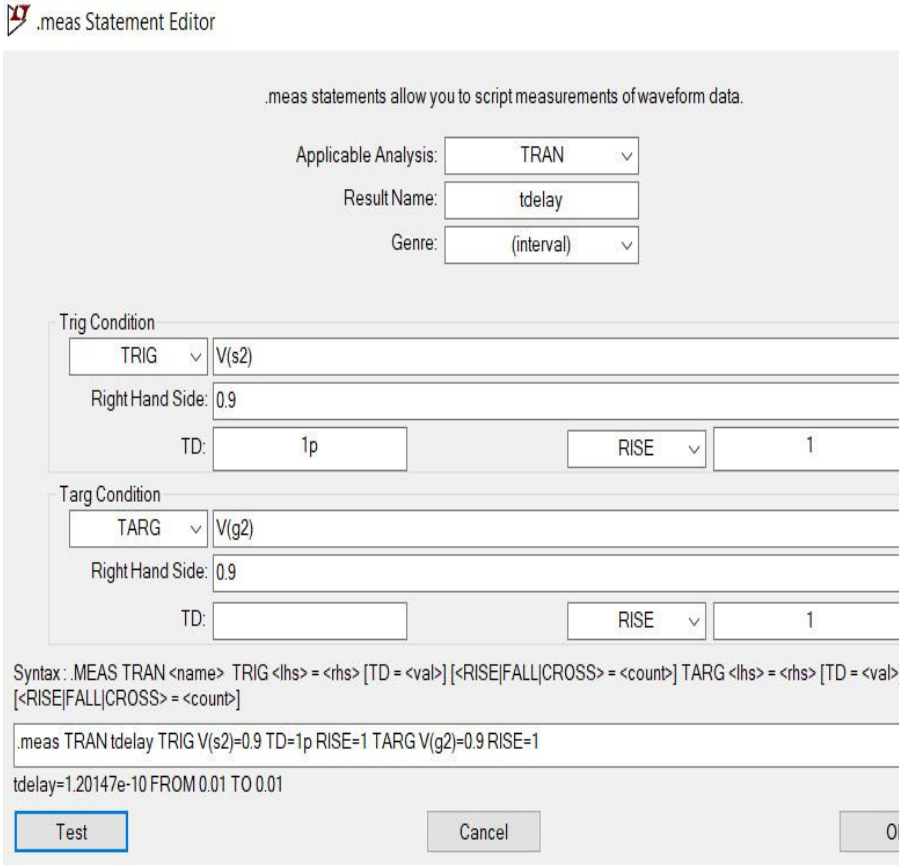


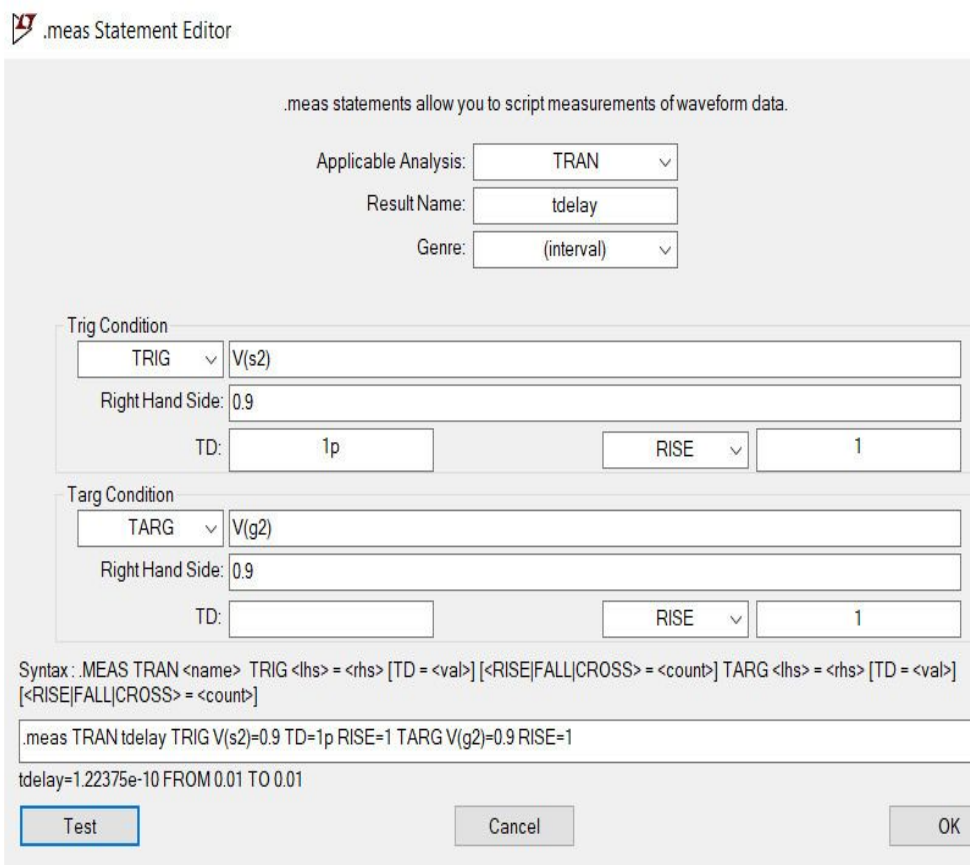
Verification of simulation results :

LTSpice Simulations with Power supply: 1.8V.
TSMC 180nm Technology
Transistor sizing: $(W/L)_p$ = 720nm /180nm $(W/L)_n$ = 360 nm /180 nm.
Delay of Design-I < Delay of Design-II.
Design-II performs better in terms of Power and transistor count.
(Here Average Power is calculated over all 8 possible operations of ALU).

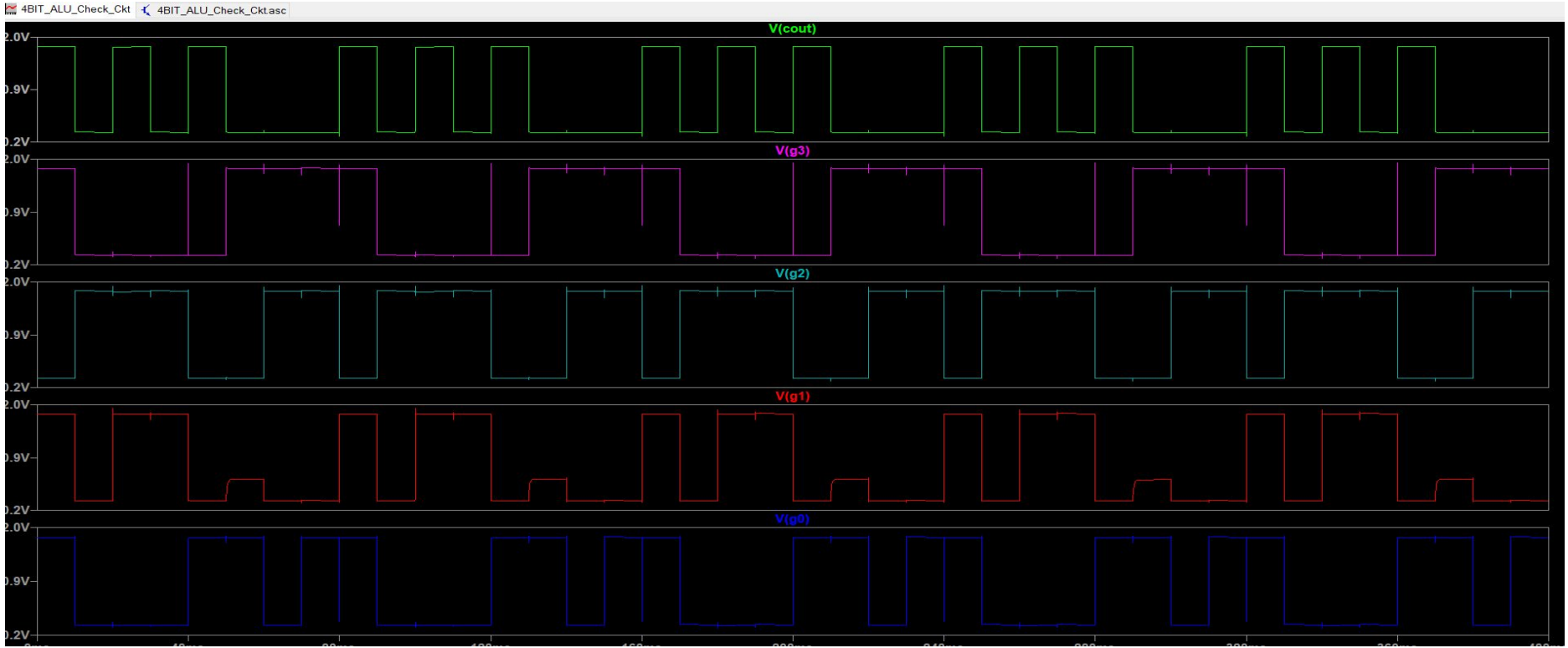
	Delay	Power	Transistor Count
ALU Design Type - I	0.122375 ns	54.927 µW	294
ALU Design Type - II	0.120147 ns	48.627 µW	286

Delay for Design I & Design II :





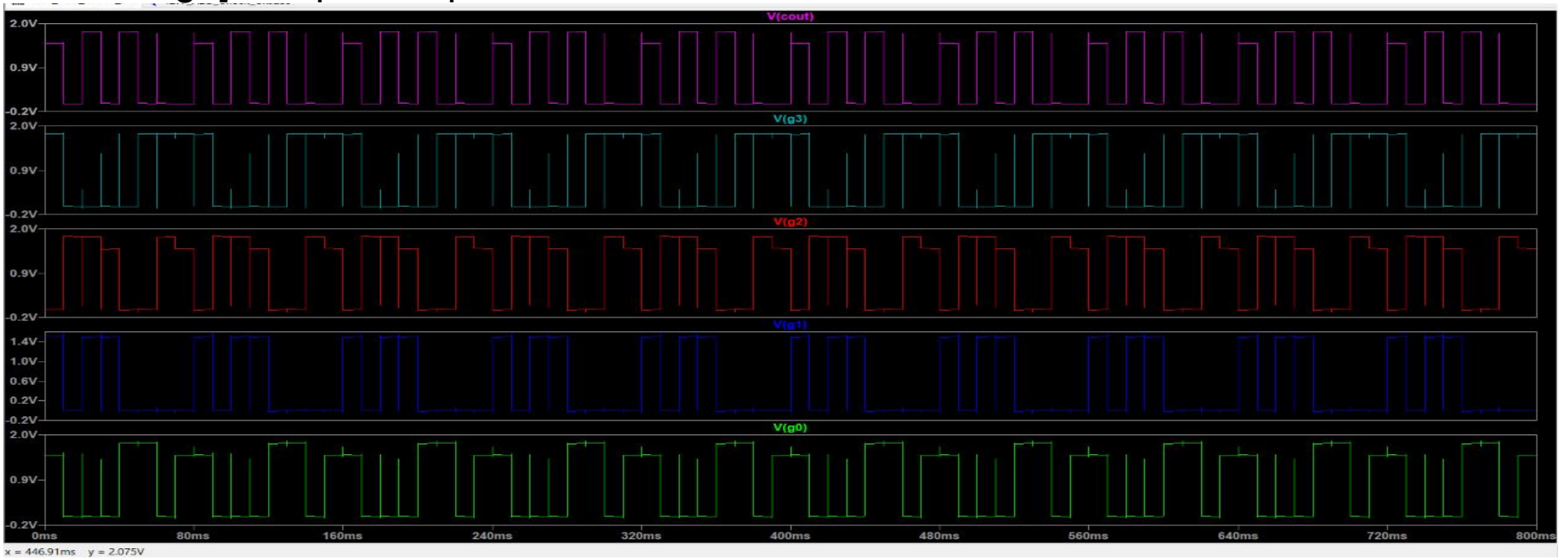
The output waveform of the four-bit alu for the test inputs A = 1100 B = 0101 is:



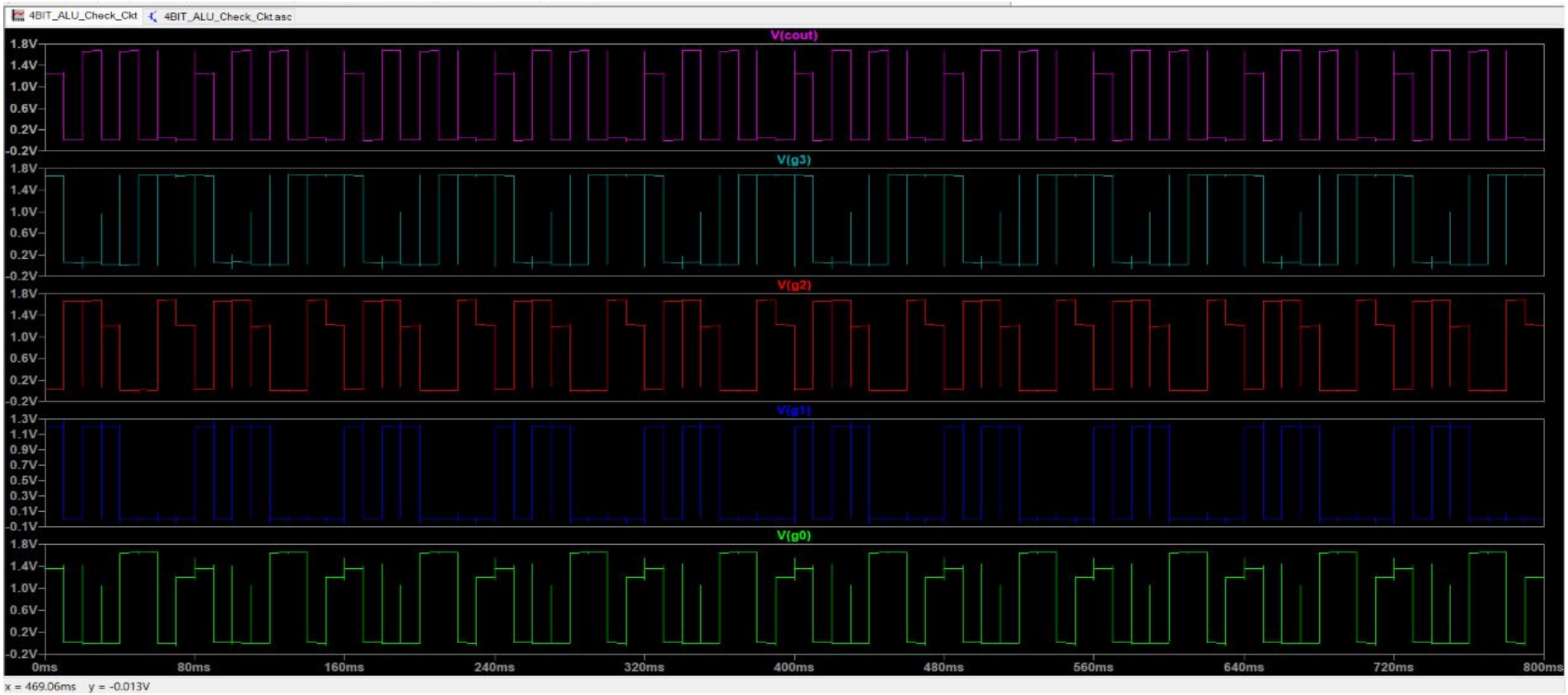
Presenter - Ruchitha Vucha

Voltage Variation:

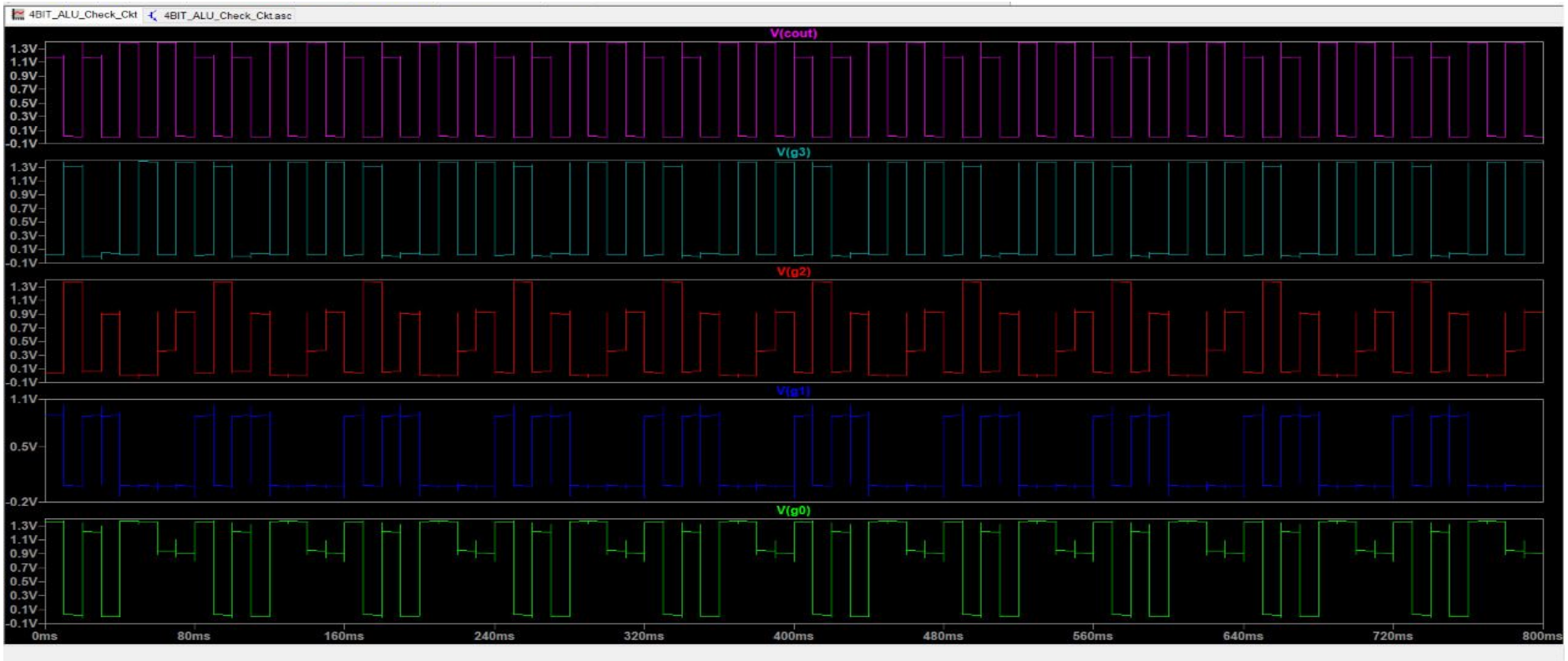
Found that the circuit works for input voltage greater than 0.9 [assuming voltage > 0.9 as high]. Output Graph for V = 1.5 :



Output Graph for V = 1.2 :

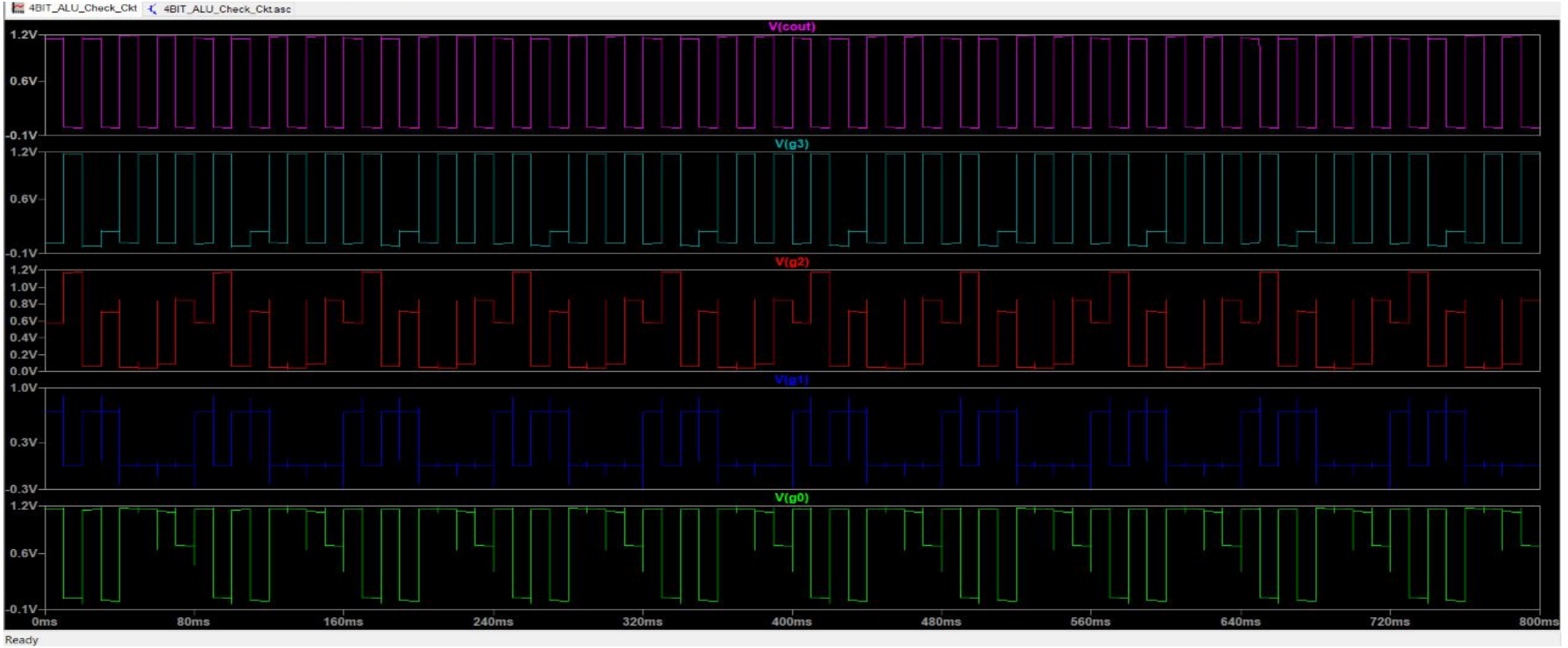


Output Graph for V = 0.9 :



Presenter - Ruchitha Vucha

Output Graph for V = 0.7 :



Presenter - Ruchitha Vucha

Power Optimization of ALUs

Presenter: Surya Poondla (20161194)

Power Optimization

- The ALU is always on, to perform various operations!!
- Power optimization can be done in many ways
    - Transistor Sizing
    - Threshold voltage scaling
    - Clock gating and power gating
    - Dynamic Voltage Scaling
- Implemented power optimization(Dynamic) based on a combinations of methods

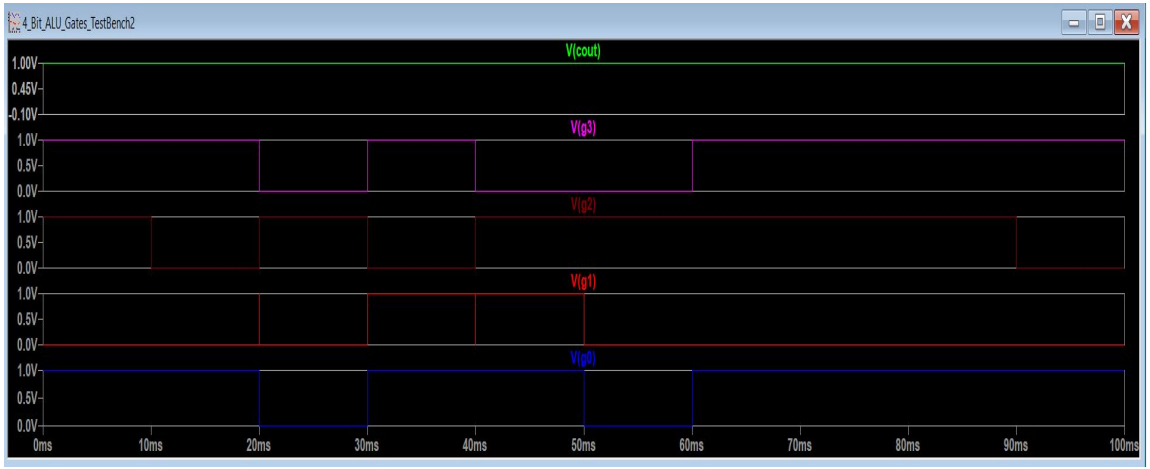
Power Optimization Architectures

- Implemented Architectures
    - CMOS Based Logic Tree Architecture
    - CMOS Based Chain Architecture
    - GDI based optimized 8T Architecture
- The above are implemented on LTSpice
- Power optimization circuits are mostly application specific!!

ALU Functionality

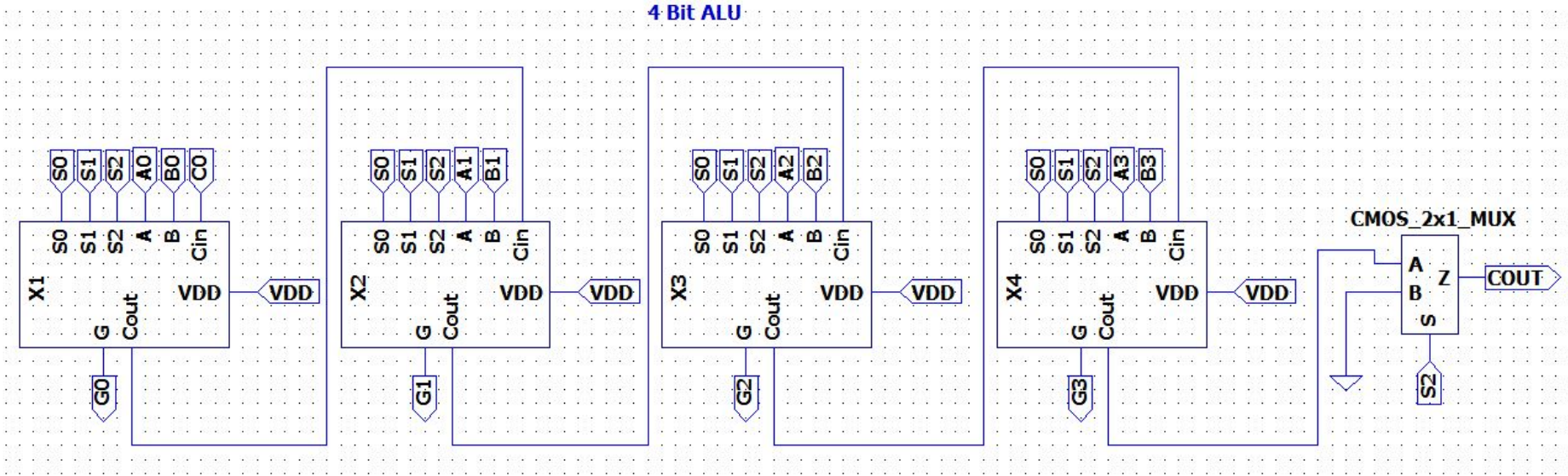
S0	S1	S2	Operation	ALU Functions
0	0	0	G=A-1	DECREMENT
0	0	1	G=A^B	AND
0	1	0	G=A+B´+1	SUBTRACTION
0	1	1	G=Ā	XNOR
1	0	0	G=A+B	ADDITION
1	0	1	G=A⊕B	XOR
1	1	0	G=A+1	INCREMENT
1	1	1	G=A∨B	OR

Truth Table



Testbench Simulations

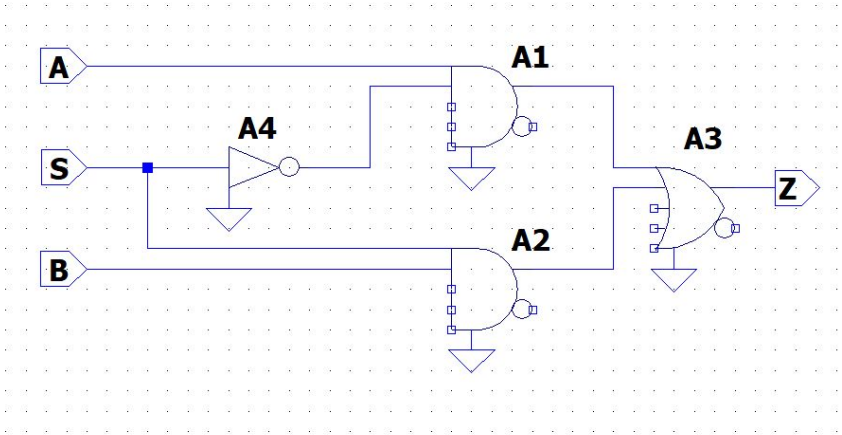
4-Bit ALU Architecture



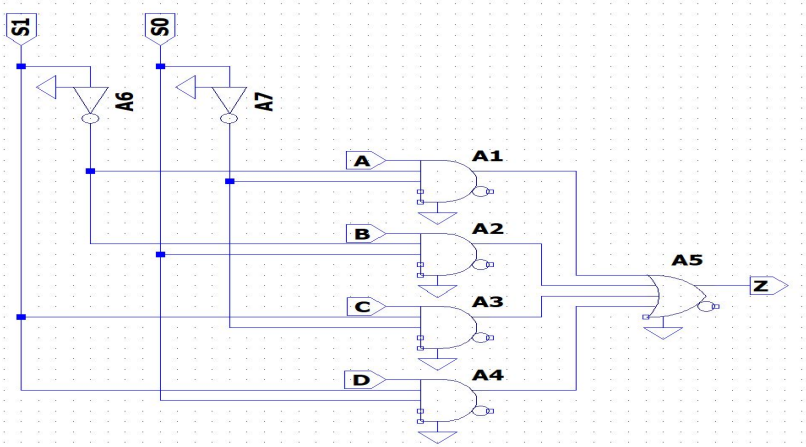
4 Bit ALU block in LTSPICE

For all architectures this is the generic 4-bit ALU Architecture. The 1-Bit ALU's architectures are different for all the designs.

CMOS based Tree Architecture Implementations



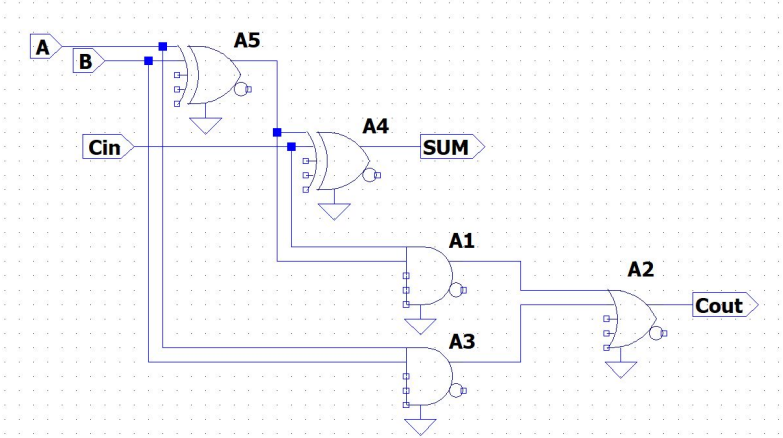
2x1 Mux CMOS Logic



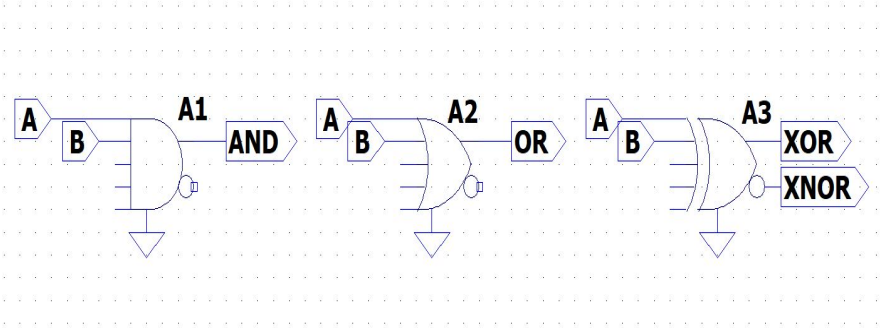
4x1 Mux CMOS Logic

Used 180nm Technology to implement CMOS architectures.
Ln = Lp = 180nm

Implementations

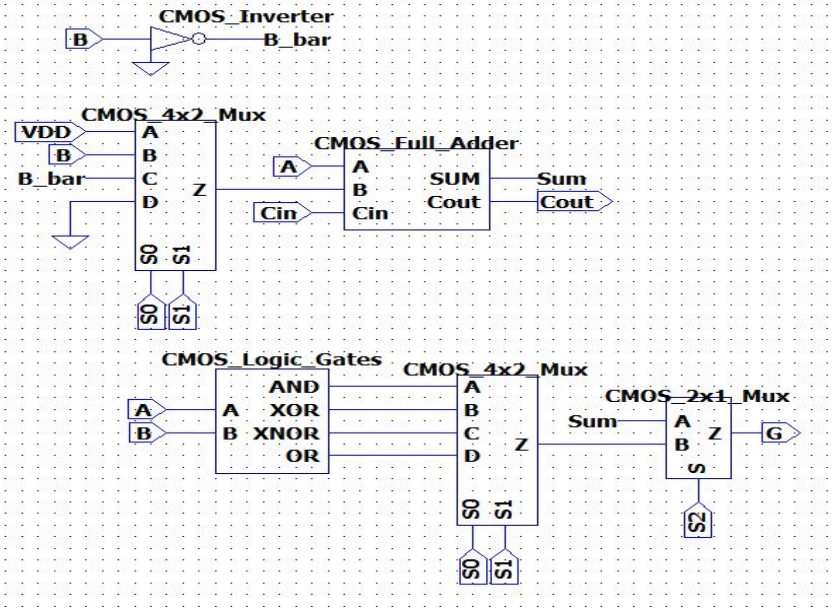


Full Adder CMOS Logic



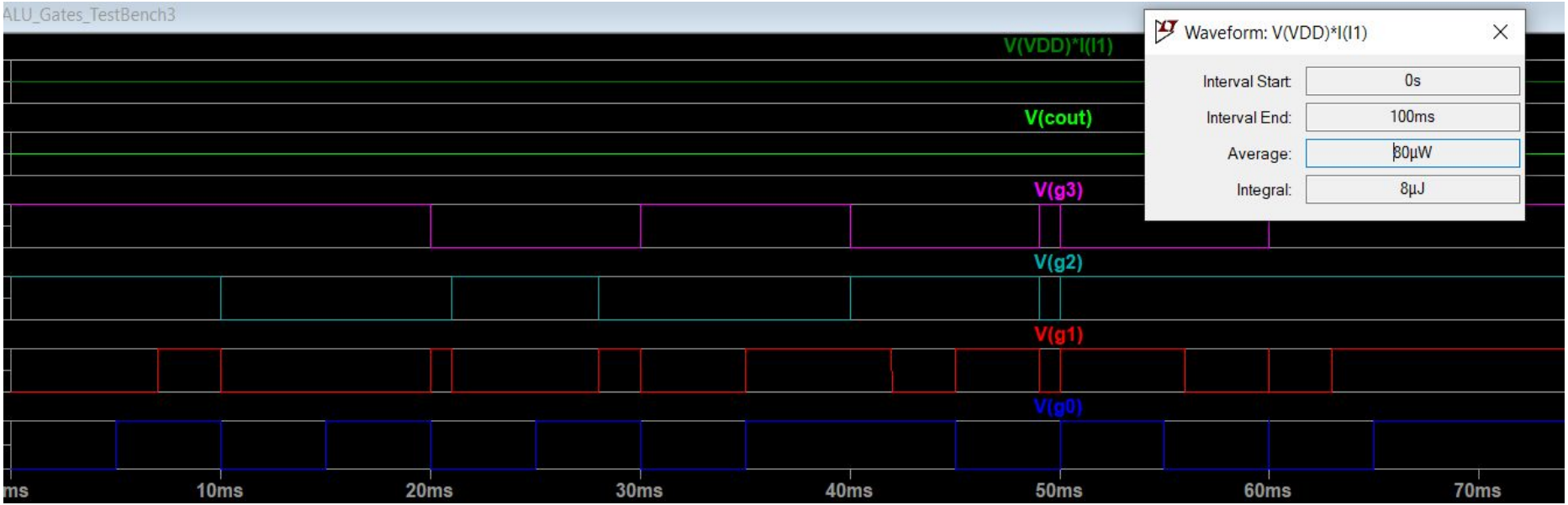
Logic Block CMOS Logic

1-Bit ALU Tree Architecture



1 Bit ALU block in LTSPICE

Simulations of Tree Structure



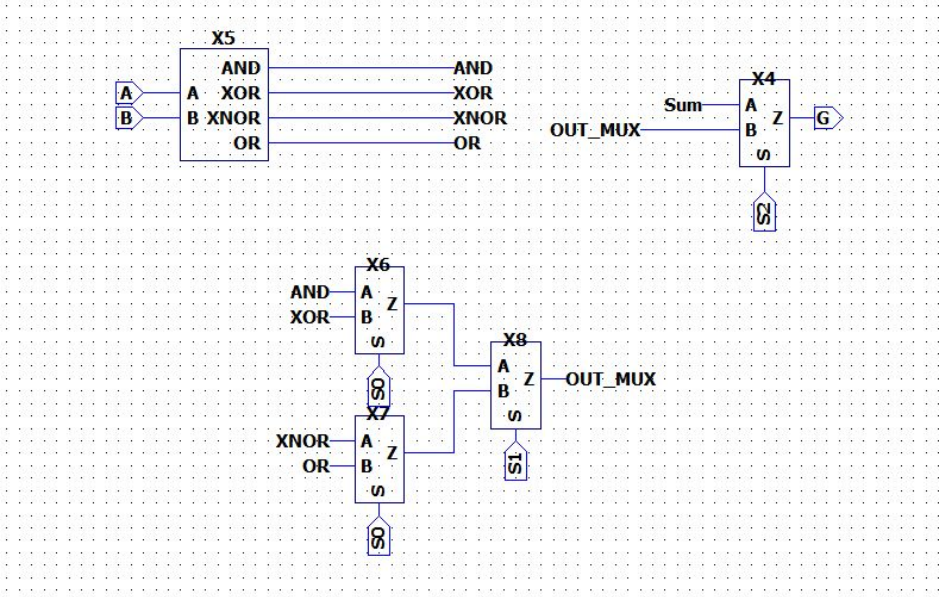
Average Power during Switching

Inputs :- Pulses to examine the different switching powers(All Combinations)

CMOS based Chain Architecture

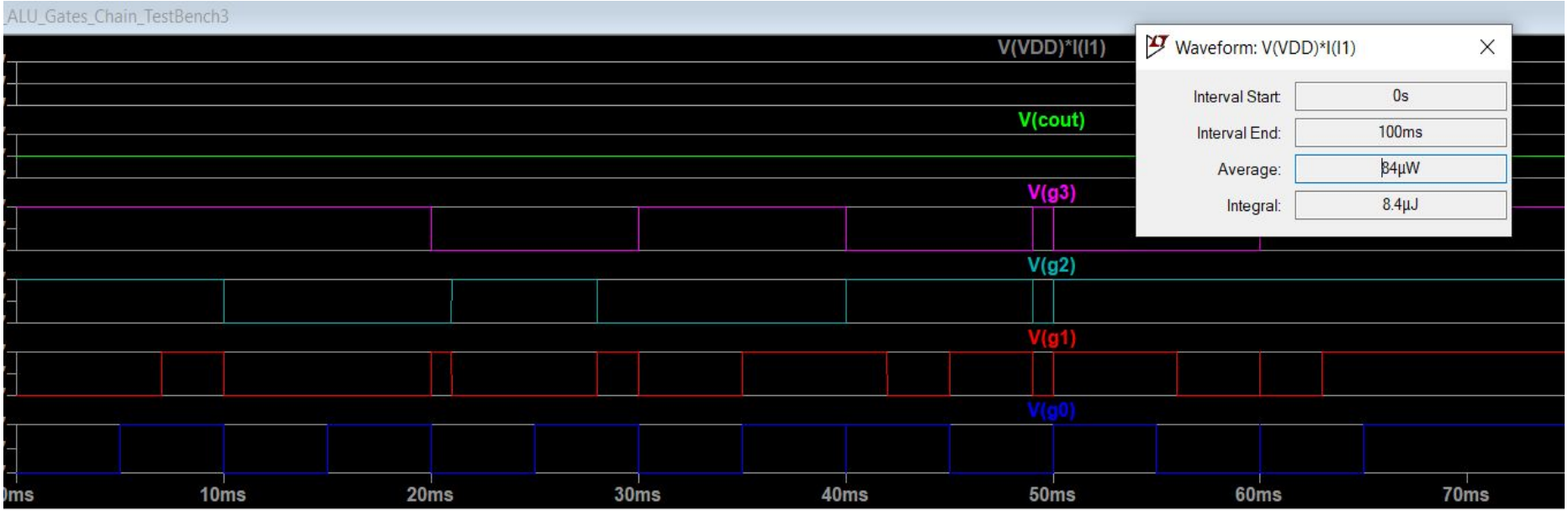
- The Chain Structure is mostly based on Applications!!
- If our applications demand too many AND operations or any other operations we modify our ALU Architecture.
- The Chain structure has comparatively lower area but has higher power dissipation

1-Bit ALU Chain Architecture



1 Bit ALU Chain in LTSPICE

Simulations of Chain Structure

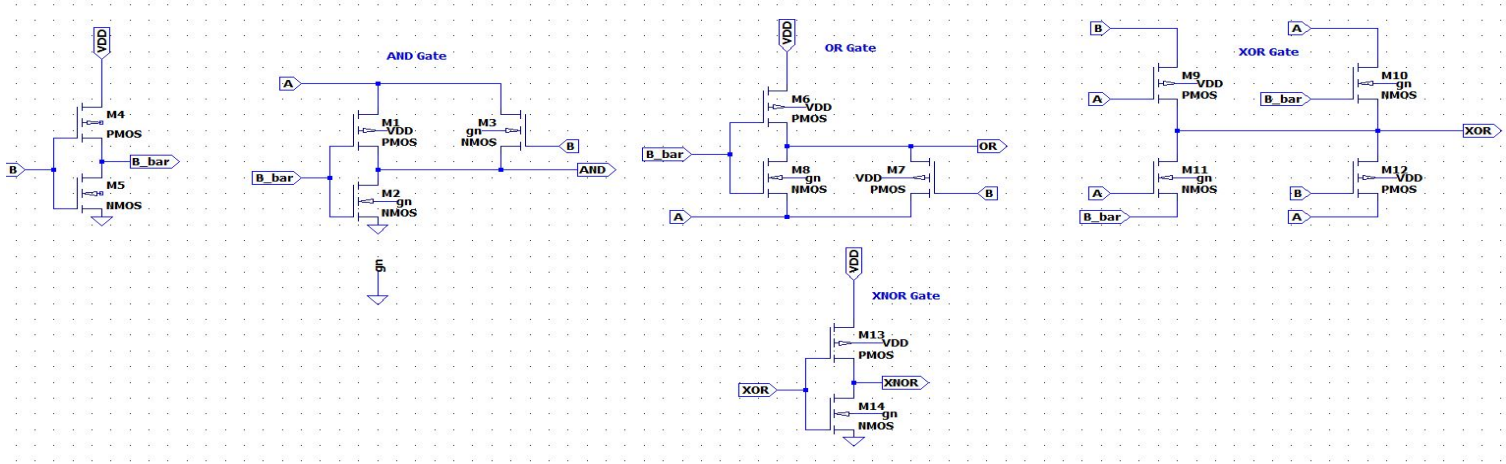


Average Power of Chain Architecture at Switching

Inputs :- Pulses to examine the different switching powers(All Combinations)

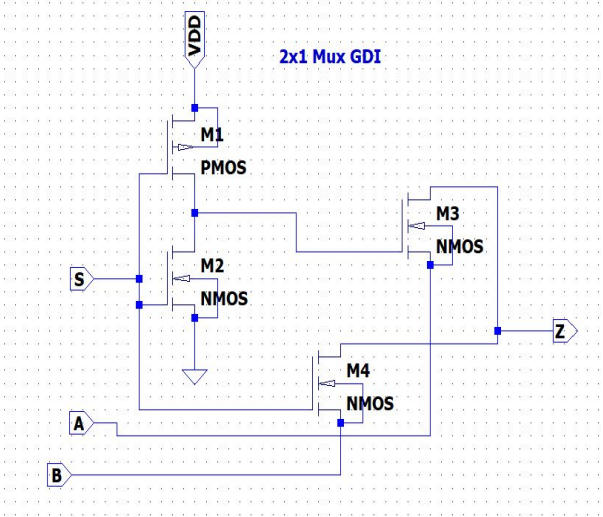
GDI based Architecture

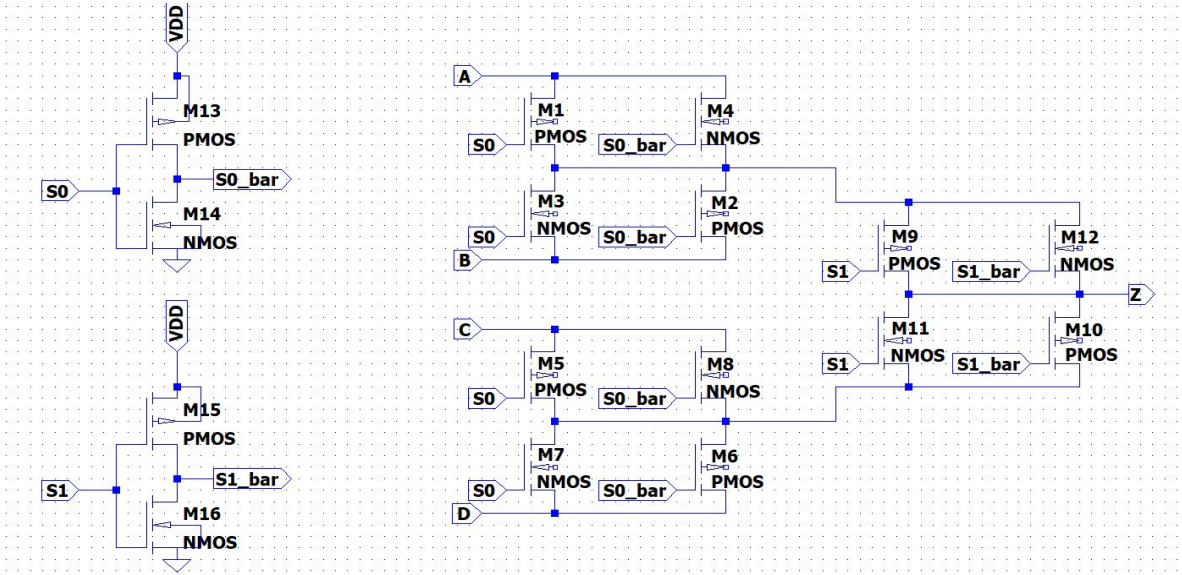
- The GDI based architecture is very efficient to reduce power and also keep the delays in bounds!!
- Implemented 4-bit ALU in GDI design. Used Technology 180nm.



Logic Block in GDI

GDI implementations in LTSpice

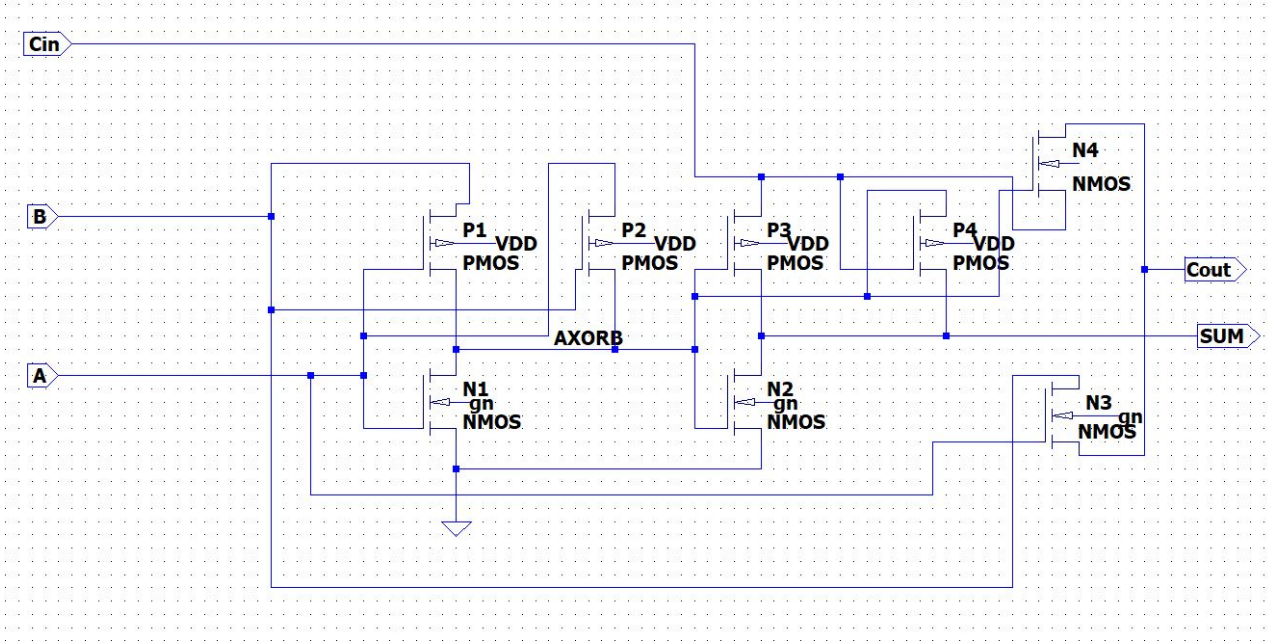




2x1 MUX using GDI

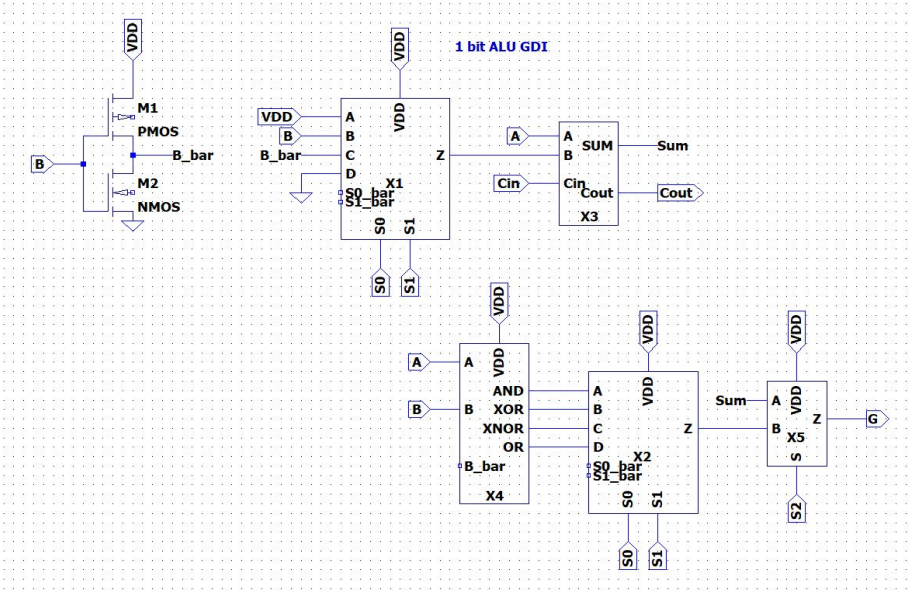
4x1 MUX using GDI

GDI implementations in LTSpice



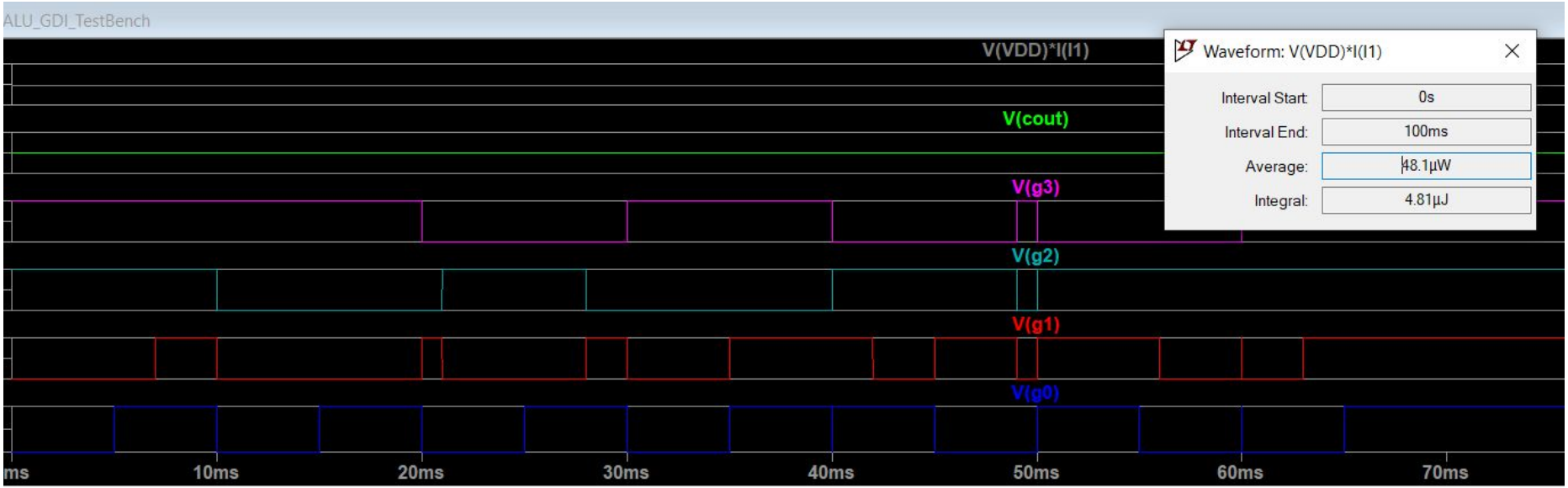
8 Transistors based Full Adder using GDI

1-Bit ALU Using GDI Architecture



1 Bit GDI ALU block in LTSPICE

Simulations of GDI Structure



Average Power of GDI Architecture at Switching

Inputs :- Pulses to examine the different switching powers(All Combinations)

Comparison between the Architectures

Design	Avg Power Dissipation	Min Avg Power	Total Number of Transistors	Area
CMOS based Tree Architecture	80u Watts	50u Watts	312	Moderate
CMOS based Chain Architecture	84u Watts	52u Watts	320	Less
GDI Based 8T Architecture	48.1u Watts	22u Watts	184	Least

GDI method also reduces Static Power too!!

Testability in ALUs

Presenter: Sai Praneeth Chokkarapu (20161009)

Why is testability important?

- We wish our circuit design to be such that it is easily and quickly testable.
- With the advent of technology, the reduction in feature size will definitely increase the chance of a manufacturing defect.
- It takes only one faulty transistor or wire to make the entire chip fail to function properly.
- So, the designs have to be testable and hence Design for Testability (DFT) is important.

Fault models in DFT

- Input pattern (IP) fault model
- Cell fault model
- Delay fault model
- Single stuck-line (SSL) fault model
- And many others in literature

Different types of Testability

Let’s say the input operand size is N. The widely used levels of testability in the literature are,

- Lin-testability
- C-testability
- I-testability
- CI-testability
- pI-testability

A circuit design is evaluated based on the fault model assumed and then testability of the circuit changes as per the design and fault model.

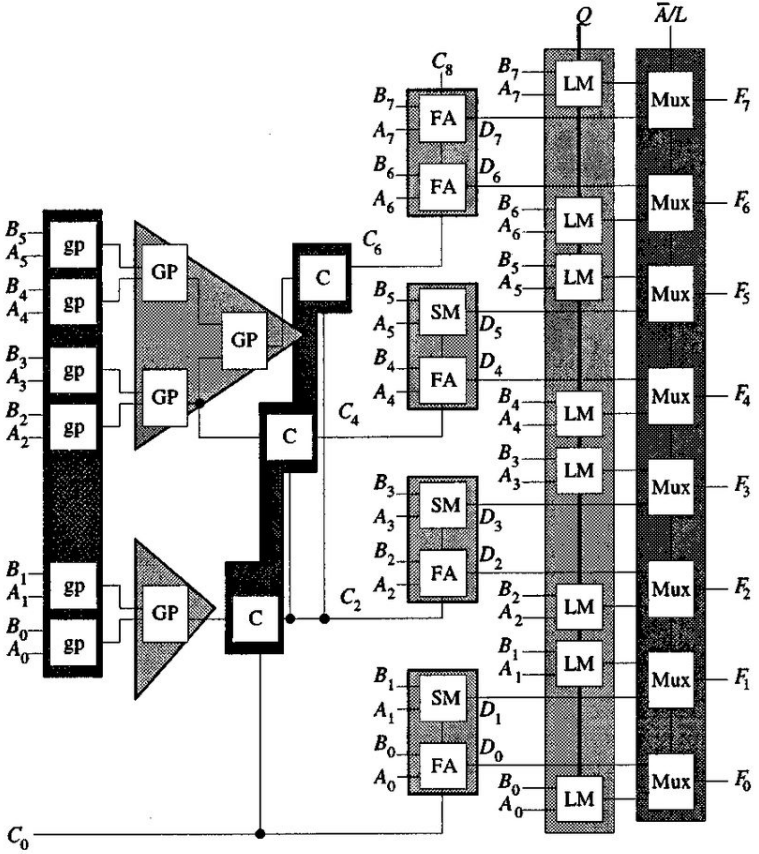
Literature Review

- Prior work has shown that an array structure improves an ALU's testing properties. The disadvantage here is that its worst case delay is linearly proportional to N, where N is the input operand size of the ALU.

- Carry-lookahead circuitry in the adder changes the worst case delay from being linear in N to logarithmic in N but increases the hardware cost and the C-testability of the circuit is lost.

8-bit tree ALU example

- Each LM module executes various logical operations controlled by the input bus Q. Arithmetic operations are performed using the modules gp, GP, C, FA, and SM.
- Much of the tree ALU's circuitry is easy to test. The testing difficulty associated with the tree ALU arises from the carry-lookahead circuitry.
- So, some design changes have to be done so that it becomes fully testable i.e all the possible faults have to covered.



Analysis and Observations

- Methods for designing Lin-testable and C-testable ALU's are proposed in a few papers.
- From this table, we can observe that the gates overhead increases as the testability of the circuit improves, from typical to Lin-testable to C-testable.
- Based on the application and the design specification, there will be a trade offs.

Circuit property	Circuit type	8-bit ALU	16-bit ALU
Number of gates overhead (%)	Unmodified	481	915
	Lin-testable	494 (2.7)	1052 (15.0)
	C-testable	603 (25.4)	1249 (36.5)
Number of IP faults	Unmodified	752	1624
	Lin-testable	824	1768
	C-testable	824	1768
Number of IP tests and coverage (%)	Unmodified	312 (85.11)	672 (83.5)
	Lin-testable	384 (100)	816 (100)
	C-testable	195 (100)	195 (100)

Analysis applies to different designs

- Such a testability analysis applies to other carry-lookahead structures also. For example, carry modules might use both carry-lookahead circuitry and ripple-carry circuitry.
- This allows the speed/area tradeoffs to be made without affecting testability

Design techniques in DFT

- While scan-path design techniques have steadily improved the testability of complex digital circuits, built-in self-test techniques can improve it more.
- In BIST, the input test vectors and responses do not have to be shifted in and out and so the BIST designs significantly improve increase testing speed.
- Moreover, we can use BIST and scan-path techniques together.

Cons of BIST

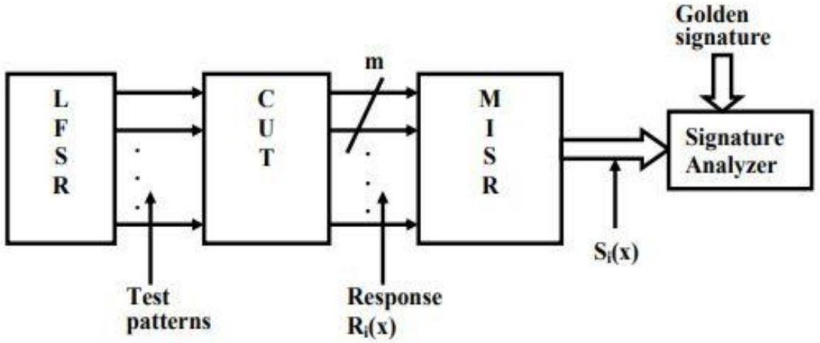
- BIST circuitry wastes space. Implicitly increases the circuitry that must be tested.

- Faults that affect memory, which are stuck-open faults, are hard to detect, may convert a combinational CMOS circuit into a sequential one.

BIST design methodology

In BIST, it’s not possible to store all the outputs of a CUT(circuit under test) on chip. So, signatures are generated by compressing the outputs.

- LFSR generates test patterns for inputs of CUT.
- The signature analyser compares the test signature value with reference signature value from MISR.



BIST in ALU

- In the literature, several BIST designs have been proposed for different ALU architectures assuming different fault models resulting in a trade off with power, area and delay.
- While BIST is considered to be the best and ideal technique in design for testability, one must remember that the BIST additional circuitry consumes a non-negligible on chip space.

Summary

- Presenter: Sai Krishna Charan Dara, 20171140
  - Summary: Looked advantages of GDI technique over others and implemented optimized delay circuit with Ruchitha
- Presenter: Ruchitha Vucha, 20161139
  - Summary: Explored different circuit designs to optimise delay and tested voltage variation of the circuit. Implemented using LTSpice
- Presenter: Surya Poondla, 20161194
  - Summary: Looked into various architectures and techniques to optimize dynamic power. Implemented using LTSpice
- Presenter: Sai Praneeth Chokkarapu, 20161009
  - Summary: Explored on design for testability, looked into different testable designs for ALU, BIST for ALU and analysed them.