

Fault-Tolerant Full Adder using Double-Pass CMOS Transistors in eSim

**A Self-Checking Full Adder in Double Pass Transistor (DPL) Logic
eSim Implementation FOSSEE Research Migration Project Report**

Name: Bhargavi Ghanshyam Hulke

Institute / College: VIT Bhopal University

**FOSSEE Project Title: Simulation of a Fault-Tolerant Full Adder
using Double-Pass CMOS Transistors Using eSim**

Submission Date: 5 September 2026

Based on: A. Ayachi, B. Hamdi, "A Fault-Tolerant Full Adder in Double Pass CMOS Transistor", WASET IJECE Vol:10, No:1, 2016

1. Theory / Description

This project implements and simulates a self-checking, fully differential 1-bit full adder using Double Pass Transistor (DPL) logic, as proposed in the reference paper. The design uses dual-rail (two-rail) encoding for both inputs and outputs, meaning every signal is represented by a true/complement pair (e.g. $a / \bar{a}$, Sum / Sum , $\text{Cout} / \text{Cout}$). Under fault-free operation, each pair is always logically complementary; a mismatch between a signal and its complement indicates a fault, which is what makes the circuit self-checking.

1.1 Double Pass Transistor Logic (DPL)

Unlike standard static CMOS, DPL uses transmission gates (a PMOS and an NMOS transistor in parallel) whose source terminals are driven by input signals rather than fixed supply rails. This reduces transistor count but produces a degraded (weakened) logic swing, which is why restoring CMOS inverters are placed at the output of every stage to bring the signal back to full rail-to-rail swing.

1.2 Circuit structure

The adder is built in three functional blocks:

- Core XOR/XNOR network — generates an internal signal A (and its complement $\bar{A}$) from inputs B and C_{in} , using two complementary pairs of transmission gates.
- Sum stage — two transmission gates (gated by $C_{in}/\bar{C}_{in}$) followed by two restoring inverters, producing the differential outputs Sum and Sum .
- Cout stage — same structure as the Sum stage, producing the differential carry outputs Cout and Cout .

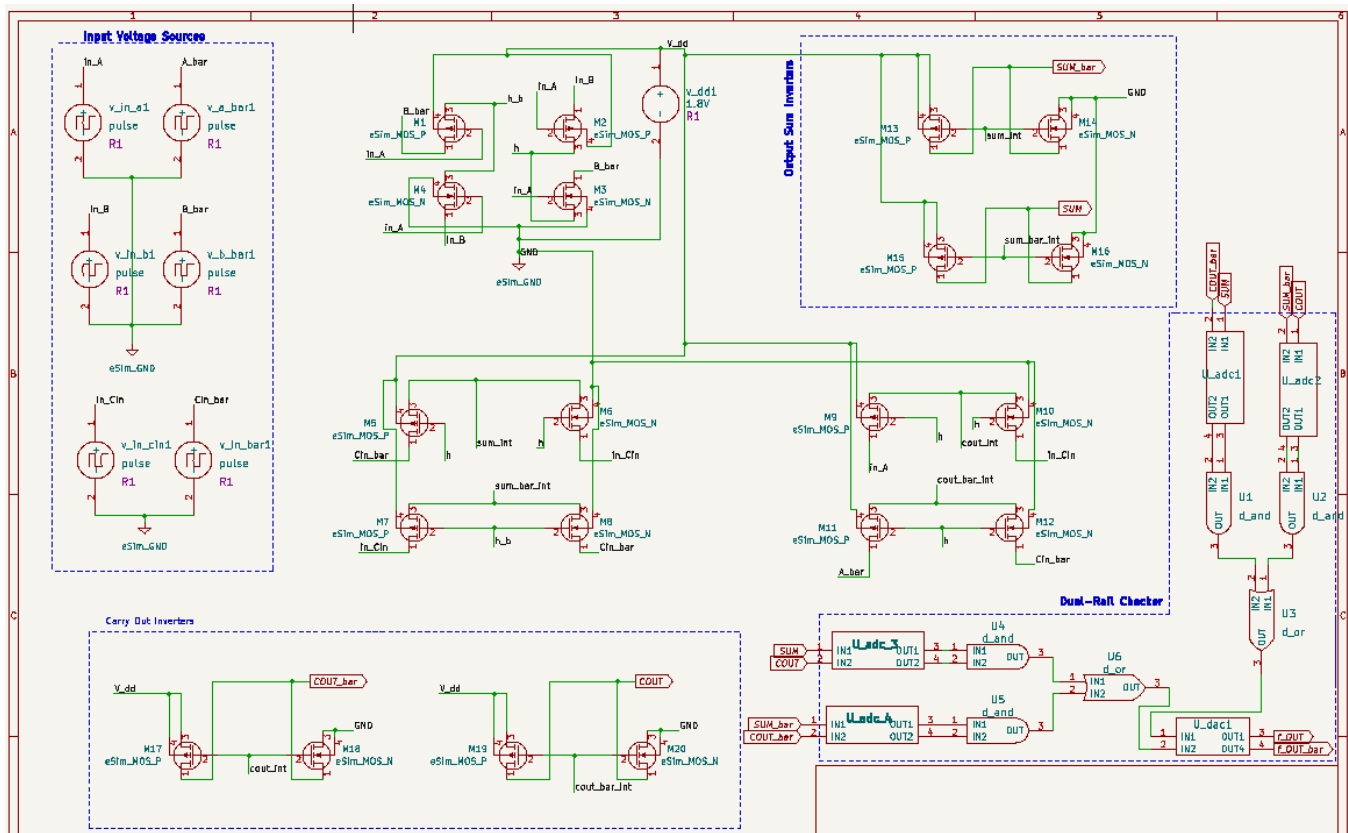
In total the design uses 20 transistors (4 in the core network, 8 in the Sum stage, 8 in the Cout stage), a 28.57% reduction compared to a standard static CMOS full adder (28 transistors).

1.3 Self-checking / dual-rail checker

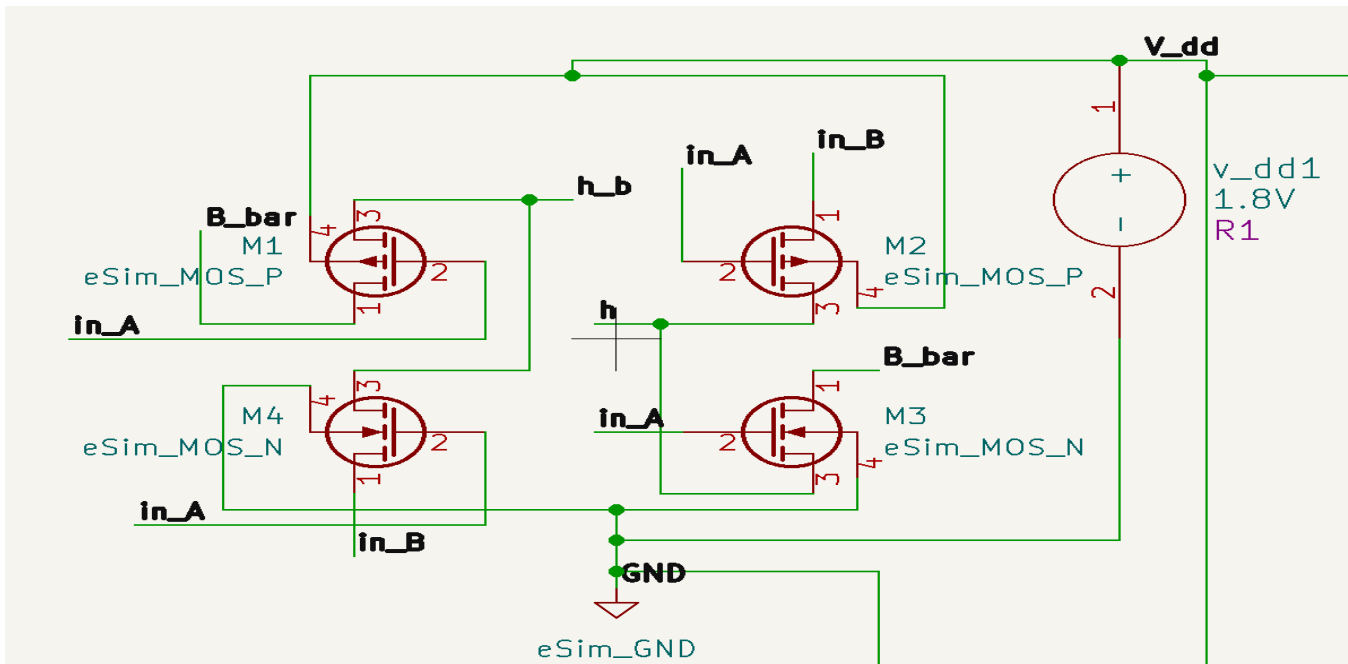
A dual-rail checker circuit compares each output pair (Sum/Sum and Cout/Cout) and produces a pair of checker outputs (f_{out} / f_{out}) that indicate whether the received pair is a valid codeword (complementary) or a non-valid codeword (a fault). This is implemented using AND-OR logic per the dual-rail checker cell structure described in the reference paper.

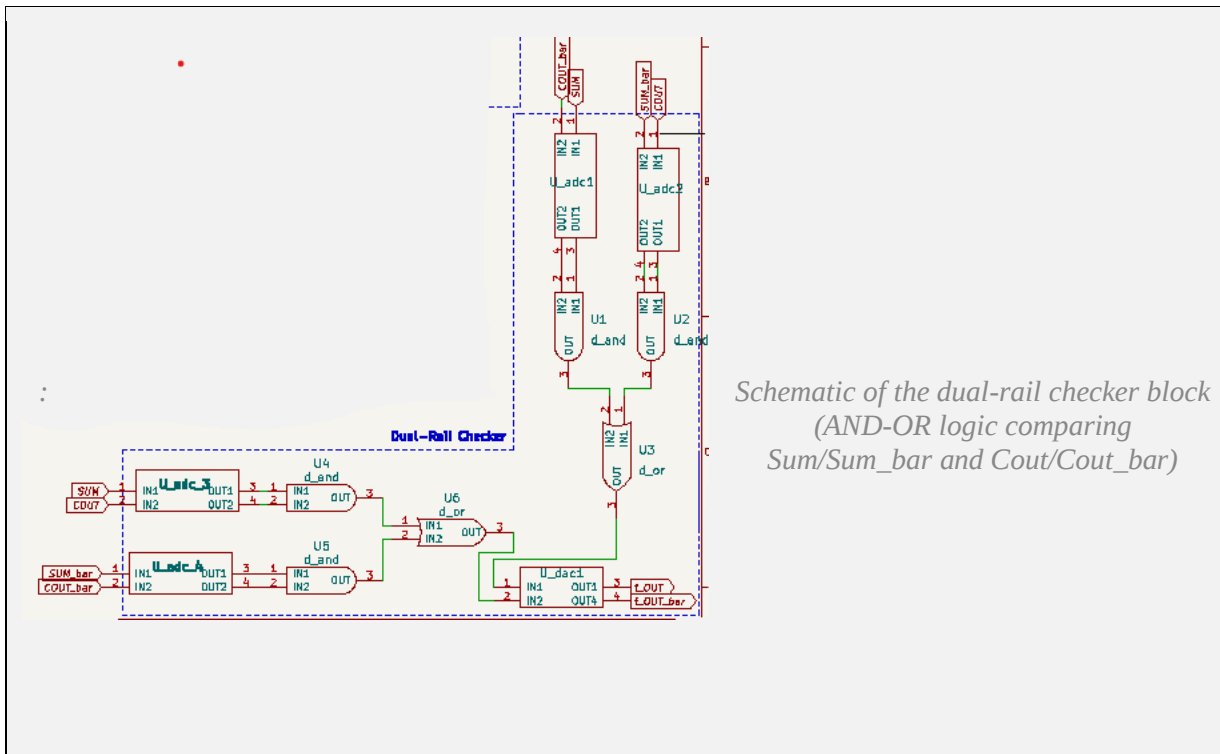
2. Circuit Diagram(s)

2.1 Full schematic (eSim / KiCad capture):



2.2 Core XOR/XNOR network

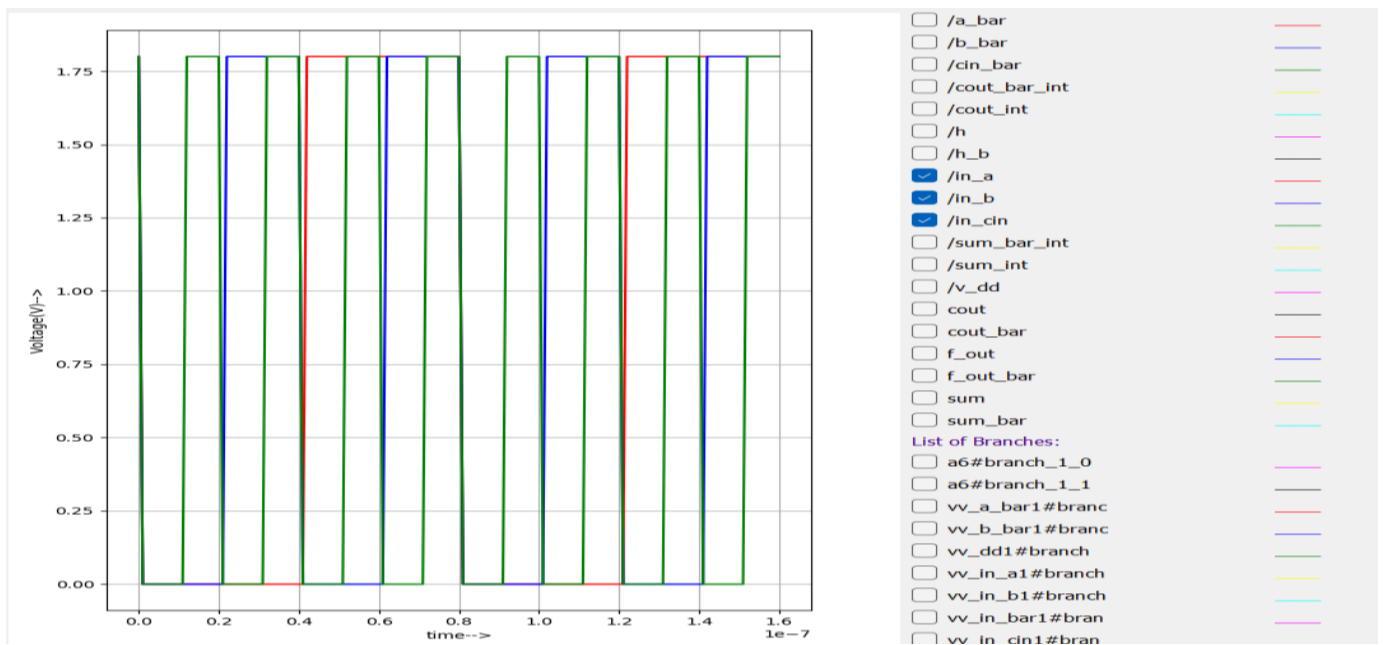




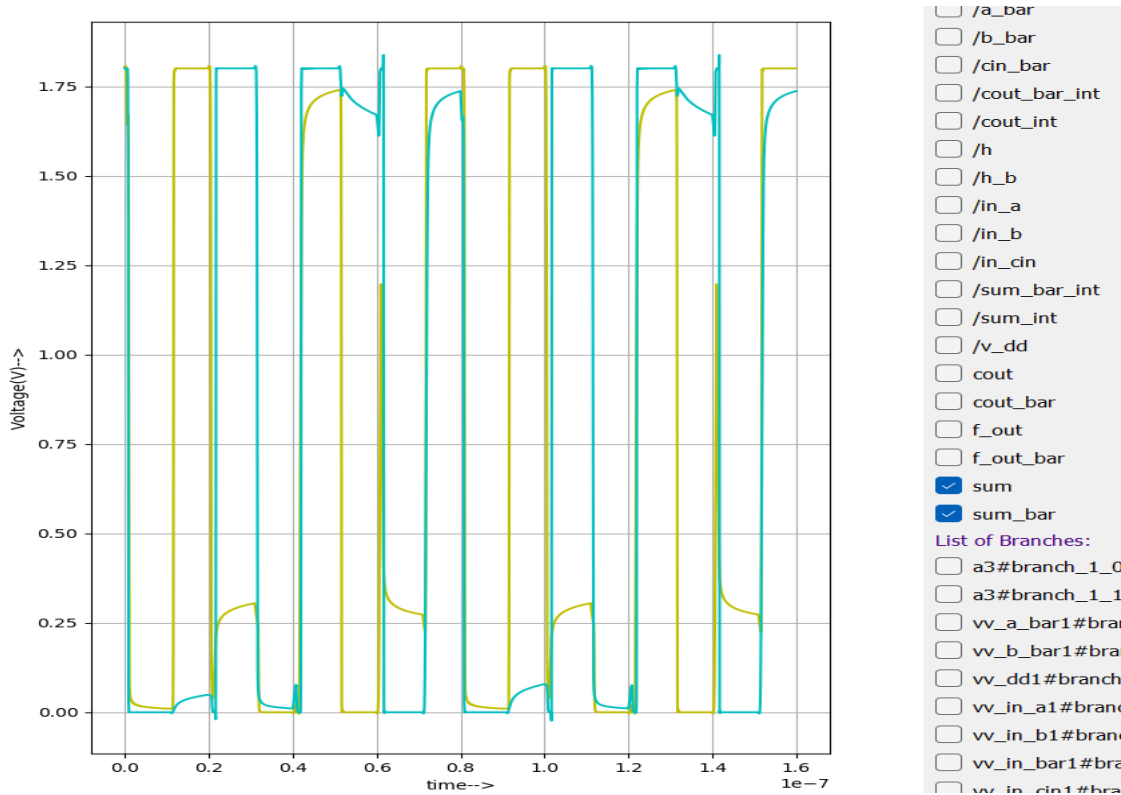
3. Results / Output

3.1 Input stimulus

The primary inputs in_a, in_b, and in_cin were driven with staggered PULSE sources so that all 8 input combinations (000 through 111) are exercised within the simulation window.

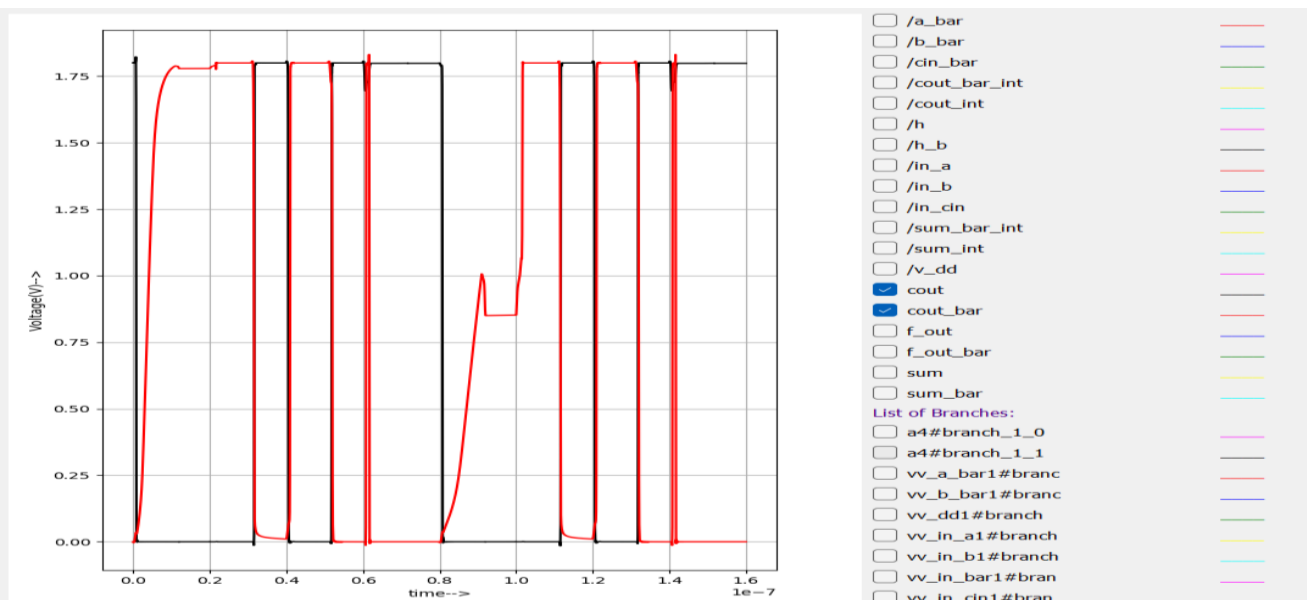


3.2 Sum / Sum_bar (fault-free)



Observation: Sum and Sum_bar are mostly opposite as expected, but at a few input combinations (like A=1,B=1,Cin=0) both sit near 0V at the same time instead of one being high is a real glitch, not just a slow edge. This lines up with the 180nm substitution and residual pass-transistor signal degradation.

3.3 Cout / Cout_bar (fault-free)



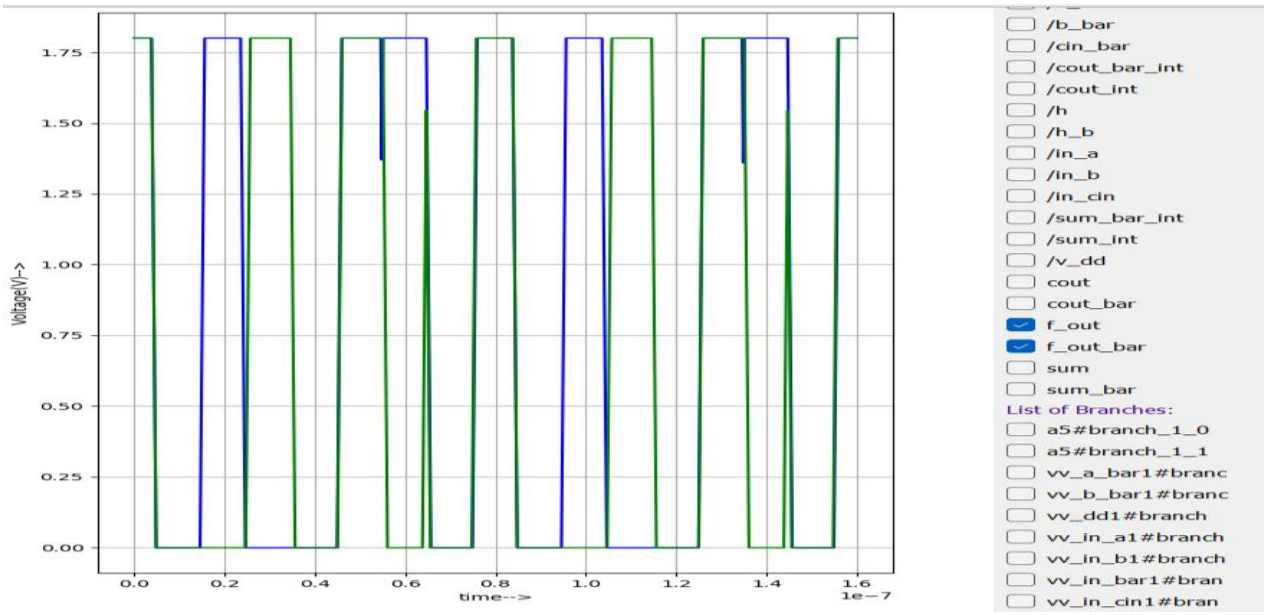
Observation: Cout / Cout_bar stay opposite at every steady point — never equal. A couple of transitions show a brief curved dip/spike instead of a sharp edge, but it always settles back to the correct value shortly after.

3.4 Truth table verification

The following table cross-checks simulated Sum and Cout values against the expected full-adder truth table for all 8 input combinations. The highlighted value is the only anomaly which occurs because of the difference in the technology of CMOS transistors used. Instead of a 32nm technology, 180nm technology was used in this simulation due to unavailability of in the eSim component library.

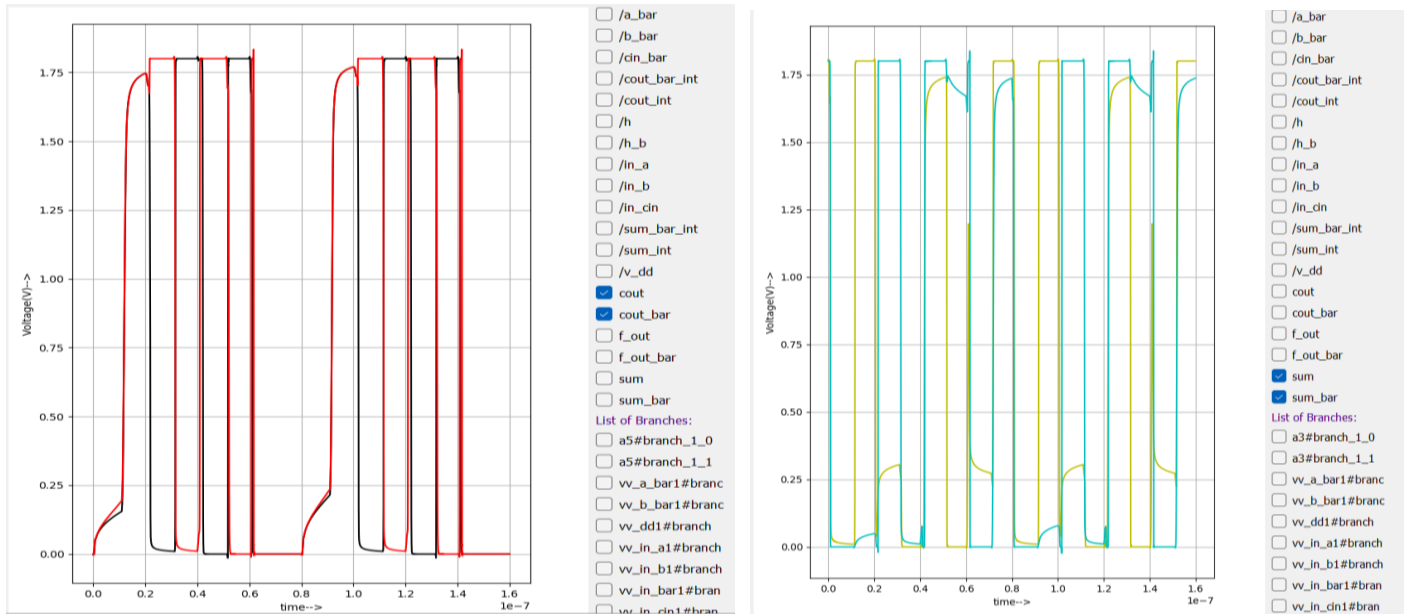
A	B	Cin	Expected Sum / Cout	Simulated Sum / Cout
0	0	0	0 / 0	0 / 0
0	0	1	1 / 0	1 / 0
0	1	0	1 / 0	0 / 0
0	1	1	0 / 1	0 / 1
1	0	0	1 / 0	1 / 0
1	0	1	0 / 1	0 / 1
1	1	0	0 / 1	0 / 1
1	1	1	1 / 1	1 / 1

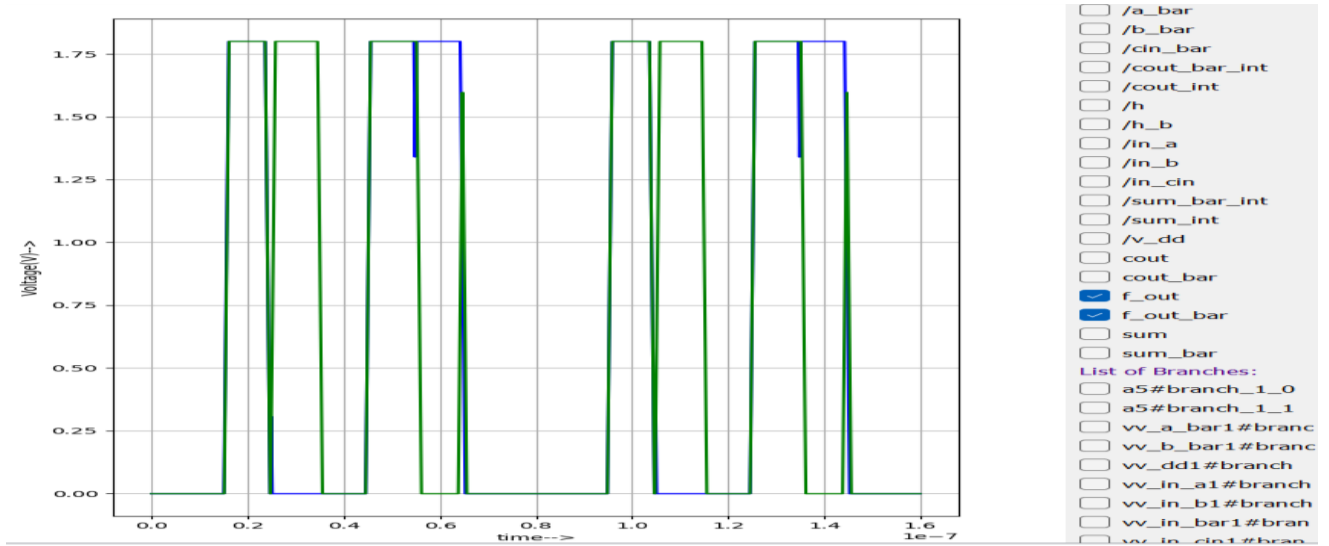
3.5 Dual-rail checker output



3.6 Fault injection

To verify the self-checking property, a fault was manually injected at a primary input node ($a = \bar{a}$) during a defined time window, following the same methodology as the reference paper's Fig. 4.





Observation: f_{out} and f_{out_bar} correctly stay opposite for most input combinations, flagging valid codewords but in the same spots where Cout misbehaves, the checker output also looks off, since it's just reporting what it's fed. During the fault window, Cout and Cout_bar briefly read the same value (both low) instead of opposite — a clear invalid codeword. Sum/Sum_bar stay unaffected the whole time. The checker (f_{out}/f_{out_bar}) responds right at the fault window, correctly signaling the error the moment it happens.

3.7 Observations for fault injected Sum and Cout

- Sum/Sum_bar remain unaffected by the injected fault $in_A = A_bar$. This is consistent with the circuit topology: the Sum stage is derived entirely from h/h_bar (generated from A/A_bar and in_b/B_bar in the core network) and Cin/Cin_bar the primary input a (and its complement) has no direct electrical path into the Sum computation. Since the fault is injected only at in_A/A_bar , and Sum has no dependency on this signal, Sum correctly stays clean throughout the fault window.
- Cout/Cout_bar are visibly disturbed by the same fault. This is because the Cout stage's pass-transistor network explicitly uses in_A and A_bar as direct data inputs ($cout_int_bar = A_bar ? Cin_bar : A_bar$, it uses A_bar). Corrupting in_A/A_bar therefore directly corrupts the value being passed through to Cout, which is why the complementary-pairing breakdown appears specifically on the carry output and not on the sum output.
- Taken together, these two observations show that fault propagation in this design is topology-dependent, not global: an injected fault only affects the outputs that have a direct data-path dependency on the corrupted node. This matches the expected behaviour of pass-transistor/DPL circuits, where each output is driven by a distinct subset of the primary inputs rather than a shared global network.
- This selectivity is itself a useful diagnostic property: in a larger self-checking system, the fact that only Cout (and not Sum) flags an error for an in_A/A_bar fault could, in principle, help localize which primary input was corrupted, purely from which dual-rail checker output raised the flag.

3.7 Technology node note

⚠ This implementation uses eSim's generic 180nm CMOS device models rather than the 32nm double-pass-transistor technology used in the reference paper, since a 32nm PDK was not available in the eSim component library. This substitution is expected to affect absolute switching speed and current magnitudes, and may account for degraded (non-ideal) logic levels and settling behaviour observed in some input transitions in the plots above. The circuit topology that is the transistor count, transmission-gate structure, and dual-rail encoding remains true to the design proposed in the paper; only the underlying device technology differs. Absolute timing/current values in this report should therefore not be directly compared to the paper's Fig. 3/4 numeric values. Only the logical/functional behaviour (truth table correctness and dual-rail complementarity) is being verified against the paper.

4. References

Reference image from the research paper given below:

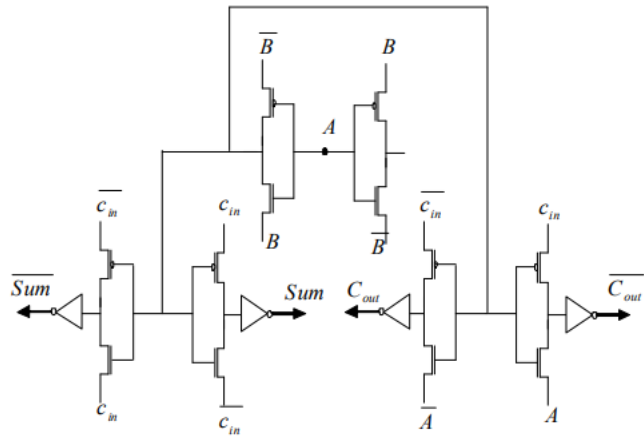


Fig. 1 The proposed self-checking full adder

[1] A. Ayachi, B. Hamdi, "A Fault-Tolerant Full Adder in Double Pass CMOS Transistor," World Academy of Science, Engineering and Technology, International Journal of Electronics and Communication Engineering, Vol:10, No:1, 2016.

[2] eSim documentation, FOSSEE, IIT Bombay — <https://esim.fossee.in>