Circuit Simulation Project

https://esim.fossee.in/circuit-simulation-project

6-Bit Unsigned Multiplier Circuit With Dadda Tree Reduction And Brent-Kung Adder

By

Reuel Reuben

Under the Guidance of Prof. A. Prabhakar and Prof. Swapnil S. Thorat

Theory/Description :

Adders and Multipliers are integral part of any modern day SOC/Processor. They are usually a part of the arithmetic unit, or ALU. The ALU can be found at the core of every digital computer and are one of the most important parts of a CPU.

Since we need more and more performance for up-to-date applications there is a need to increase the speed as well as the performance and reduce the delay of these Adders and Multipliers. This kind of performance boost and delay reduction can be seen in the Parallel Multipliers such as the Dadda Tree Multiplier and the Parallel Prefix Adders such as the Brent Kung Adder.

Dadda Tree Reduction Multiplier:

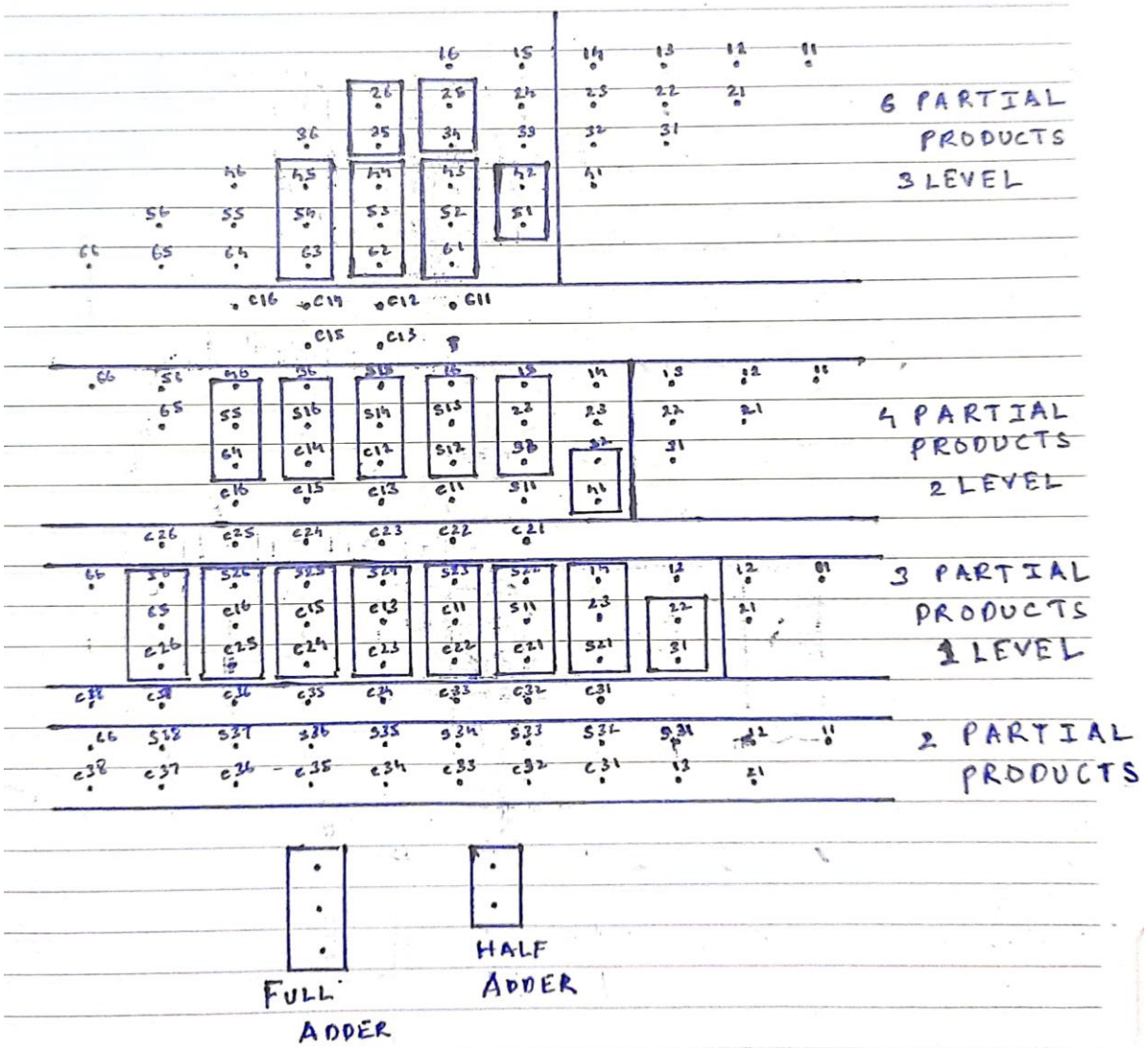
This Multiplier is very similar to the Wallace tree Multiplier but this multiplier has some key advantages over the Wallace tree Multiplier such as using lesser number of gates and decreasing the overall delay of the multiplier by the Dadda reduction technique.

The Dadda reduction technique is based on the Wallace Tree Reduction table which gives us the information of number of levels present in the Wallace reduction tree based on the number of partial products.

NUMBER OF LEVELS IN WALLACE TREE

Number of Partial Products,(k)	Number of Levels in the Wallace Tree
3	1
4	2
$5 \leq k \leq 6$	3
$7 \leq k \leq 9$	4
$10 \leq k \leq 13$	5
$14 \leq k \leq 19$	6
$20 \leq k \leq 28$	7
$29 \leq k \leq 42$	8
$43 \leq k \leq 63$	9

From the table we know the number of levels we have to reduce in order to reach just 2 partial products. Since we are designing a 6x6 Dadda Tree Multiplier we can conclude that we need to reduce 3 levels in order to reach 2 partial products.

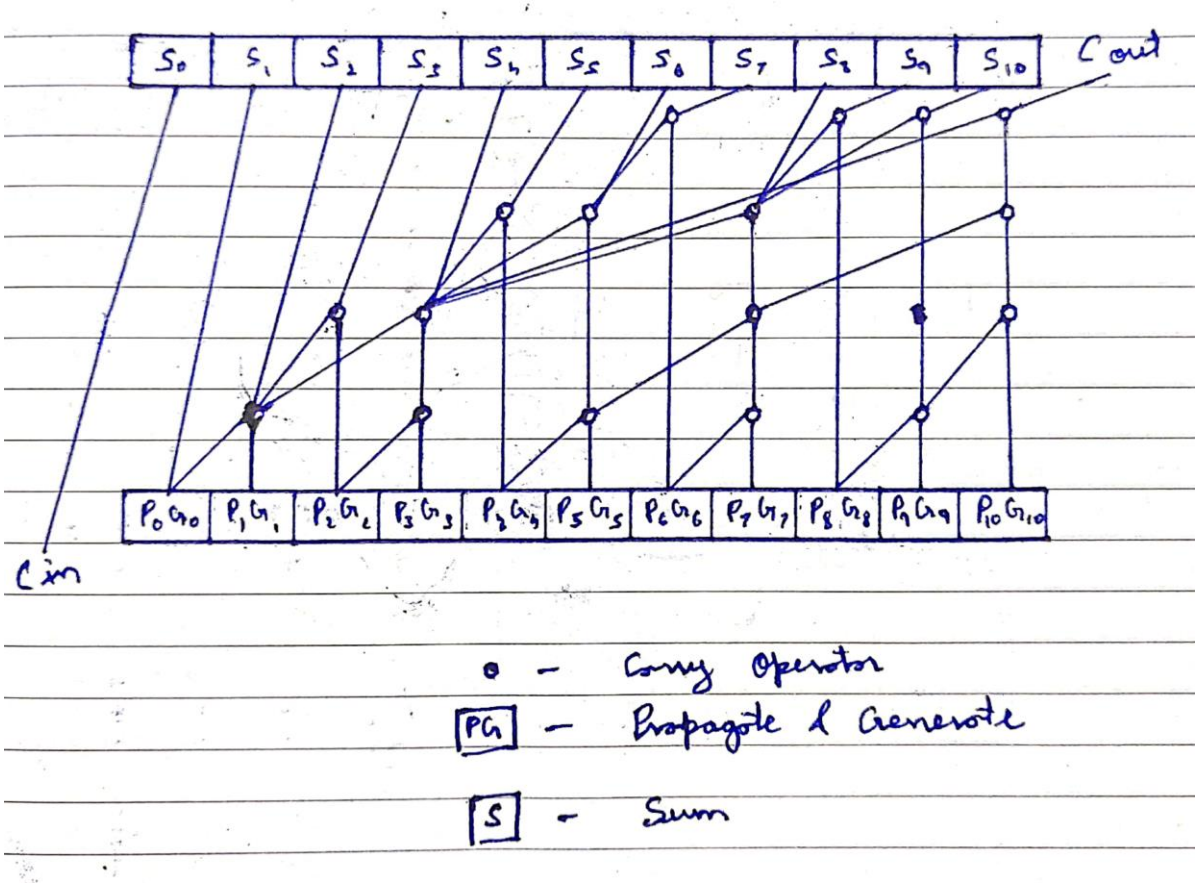


As seen above we can reduce the 6 partial products to 2 partial products. Once we reach 2 partial products, we can easily add both the partial product to get our multipliers result. In this project I have used a Brent Kung Adder to add the two partial products in order to get our result.

Brent Kung Adder:

Brent Kung adder is a parallel prefix adder. Some of the advantages of the Brent Kung adder is that it introduces higher consistency to the adder structure, it has lesser wiring cramming and lesser chip area to implement compared to the Kogge Stone adder.

Since the output of the Dadda Tree Multiplier is 2 11-bit partial products we need to make a 11-bit Brent Kung Adder.



For the brent kung we use the Generate and Propagate logic to make the parallel prefix adder

First we create the generate and propagate for all the inputs using formula

G(i) = Xi * Yi

P(i) = Xi ⊕ Yi

C(i+1) = G(i) + P(i) * C(i)

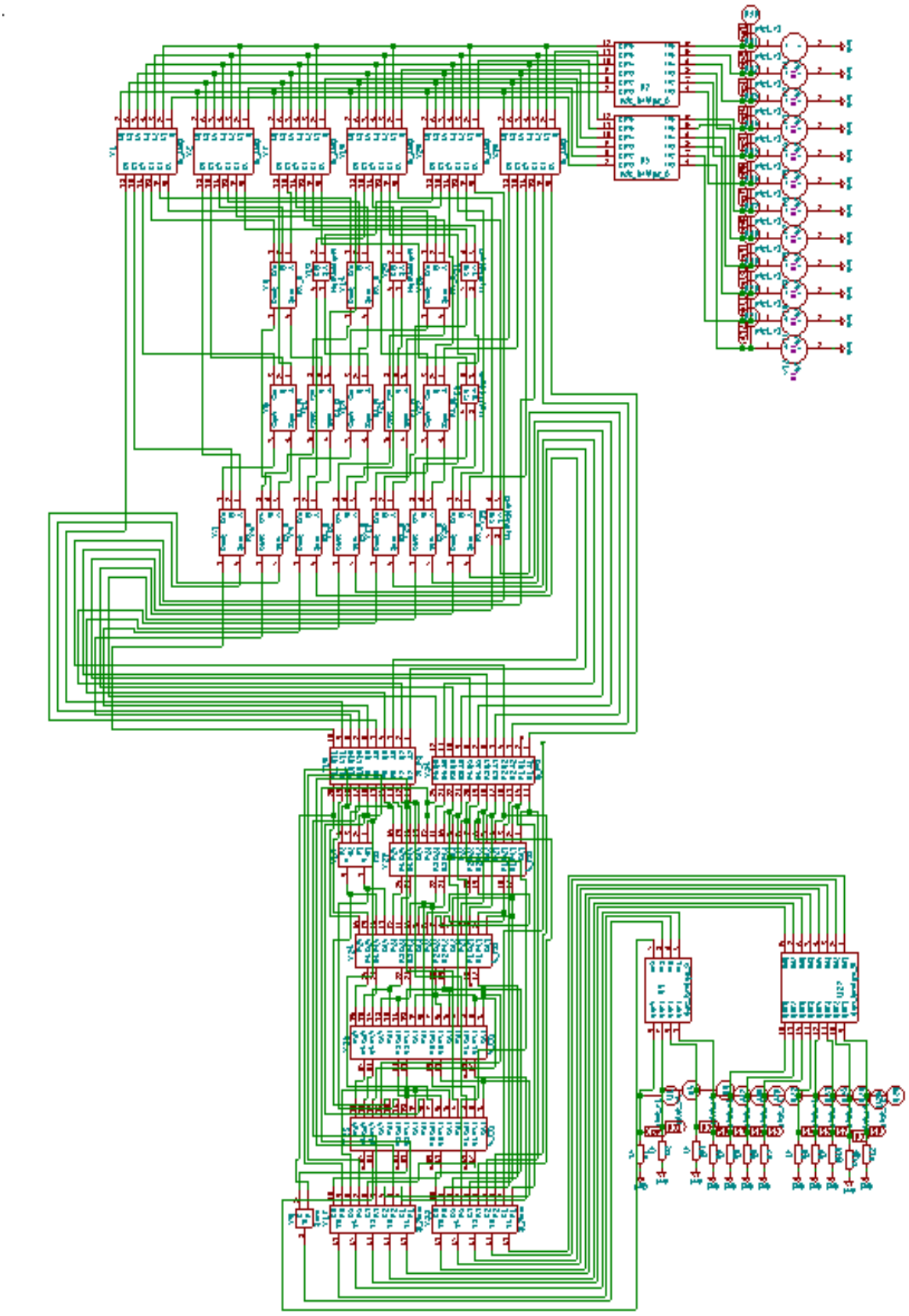
Then using the carry operator

(G'', P'') ¢ (G', P') = (G''+G'·P'', P'·P'')

Once we get all the carry's we can get the sum by the formula

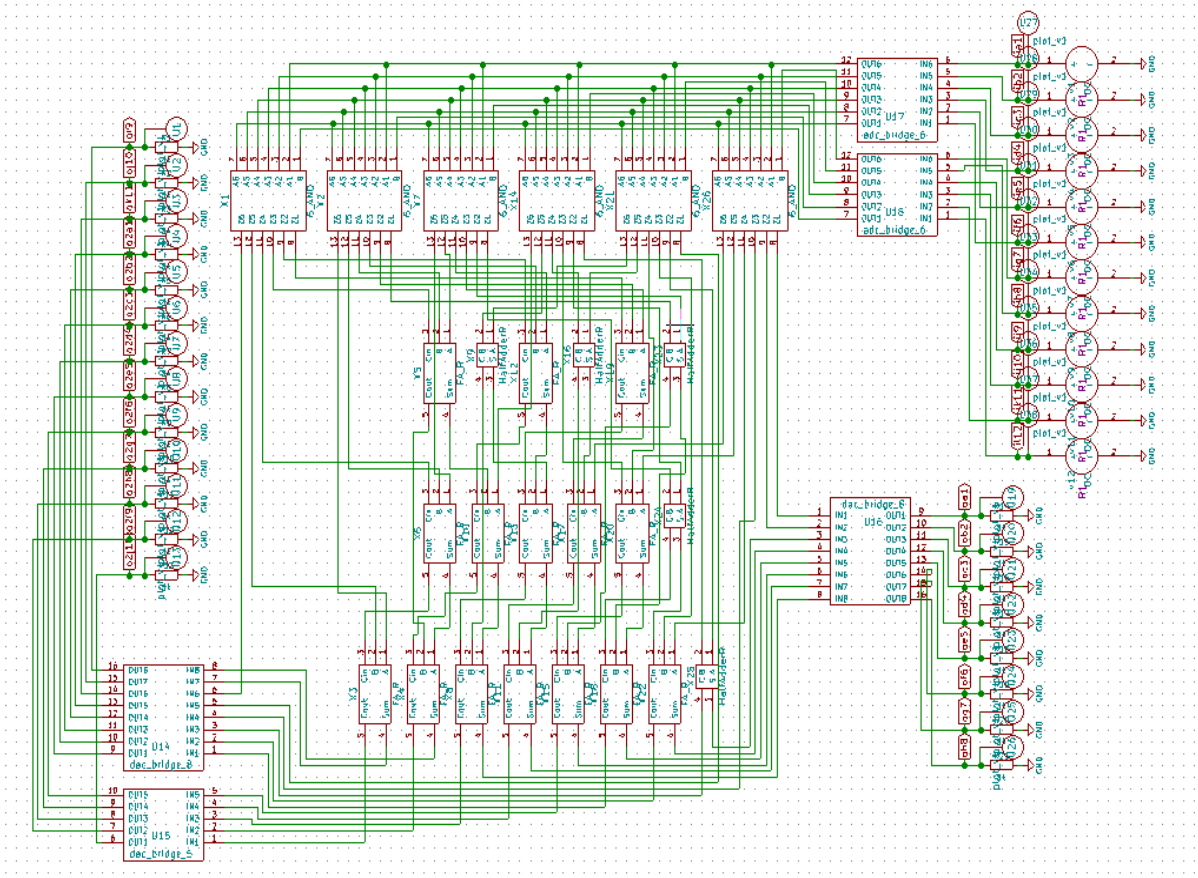
S(i) = P(i) ⊕ C(i)

Circuit Diagram(s) :

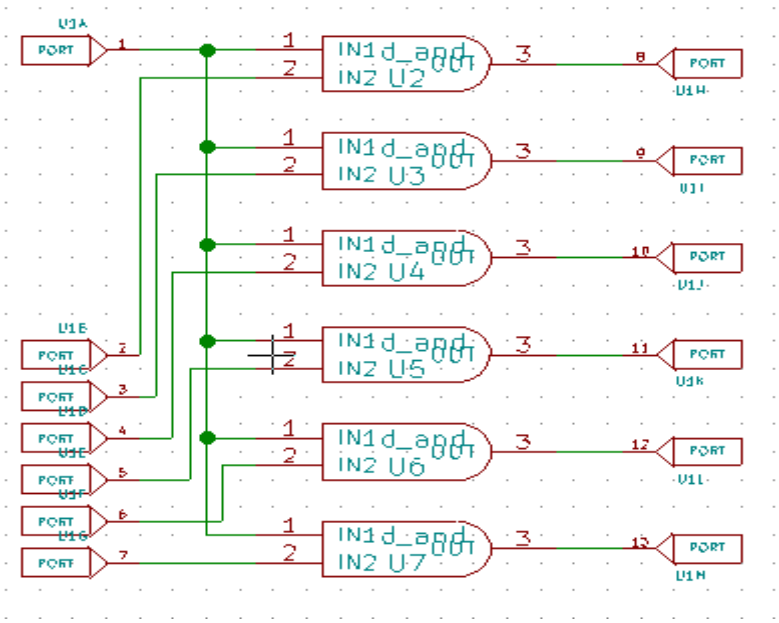


6-Bit Unsigned Multiplier Circuit With Dadda Tree Reduction And Brent-Kung Adder

Dadda Tree Reduction Multiplier and It's Sub-circuit's:

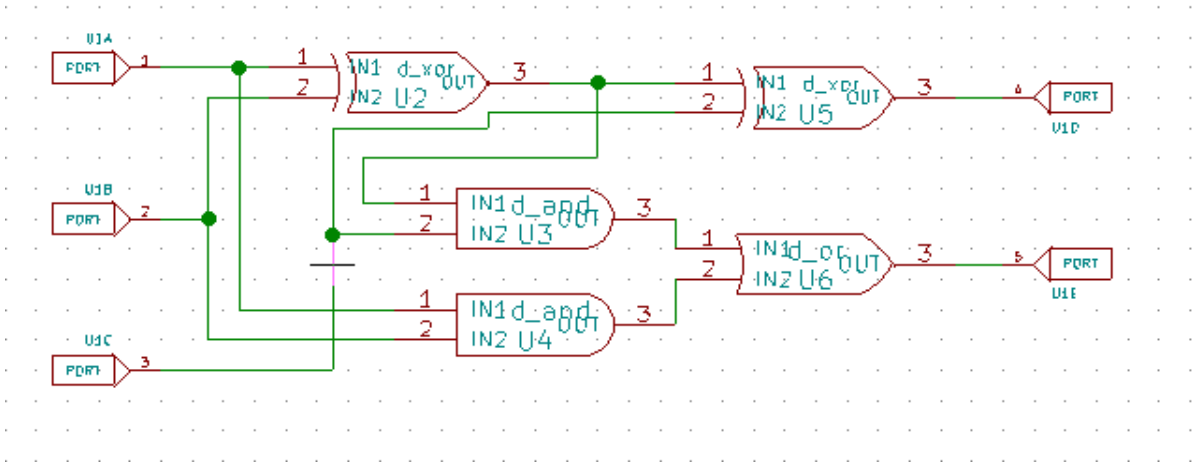


6x6 Dadda Tree Reduction



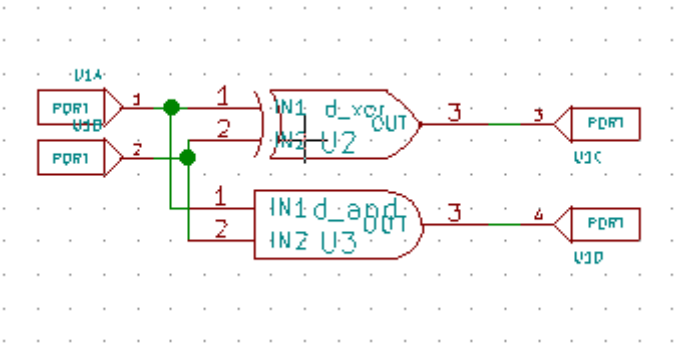
6 And Gates for Generating the Partial Products

6_and (AndArrayTest)



Full Adder

fa-r (Full_Adder_R)

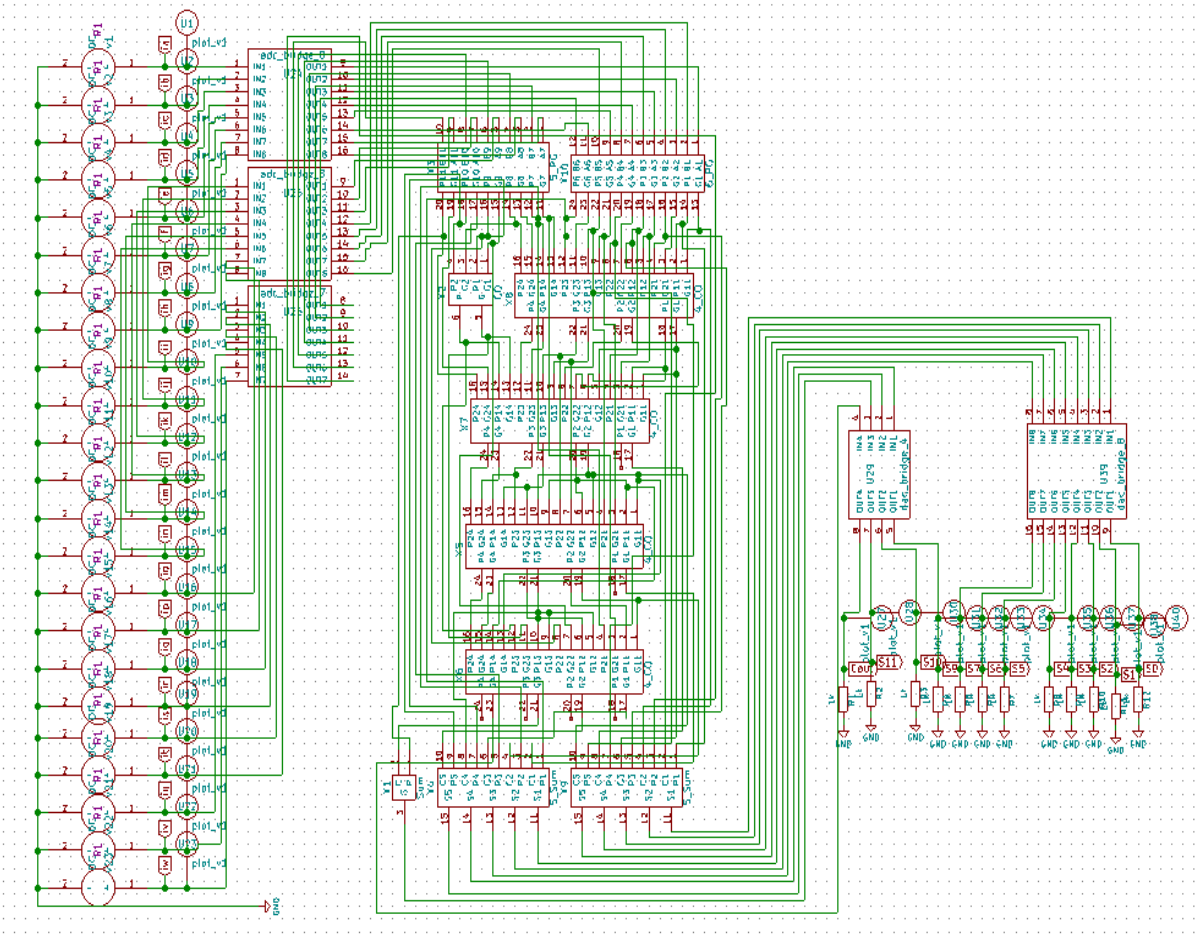


Half Adder

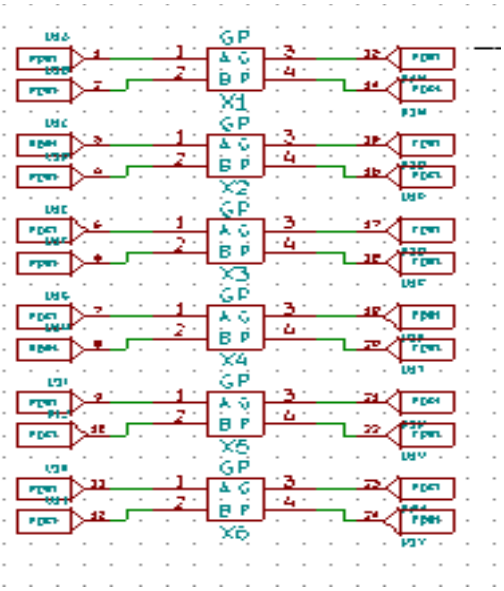
halfadderr (HalfAdderR)

***Note:** **Sub-Circuit Abbreviation (Sub-Circuit File Name in the Project Files)**

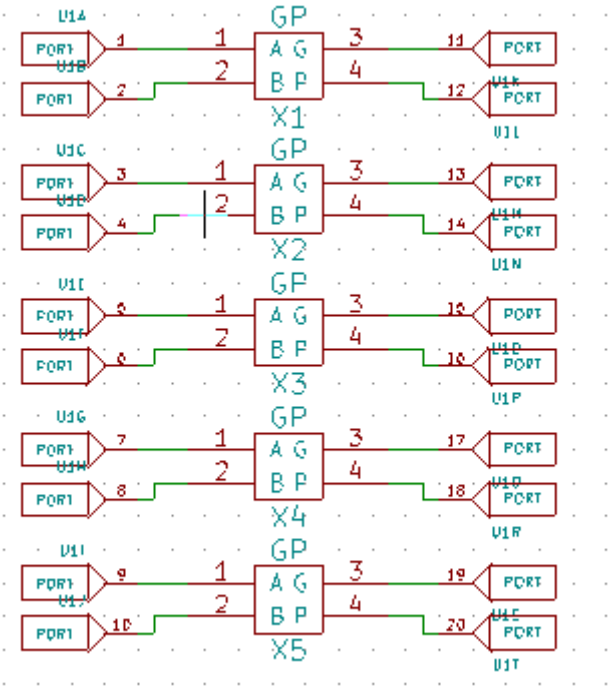
Brent Kung Adder and It's Sub-circuit's:



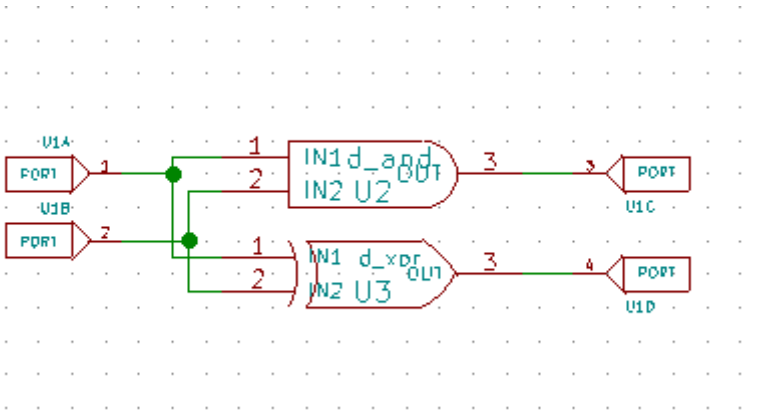
11-bit Brent Kung Adder



6 Propagate and Generate Blocks
6_PG (Gen&PropArray)

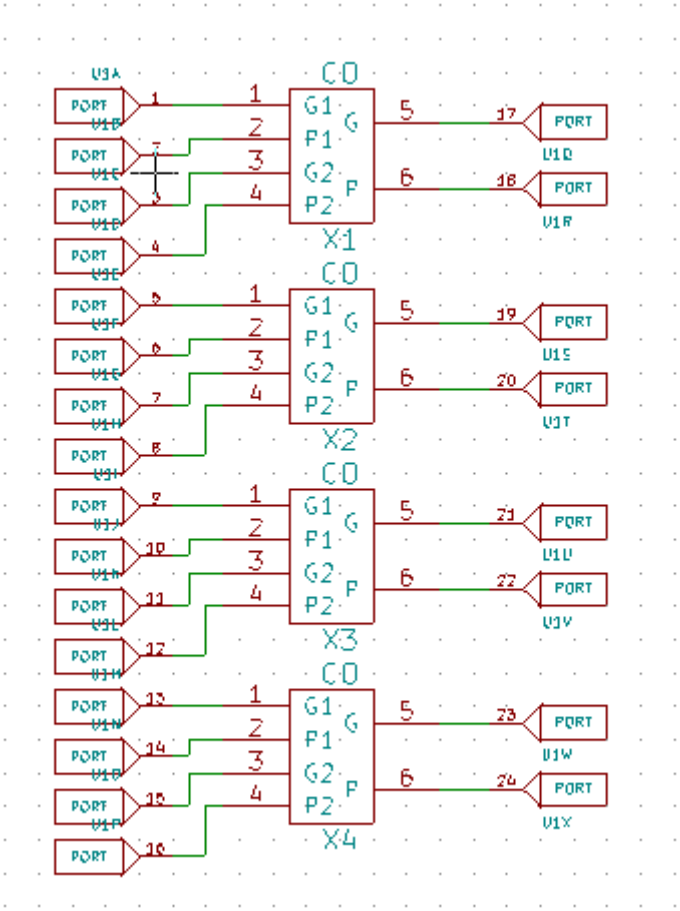


5 Propagate and Generate Blocks
5_PG (Gen&PropArray)

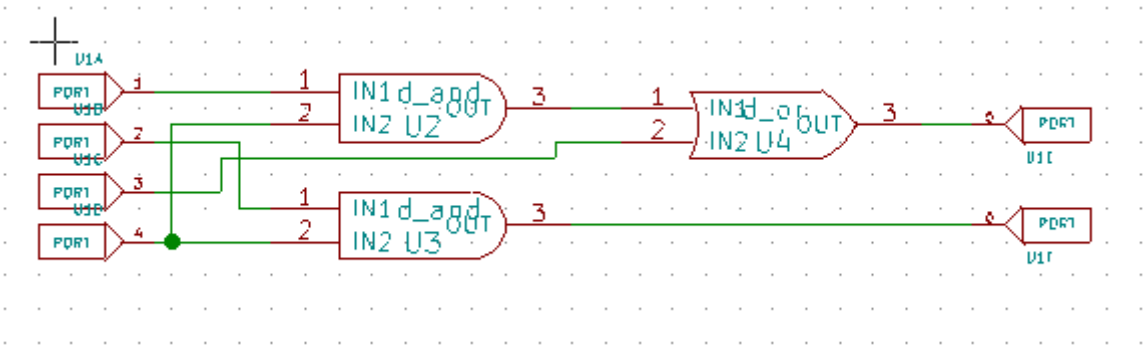


Propagate and Generate Block
GP (Gen&Prop)

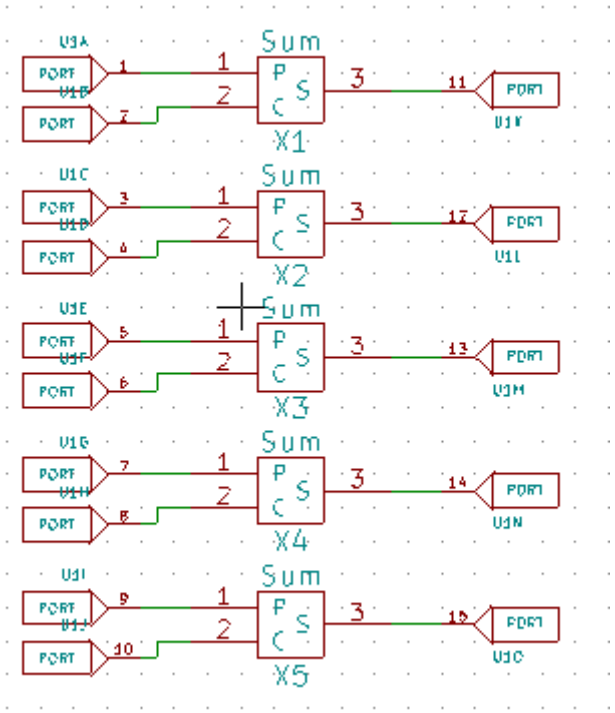
***Note:** **Sub-Circuit Abbreviation (Sub-Circuit File Name in the Project Files)**



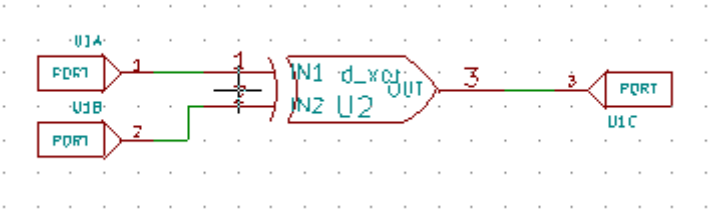
4 Carry Operator Blocks
4_CO (CarryOperatorArray)



Carry Operator Block
CO (CarryOperatorR)



5 Sum Blocks
5_SUM (SumArray)



Sum Block
SUM (SumR)

***Note:** **Sub-Circuit Abbreviation (Sub-Circuit File Name in the Project Files)**

Results (Input, Output waveforms and/or Multimeter readings) :

We will be multiplying to 2 6-bit binary numbers to verify our simulation

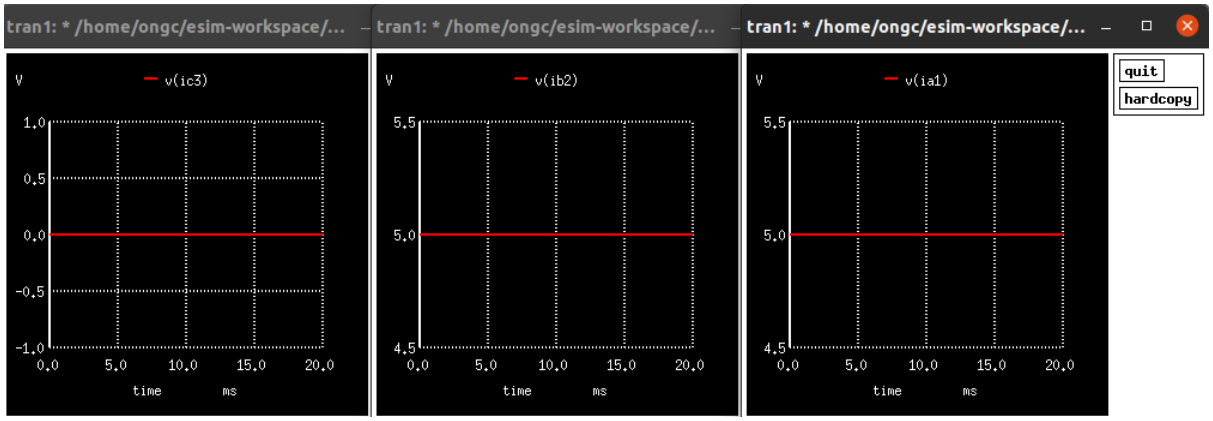
A = 110011

B = 111110

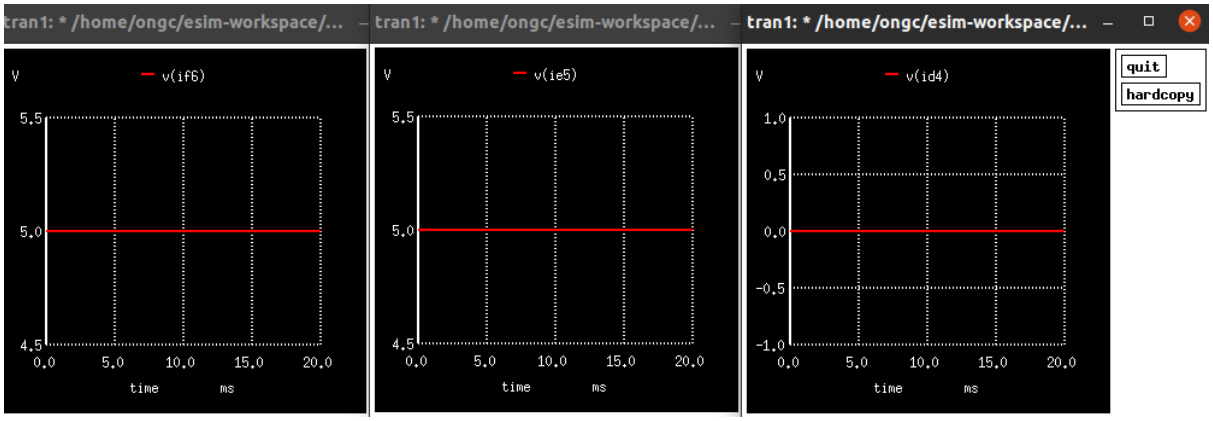
A X B = 110011 X 111110 = 110001011010

Ngspice:

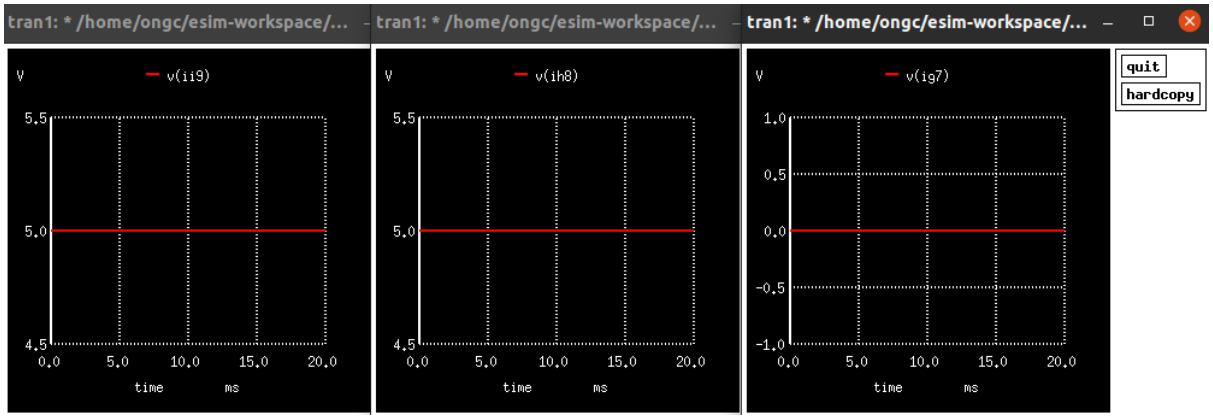
Input:



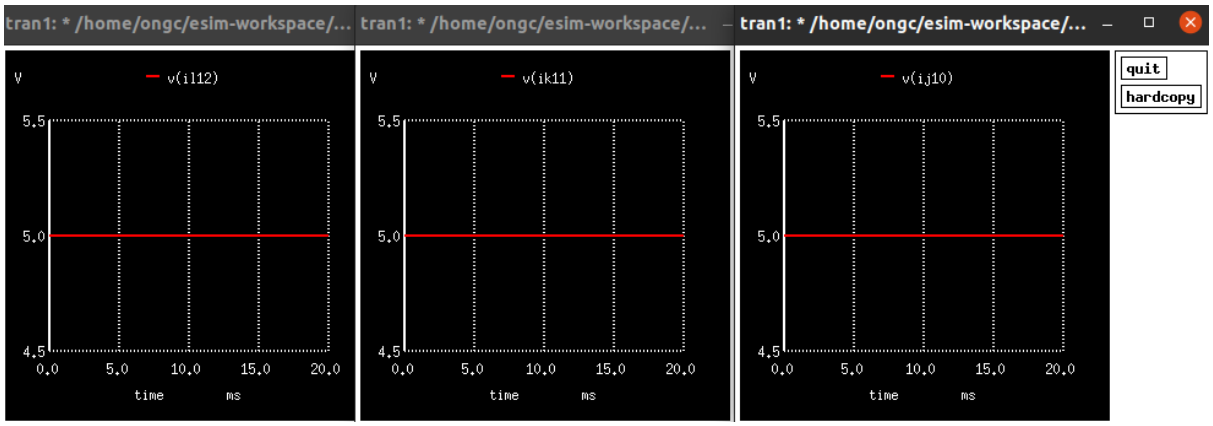
A3 **A2** **A1**



A6 **A5** **A4**

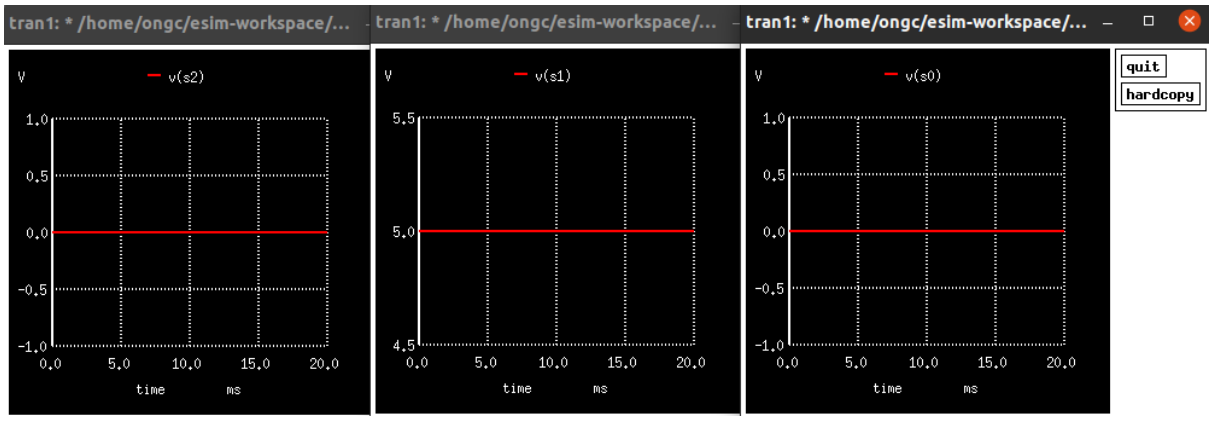


B3 **B2** **B1**

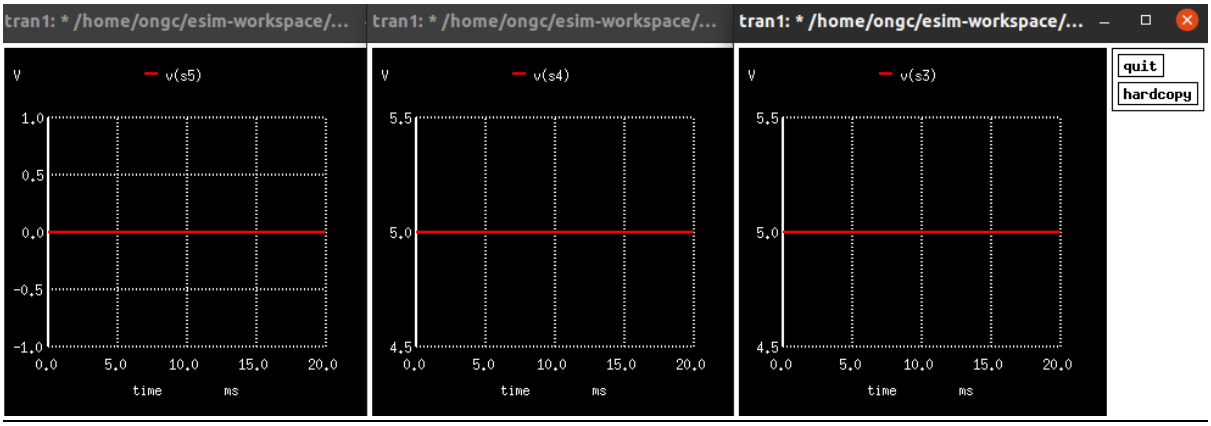


B6 **B5** **B4**

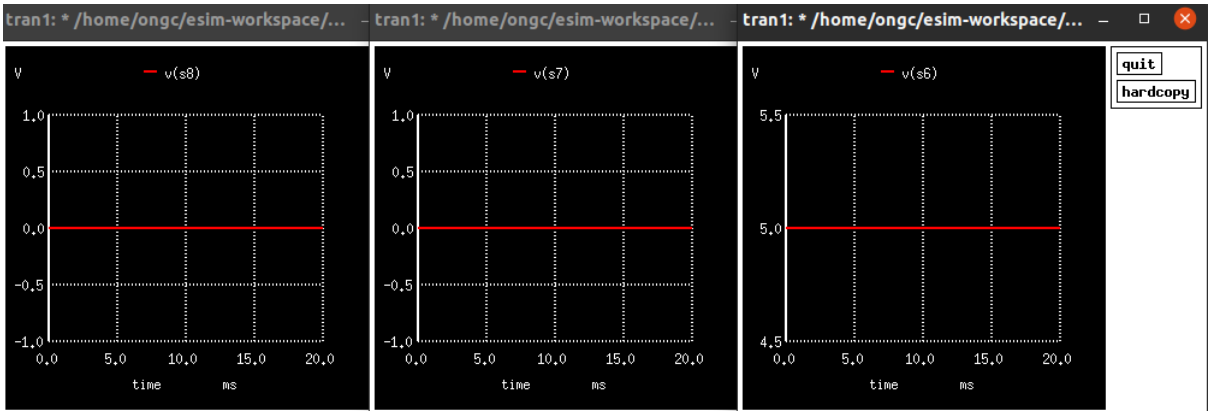
Output:



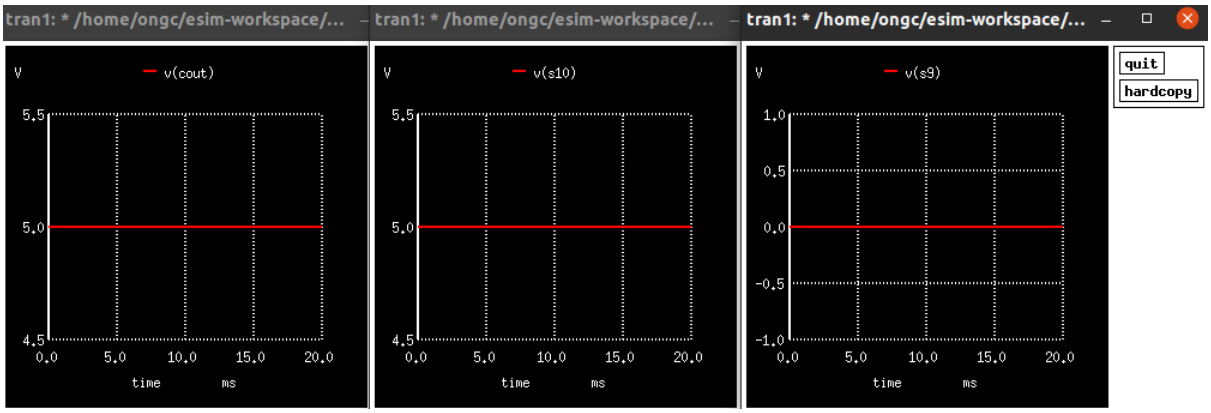
S2 **S1** **S0**



S5 S4 S3



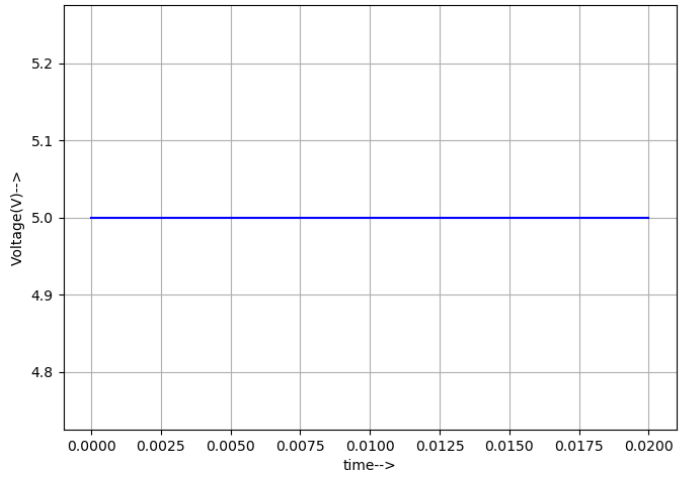
S8 S7 S6



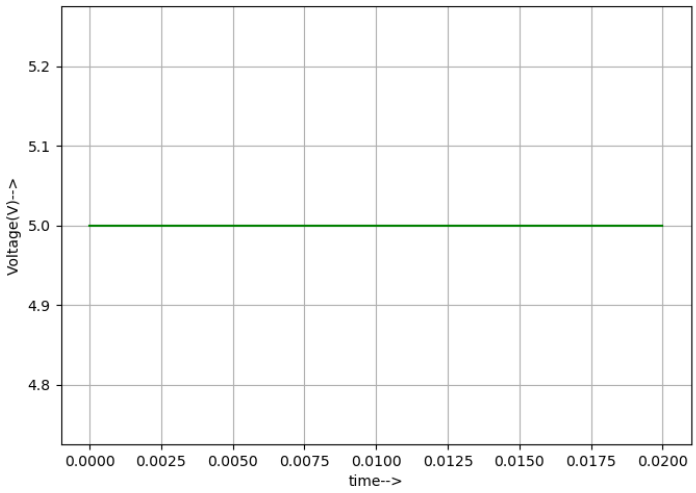
Cout S10 S9

Python Plots:

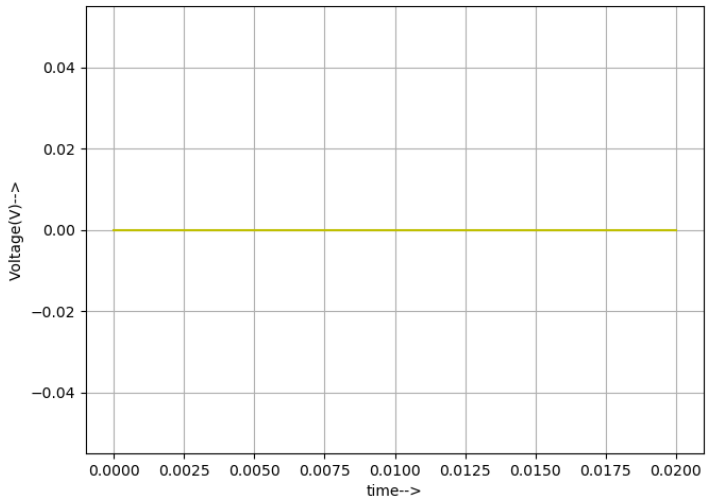
Input:



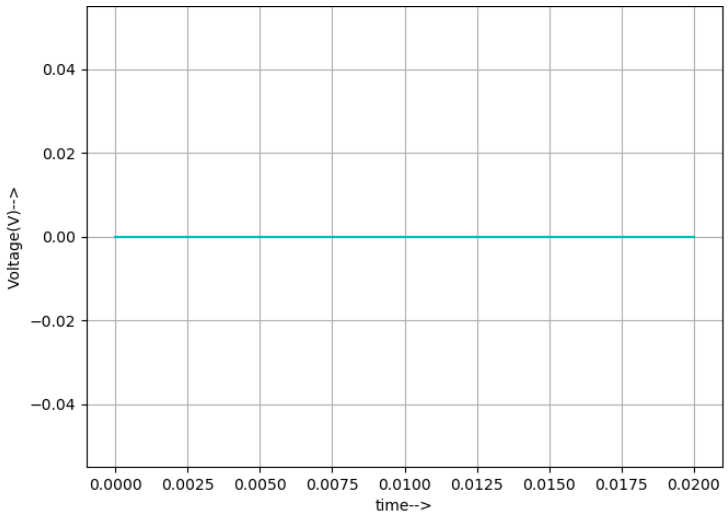
A1



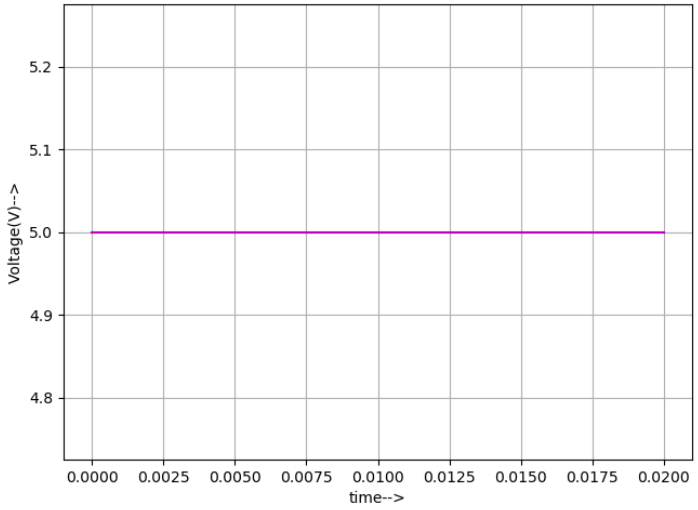
A2



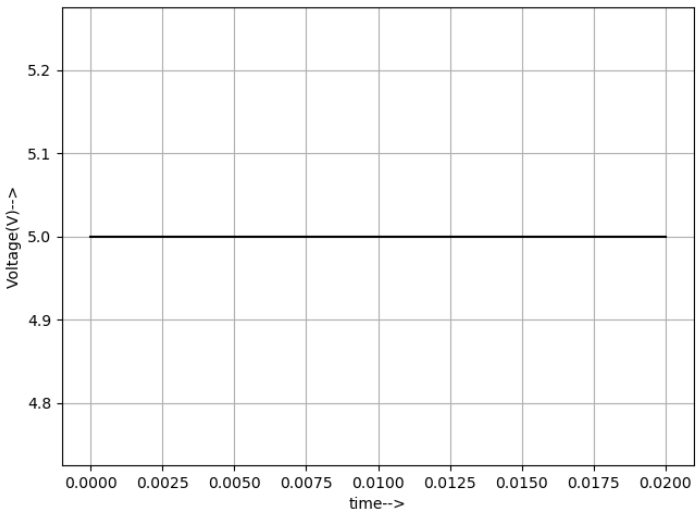
A3


0.04
0.02
0.00
−0.02
−0.04
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

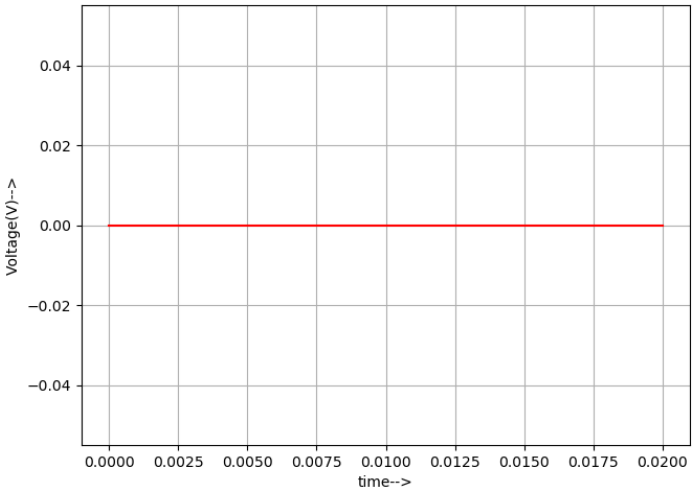
A4


5.2
5.1
5.0
4.9
4.8
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

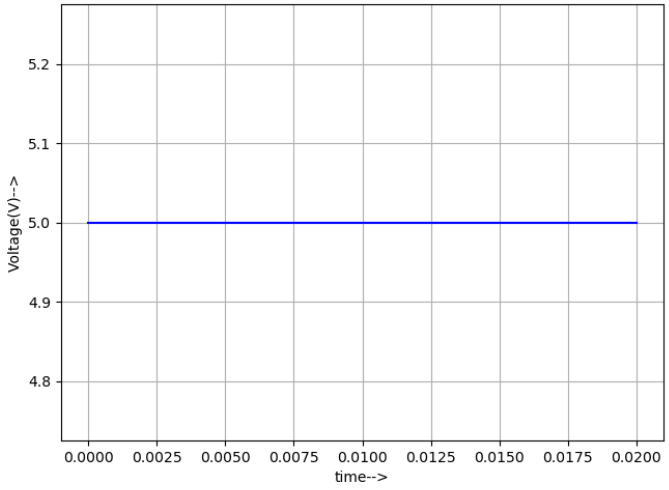
A5


5.2
5.1
5.0
4.9
4.8
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

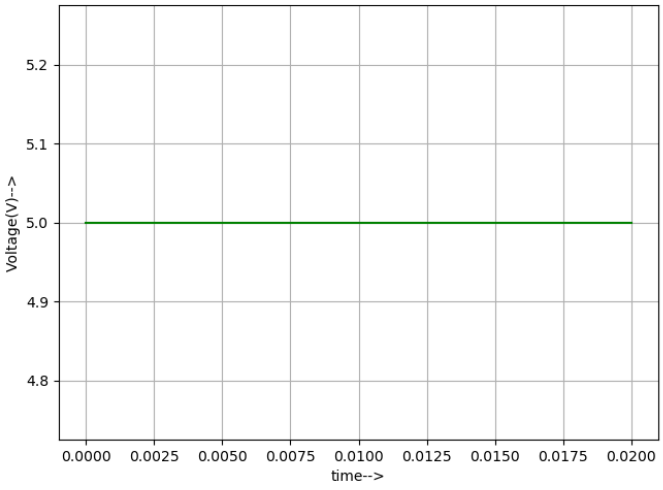
A6


0.04
0.02
0.00
−0.02
−0.04
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

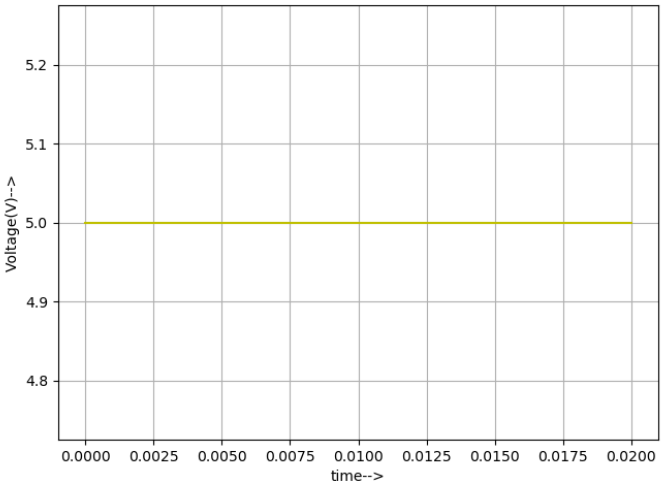
B1


5.2
5.1
5.0
4.9
4.8
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

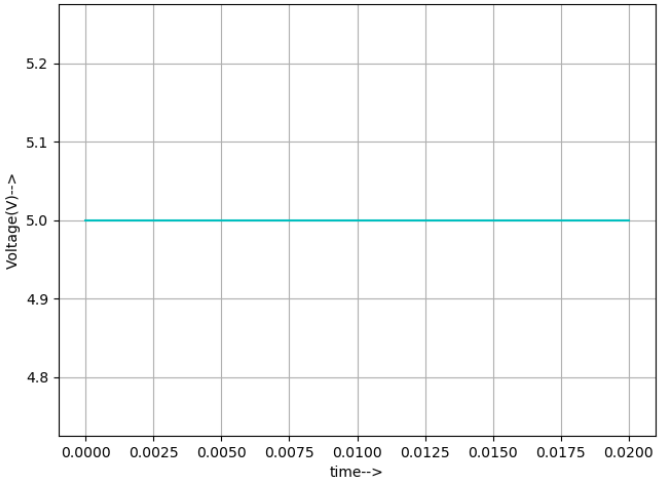
B2


5.2
5.1
5.0
4.9
4.8
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

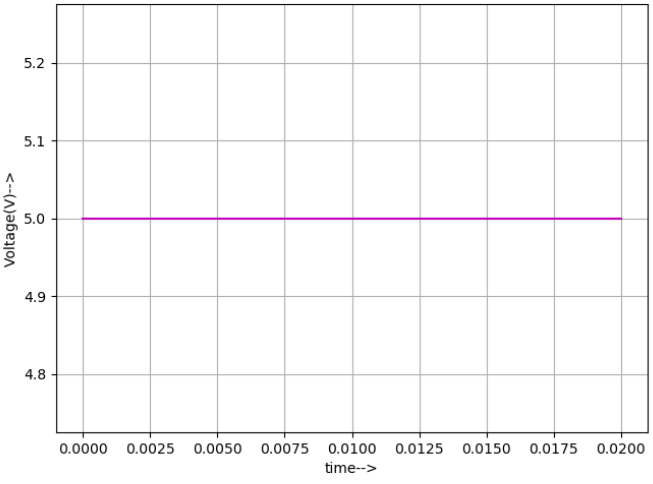
B3


5.2
5.1
5.0
4.9
4.8
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

B4

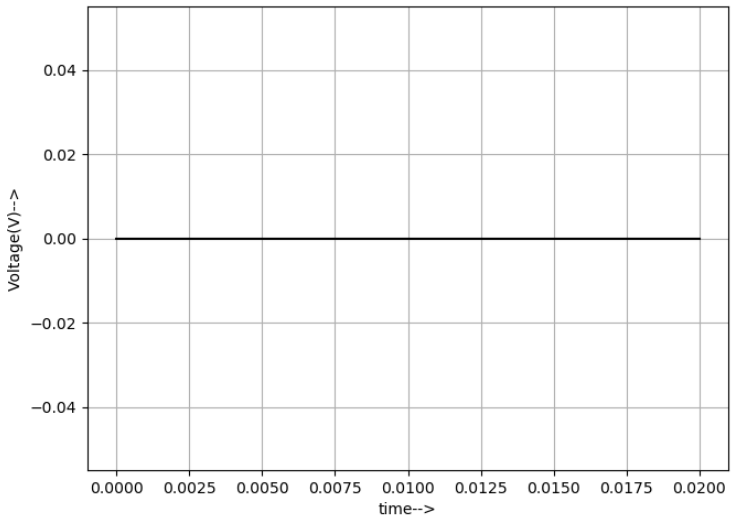

5.2
5.1
5.0
4.9
4.8
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

B5

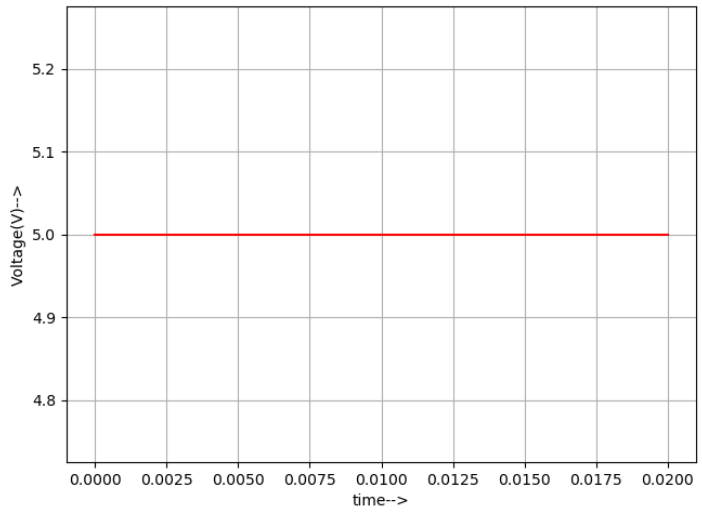

5.2
5.1
5.0
4.9
4.8
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

B6

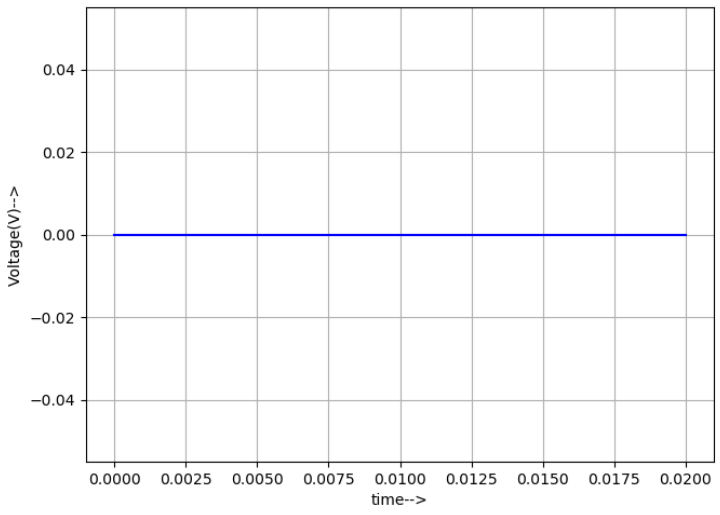
Output:



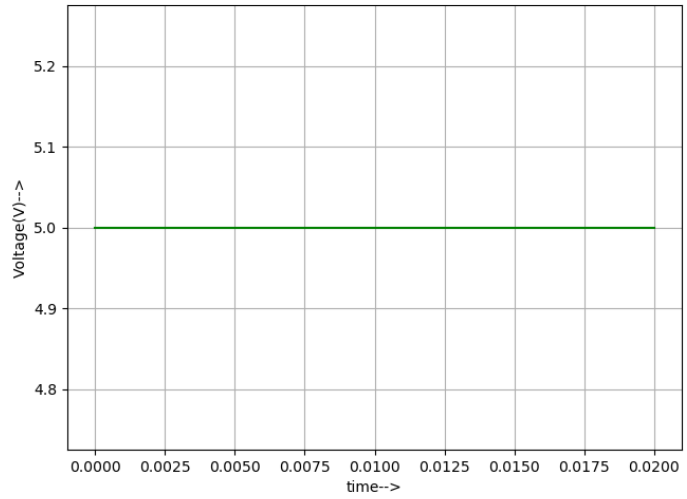
S0



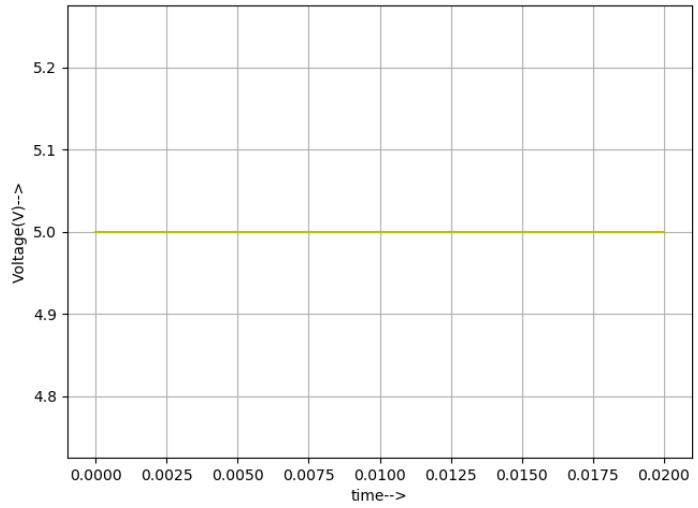
S1



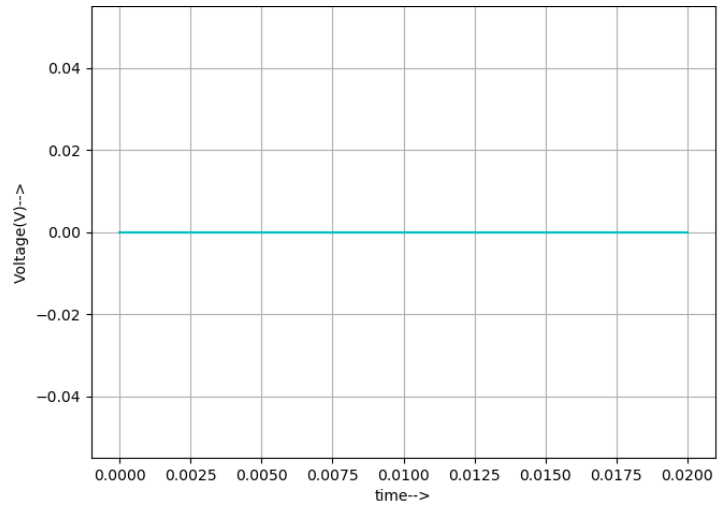
S2


5.2
5.1
5.0
4.9
4.8
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

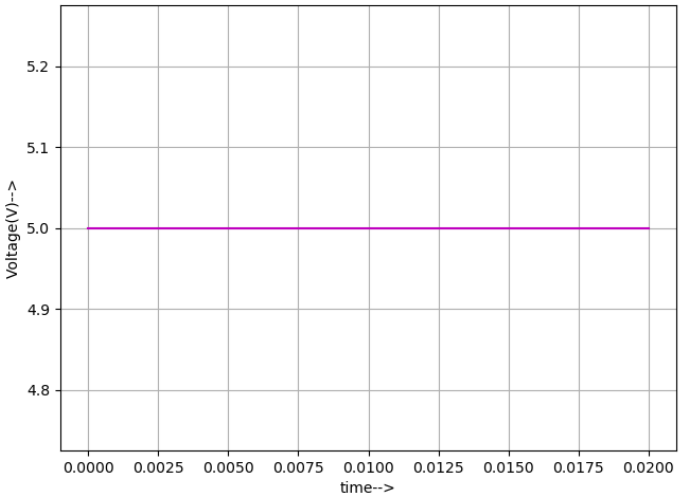
S3


5.2
5.1
5.0
4.9
4.8
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

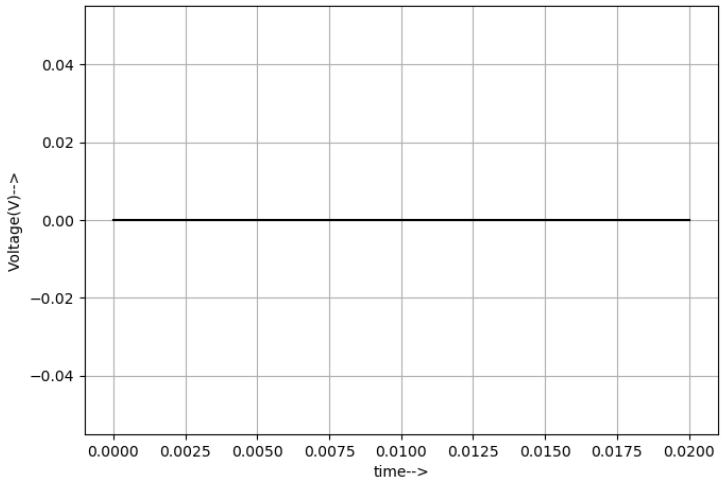
S4


0.04
0.02
0.00
−0.02
−0.04
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

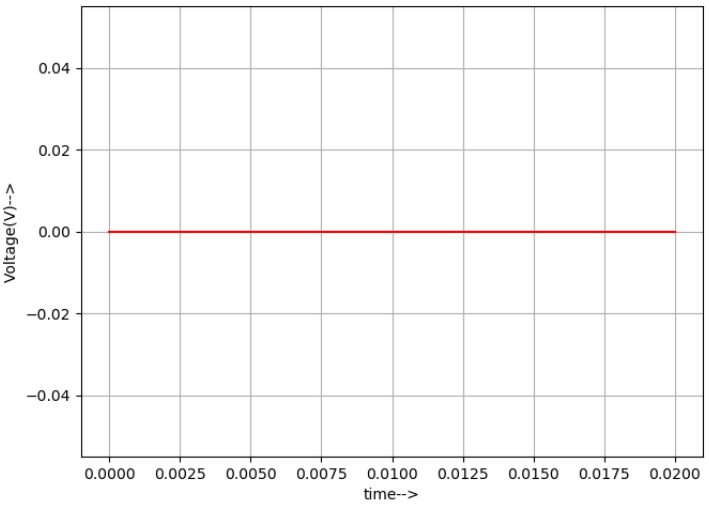
S5


5.2
5.1
5.0
4.9
4.8
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

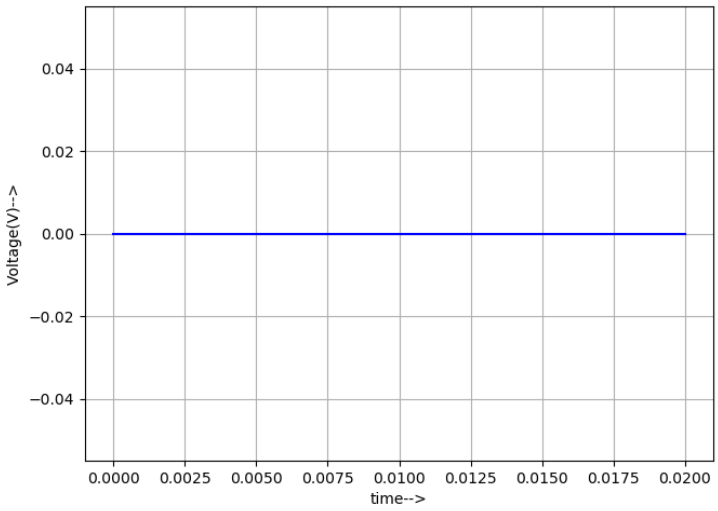
S6


0.04
0.02
0.00
−0.02
−0.04
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

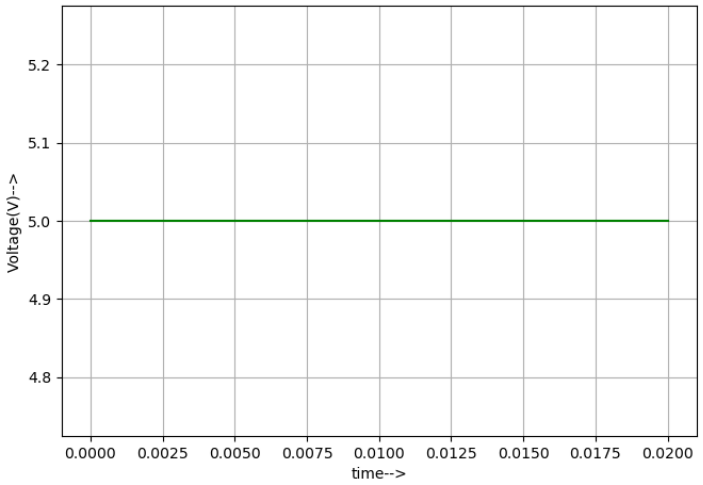
S7


0.04
0.02
0.00
−0.02
−0.04
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

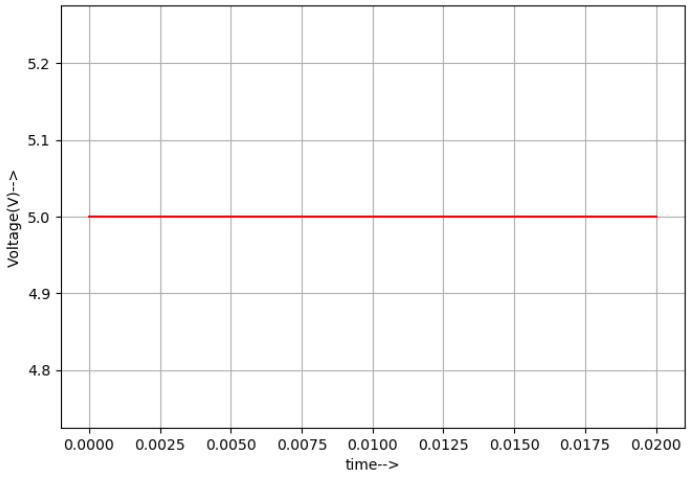
S8


0.04
0.02
0.00
−0.02
−0.04
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

S9


5.2
5.1
5.0
4.9
4.8
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

S10


5.2
5.1
5.0
4.9
4.8
Voltage(V)-->
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200
time-->

Cout

Source/Reference(s) :

1. https://en.wikipedia.org/wiki/Brent%E2%80%93Kung_adder
2. https://www.ijitee.org/wp-content/uploads/papers/v8i9S3/I31300789S319.pdf
3. https://en.wikipedia.org/wiki/Dadda_multiplier