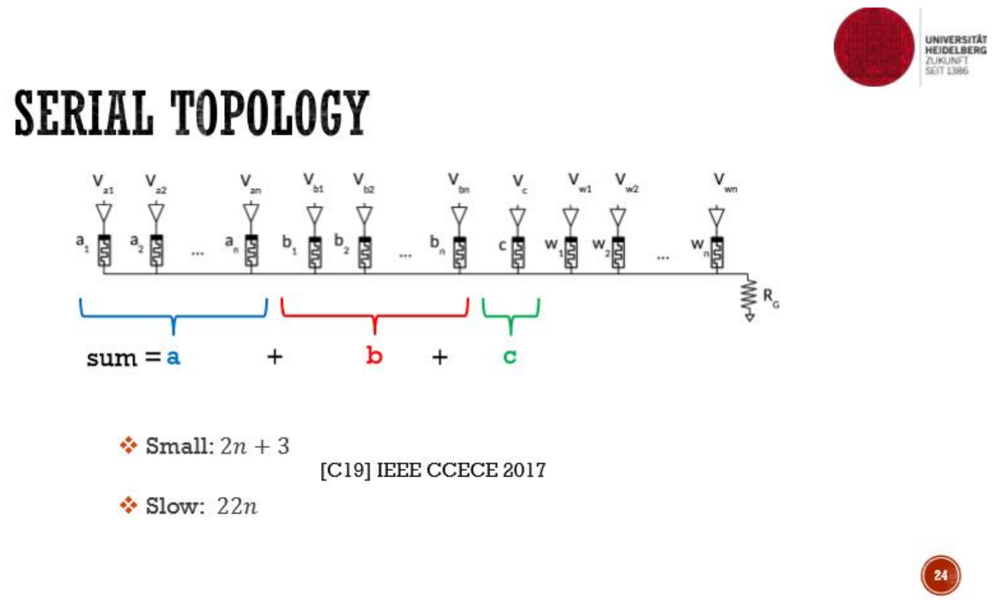
1. Description of the 3-bit adder circuit implementation:

The solution shown is a 3-bit serial full adder. The circuit is using 6 input bits, 3 for input variable a and 3 bits for input variable b. Furthermore, 1 carry bit and 2 working bits are used.

In difference to Figure 1, shown in lecture 08 – memristive in memory computing, we decided to only take two working bits into considerations. As shown in Figure 2, the implementation for a full adder algorithm for a serial computation is only using s1 and s2. Our tests have shown that nevertheless correct results were achieved.

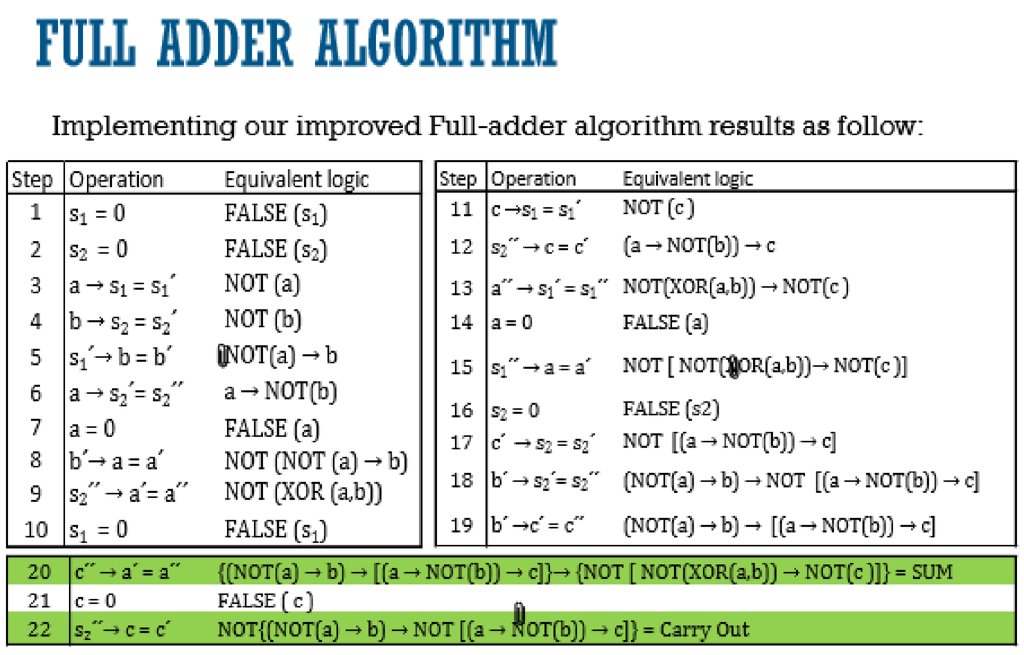

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386
V_{a1} V_{a2} V_{an} V_{b1} V_{b2} V_{bn} V_c V_{w1} V_{w2} V_{wn}
a_1 a_2 ... a_n b_1 b_2 ... b_n c w_1 w_2 ... w_n
R_G
sum = a + b + c
24

SERIAL TOPOLOGY

- Small: $2n + 3$
- Slow: $22n$

[C19] IEEE CCECE 2017

Figure 1: serial topology memristor full adder

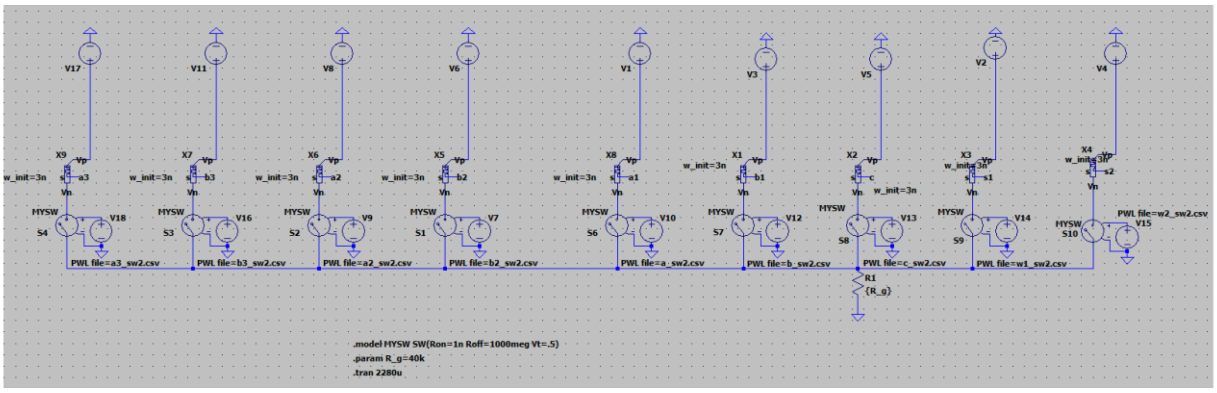


FULL ADDER ALGORITHM

Implementing our improved Full-adder algorithm results as follow:

Step	Operation	Equivalent logic
1	$s_1 = 0$	FALSE (s_1)
2	$s_2 = 0$	FALSE (s_2)
3	$a \rightarrow s_1 = s_1'$	NOT (a)
4	$b \rightarrow s_2 = s_2'$	NOT (b)
5	$s_1' \rightarrow b = b'$	NOT(a) → b
6	$a \rightarrow s_2' = s_2''$	a → NOT(b)
7	$a = 0$	FALSE (a)
8	$b' \rightarrow a = a'$	NOT (NOT (a) → b)
9	$s_2'' \rightarrow a' = a''$	NOT (XOR (a,b))
10	$s_1 = 0$	FALSE (s_1)
11	$c \rightarrow s_1 = s_1'$	NOT (c)
12	$s_2'' \rightarrow c = c'$	(a → NOT(b)) → c
13	$a'' \rightarrow s_1' = s_1''$	NOT(XOR(a,b)) → NOT(c)
14	$a = 0$	FALSE (a)
15	$s_1'' \rightarrow a = a'$	NOT [NOT(XOR(a,b))→ NOT(c)]
16	$s_2 = 0$	FALSE (s2)
17	$c' \rightarrow s_2 = s_2'$	NOT [(a → NOT(b)) → c]
18	$b' \rightarrow s_2' = s_2''$	(NOT(a) → b) → NOT [(a → NOT(b)) → c]
19	$b' \rightarrow c' = c''$	(NOT(a) → b) → [(a → NOT(b)) → c]
20	$c'' \rightarrow a' = a''$	{(NOT(a) → b) → [(a → NOT(b)) → c]}→ {NOT [NOT(XOR(a,b)) → NOT(c)]} = SUM
21	$c = 0$	FALSE (c)
22	$s_2'' \rightarrow c = c'$	NOT{(NOT(a) → b) → NOT [(a → NOT(b)) → c]} = Carry Out

Figure 2: full adder algorithm



```
.model MYSW SW(Ron=1n Roff=1000meg Vt=.5)
.param R_g=40k
.tran 2280u
```

Figure 3: implementation 3-bit full adder

Figure 3 shows the implementation in LT Spice for a 3-bit serial full adder circuit. The circuit is implemented in a serial way.

In this circuit the flow of computation is as follows: First the sum of bit a1, b1 and Carry is calculated, the result is written back to a1. Afterwards carry is newly calculated for the next iteration.

Iterations 2 and 3 are doing the exact same, except for bits a2, b2, a3 and b3.

The result is stored in a1-a3 and carry.

Following is the complete imply logic for this circuit.

s1 = 0
s2 = 0
a1 -> s1 = s1'
b1 -> s2 = s2'
s1' -> b1 = b1'
a1 -> s2' = s2"
a1 = 0
b1' -> a1 = a1'
s2" -> a1' = a1"
s1 = 0
c -> s1 = s1'
s2" -> c = c'
a1" -> s1' = s1"
a1 = 0
s1" -> a1 = a'
s2 = 0
c' -> s2 = s2'
b1' -> s2' = s2"
b1' -> c' = c" first two bit added

s1 = 0
s2 = 0
a2 -> s1 = s1'
b2 -> s2 = s2'
s1' -> b2 = b2'
a2 -> s2' = s2''
a2 = 0
b2' -> a2 = a2'
s2'' -> a2' = a2''
s1 = 0
c -> s1 = s1'
s2'' -> c = c'
a2'' -> s1' = s1''
a2 = 0
s1'' -> a2 = a2'
s2 = 0
c' -> s2 = s2'
b2' -> s2' = s2''
b2' -> c' = c'' second two bit added

s1 = 0
s2 = 0
a3 -> s1 = s1'
b3 -> s2 = s2'
s1' -> b3 = b3'
a3 -> s2' = s2''
a3 = 0
b3' -> a3 = a3'
s2'' -> a3' = a3''
s1 = 0
c -> s1 = s1'
s2'' -> c = c'
a3'' -> s1' = s1''
a3 = 0
s1'' -> a3 = a3'
s2 = 0
c' -> s2 = s2'
b3' -> s2' = s2''
b3' -> c' = c'' third two bit added

final result in a1, a2, a3, c

2. Input and Output

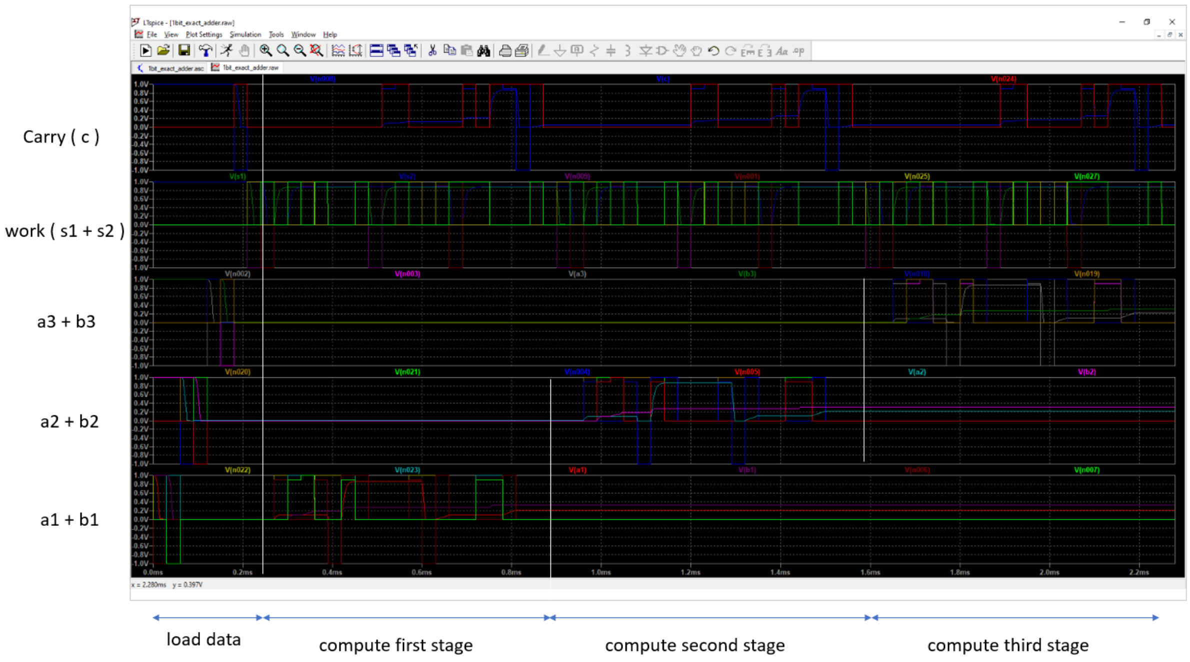


Figure 4: input Output

Figure 4 shows all the relevant input and output signals in the circuit for the 4 computational phases.

1. Loading data
2. Compute first stage
3. Compute second stage
4. Compute third stage

3. Tests:

To check the correct behaviour of the circuit, six tests were run.

3.1 Test 1:

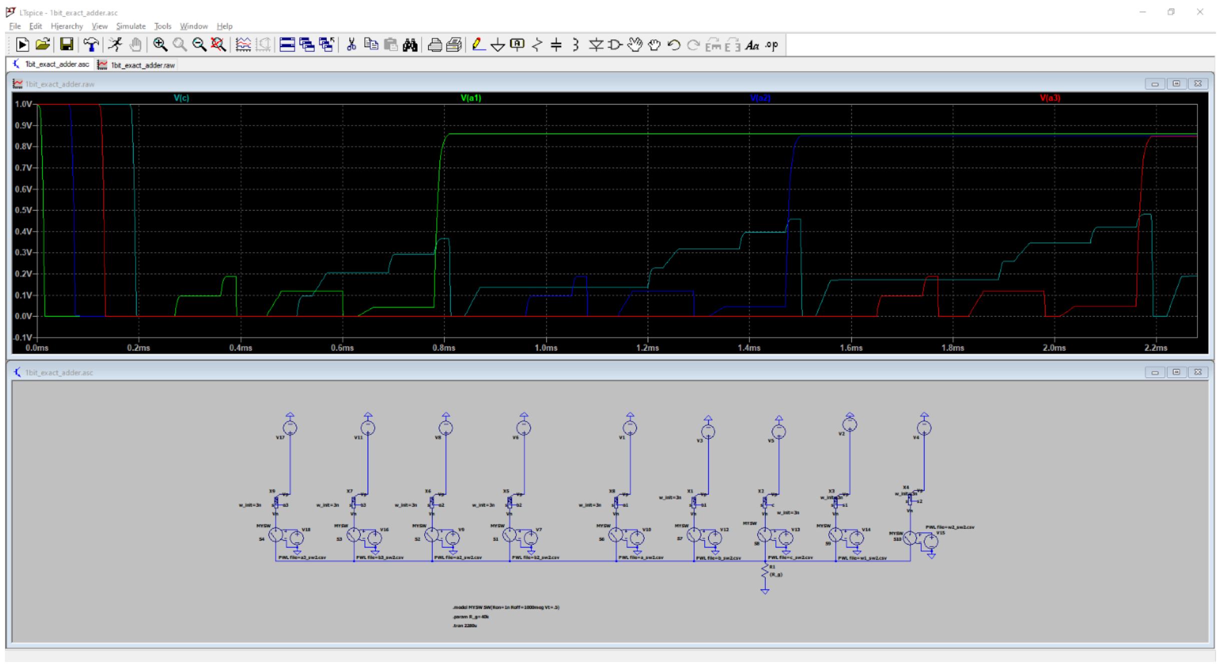


Figure 5:Test 1

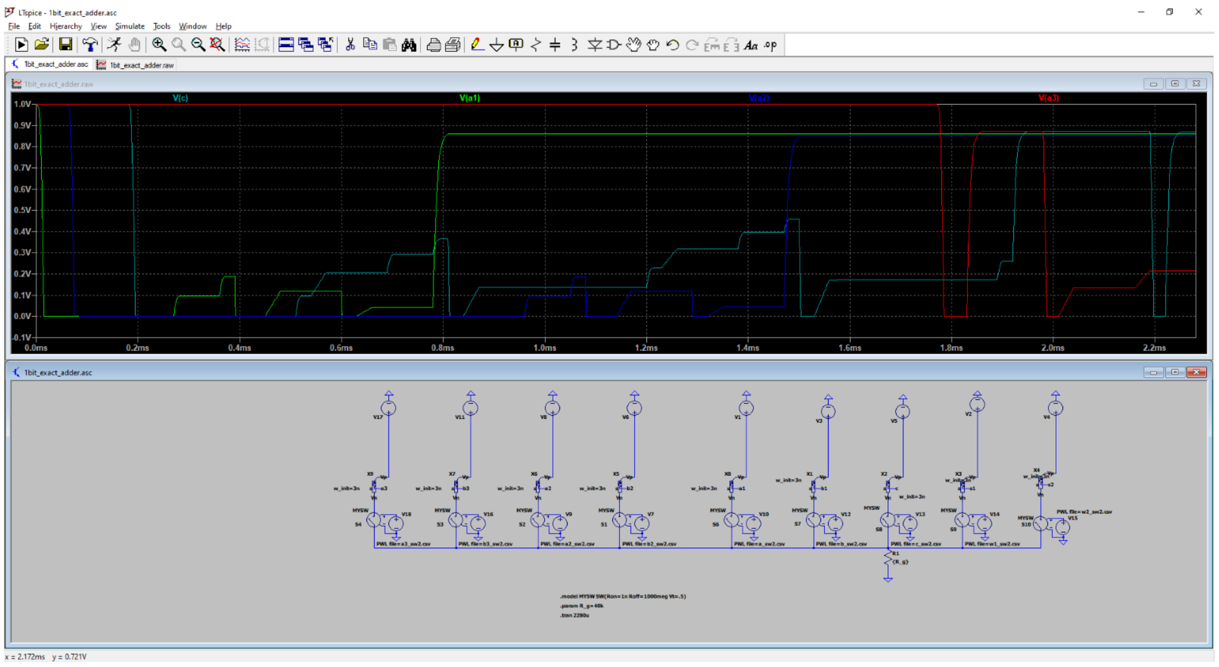
Test 1 loads at the beginning following data:

a(2:0) = 3'b000
b(2:0) = 3'b111
C = 0

On the output a result of a(2:0) = 3’b111 and Cout = 0 can be observed

The result is correct.

3.2 Test 2



```
.model MYSW SW(Ron=1n Roff=1000meg Vt=.5)
.param R_g=40k
.tran 2280u
```

Figure 6: Test 2

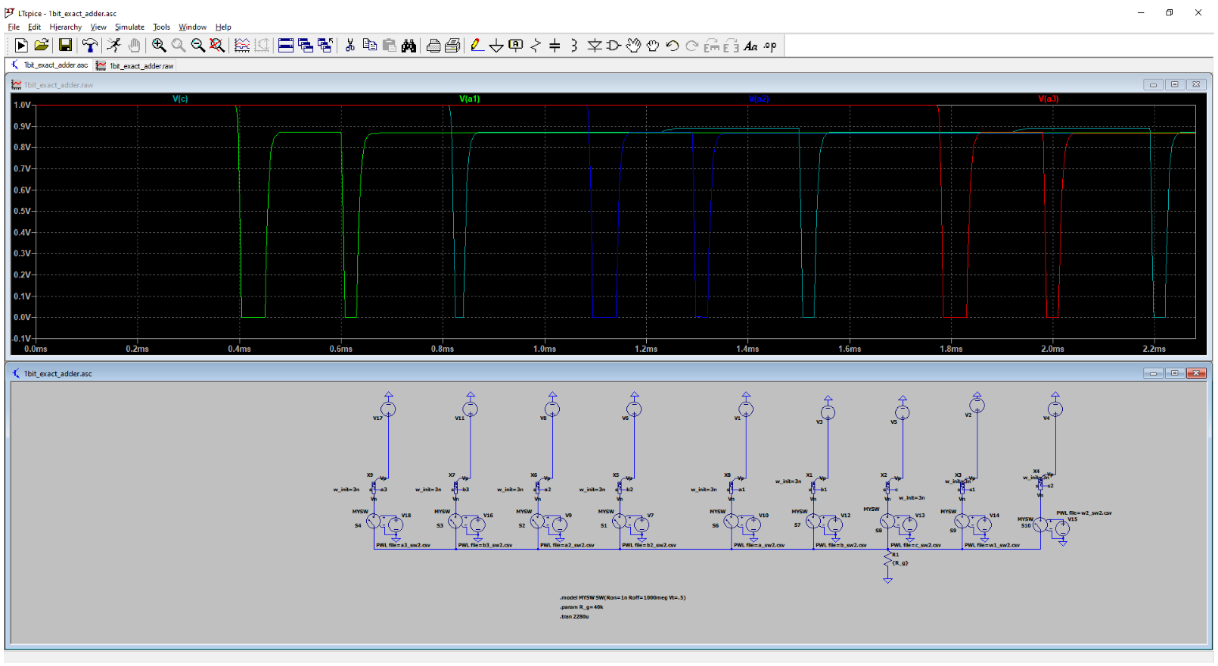
Test 2 loads at the beginning following data:

a(2:0) = 3'b100
b(2:0) = 3'b111
C = 0

On the output a result of a(2:0) = 3’b011 and Cout = 1 can be observed

The result is correct.

3.3 Test 3



```
.model MYSW SW(Ron=1n Roff=1000meg Vt=.5)
.param R_g=40k
.tran 2280u
```

Figure 7: Test 3

Test 3 loads at the beginning following data:

a(2:0) = 3'b111
b(2:0) = 3'b111
C = 1

On the output a result of a(2:0) = 3’b111 and Cout = 1 can be observed

The result is correct.

3.4 Test 4

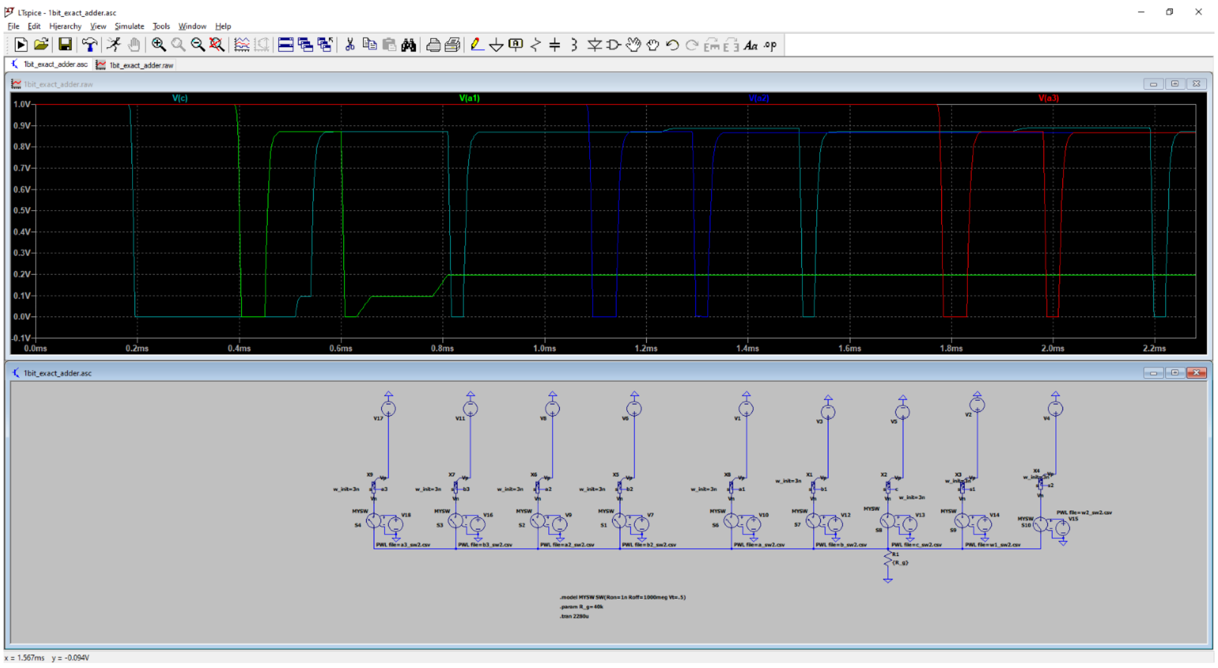

LTspice - 1bit_exact_adder.asc
V(c)
V(a1)
V(a2)
V(a3)
.model MYSW SW(Ron=1n Roff=1000meg Vt=.5)
.param R_g=40k
.tran 2280u
x = 1.567ms y = -0.094V

Figure 8: Test 4

Test 4 loads at the beginning following data:

a(2:0) = 3'b111
b(2:0) = 3'b111
C = 0

Out = 110
Cout = 1

On the output a result of a(2:0) = 3’b110 and Cout = 1 can be observed

The result is correct.

3.5 Test 5

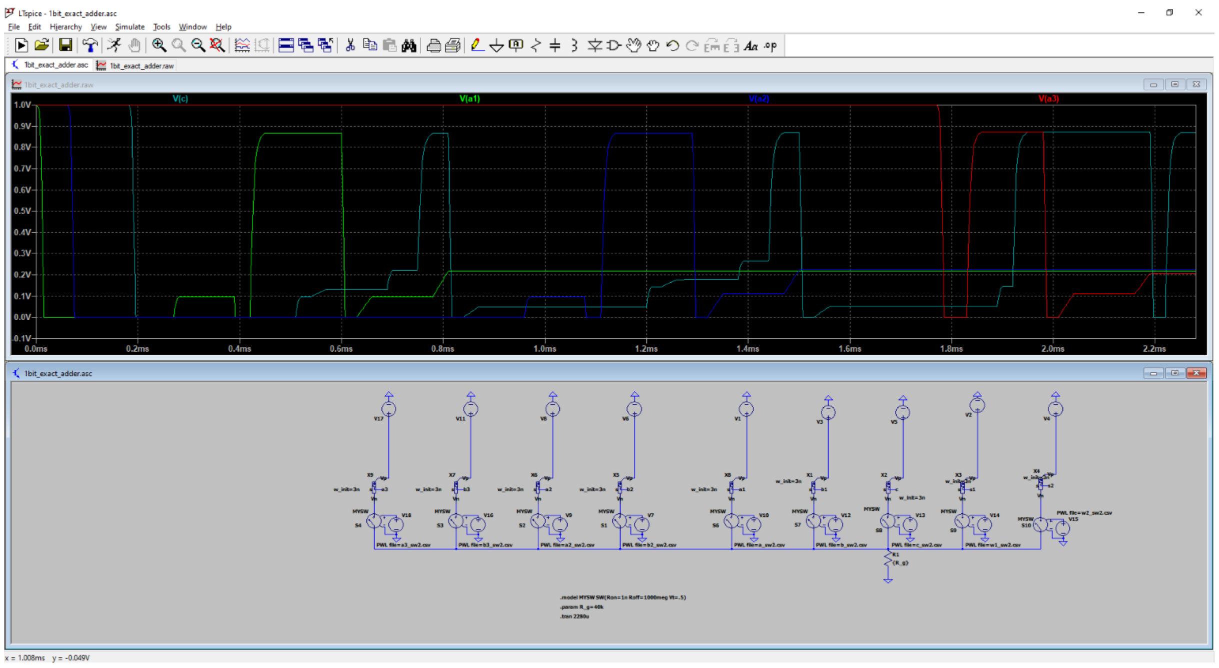

LTspice - 1bit_exact_adder.asc
V(c)
V(a1)
V(a2)
V(a3)
.model MYSW SW(Ron=1n Roff=1000meg Vt=.5)
.param R_g=40k
.tran 2300u
x = 1.008ms y = -0.049V

Figure 9: Test 5

Test 5 loads at the beginning following data:

a(2:0) = 3'b100
b(2:0) = 3'b100
C = 0

Out = 110
Cout = 1

On the output a result of a(2:0) = 3’b000 and Cout = 1 can be observed

The result is correct.

3.6 Test 6

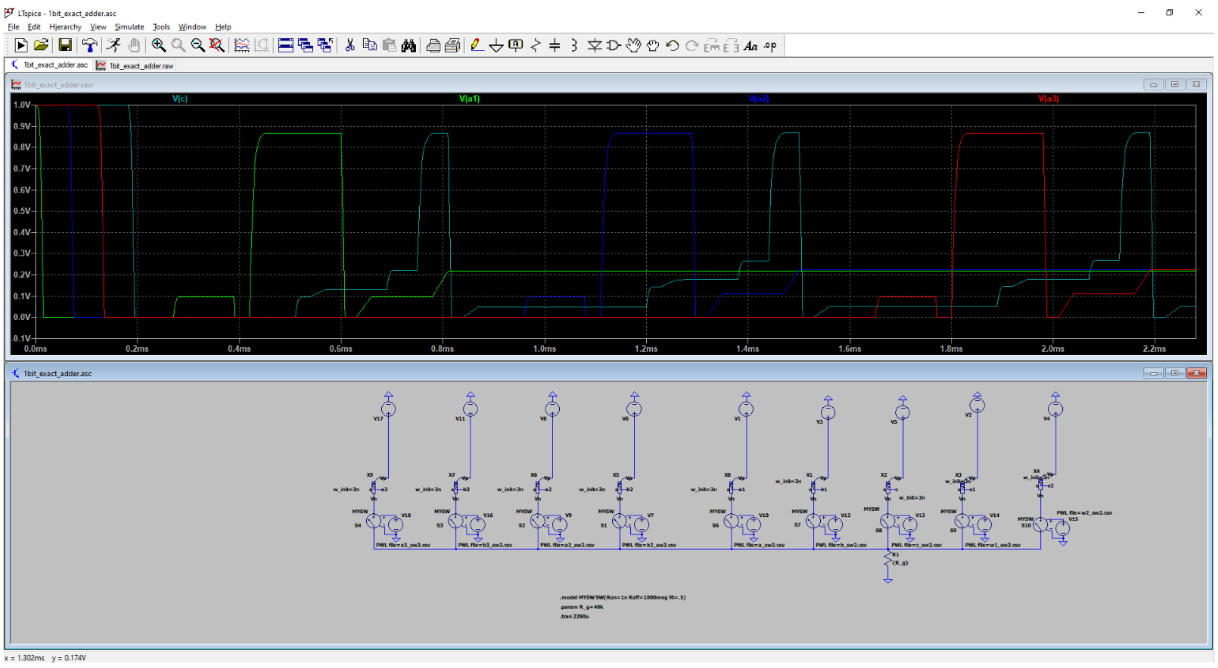


Figure 10: Test 6

Test 5 loads at the beginning following data:

a(2:0) = 3'b000
b(2:0) = 3'b000
C = 0

Out = 110
Cout = 1

On the output a result of a(2:0) = 3’b000 and Cout = 0 can be observed

The result is correct.

4. Summary

The tests showed that the circuit is working as accurate as expected, no failures could be found. During design of the circuit it came up that for each round of calculation the same steps need to be done. For sure this implementation can be optimised for higher speeds.

List of illustration

Source directory

Figure 1: lecture 08 – memristive in-memory computing by Prof. Dr. Nima Taherinejad slide 24 (6.12.2024)

Figure 2: lecture 08 – memristive in-memory computing by Prof. Dr. Nima Taherinejad slide 25 (6.12.2024)

All other pictures are created by the authors.