ECE 45600 - Project 1

Nicholas Marks

35757791

1 Design Diagrams

1.1 Top-level Design

The Manchester Adder consists of three top-level blocks: the propagate-generate block, the carry chain and the summation.

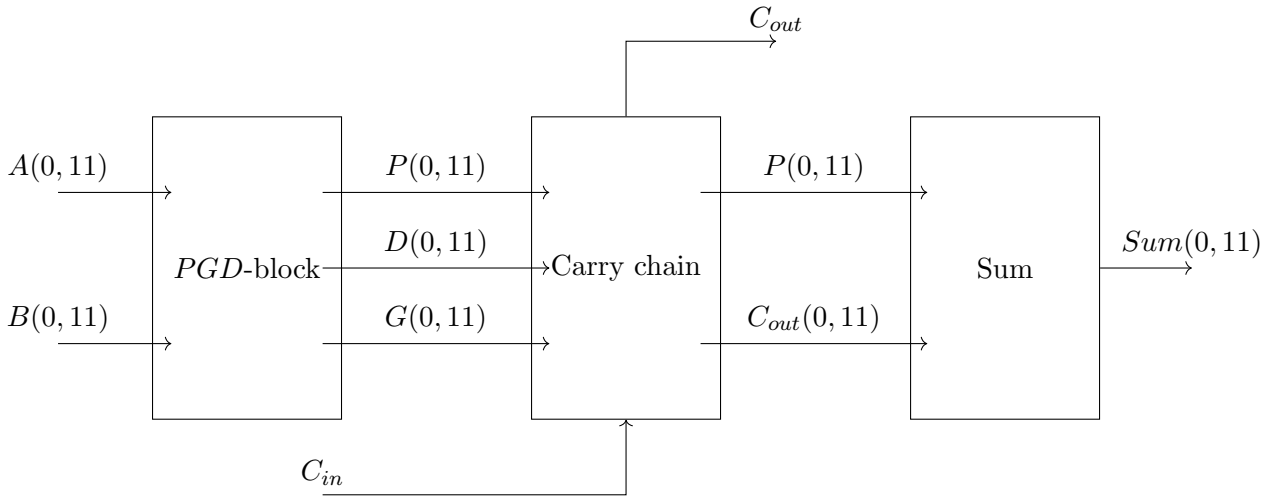

Figure 1: Top-level design of Manchester Adder

In Figure 1, $A(0,11)$ and $B(0,11)$ represent the two 12-bit input numbers to sum and $P(0,11)$ and $G(0,11)$ represent the propagate and generate bits respectively.

We will now proceed to describe the low-level implementation for each block in turn.

1.2 PGD Block

From the lecture notes, we have that the propagate bit P_i and generate bit G_i are given by the following equations:

$$P_i = A_i \oplus B_i \tag{1}$$
$$G_i = A_i \wedge B_i \tag{2}$$
$$D_i = \overline{A_i} \wedge \overline{B_i} \tag{3}$$

using the truth tables and content from the lecture slides.

To implement these two expressions in complimentary CMOS, we will generate a NAND gate and an XNOR gate as shown below:

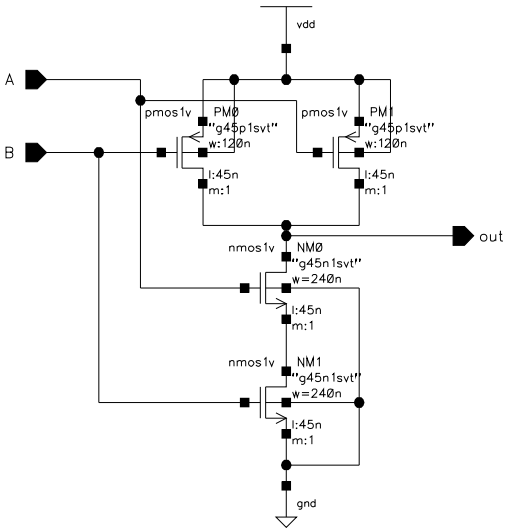

Figure 2: NAND gate output for generating G_i

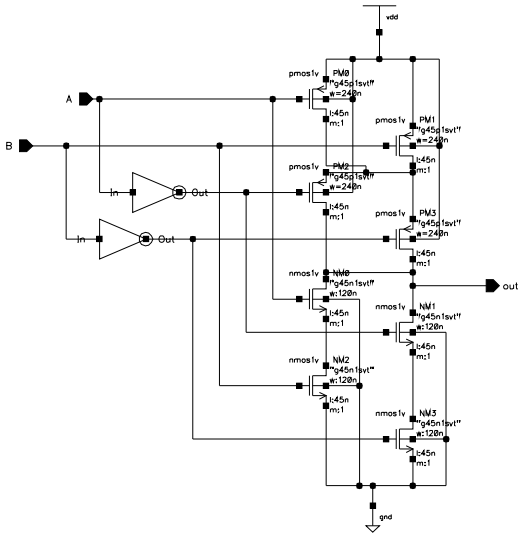

Figure 3: XNOR gate for generating P_i

The unit tests for each of these gates can be found in the Appendix. Since we have implemented the compliments of each of these functions, we may need to include an inverter on the output depending what block the output signal feeds into. We will see that in actuality no such inversion is really required, since all subsequent blocks are themselves complimented.

As far as design is concerned, we use assume a minimum-size inverter of $W_{MOS} : L_{MOS} = 120n : 45n$ and so each double-stacked NMOS or PMOS configuration, such as the pull-down network for G_i has been sized up by a factor of two to account for the increased logical effort.

Finally, for the D_i (delete) signal, we can reuse the NAND gate designed for G_i, this time inverting the input. However, with the specification of increasing speed, it is faster to generate D_i via de Morgan's Law as a NOR gate:

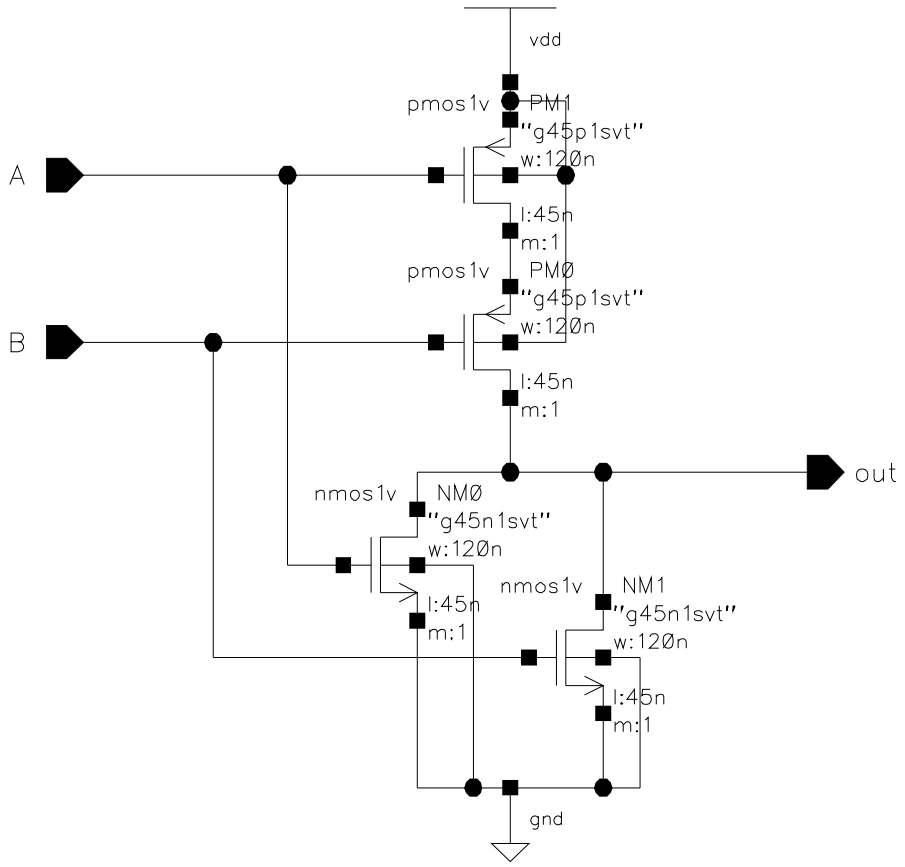

Figure 4: The NOR gate used to generate D_i

1.3 Carry Chain

For the carry chain, the carry out of each bit addition is described by the logic function

$$C_0 = G_i + PC_i \tag{4}$$

and can be generated by the following schematic, drawn from the lecture slides.

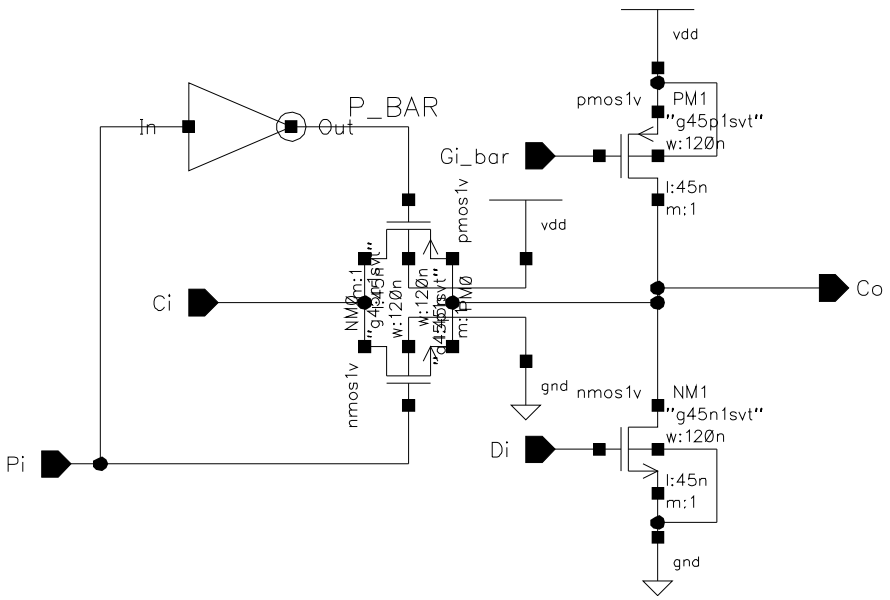

Figure 5: The schematic for generating the carry-out bit at each stage

The use of pass transistor logic here vs. complimentary CMOS enables for both a smaller design area and faster performance. One change that is made from the above figure is to generate $\overline{P_i}$ separately from P_i so that we can remove the delay of the input inverter above from the overall propagation delay. By moving $\overline{P_i}$ to its own function, we do however increase the overall area of the design.

1.4 Sum Block

Finally, for the sum bit S_i at each stage and the carry-out of each stage $C_{0,i}$ we have the following generating expressions:

$$S_i = P_i \oplus C_{in,i} \tag{5}$$

$$C_{0,i} = G_i + PC_{in,i} \tag{6}$$

The implementations of S_i is given below:

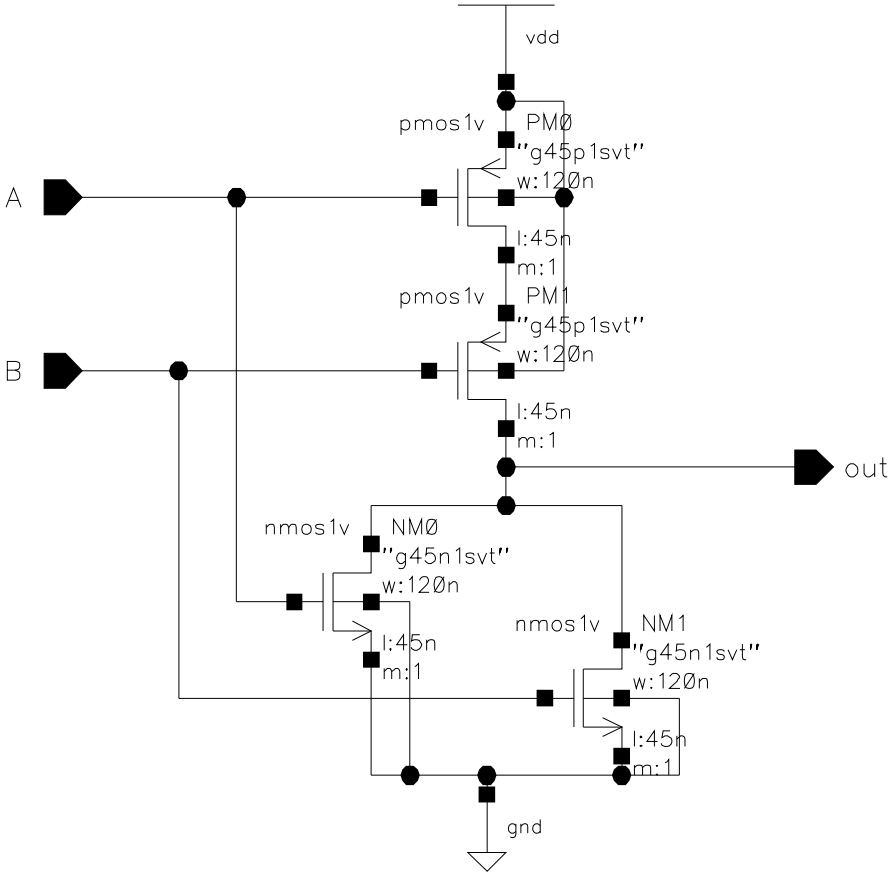

Figure 6: Schematic for each summed bit

1.5 Full Adder and Verification

We now have all the pieces for testing a 1-bit adder. It would seem that if this works succesfully, then generating the full 12-bit adder is simply a matter of concatenating 12 of these units together...

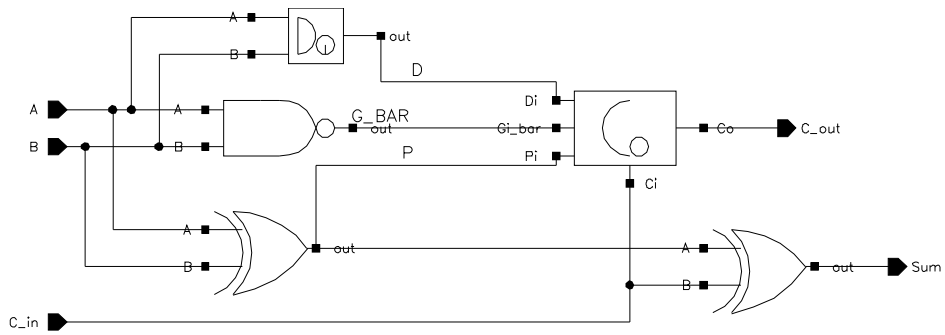

Figure 7: A 1-bit adder for two input bits A and B. Note that the block above the NAND gate represents the delete block (see Figure 4) and the block to its right the carry-out block (see Figure 5

The test outputs for all possible 2-bit combinations is as follows:

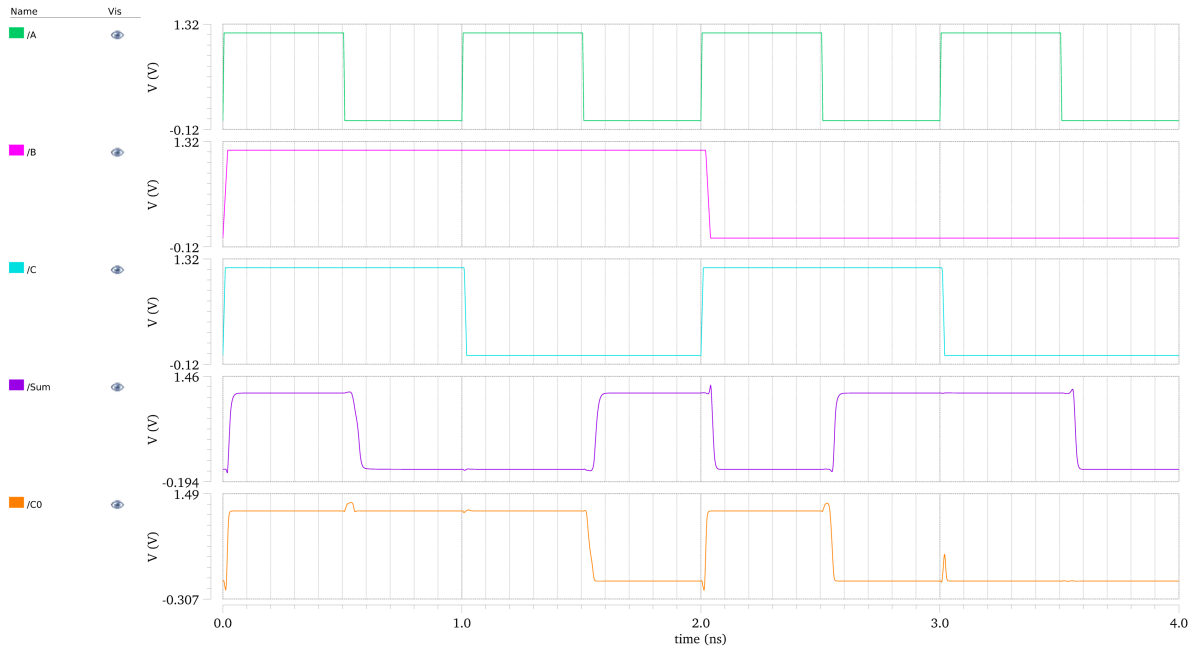

Figure 8: Test outputs for the 1-bit adder. There are a few small bits of noise in the transition regions when running A at a period of 1 ns.

In combining twelve 1-bit adders naively into a full 12-bit adder, it was found that the signal C_{out}, being passed repeatedly through logic gates, was attenuated over time such that by the 12th bit, the high voltage had become sufficiently weak so as to produce an incorrect value. By trialing different combinations, it was found that splitting the full adder into three 4-bit adders each proceeded by a buffer (double inverter) on the C_{out} line rectified the correctness of the output whilst still maintaining low latency.

The 4-bit adder blocks look as follows:

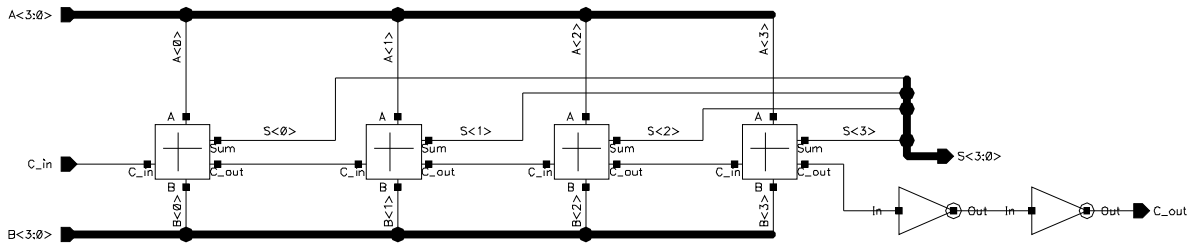

Figure 9: A 4-bit adder block including the buffer on C_{out}

and the final 12-bit Manchester adder:

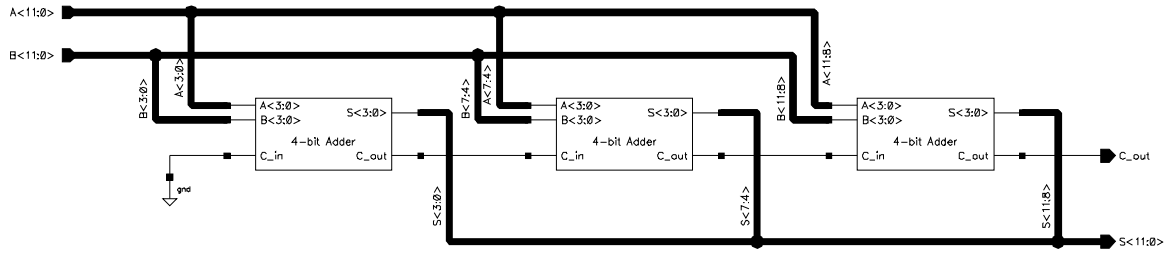

Figure 10: The Manchester adder built from three 4-bit adder blocks

2 Testing

Using the provided test cases in the project document, the adder clearly gives the correct input. See the appendix for the vector file used.

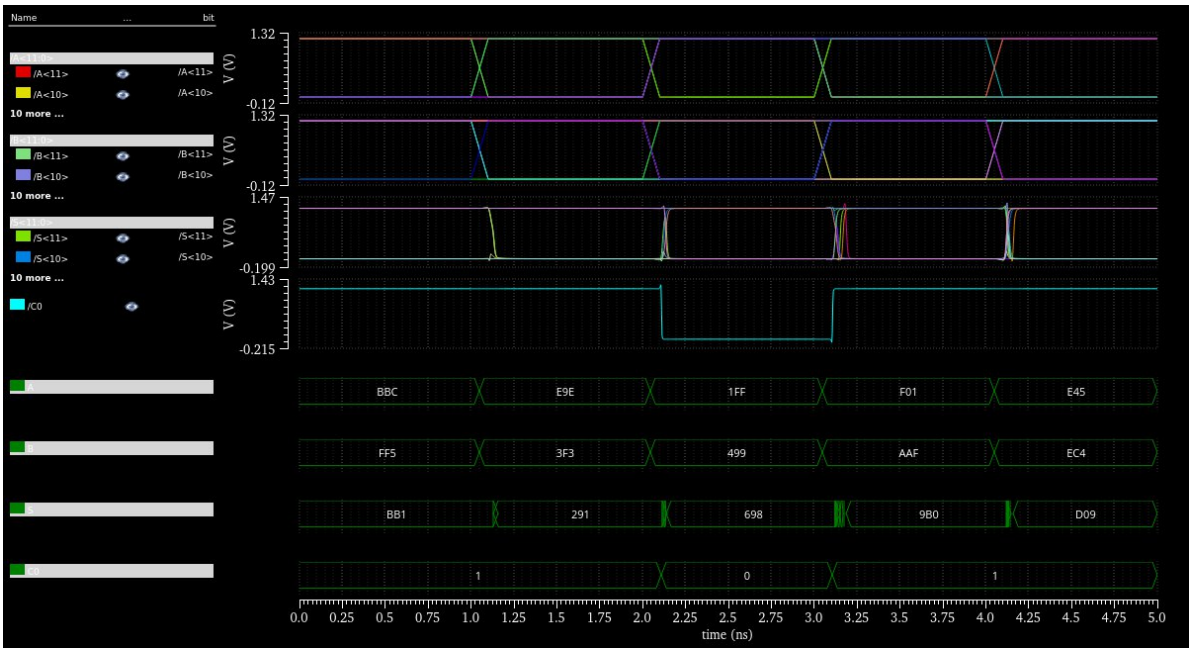

Figure 11: The verification of the adder, digital signals provided at the bottom

The circuit could be run up to a period of 0.5 ns before incorrect output was registered. Above, the circuit has been run at a period of 1.0 ns and the propagation delay can be calculated as 0.17 ns from the plots.

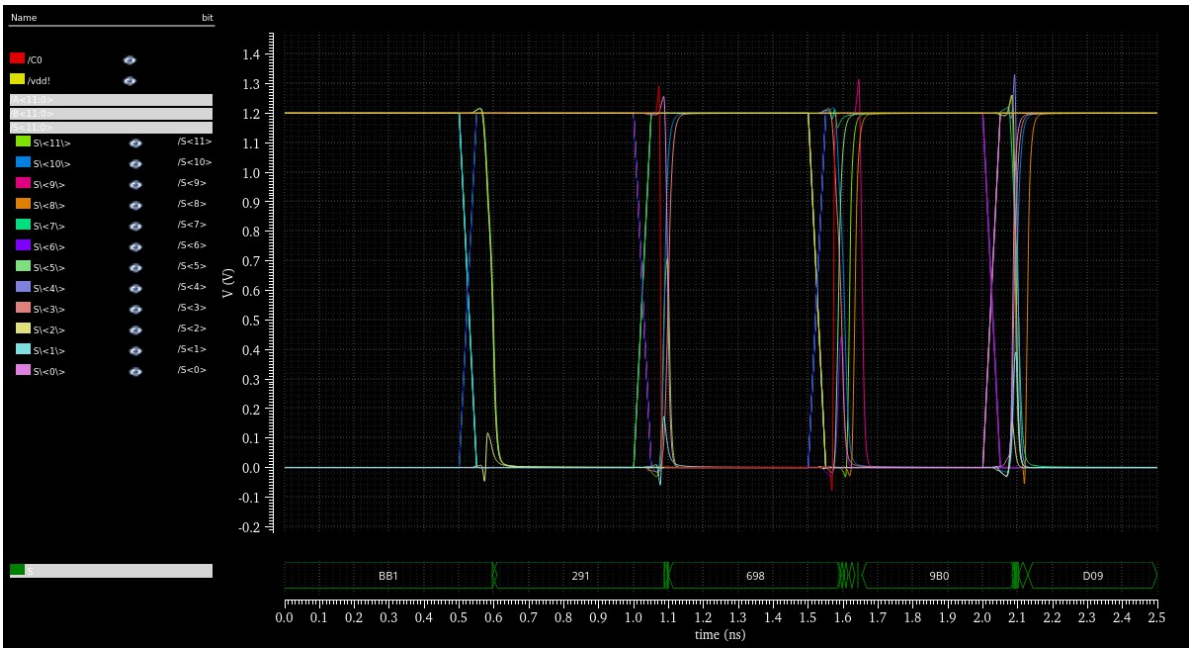

Figure 12: Running at a period of ≤ 0.5 ns causes incorrect outputs during the third-fourth and fourth-fifth transitions

3 Power Consumption

The power consumption is calculated from the testing[1]as directed in the project notes. The power consumption at the fastest possible, correct period (0.5 ns) was 53.6 μW. This was taken as the average of the product between the source current I_{DD} and the source voltage V_{DD}.

Figure 13: Power consumpption of the adder - note the negative sign indicates a power draw.

4 Adder Comparison

An ideal prefix network would comprise $\ln N$ logic stages, a fanout ≤ 2 at each stage, and no larger than one horizontal branch across a particular stage of computation. Each of the prefix adders under examination here consist of $2\ln N$ stages, $N/2+1$ fanout and $N/2$ tracks between gates at the same level.

If we break up the prefix adder into three steps,

1. Precomputation, the process of generating propagate (P), generate (G) and delete (D) vectors from the N-bit inputs A and B
2. Prefix, the generation of subsequent P, G and D bits from lower-order bits and
3. Postcomputation, the computation of the actual sum and final carry out

then each type of prefix adder (HC, BK, LF, KS and M (Manchester)) can be differentiated by the particular approach to prefix computationHC, BK, LF, KS and M (Manchester)) can be differentiated by the particular approach to prefix computation. There is generally a trade-off between lower latency and design area. Designs like BK and HC opt for a greater number of logic levels at the reward of reducing fanout by a factor of two. Reducing fanout can reduce the overall wiring length of the design, thereby minimising (interconnect) capacitance in the synthesised layout. LF, KS conversely minimise the number of logic levels at the expense of increased fanout.

Considering inverting static CMOS-logic congruent to that used in our Manchester adder, Harris finds that with a wiring capacitance ratio of $w = 0.5$, the architectures in order of increasing delay are: HC, KS, BK, LF. Knowles describes a similar result, namely that KS outperforms LF. It would seem then that prioristing logic levels and layout over fanout produces performant adders.

As far as our Manchester adder is concerned, we have a fanout ≤ 2 as there is no point in our design where one output drives more than two consequent inputs. The design has a logic depth of 4, which means between these two factors our design is similar in respects to BK, but with a slightly lower fanout meaning the overall adder has a larger area - as discussed in the preceding paragraph, a lower fanout at the expense of design area generally results in a slower computation.

[1]Using the new test values for A and B

References

1. S. Knowles, "A family of adders," in Proc. 14th IEEE Symp. Computer Arithmetic, Apr. 1999, pp. 277–281.

2. D. Harris, "A Taxonomy of Parallel Prefix Networks," in Proc. 37th Asilomar Conf. Signals Systems and Computers, pp. 2213–2217, 2003.

5 Appendix

Note that throughout these test benches, I have assumed a $V_{DD} = 1.2\,V$.

5.1 Sub-design Testing

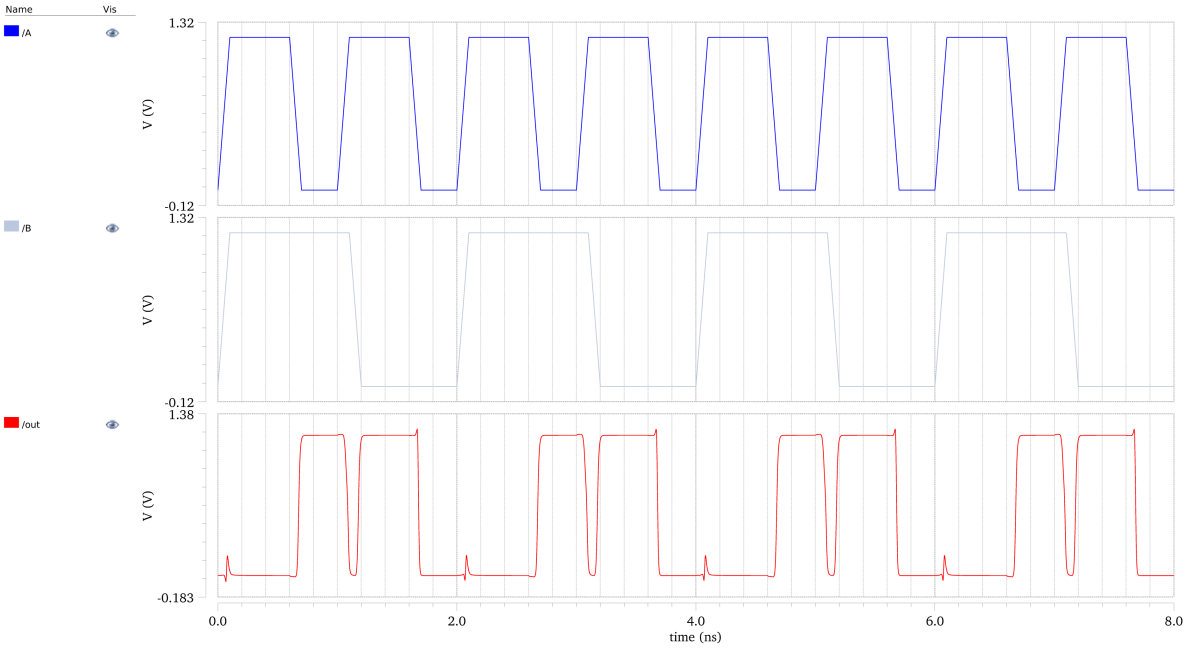

Figure 14: Test of XNOR gate

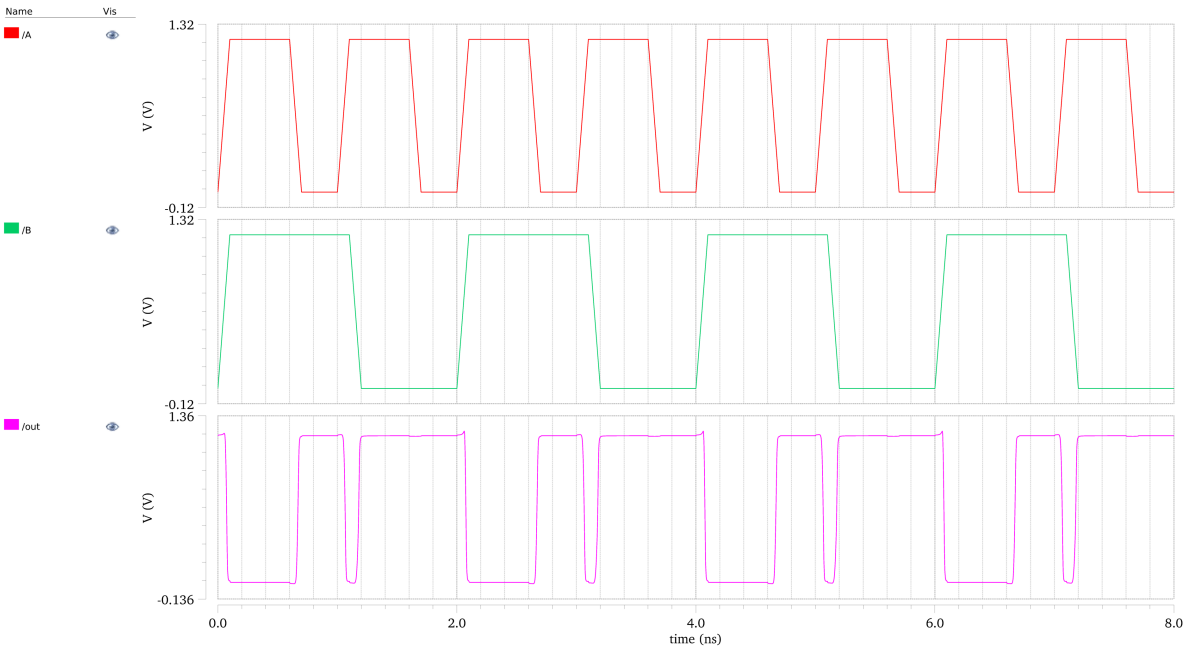

Figure 15: Test of NAND gate

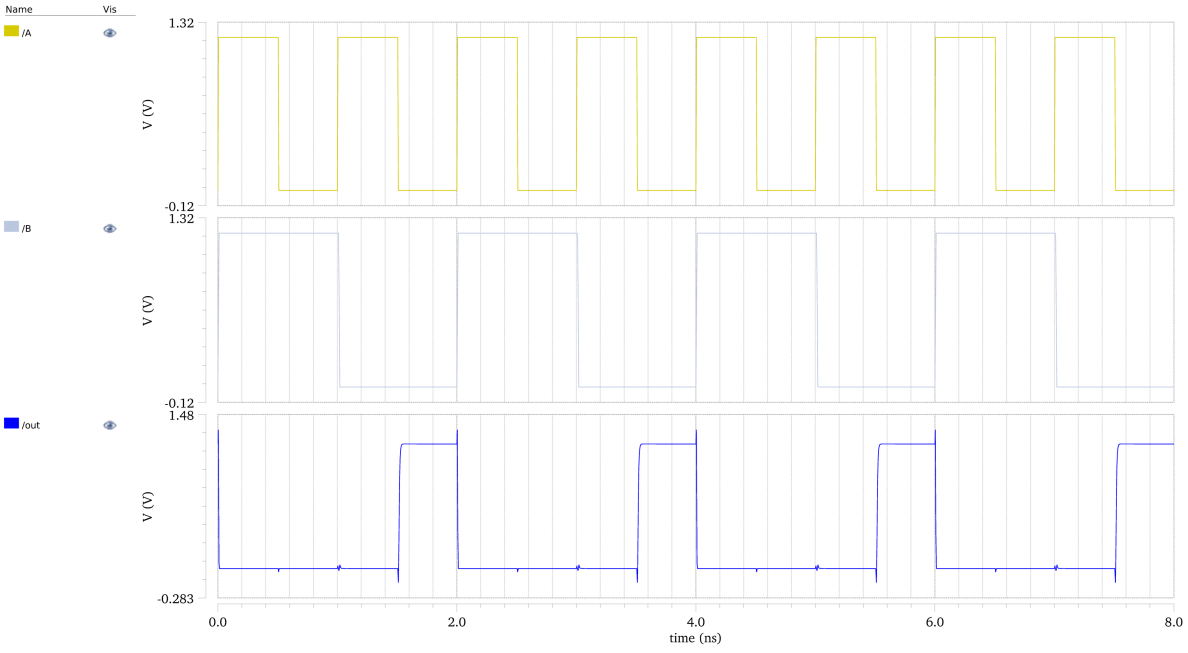

Figure 16: Test of generating D_i

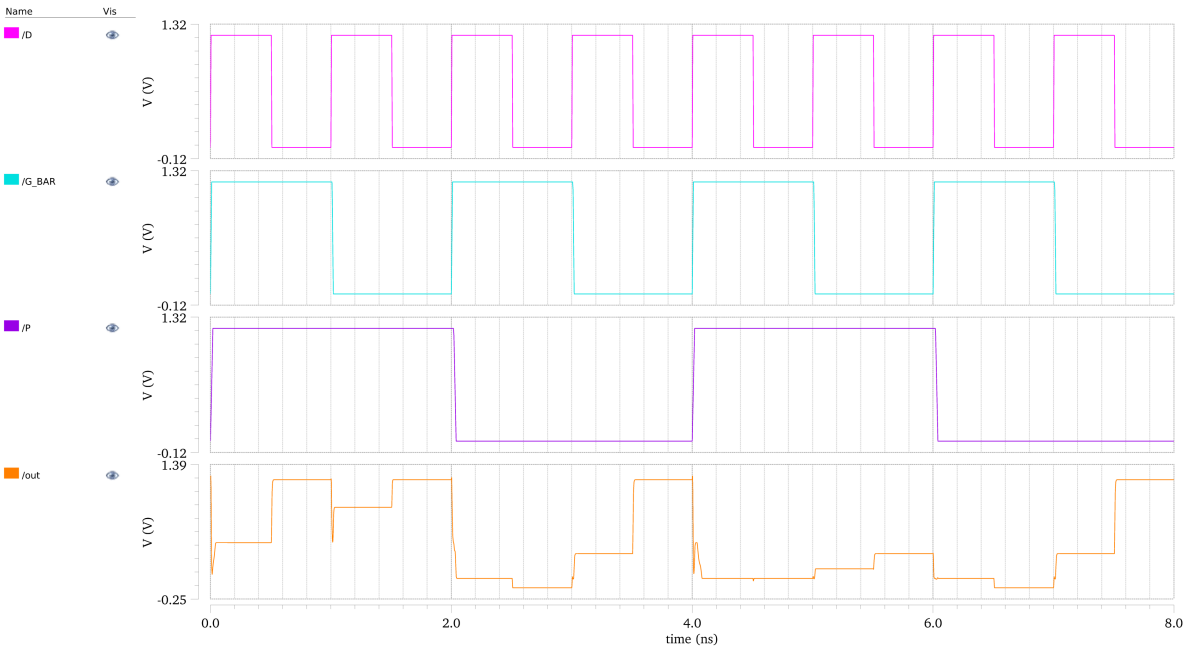

Figure 17: Test of generating $C_{out,i}$

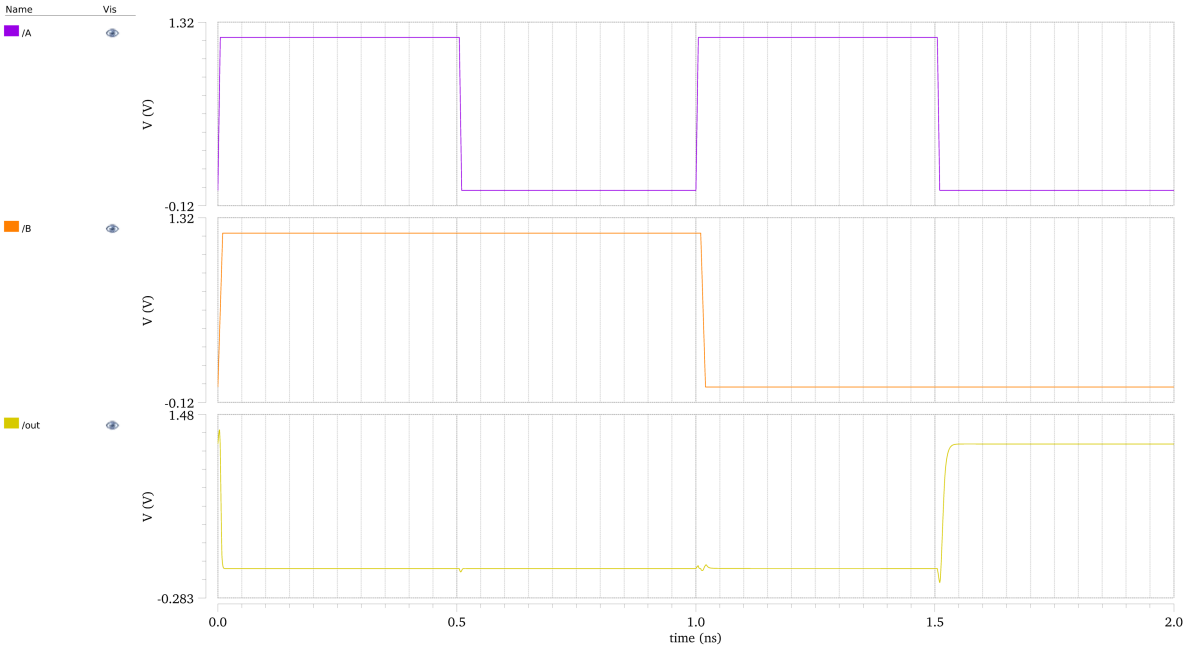

Figure 18: Test of generating S_i

5.2 Vector Files

5.2.1 4-bit Adder

```
; Enable waveform output
output wf_0

; RADIX specifies num of bits for each wire/bus
radix 4 4 4 1

; VNAME for naming each vector
vname A<[3:0]> B<[3:0]> S<[3:0]> C0

; IO specifies direction of wire/bus
io i i o o

; tunit for each time value
tunit 1ns

; CHK_window for where to check for output
; chk_window -19 20 1 period=10

; PERIOD of signal
period 1

; TRISE and TFALL of signal
trise 0.2
tfall 0.2

; VIH and VIL for input high/low voltages
vih 1.2
vil 0.0

; VOH and VOL
voh 0.7
vol 0.5

; Tabular test data
0 0 0 0
A B 5 1
F F E 1
D 5 2 1
5 5 A 0
```

5.2.2 12-bit Manchester Adder

```
; radix specifies the number of bit of the vector.

radix 444 444 444 1

; io defines the vector as an input or output vector.

io   iii iii ooo o

; vname assigns the name to the vector.

vname A<[11:0]> B<[11:0]> S<[11:0]> CO

; tunit sets the time unit.

tunit 0.2 ns

; trise specifies the rise time of each input vector.
```

```
18
19 trise 0.1
20
21 ; tfall specifies the fall time of each input vector.
22
23 tfall 0.1
24
25 ; vih specifies the logic high voltage of each input vector.
26
27 vih 1.2
28
29 ; vil specifies the logic low voltage of each input vector
30
31 vil 0.0
32
33 ; voh specifies the logic high voltage of each output vector
34
35 voh 1.0
36
37 ; vol specifies the logic low voltage of each output vector
38
39 vol 0.2
40
41 ; time per bit
42 period 1.0
43
44 ; data
45 BBC FF5 BB1 1
46 E9E 3F3 291 1
47 1FF 499 698 0
48 F01 AAF 9B0 1
49 E45 EC4 D09 1
```