Dadda Digital Multiplier

(April 2019)

*Abstract*—Multiplier as an important component of today’s digital circuit has a variety of usage in IC. Almost all kind of microprocessor, microcontroller, CPU cannot work without multiplier. There are a large variety kind of designs in multiplier. This design is based on Dadda multiplier which use Dadda reduction to reduce the carry and sum bit in order to reduce the delay caused by propagation. This design does not include any optimization or changing from the regular version of Dadda multiplier. The partition of the partial products is not further separated because it does not really help the performance for 8\*8 multiplier. The design simulation is provided with the operating limit of this multiplier.

*Index Terms*—Dadda multiplier, partial products, column compression.

# Introduction

Multiplier as an important component of today’s digital circuit has a variety of usage in IC. Almost all kind of microprocessor, microcontroller, CPU cannot work without multiplier. There are large variety kind of designs in multiplier. Compared to regular multiplier, the column compression technique-based multiplier is preferred. Since the column compression technique will make the total delay proportional to the logarithm of the operand word lengths, at the same time the array-based multiplier’s delay is proportional to the word length.

Two of the most popular column compress-based multiplier is Wallace multiplier and Dadda multiplier. Compared to Wallace multiplier, Dadda multiplier is slightly faster than Wallace multiplier, and the space and hardware requirement is less than Wallace multiplier as well. The mainly difference behind a Wallace multiplier and Dadda multiplier is the grouping of partial product. In this design, team implement a regular Dadda multiplier.

The delays of a multiplier can be divided into three part which is partial product generation (PPG), partial product summation (PPST), and the last stage adder. The PPST and last stage adder is the main contributor to the multiplier delay. The PPG’s delay is relatively low since every partial product is generated parallelly which makes the delay not that important to the total delay. The column reduction technique mainly reduces the delay of PPST. There is some technique to reduce the delay of last stage adder as well, but in this design, there are not optimization for the last stage adders. This report is structured as follow: Section II and III illustrate the design of parallel structure for PPST. Section IV reports the simulation result, and power calculation for this design. And finally, section V summarize the simulation results. In this report, we only discuss 8\*8 multiplier.

# design review

The basic idea of multiplier is pretty straightforward. It computes the partial sum of each bit. Add them together according to their relative position. The main challenge in designing a multiplier is to make it as fast as possible. As stated before, the main idea of this design is to use the column reduction approach to make the multiplier faster.

## Partial product generation

This is the first part of this design and basically the first part of all the multiplier design. It requires product of all the combination of the input operand ranges from A0B0, A0B1…. A7B7(8 bits). The idea here is that the sum of two bit is actually the same thing as doing AND operation. The carry is ignored in this stage which will be covered the next step. To generate this partial product, the only thing needed is 64 AND gates which computes all the partial product simultaneously. Since it is paralleled that means the delay of the generation of partial product will not pass to the next stage. As a result, the delay minimization for this stage is not interesting for most of the design. The way team implement the AND gate is using the transmission gate logic. It contains two transmission gates; the logic analysis is straightforward. The input is Ai, the upper logic output is Ai\*Bi, while the lower output is Bi\*Bi’. Then for the intrinsic behavior of transmission gate, it adds the upper logic and lower logic together. So the output is:

Ai\*Bi+Bi\*Bi’ = Ai\*Bi

This gate realizes the AND operation with only 4 transistors. Compared to the static CMOS logic gate which requires, it is basically the same, but we no longer need to worry about the wiring of Vdd and GND which ease the difficulty of wiring.

## Dadda reduction and partial product summation

This is the core part design of a Dadda multiplier. The idea is based on the reduction produce the partial summation while push the carry to the next column to get the length of each column reduced by certain amount. The tool we used here is half adder and full adder which will be discussed in detail the next section. In this design, we use half adder to sum 2 bits together to produce 2 bits; one is sum and one is carrying. Sum will stay at the same column; however, 2 bits only produce one sum, that means at this column the length is reduced by one. At the same time, carry will go to the next column which will not contribute to the length of this column. Although, the length of next column seems to increase at this moment, the length of the next column has already been reduced by using half or full adder which has already decreased a certain amount. What we need to make sure is that the total length is reduced.

The next question is to determine how much should be deducted each time. The inventor of this multiplier has already defined a criterion for reduction. First, it is easy to see that the reduction cannot be done for just one stage. The idea is to limit the largest length of each column in one stage. The largest number is determined by a series of numbers:

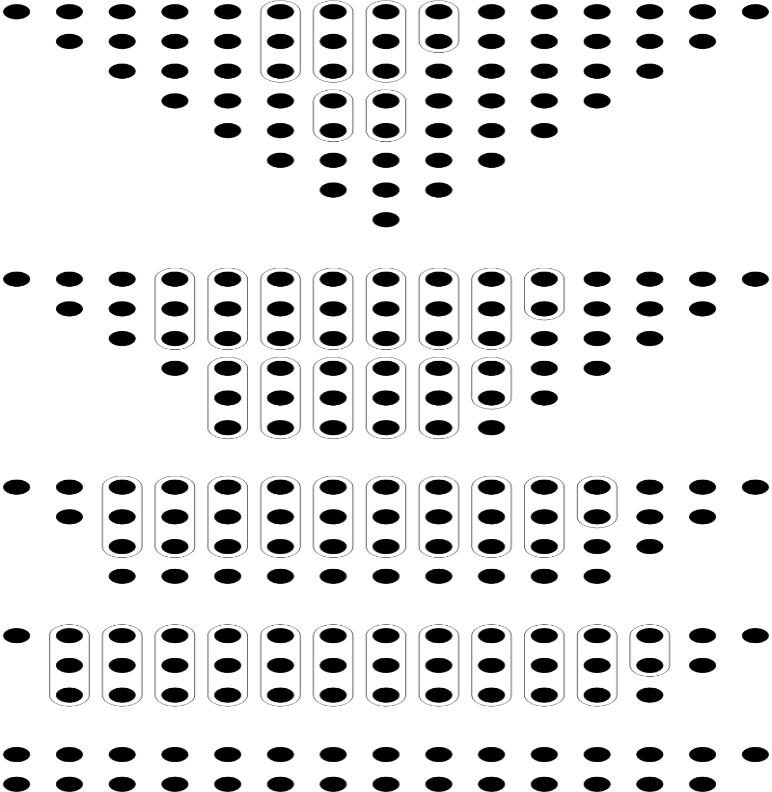
dj = floor(1.5dj-1)

j is the stage number while d is the largest length of column that is permitted by all the columns. So, the sequence is 2, 3, 4, 6, 9, 13…

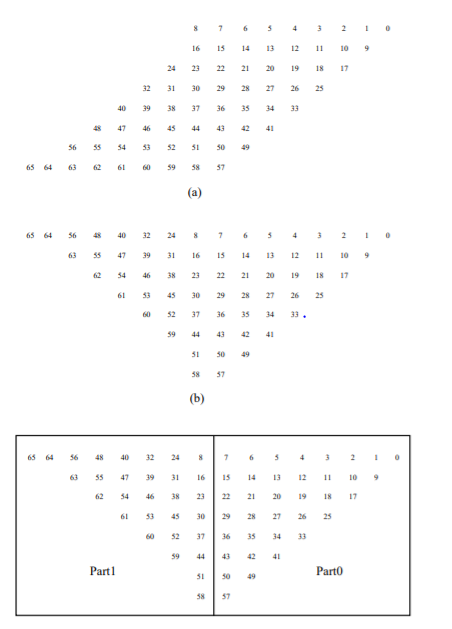
In this design, the length of operand is 8 so that the largest length originally exist is 9. As a result, the first stage should reduce 9 to 6. Then the second stage from 6 to 4, then from 4 to 3…

For the first stage the first column we need to deal with is the column 6 (count begin at 0). The length of this column is 7, and there is no carry in for the previous stage which makes it is enough for a half adder to reduce it. After applied a half adder, the length of column reduced by 1. However, the carry bit of column 6 is propagate to column 7 which makes the length of column 7 become 9. Generally, we put the carry bit to the last position so that we will not deal with the propagation bit in this stage. This technique will make the design more tractable. Since the length of column 7 is 9 now, we need to use one full adder and one half adder to make the length to 6. Then it is worth noting that there are 2 carry bits generated by this column. As a result, we need to put 2 carry bits at the end of column 8. Now, the length of column 8 become 9 as well. So, we need one full adder and one half adder to make it 6 again. The same process proceeds until the column 10 since column 10 has only 4 bits add one of the carry bits from column 9 make it has a length of 5 which is not exceeds the 6 limitation. Now we are done with the first stage.

The next stage is trying to make every column has no larger than size of 4 which requires the same technique. Again column 0 to column 3 and column 12 to column 14 do not need anything. The column 4 need a half adder; column 5 need a full adder and a half adder. Column 6 to 10 needs two full adders and column 11 needs one full adder. The idea is exactly the same which makes this structure looks good. After done with this stage, the next stage is to try to go to 2 and finally the last stage which produce the result. The general flow of this process is shown in the graph.



There can be some optimization of this adder but due to the time limitation, we did not do that. One of the approaches is to try to separate it by two parts. So that the large number of carries will not propagate through the longest column to make the calculation much easier. This process is attached below. The bit 8 is added for the symmetric reason. It is fine to delete it, but the structure will no longer be symmetric. The calculation of the partial summation is exactly the same. The only difference is the last stage need to have some overlap summation. Since the carry bits is not propagating through when calculate the sum separately. In this specific example, the first stage could contain at most 3 overlap carries that need to be added to the bit 8 ,9 and 10 for the final solution.



## Half and full adder

The design of half adder and full adder are based on different strategy just because I want to try different kind of design. The full adder is actually a Manchester adder which use the formula:

G = A\*B

P = AxorB

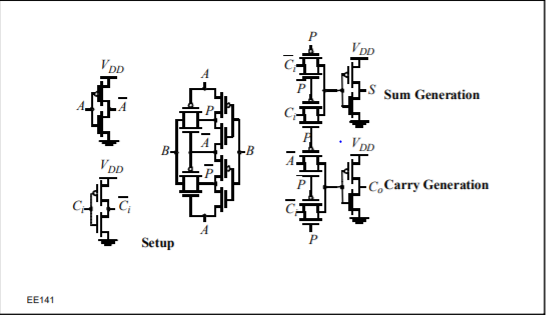
D = (A+B)’

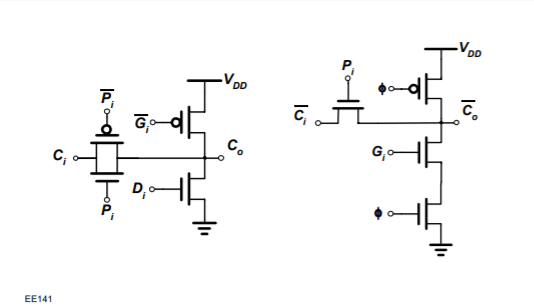
Co = G + PCi

S = P xor Ci

The implementation of single gates are based on the static CMOS logic which we know is not the best design in terms of the space complexity. The feature of this adder is to use Manchester carry chain to minimize the delay of the critical path. In this design the carry chain is static logic.

The graph attached below is the basic component design of a full adder and the propagation chain.

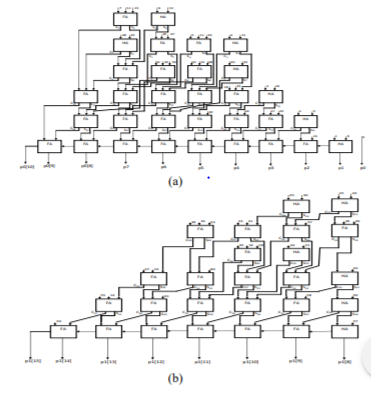




For the half adder, it is pretty straightforward. From the truth table, we can easily see the carry is just Ai\*Bi and sum is Ai xor Bi. In this design, we use transmission gate logic to implement it which will save a lot of space in terms of xor gate.

The schematic of the whole PPST design is shown below (no last stage involved). It is worth noting that this is actually not what we implement but the basic idea should not be any different.

For the final stage adder,

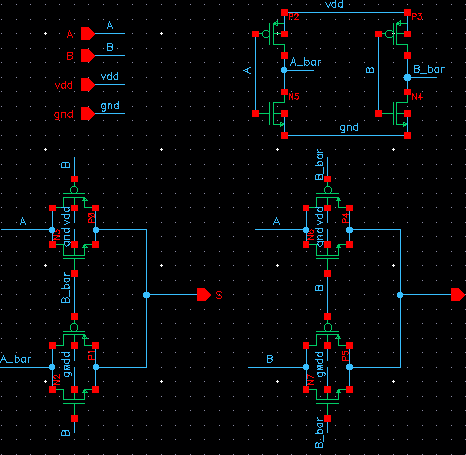


## Final stage adder

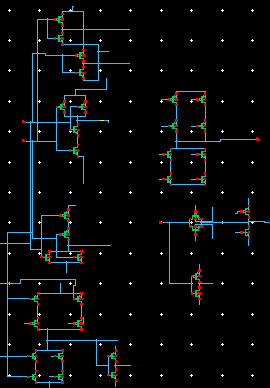
For the final stage adder, there are a bunch of method optimize the delay. In this design, due to the time limitation we do not optimize it. What team did is just use the Manchester full adder to sum the last stage of PPST together to get an answer. The performance is pretty good because of the size of this design is pretty small which is just 8\*8 design. This should be much worse as the size goes up.

# SIMULATION and RESULT

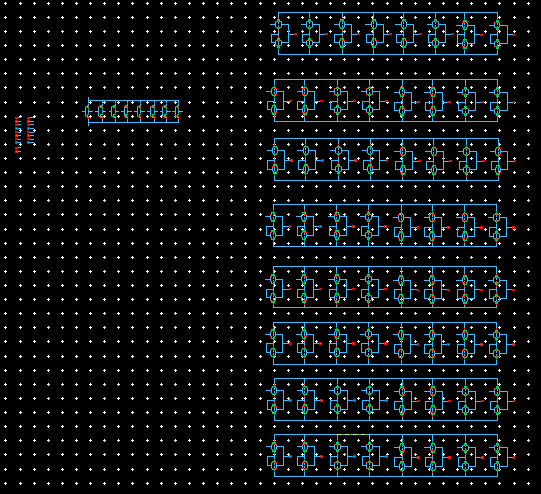
This design is all in full custom fashion which means all of the design are done in transistor level. The simulator we use is Spetre simulator. The transistor model we used is TSMC 180nm technology. The design contains only gate level optimization which we assume the optimal ratio of wp:wn = 2:1. Based on this fact, we optimize the gate for symmetrical output. It is possible to make it faster, but in this small design this performance is good enough. There is no layout provided in this design, the only thing available is the schematic for all the design. The schematic graphs are attached below.



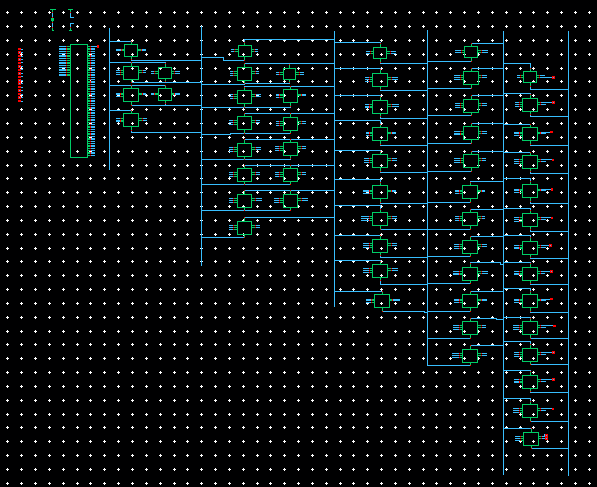
Half\_adder



Full\_adder

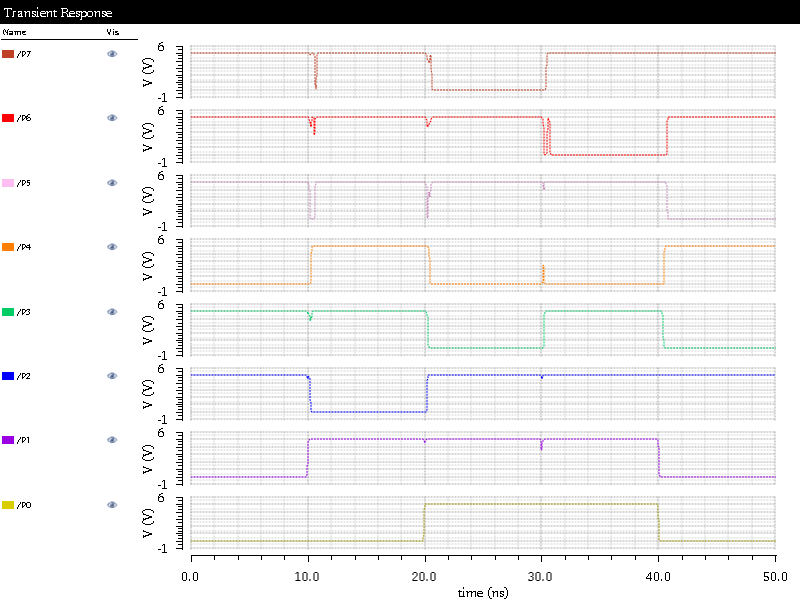


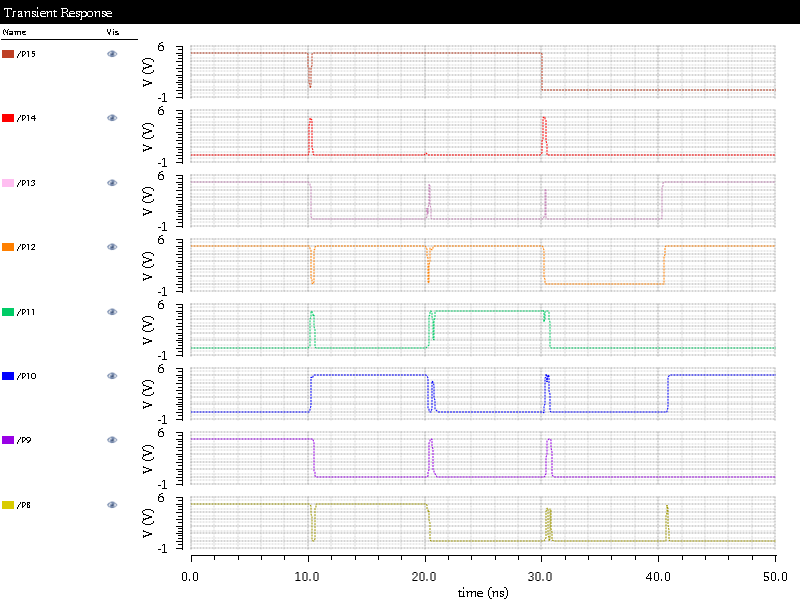
PPG\_generation

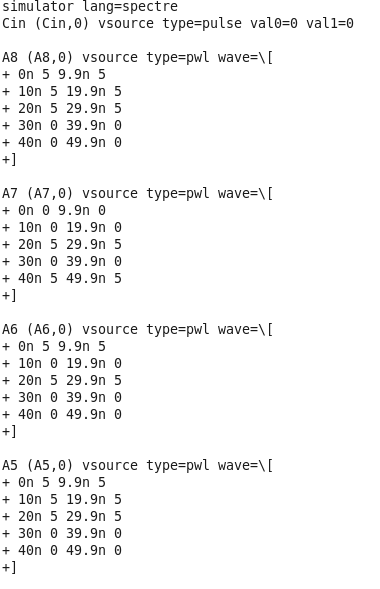


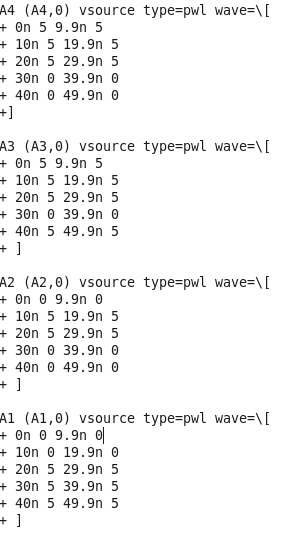
Top\_level

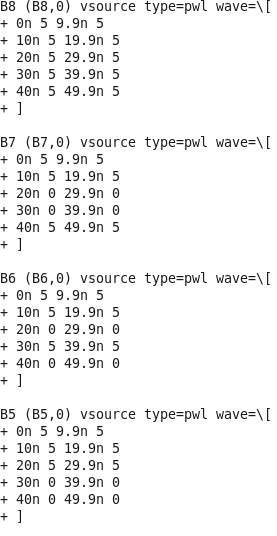
The simulation result is shown below, the result is checked against the online binary multiplication calculator which is correct. The stimulus file and the output wave form are attached. The run time is 50n sec.

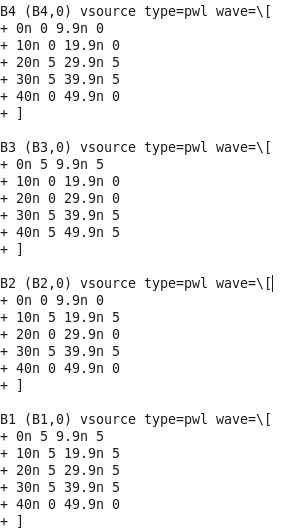






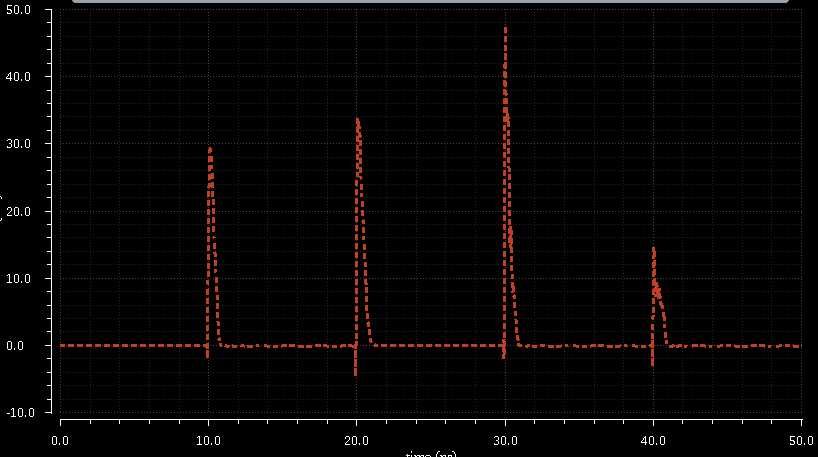


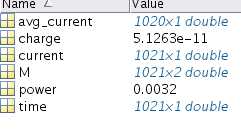




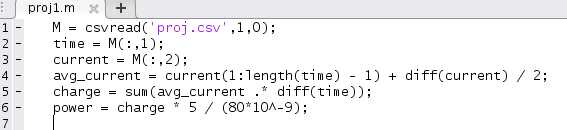
# Power analysis

The power analysis is done by measuring the total power which attach a probe to the negative pole of voltage source. This will give us a matrix of time and current value. Then we will import this matrix to the MATLAB and then calculate the power dissipation with a script. The most power dissipation is due to the dynamic transition of the transistor. The static power is low because of there is no static path from Vdd to ground. The simulation result of current is shown below.





The final result is 32mW as calculated above. The power is kind of large because that the power supply we use is 5 volts which is not necessary. The MATLAB script of calculation is shown below.

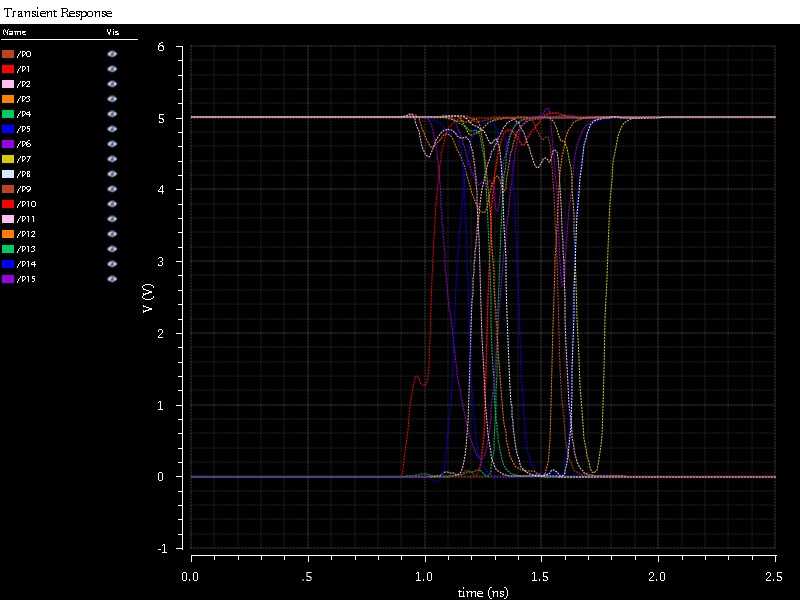


# TIMING ANALYSIS AND SUMMARY

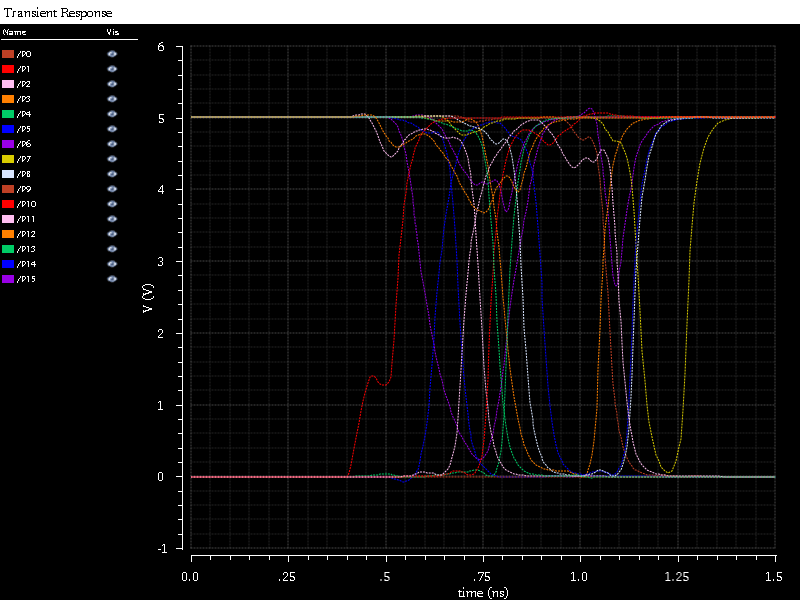
In the previous analysis, the input is 10n second for each input, the result is correct. In this section, we will set the input signal to be very short in order to find the limit of shortest time that the result is still correct.

|  |  |
| --- | --- |
| Time(duration) | Correctness |
| 10nsec | Yes |
| 5nsec | Yes |
| 1nec | Yes |
| 0.5nsec | No |

As we can see, during the 1nsec simulation, it is possible to input data at 1nsec. After measuring the delay is about 0.6 nsec. It will not be true if the input data is faster than 0.5nsec each which is 2GHz. If the data input is morethan 2GHz the result of the multiplier is not reliable anymore. It is best to keep the frequency of input below 1GHz to ensure the output’s reliability.



1 nano second



0.5 nano second

# REFERENCE

[1] B.Parhami, "Computer Arithmetic", Oxford University Press, 2000.

[2] E. E. Swartzlander, Jr. and G. Goto, "Computer arithmetic," The Computer Engineering Handbook, V. G. Oklobdzija, ed., Boca Raton, FL: CRC Press, 2002.

[3] C. S. Wallace, “A Suggestion for a Fast Multiplier,” IEEE Transactions on Electronic Computers, Vol. EC-13, pp. 14-17, 1964.

[4] Luigi Dadda, “Some Schemes for Parallel Multipliers,” Alta Frequenza, Vol. 34, pp. 349-356, August 1965.

[5] K.C. Bickerstaff, E.E. Swartzlander, M.J. Schulte, Analysis of column compression multipliers, Proceedings of 15th IEEE Symposium on Computer Arithmeitc,2001.

[6] W. J. Townsend, Earl E. Swartzlander and J.A. Abraham, “A comparison of Dadda and Wallace multiplier delays”, Advanced Signal Processing Algorithms, Architectures and Implementations XIII. Proceedings of the SPIE, vol. 5205, 2003, pages 552-560.

[7] P. R. Cappello and K Steiglitz: A VLSI layout for a pipe-lined Dadda multiplier, ACM Transactions on Computer Systems,pp. 157-174, 1983.