The Use of CMOS Integrated Circuits for Spot Counting in LDI-TOF Techniques for Disease Biomarker Recognition

A thesis submitted in partial fulfillment of the requirement
for the degree of Bachelors of Science in Physics from
The College of William and Mary

by

Dillon Patrick Roach

Accepted for ______________________________________

____
Dr. William Cooke, Advisor

____
Dr. Christopher Carone

____
Dr. Todd Averett

____
Dr. Michael Como

Williamsburg, VA
April 25, 2005

Abstract

We are working to enhance the resolution of laser desorption and ionization time of flight (LDI-TOF) techniques for disease biomarker recognition. In order to take useful data from the LDI-TOF system, we will need a device capable of taking digital inputs and effectively counting the number of inputs that are "on" once every two to four nanoseconds. For this we have investigated the use of metal-oxide-silicon integrated circuits (MOS ICs) for use in a microcircuit capable of accomplishing our goals. The purpose of this paper is to give an overview of the problems that need to be addressed when building such a summation circuit, how we have addressed those issues, and to give suggestions for how one might accomplish this task.

Table of Contents

Introduction

A team of physicists here at The College of William and Mary, in cooperation with the informatics company INCOGEN, as well as the math and biology departments at the college, are developing a new imaging detector aimed at enhancing laser desorption and ionization time of flight (LDI-TOF) techniques for disease biomarker recognition. Simplified, LDI-TOF devices work in the following manner: a substance in question is attached to a specially prepared surface and placed in the LDI-TOF device. Lasers are then used to blast the surface, hopefully releasing particles of the test substance from the surface and ionizing them. These ions are allowed to travel a distance through a charged field and eventually hit a detector. The time that an ion takes to travel the distance to the detector indicates its charge-to-mass ratio, a value that can help researchers determine the makeup of the substance in question.

The goal of our project is to significantly increase the resolution of the LDI-TOF technique. LDI-TOF devices produce large numbers of ions, and often times the detectors in these devices simply produce an analog signal where larger signals correlate to greater numbers of ions striking the detector at once. These detectors are quite susceptible to noise, which make high-resolution data difficult to analyze. To avoid this difficulty, our detector will be segmented to count multiple shots, which must then be summed quickly by our device (in a matter of two to four nanoseconds). My contribution to the project has been the investigation of the electronics systems that will eventually be capable of receiving signals from the LDI-TOF detector, summing the number of segments that are "on," and sending the results to a computer for analysis and storage. It is the purpose of this paper to discuss what has been learned about the execution of such a task, how to optimize the summing device for speed, and how this information may be used to fabricate such a device.

Background

In order to build a small, fast electronics device it is natural that we turn to the transistor-based integrated circuitry (IC), the technology used to create electronics chips like the CPUs found in computers. More specifically, we have chosen to investigate the use of the industry standard Metal-Oxide-Silicon Field Effect Transistors (MOSFET, or sometimes simply MOS transistors). Thanks to a service provided by a group called MOSIS (http://www.mosis.org/), universities can have IC prototypes fabricated at effectively no cost, making MOS ICs not only the cutting edge for speed, but incredibly affordable.

The basic component of a MOSFET chip is the transistor itself, a voltage controlled device that will pass a current between two gates depending on what voltage is applied to its control gate. There are two basic types of MOS transistors, the n-channel enhanced MOS transistor which will pass a current when a positive voltage is applied and the p-channel enhanced MOS transistor which will pass current when negative voltage is applied. As well, some transistors are built with a bias so that they will pass current as the applied voltage goes to zero. Whether pMOS or nMOS, all MOSFETs are "unipolar" devices, meaning only one charge carrier dominates the current [1]. In the case of nMOS, electrons are the charge carriers, while pMOS transistors use "holes" (or atoms that are missing an electron) as charge carriers [2]. These "holes" are slower moving than the electrons due to their larger mass-to-charge ratios, and because pMOS must build an inversion layer out of holes, they are slower to transition than nMOS transistors.

As shown in Fig. 1 the typical MOSFET has three gates: drain, source, and control. In the case of the nMOS transistor, when positive voltage is applied to the control gate, electrons in the doped region between the source and drain are attracted to towards the control gate, while positive ions are repelled. This creates an inversion layer between the source and drain, which allows a current to flow [1]. The MOSFET is symmetrical, meaning the drain and source can be interchanged with no change in function, yet the drain is conventionally the gate where output current is read.

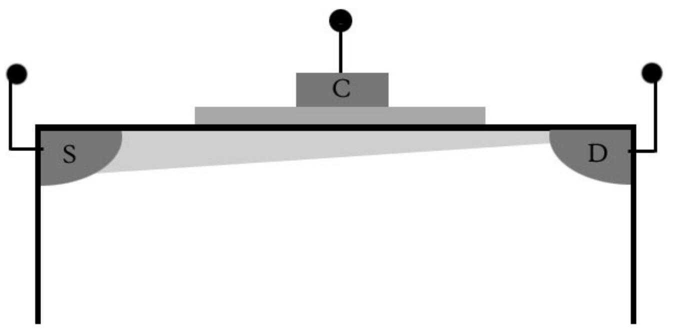

Figure 1. The typical nMOS transistor with exaggerated conduction band between source (S) and drain (D), as positive voltage is applied to control gate (C). The control gate is separated from the body of the transistor by a non-conducting oxide layer, so that no current flows from the control gate to the body. The source and drain gates are made of doped n-type material, while both gates are imbedded in a larger p-type well where the conduction and inversion layers form [3].

One creates logic circuits by connecting transistors together so that they either pass or do not pass a current, depending on inputs. For example, say one connects a pMOS to a power supply and an nMOS to ground then gives each transistor the same input. In one case the input will allow the pMOS to pass a current while turning off the nMOS, pulling the output high. And in the other case the nMOS passes a current, the pMOS turns off, and the output is pulled low by the ground. This type of configuration is how one would build an inverter and will be seen many times through this paper. The use of nMOS and pMOS transistors in the same circuit is known as complementary MOS, or CMOS, technology.

Because there are a number of different configurations that could be employed to achieve the same logical effect it is the task of a circuit designer to optimize their circuit for the intended use. We wish to optimize the response time and the repetition rate of our circuit and do not care as much for issues like power-drain or overall size.

Data and Analysis

In order to simulate our circuit and transistor designs we used the freeware program "Electric 6.08," which can be downloaded at http://www.staticfreesoft.com/. While the program lacks the ability to simulate the current and voltage effects of the transistors, it is sufficient for simulating time-delays, which is our primary concern.

Our simulations have addressed a few basic issues. First, how circuitry responds to changes in the overall fabrication size (a set value for the entire fabrication process); second, how transistors react to individual changes in scale, i.e. width and length of a given transistor; third, how to optimize the speed of an entire network of transistors; fourth, how to use basic adders to sum an arbitrary number of bits; and finally, the difficulties of using three-input adders as the basis for a summation circuit and how such problems may be solved.

A. Overall Fabrication Size

When submitting an IC design to be fabricated, a designer must specify a fabrication process that will determine the scale for all components on the chip. For example, the fabrication processes available through MOSIS at the moment have scales ranging from 0.13μm to 1.5μm. To test the effects of choosing one process over another we turned to Electric 6.08. However, we were limited by the program, because it would only simulate fabrication values of 0.3μm and 1.0μm. Fig.2 gives examples of how the time-delays of an inverter change when the fabrication process is altered. Importantly, each line has a different slope, meaning that time delays do not simply scale when altering your fabrication size; thus you must simulate your circuit at the appropriate fabrication size for correct delay information.

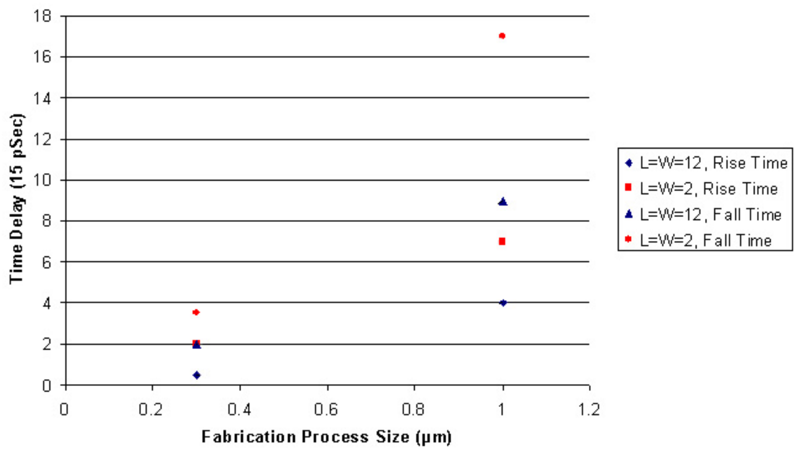

Figure 2. The set of points on the left represent time delays from a 0.3µm process. These points are connected to points displaying delay times for the equivalent circuit in a 1.0µm process on the right. The darker pair represent circuits with transistor lengths and widths of 12, while the lighter pair represent transistor lengths and widths of 2. In both cases, smaller fabrication values mean faster responses, though the slopes of each of the lines are different. When finalizing a circuit design for fabrication it should be important to simulate that design with the appropriate fabrication scale.

As in Fig.2, smaller fabrication values mean speed increases. As well, small fabrication values make for smaller overall chip sizes. We wanted to know, however, if we would duplicate the 1.0µm process time delays by scaling the length and width of transistors fabricated with a 0.3µm process to the size they would be in a 1.0µm process. We simulated an inverter again and found time delays, as seen in Table 1.

Fabrication (µm)	Transistor Size	Scale	Rise Time (15ps)	Fall Time (15ps)
1.0	12W;2L	1	**3.94**	**8.50**
0.3	24;4	2	1.31	3.80
0.3	40;7	3.33	2.63	7.26
0.3	48;8	4	3.27	**8.50**
0.3	60;10	5	**3.88**	11.1
0.3	55;8	Mixed	3.50	8.50

Table 1. *Of note: the transistor size is relative to the fabrication process*. If the only thing that fabrication size changed was the size of the transistor length and width, then the 0.3µm fabrication scaled to 3.33 should have returned the same rise and fall times as the 1.0µm, because the transistors would be the same size physically. However, we don't see similar time-delays until we scale the 0.3µm transistors up to four or five times in scale.

If fabrication scale simply affected the size of the transistors, the times from the first and third row of Table 1 should be equal, because they would be identical circuits. However, in order to see delay times close to those found in the 1.0μm process we need to enlarge the 0.3μm process transistors by about four or five times, rather than the 3.33 that would mean equivalently sized transistors. These numbers seem to indicate that there is more to fabrication size than simply the size of the transistors. Perhaps this is due to the wiring tracks and other connection features on the chips, which will also be smaller. Therefore we should design our circuits using relative units, and then choose the appropriate fabrication size to yield a balance between speed and the cost of producing a chip at high resolutions outside of MOSIS.

B. Individual Transistors

After choosing the fabrication size for a circuit, the characteristics of a given transistor that the designer will usually alter are the length and width of the transistor. The length of the transistor is the physical length of the channel between the drain and the source, while the width is the broadness of that channel. To test the effects of altering either size characteristic of the transistors we created the simplest logic circuit, an inverter (using one pMOS and one nMOS). We then simulated the time it took for the circuit to respond to the input switching from 1 to 0, and repeated the simulation using a number of different size values for both of the transistors. As Fig.3 shows, an expanding transistor width will decrease the transistor's delay time as the width gives the transistor more charge carriers with which to form a current, however the benefits of enlarging the width of the transistor trail off after a certain point. For speed purposes, the width of a transistor should be at least the size of its length.

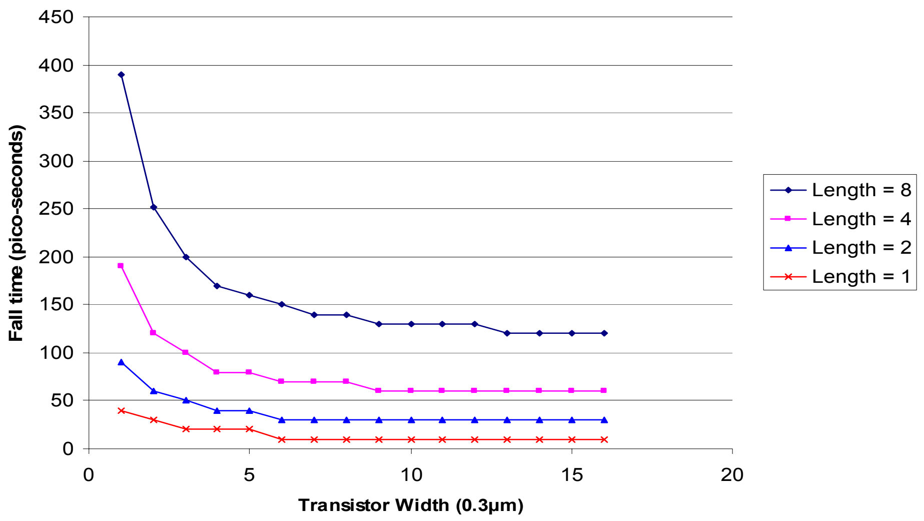

Figure 3. Effects of altering transistor width on inverter response to a falling input. As width of the transistor increases, the response time of the circuit quickly falls and tapers to a near constant.

Fig.4 demonstrates the effects of altering the transistor length. The points on the graph are strongly linear and suggest that the smaller the transistor length can be the better. This makes physical sense, because the length of the transistor is the distance that the charge carriers must travel. As well, the saturation of the speed due to altering the transistor width makes sense, because the carriers still feel effects of the transistor length and must still travel the distance of the channel.

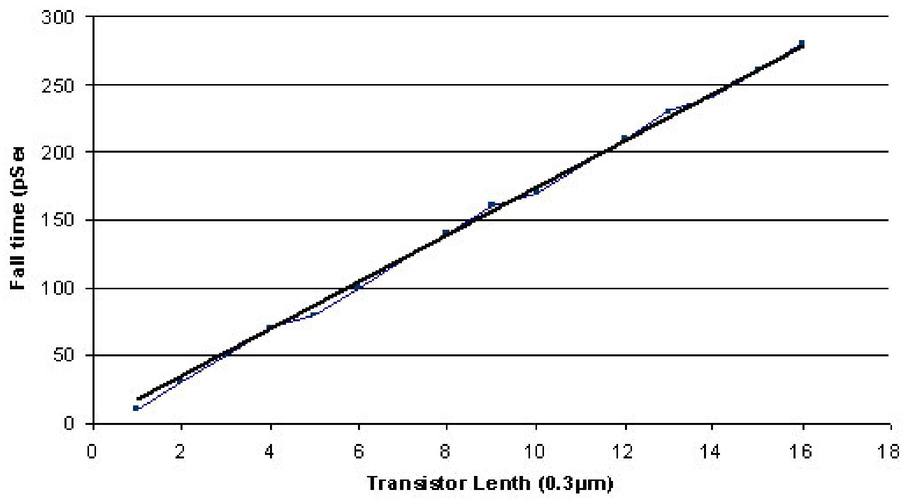

Figure 4. Effects of altering transistor length on inverter response time to falling input. As length goes up, so does response time; the effect over the values we tested is linear.

Fig.3 and Fig.4 indicate that we can expect to see the fastest circuit responses by using transistors with large widths and small lengths. However, since the time saved by increasing transistor width diminishes at higher values, we should choose width values that are fast "enough," simply because larger transistors mean larger chip sizes and greater power consumption.

C. Logic Gates

i. Pseudo-nMOS Technique

As was mentioned in the background section of this paper, pMOS transistors use different charge carriers than nMOS transistors, ones that happen to travel significantly slower than the electrons used in the nMOS transistors. This, in part, lends to the fact that pMOS transistors are significantly slower than their nMOS counterparts. In fact, because pMOS transistors are enough slower than the nMOS transistors at building up a conduction band, circuit speeds can be greatly increased by decreasing the number of pMOS transistors in the circuit. One method for decreasing delays due to pMOS speed, called pseudo-nMOS, involves tying the pMOS to ground so that it is always in the "on" mode, essentially making it a resistor, and creating a stronger "pull down" network of nMOS transistors that can selectively pull the output low. This way all dynamic inputs to the pMOS are cut off, so that the transistor does not need to make any transitions; and because the pMOS can only supply so much current, the circuit is still able to selectively pull its output to ground using only nMOS for logic. We simulated this technique using an AND2 circuit and found that the pseudo-nMOS technique shaved off a good portion of the time delay. Really, the only difference between the two circuits (shown in Fig.5) is that the pseudo-nMOS version has one significant pMOS transistor tied to ground. Yet, Fig.6 graphs the delay times for each of the circuits and we see that the pseudo-nMOS circuit is 140 picoseconds faster regardless of transistor width. Therefore we should attempt to use the pseudo-nMOS technique whenever applicable.

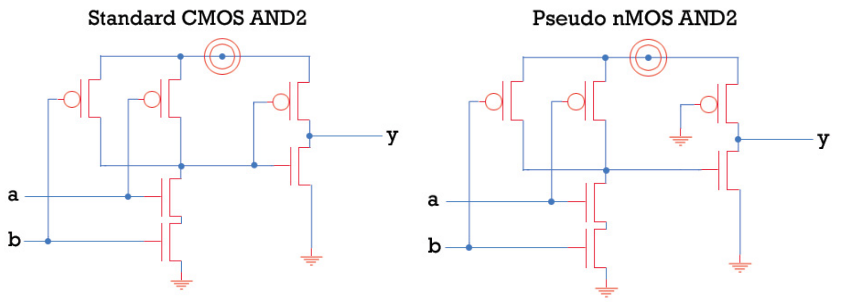

Figure 5. On the left is the standard CMOS 2-input AND circuit, while the right has one pMOS tied to ground, in order to use the "pseudo-nMOS" technique. In both figures pMOS transistors are represented by the red figures with circles, while the nMOS transistors lock circles. The blue lines represent wiring, "y" is the output, and "a" and "b" are both inputs.

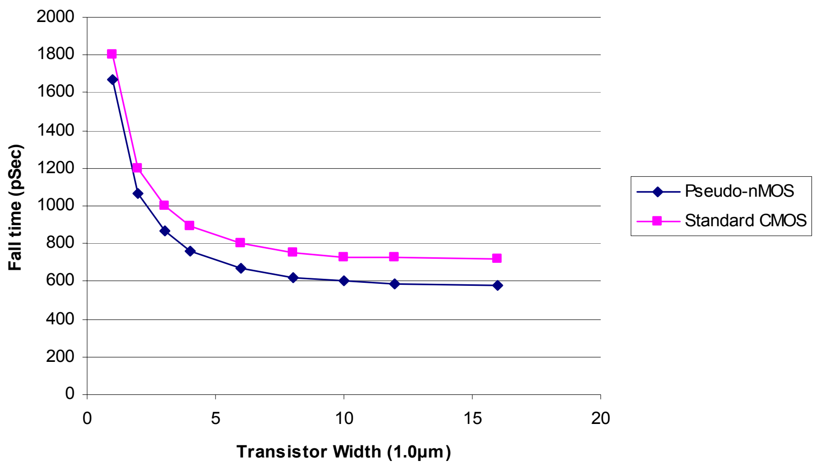

Figure 6. While the pseudo-nMOS technique decreases the time delay significantly, it only seems to save the circuit a set amount of time (in this case about 140 picoseconds), rather than a percentage of the total.

ii. Dynamic CMOS Technique

The pseudo-nMOS technique is helpful for decreasing a fixed amount of timing delay, yet it is not as effective as we would like. However, there is another method, called dynamic CMOS, which removes all but one pMOS transistor in a circuit that is then clocked. While the clock is low the nMOS network is cut off from the ground and the pMOS transistor pulls the output high. Then, when the clock goes high, the pMOS turns off and we build a network of nMOS logic to selectively pull the signal low on evaluation. We see much faster response times with this method, with the introduction of a clock as a penalty. When applying this technique to the Full Adder (a circuit that adds three inputs together for a 2-bit digital output, see Fig.7) we see a 36% decrease in time delays over the pseudo-nMOS version of the circuit (160ps versus 250ps, when simulated at 0.3µm; max clock cycles were 200ps and 500ps, respectively), though the dynamic CMOS circuit does also utilize the pseudo-nMOS technique for its inverters.

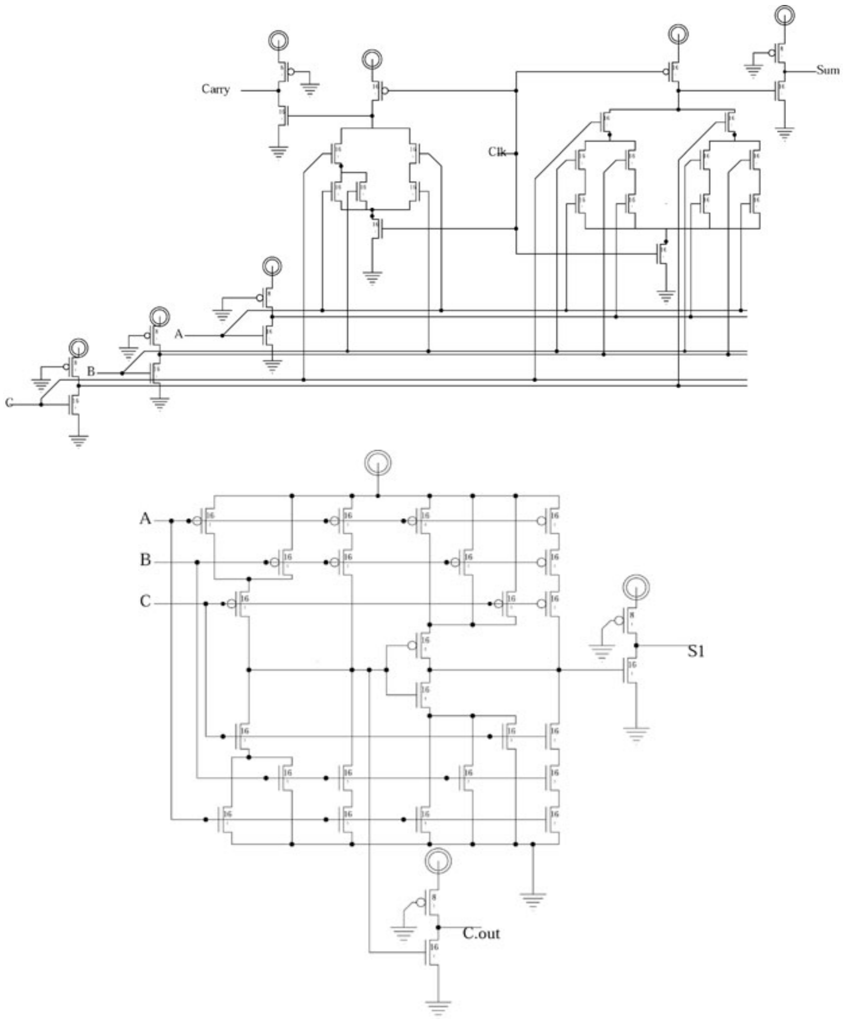

Figure 7. Above are two different circuits that essentially do the same thing. Both take inputs A, B, and C, and add them together in binary to outputs Sum and Carry. The top circuit, however, introduces a clock and uses dynamic CMOS logic to achieve its goal, while the circuit on the bottom uses standard CMOS techniques, with pseudo-nMOS inverters. Simulated delays for the two circuits were 160 picoseconds and 250 picoseconds, respectively. However, while the bottom logic operates a full cycle in roughly twice its rise time give above, the dynamic CMOS adder can operate a full cycle in roughly 200 picoseconds.

So, not only is it advisable to use pseudo-nMOS techniques wherever possible, it is also very worthwhile to attempt to build our circuitry out of mostly nMOS transistors and to introduce a clock in the style of dynamic CMOS.

iii. AND vs. OR

When designing circuits using dynamic CMOS logic the nMOS pull-down network is simply a series of ANDs and ORs connected together. nMOS networks in series function as AND operators, while nMOS networks in parallel function as OR operators. Since desired logic may be accomplished using different combinations of AND, OR, and NOT operators, it is important to know the time delays involved in using an AND operation in place of an OR (choosing transistors in series over parallel). To test this we built a simple dynamic CMOS configuration (Fig.8) and simulated time delays for increasing numbers of nMOS transistors in series and then in parallel.

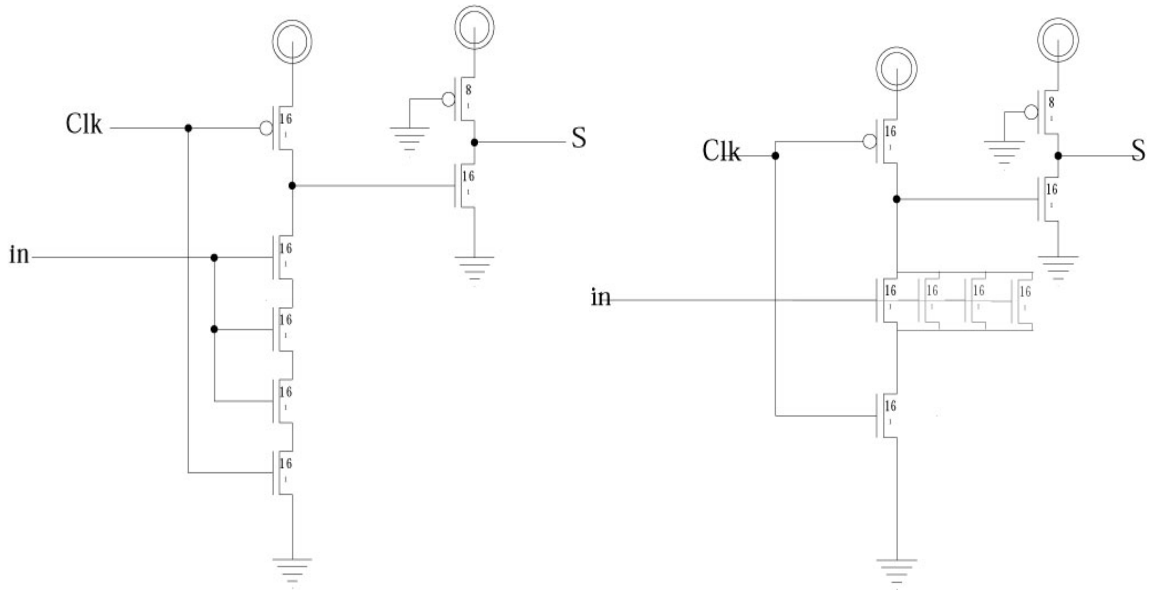

Figure 8. The basic dynamic CMOS configurations used to test the time delays due to nMOS transistors in series (left) and parallel (right).

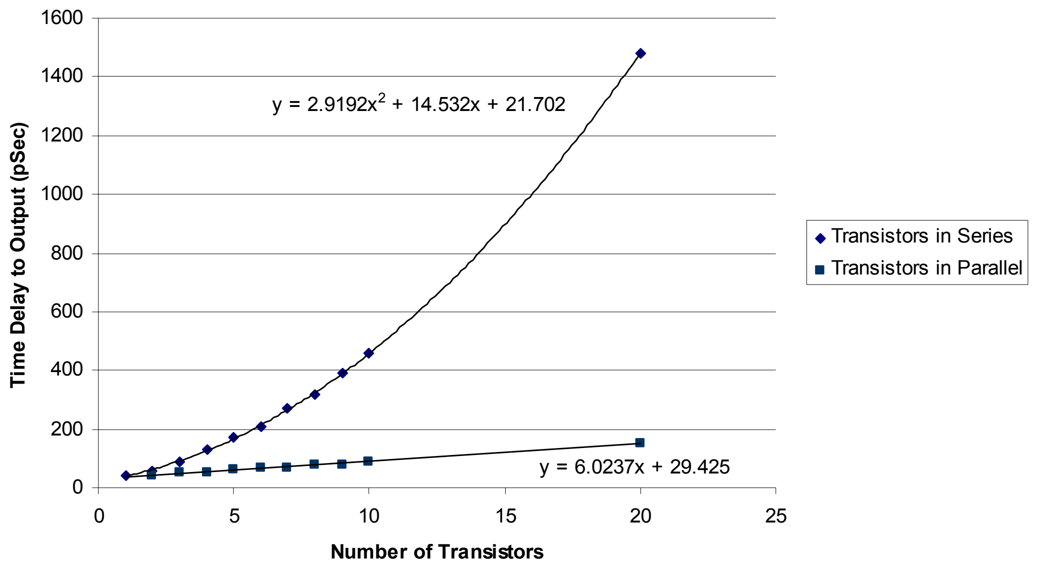

Figure 9. Transistors in parallel increase total delay time linearly and gradually as the number of transistors increases. However, the total delay time due to transistors in series is quadratic as transistors are added.

Fig.9 shows the effects of using transistors in parallel, as opposed to transistors in series. While parallel transistors add delay time slowly, transistors in series fit a strongly quadratic curve. The series transistors start taking so long, because transistors will conduct current in either direction and before the low signal can propagate up from ground the high signal from the pMOS is still trying to pull each of the transistors high. Right as the clock turns on the ground begins pulling transistors low, and the high signal continues to pull transistors high. The two signals will likely meet somewhere in the middle of the nMOS network and the ground must begin pulling those transistors low that were already high, doing effectively twice the work on them, as it has to pull them from high to neutral and then to low. From this we learn that in order to optimize our logic speed we should refrain from using many-input AND operations and rely more heavily on OR operators, transistors in parallel.

D. How to add Inputs using Full Adders

As shown in Fig.7 we already have the design for a circuit that can add three inputs together for a digital output of two bits. However, it certainly won't be the case that our LDI-TOF device will only need three inputs added together, so we need to develop a way to add together an arbitrary number of bits using similar techniques. It has been customary in circuit design to simply cascade the desired number of three-input adders like the ones in Fig.7 in order to add two binary numbers together. However, our process is slightly more complex in that we are not given binary numbers, but a large number of binary bits that we need to convert to binary numbers. In this way an input of 10110 would need to be converted to an output of 00011 (binary 3) since there are three 1s in the input (likewise, 11111 would become 00101, or binary 5). As we will show, it is possible to accomplish this conversion task with the use of three-input adders, but their use comes with a number of other considerations.

In order to convert a number of inputs into a digital output using three-input adders there is only one basic concept that needs to be understood. It is simply that the bits that the adder receives may represent values other than one. A normal adder receiving inputs and then outputs bits representing a two and a one (10 and 01, respectively). However, if we added bits representing twos together, like the red adder in Fig.10, the first bit would represent a two and the second bit would represent a four. Fig.10 shows the first step towards adding nine inputs together. In order to come to a final digital output we simply add outputs together further, as illustrated in Fig.11 using 63 inputs, until we are at a point where the bits can be taken together to represent a binary number or two binary numbers that may be added together in a conventional fashion.

Fig.11 demonstrates that this process is applicable to much larger numbers, and in fact the full adders can reduce the sixty-three inputs down to ten outputs in only six cycles.

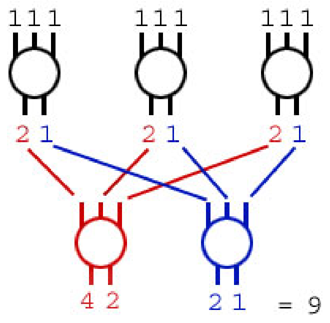

Figure 10. In this figure each circle represents a three-input adder. The nine inputs at the top are added together to give six outputs in the middle, which are then added together once more to give four outputs. While the outputs of the blue adder are equivalent to those of the first set of adders, the red adder is adding together twos, and so its possible output values are 0, 2, 4, or 2 and 4 equal 6.

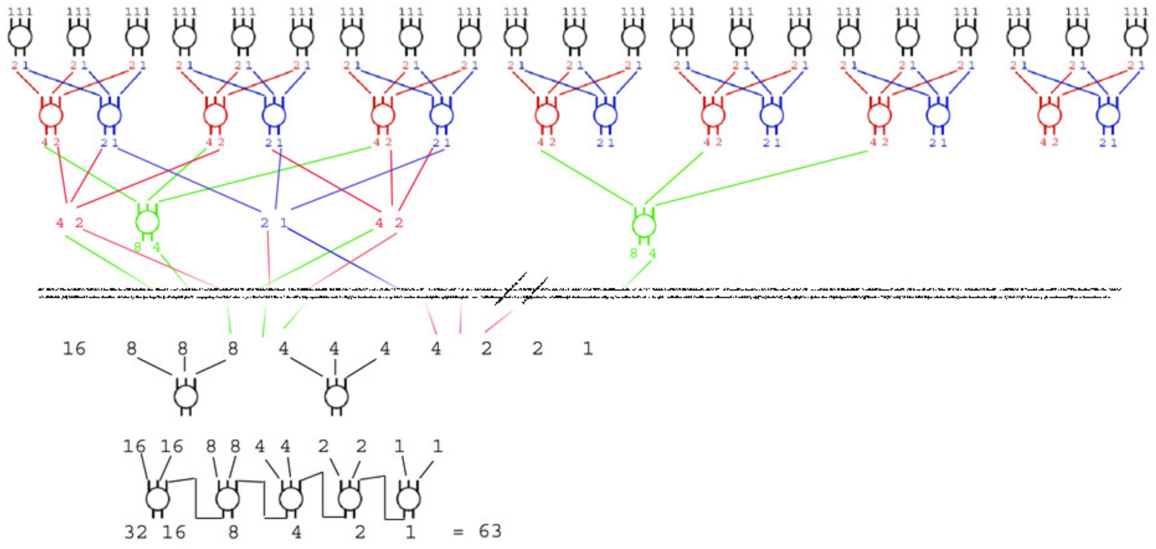

Figure 11. Steps are left out in the picture above, favoring small detail to the full scheme. Again, each circle represents a three-input adder. By taking the outputs of each row of adders and using them as inputs for the next, the input information may be reduced down to a manageable few bits, which may then be added together with a series of three-input adders in the same fashion as one would when adding two binary numbers. This process is actually less complex than it may seem, the two lines in the center of the figure represent the skipping of only two rows of full adders. The information is quickly narrowed down to two sets of 16, 8, 4, 2, and 1, which can then be added together using the cascaded full adders.

The next thing we wish to address is simply this: why three inputs per adder? Why not two, or seven, or any other number for that matter? We are using the three-input adder as the basic component of our summation circuit for a few reasons. First, the two input adder, or half-adder, gives its output in two bits as well; the output can be 00, 01, or 10. The fact that it takes two inputs and gives two outputs means that there is very little reduction of information going on and so a great many more steps are required to come to a reasonable output than would be required with the three-input adder, which takes three inputs and reduces that information down to two bits. The fact that the three-input adder, or full adder, uses the full range of possible outputs (00, 01, 10, and 11) makes adding the output of many adders together a much simpler process and requires fewer steps, because it does not require the many "or" operators that become necessary when adding outputs from the half adders. Fig.12 demonstrates this using a small number of bits.

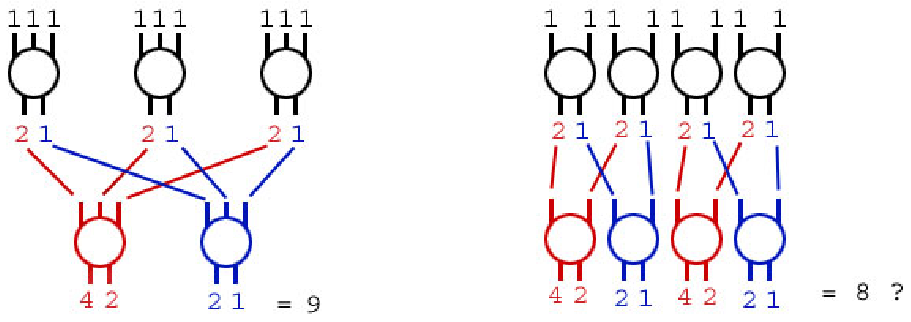

Figure 12. Above are two schemes for adding inputs together using 3-input adders (left) and 2-input adders (right). In each the output of the first set of adders is fed into the second set of adders, with each circle representing an adder circuit; the second row would run after receiving input from the first. The first outputs representing 2s are added together again in the red adders that output bits representing 2 or 4 (so, in the example of the 3-input adder, the red adder can output, 0, 2, 4, or 2 and 4 = 6).

In Fig.12 the output from the full adders clearly adds to the number of original inputs, nine, while the half adders do little to simplify information and would need a further three steps to come to a digital output. The bits from the full adders only need two further operations performed on them at this point. Though this particular addition of eight bits using half adders can be solved rather quickly with a pair of "or" circuits towards the end of the adding process, when much larger numbers of bits are involved the amount and placement of these “or” circuits becomes quite a hassle. Because the “or” circuit can operate so much faster than the adders, when you use them in a circuit you must either have a different clock signal to address them, or you are wasting a good deal of time with them waiting an entire clock cycle tailored to the speed of the adder. Two input adders can be used effectively to build a summation circuit, however we have chosen the full adder for its ease of use and a simpler clock signal.

So if the key usefulness of the full adder is that it can use all of the possible outputs and that it reduces its information from three bits to two, why not look at the next number that works similarly: seven? A seven input adder would reduce information from seven inputs down to three, and would also utilize every three bit combination in its possible output (000, 001, 010, 011, 100, 101, 110, and 111). To test this possibility we made a seven input adder based on the same dynamic CMOS technique as the full adder. We tested its fastest bit, the third output, against the speed of the slowest of the full adder’s outputs. The figures in Fig.13 show the output response to summing inputs for a cycle, and then waiting an appropriate amount of time (left) before triggering the clock once more, compared to a shorter amount of time (right). The black areas represent undefined output values. If the speed of the seven input adder was comparable to the speed of the full adder (that is, if the seven input adder could function at a speed roughly 7/3rds the speed of the full adder) we might consider building summation circuits off of adders with other numbers than three. However, as can be seen in the figure, the seven input adder only functions equivalently well in the time that it takes to get a rising output. The seven input adder takes miserably long to return to a state where new input can be taken once more, making it nearly useless when compared to the 200 picoseconds needed to run a full cycle with the full adder.

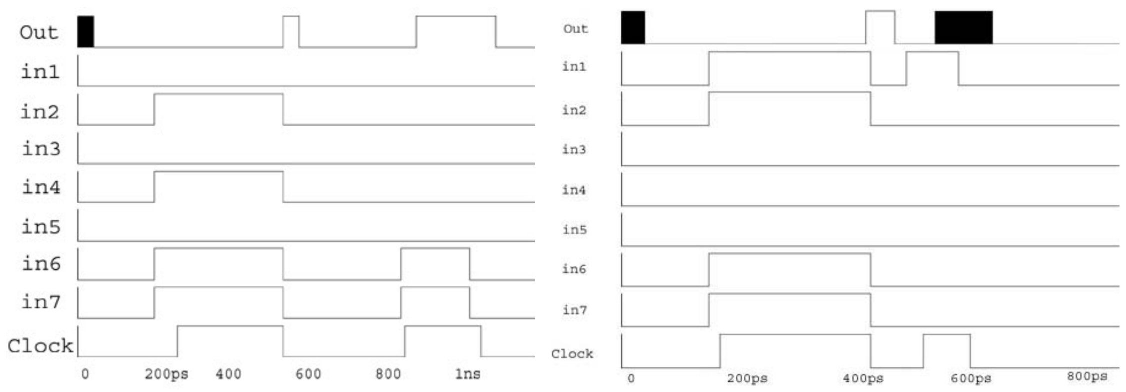

Figure 13. Above are two graphs of simulated inputs (output on top) to the seven input adder. On the right a proper amount of time is given to the circuit after inputs have gone low before triggering the clock high again; but on the right is what happens when inputs and the clock are triggered too soon. While only one input is sent high after the first reading, it is enough to send the output into an undefined state, rendering the circuit useless until it is reset again and given a full 250+ picoseconds time with no inputs.

It is our opinion, then, that the three-input adder is by far the best choice as a basic component of our summation circuit. However, the three-input adder does come with its own issues that we wish to address now.

E. Difficulties with Full Adders

Problematically, the full adder requires constant inputs while operating, and it is a clocked circuit, which means we are potentially missing information while we are operating the circuit. As well, the clock timing is very important to the reliability of the circuit and we have identified a problem that can arise if the pMOS pull up transistors are not adequately strong.

i. pMOS Width Requirement

In terms of the full adder itself, we noticed a problem that could arise when two inputs were triggered with the clock simultaneously. If the pMOS acting as the pull-up network for the sum bit was not wide enough, the output at the sum bit would become undefined on clock triggering and would not return to normal until the clock had gone low once more. Fig.14 gives an example of this output; the black region being the error.

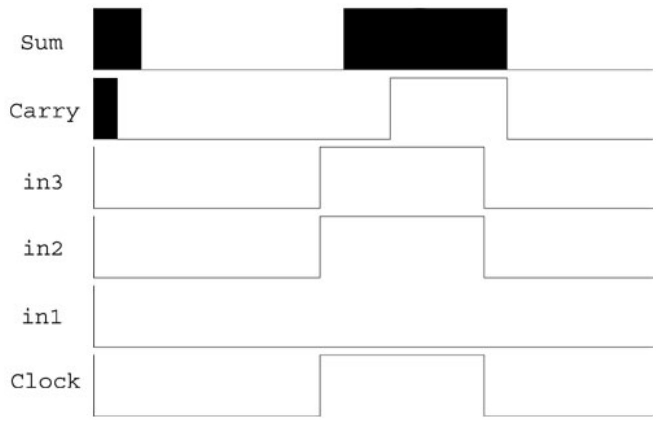

Figure 14. As we see above, without the adjustments proposed below the full adder can be fail to give proper output when inputs 2 and 3 go high (B and C in some figures).

Fig.15 illustrates our solution to this minor problem. So long as the circled pMOS transistor has a length of at least 34, in this particular arrangement, the circuit runs smoothly. When designing the full adder it is important to test the entire range of input possibilities and to ensure that this pMOS, in particular, is large enough that the error in Fig.14 does not occur.

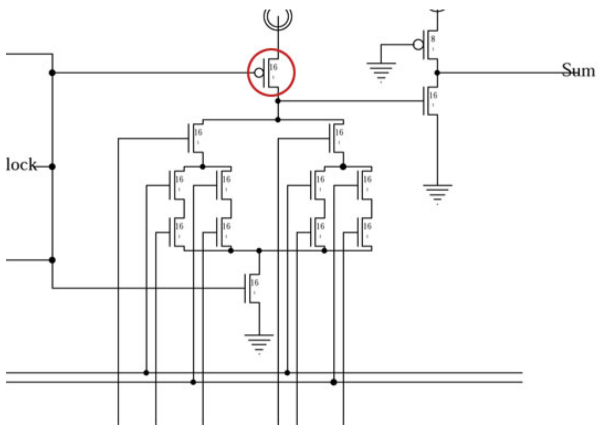

Figure 15. The adjustment is actually quite simple: you must make sure that the pMOS pulling the Sum network high is large enough that the pull-down network of nMOS transistors does not throw the output into an undefined state. In this case, so long as the pMOS is larger than 35 units wide, as opposed to the 16 of nearly all other transistors in the circuit.

ii. Down Time

Since our circuit is controlled by a clock it is natural that there should be times while the circuit is processing information that it may lose track of the input. To remedy this problem, we have developed a circuit that can account for any input information that might be missed by the full adder while it is processing. The circuit is a slightly modified latch and acts as an integrating device. That is, if the input remains low, the output of the circuit remains low as well; yet if the input goes high for even a moment (at least 10 to 20 picoseconds) then the output of the circuit will stay high until we send a short pulse signal to the "reset" input, which could act like a clock signal with a very short "on" time and would be synchronous with the clock. This integration circuit allows us to use a clock to run the full adder, while at the same time continuing to take input information and can be seen in Fig.16. Since we expect to see inputs remaining high for as long as two or three nanoseconds, the small amount of time needed to reset the integrating device (less than 0.1 nanoseconds) is essentially negligible in terms of data loss.

Under normal conditions, with both reset and input low, the circuit will keep the wiring between the top transistor and the far right transistor high. When an input is received it allows the transistor on the far right to pass the high signal to the output. The output will remain high until the current is dissipated by another circuit, or the reset is pulsed. When the reset is pulsed it ignores the input momentarily and forces the output to return low, so that it can begin waiting for a high input once more. Beyond our original design we replaced nearly all pMOS transistors with nMOS transistors, simply inverting the input to the nMOS transistor so that they worked logically like pMOS, only faster.

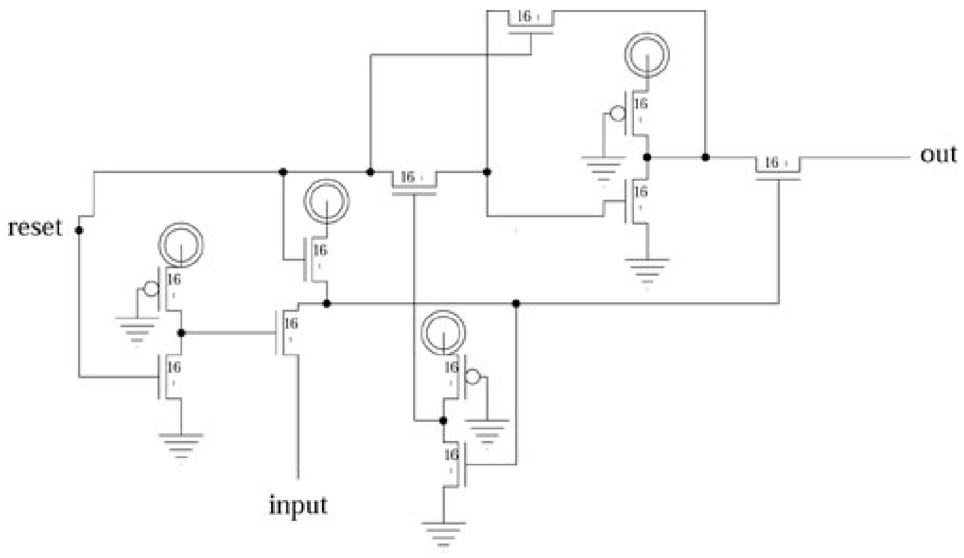

Figure 16. This is a circuit which can retain the input history long enough that our summation circuit can; if we scale the width of each of the pMOS transistors in the circuit to 20. The circuit is capable of reacting to inputs as quickly as 20 picoseconds after the reset has been pulsed, so that all but 20 picoseconds of the input history can be accounted for.

iii. Dynamic Input

While the integration circuit acts to “remember” the input history, effectively recording when inputs have gone high, it also has an output that can switch from low to high randomly. As mentioned above, the full adder requires constant inputs while its clock is high or problems can occur, as illustrated in Fig.17. The figure shows only two inputs having been high and yet both the sum and carry outputs have gone high as well, indicating that three inputs have been received. If inputs change while the clock is high, the circuit output can be unreliable and so the circuit requires something between the integrator and the full adder to hold the adder input constant while the clock is high.

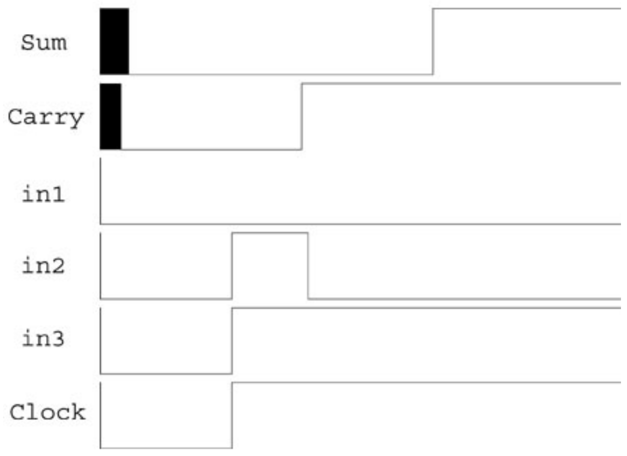

Figure 17. Because the full adder is based on the dynamic CMOS design, the circuit selectively pulls the output high (low, before inversion), but is incapable of allowing the output to return to its previous low state until the clock has gone low once more as well. The effect of this is illustrated above; if inputs are not constant while the circuit is operating it is easy to "trick" the circuit into giving the wrong sum. In this case, only two inputs have gone high, yet because one changed both the sum and the carry bits have been activated, giving the impression that actually three inputs had been received.

To address this issue, we simply insert a latch between our integration circuit and the full adder. The latch is different than the integrator, a modified latch, because it will freeze its output to a single value when the clock has gone high. When the adder has its clock low, we allow the latch to continually take input, but right before the full adder begins its processing, we trigger the latch and force the latch output to remain constant until the adder has finished its process. Because of this, integrator remains free to continue taking input information, without worry of interrupting the adder.

There are different ways that we can construct the latch, and two such examples are given in the Fig.18. While the latch based on two simple inverters is slightly faster than the second design (50 picoseconds rise time compared to 70 picoseconds), the problem with it is that it quickly dissipates the integration circuit's output to zero, rendering it useless. So, while the second design is slightly slower to respond, we are using it in our circuit because it does not interfere with the integration circuit's function.

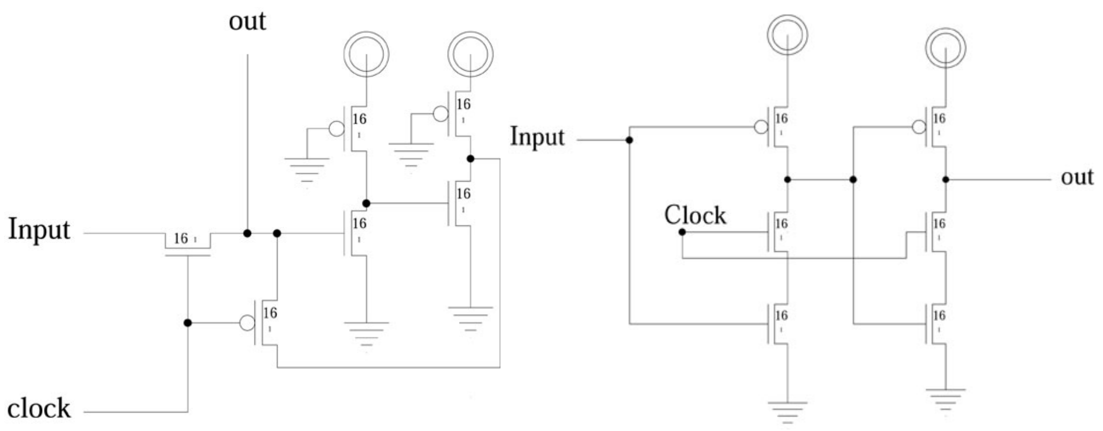

Figure 18. On the left is a very simple latch based on two inverters, however it is not functional alongside the integration circuit and so the latch on the right was chosen instead. The design on the right is slightly slower, yet will not interfere with the function of the other circuits.

The entire buffered full adder would look like Fig.19; each of the sub-circuit sections has been highlighted and labeled, in order to show the functional sections of the circuit. Then Fig.20 gives an example of the buffered full adder operation (the clock has been inverted between the latch and the full adder to delay its arrival by a short amount of time). After starting with all inputs low and pulsing the reset (far left wire) we simulate two inputs being detected, waiting and triggering the clock to demonstrate its ability to effectively "remember" the input history until the clock is triggered. The simulation then turns the adder off, pulses the reset line once more, then receives a third input and correctly sends only the Sum output high, demonstrating its ability to reset operation and correctly begin counting anew. It should also be noted that the integrator circuitry is only necessary for the full adders taking their information directly from an outside source; for the adders receiving input from other adders in the circuit the latches are sufficient for proper operation.

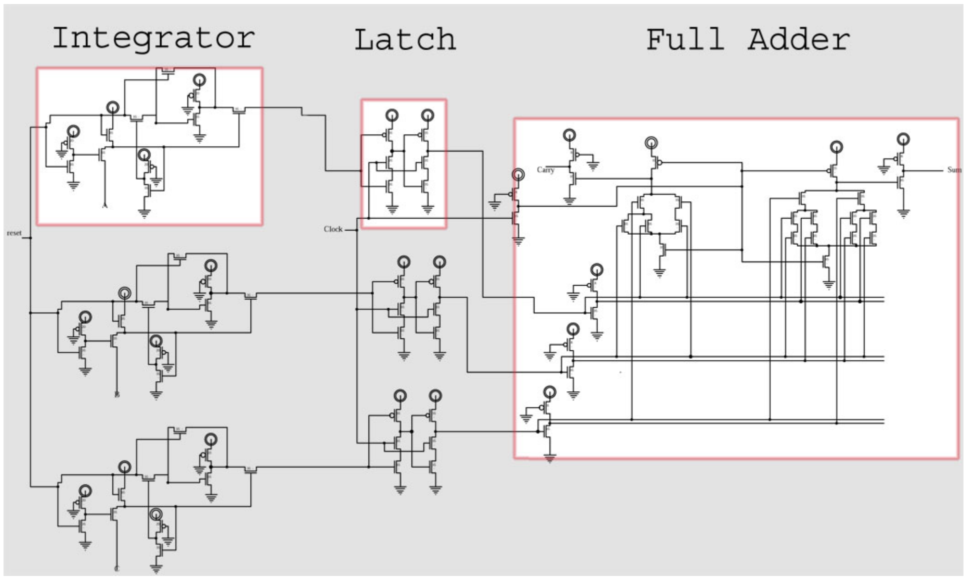

Figure 19. Here we see the full adder properly buffered and ready to take inputs. In the full-scale circuitry this buffering would only be necessary for the very first adders to receive inputs, after that it is sufficient to have the full adder with latches; the integrator can be dropped.

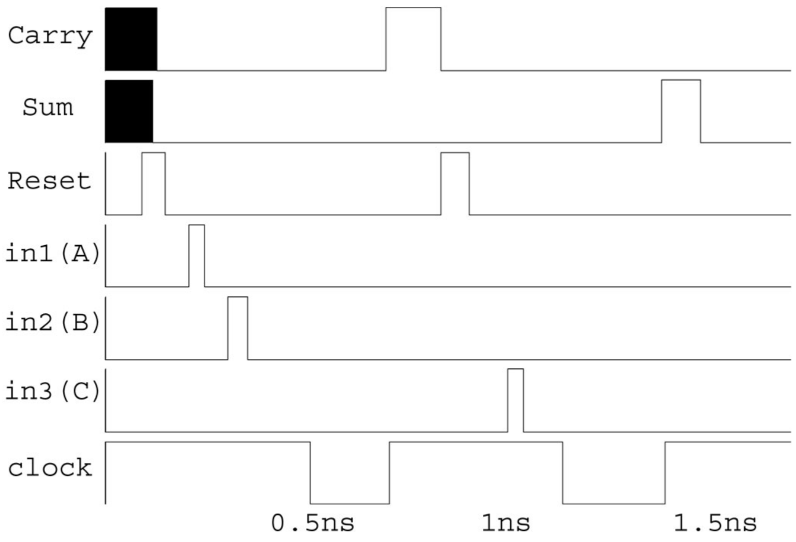

Figure 20. This figure gives an example of the buffered full adder operation (with inverted clock). After pulsing the reset to initialize the circuit, A and B are pulsed, then the clock is flipped to demonstrate its ability to effectively "remember" the input history until the clock is triggered. As the clock goes high once more, the reset line is pulsed once more, readying the circuit to begin counting anew. Finally the circuit receives a third input from C and correctly sends only Sum high when the clock goes low again. *Of note: though the output seems to follow the clocks rising edge, it is actually just delayed from the falling edge.*

While Fig.20 shows the ability of the circuit to function properly, it does not address the overall speed of the circuit. It shows two complete cycles taking place in under 1.5ns, which is very promising, yet does not imply how the circuit will slow when connected to a vast number of further networks, as will be necessary to finally reach an output.

iv. Clock Timing

As mentioned earlier, the full adder requires proper clock timing in terms of how long the clock must be on and how long it must be off before going high once more. Fig.21 shows two examples of the problems that can occur if clock timing is not set up properly. The left plot demonstrates what can happen if the clock is not left on long enough; the sum does not have time to rise and never registers. The right plot demonstrates what can happen if the clock pulses come too closely to each other; in this case, the sum was not given enough time over the entire cycle and ended up going into an undefined state until the next cycle.

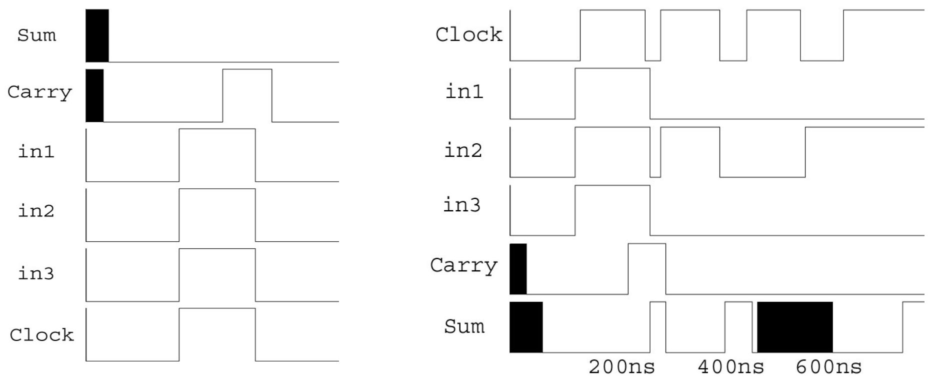

Figure 21. In the left example the clock is not on sufficiently long for the Sum (slower than the carry) to register anything, even though all three inputs are high and sum should have gone high as well. The right example shows the clock with an overall cycle that is too short, which puts the Sum into an undefined state.

In order to avoid such errors, the fastest our unloaded full adder can be run is with a cycle of 200 picoseconds (170ps high, and 30ps low), as is demonstrated in Fig.22. These numbers will change significantly when the full adder is placed in the context of the entire summation circuit, but these figures give a reference point from which to proceed when investigating the overall circuit.

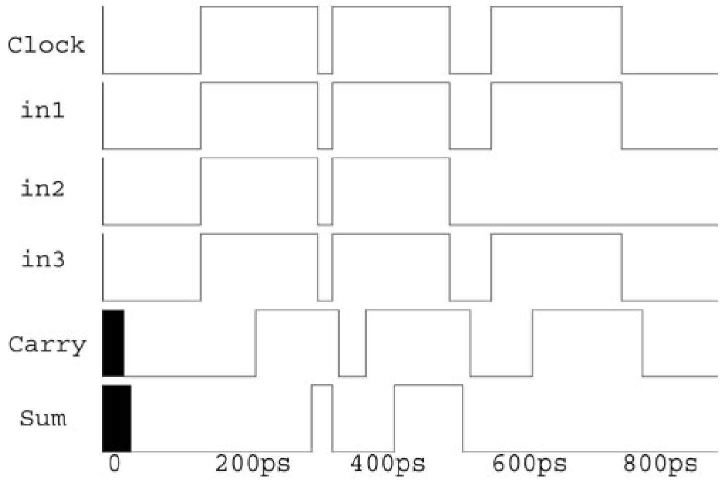

Figure 22. This is the fastest clock cycle that the full adder can handle and still produce reliable results. Of note, it is important that all of the inputs rise and fall with the clock, or timing will need to be altered.

Because the effects of loading this circuit with the vast number of networks needed to get to an output (as indicated in Fig.11) could not be estimated, in order to get an absolute best operating speed for this circuit the entire summation device must be simulated. This would be a great place for another to pick up this research.

Conclusions

In order to aid in the enhancement of LDI-TOF techniques this project has aimed to test the ability of CMOS technology to provide us with a nanosecond-scale digital summing device. Our results are promising and lay the groundwork needed to finally tackle the layout and fabrication of this device. We have learned that we want large transistor widths, short transistor lengths, a small-scale fabrication process, and we have learned a number of useful techniques for optimizing transistor networks, such as dynamic CMOS and pseudo-nMOS. We have learned that the best basic component for our summation circuit is the three-input adder, when compared to two and seven input competitors. As well, we now know that the greatest limitation on the circuit's speed is not how quickly the entire circuit can operate, but simply the repetition rate of the first set of full adders, since the circuit may operate like a shift-register, passing data to the next set of adders with each cycle and operating on new data directly after. We have designed all parts needed to fabricate a summation circuit and have tested the speed limitations, as well as operational issues, of those components. We have shown that an unloaded full adder can operate a full cycle in 200 picoseconds, and that a fully buffered full adder can run two full cycles in under 1.5 nanoseconds.

We were limited in a few ways, however, and so further investigation into our matter is strongly warranted. We have a number of suggestions to make to who ever follows up on this research. To begin, our simulation software was unable to give us more information about the circuit's operation than delay-times. As well, the software's methods available to us for simulated input control were mediocre at best. High and low vectors applied to an input could not be removed save for an entire sweep of input data, making precise simulations a tedious game of trial and error. On top of this, the software only has two available fabrication options for simulation, so we are unable to simulate circuits with fabrication sizes any lower than 0.3 microns, which would be extremely beneficial. In light of these limitations we suggest trying to find better simulation software before proceeding further. We would want the ability to simulate smaller scale fabrication sizes, ideally 0.13 or 0.18 microns, and the option to simulate the current-voltage relationships to better characterize a final circuit design.

With that in mind, the path from where we are to fabrication is not a particularly long one. With new software in hand one needs only to take our buffered full adder design and to build a system off of them similar to the diagram in Fig.11. One would then simulate this full circuit at higher resolution processes and determine the best configuration of networks to reduce the repetition time for the buffered full-adder. Then one would convert the above schematic designs to actual layout masks with wiring and transistor parameters according to MOSIS design rules, determine the exact clock timing based on the more accurate simulations resulting from these layouts and finally submit the design to MOSIS for fabrication. On the assumption that the principles discovered in this paper are applicable at lower fabrication sizes we are only a few microns away from having our summing circuit fully functional.

References

[1] T. Mouthaan, "Semiconductor Devices Explained: Using Active Simulation," (John Wiley & Sons, LTD, New York, 1999)

[2] Ed. C.Y. Chang and S.M. Sze, "ULSI Devices," (John Wiley & Sons, LTD, New York, 2000)

[3] M.M. Cirovic, "Integrated Circuits: A user's Handbook," (Reston Publishing Company, INC, Reston VA, 1977)