CMOS Architecture

David Xu

February 2021

Figure 1: Artist interpretation of Carbon Nanotubes

Abstract

With the current landscape of MOSFETs in mind, the approaching limits and shortages of the modern silicon transistor has lead to many concerns over the future of semiconducting technology. This paper aims to look through one possible successor to the long-standing Silicon based MOSFET architecture in the form of Carbon Nanotubes. Although a lot has been researched about the wonder material graphene, many research have not yet reached solid conclusions about the potential of graphene and carbon's allotropes. One example is the Standford model of a 16-bit RISC-V architecture microprocessor that sets a precedent for the earliest prototypes of a CNT based system. A following prototype designed by MIT engineers also prove the possibility of CNT processors. In the time of writing this paper, no other substantial project on CNT processor have been conducted leaving much room for future findings and improvements to existing models.

Contents

1 Introduction

There is no secret that we are fast approaching both the computational and physical limits of the long-standing silicon MOSFET designs. Before now, chips have been steadily increasing in density as transistors shrink in size while becoming more efficient overall as energy consumption remains more or less the same according to Dennard scaling [1] and hence are more cost-effective. However, recently Moore's Law, an observation by Gordon Moore stating that the number of transistors will double every 18 months, is becoming harder and harder for engineers to conform to [2] as the production process nears a nanoscopic scale of 5nm. The problem has less to do with Moore's law but is more in line with Dennard scaling since the main issue has to do with heat, the limiting factor in current circuit design. It turns out that the Dennard scaling rule only applies to transistor sizes above 65nm as chips slowly require more energy to run the smaller they get due to an exponential increase in energy leakage which increases in the amount of heat produced.

In response to this, much effort has already been placed in finding other alternatives in the form of Optical [3] and Quantum based computers that rely on the properties of light and quantum mechanics, respectively. There are also other plans set out to replace silicon with Gallium Nitride using a GANFET architecture[4]. The main problem surrounding these proposed solutions is that they are still unproven and/or are restricted to non-commercial uses making them unlikely to hit the consumer market any time soon, thus limiting them to a very specific audience. Regardless, there is one promising solution that has been under scientific analysis and experimentation over the last two decades, using the so-called wonder material graphene in the form of Carbon Nanotubes[5]. Like with all experimental technologies, CNT transistor-based computers are yet to make an appearance on the market anytime soon and were only recently proven to be practical under controlled environments in labs at MIT[6] using a 16-bit RISC-V based microprocessor.

However, its substantial increase in energy efficiency over MOSFET and high thermal conductivity make it a viable investment that can lead to massive leaps in performance over current processors[7]. The lower voltage draw required to switch a CNT transistor means that less heat is produced leading to an increase in the density of these new chips, helping us overcome the Dennard Scaling problem and follow more closely in line with Moore's Law. Although at this point in time, there are a few main concerns surrounding the manufacturing and variability of the materials used to create these Carbon Nanotubes. The benefits of such technology will not be limited to just classical computers as it may also allow for quantum computers to operate outside of their near-absolute zero temperatures [8].

2 Initial Research

Initially this project started off as a review paper of the current research climate and developments surrounding Carbon Nanotubes. However, as time passed my interests shifted to a more hands on based approach. The research started with looking at existing studies on testing the electrical properties of Carbon Nanotubes as transistors. This was followed by some further reading on two low bit CNT-based processors built by researchers from Stanford and MIT. However, when it came to designing a process of replicating these results there were many complications that resulted in the project taking a different approach. This lead to experimenting with the Reduced Instruction Set Computer architecture (RISC-V) as well as experience in using circuit simulation software such as LTSpice. Therefore much of focus was placed on operation of the RISC-V architecture including a few courses on CMOS and MOSFET devices.

2.1 Stanford Model

2.2 MIT Model

3 RISC-V

This was the first part of the project that involved simulating low level code in order to grasp a better understanding of simple low bit architecture devices. The software used for this was an online compiler (RISC-V interpreter [9]) provided by the university of Cornell. A few sample code were used to learn about the structure of registers, memory addresses and the included instruction mnemonics to the RISC-V architecture. These sample tests ran in the form of basic logical operations such as calculating the mean of a set of numbers, or basic addition and memory manipulation using registers.

One of the harder parts of this exercise was keeping track of when to store variables into the memory and when to move them into them into the register. The compiler used a 32-bit register along with 36 bits of memory.

Each code was run at a frequency of 1Hz to see the process step by step, however for the purposes of demonstration only the final state of the register and memory will be displayed

3.1 Part 1, Memory and addresses

The first study involved looking at how memory and data is handled through commands such as sw or lb by means of addition of two numbers

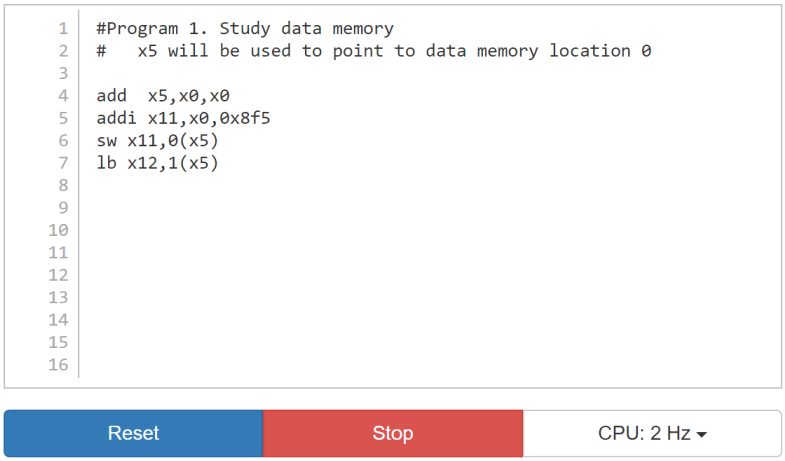

Figure 2: Simple Addition Code

The final memory address and register states are as shown below:

Memory Address	Decimal	Hex	Binary
0x00000000	-1803	0xfffff8f5	0b11111111111111111111100011110101
0x00000004	0	0x00000000	0b00000000000000000000000000000000
0x00000008	0	0x00000000	0b00000000000000000000000000000000
0x0000000c	0	0x00000000	0b00000000000000000000000000000000
0x00000010	0	0x00000000	0b00000000000000000000000000000000
0x00000014	0	0x00000000	0b00000000000000000000000000000000
0x00000018	0	0x00000000	0b00000000000000000000000000000000
0x0000001c	0	0x00000000	0b00000000000000000000000000000000
0x00000020	0	0x00000000	0b00000000000000000000000000000000
0x00000024	0	0x00000000	0b00000000000000000000000000000000

Figure 3: Memory Address

Init Value	Register	Decimal	Hex	Binary
0	x0 (zero)	0	0x00000000	0b00000000000000000000000000000000
0	x1 (ra)	0	0x00000000	0b00000000000000000000000000000000
0	x2 (sp)	0	0x00000000	0b00000000000000000000000000000000
0	x3 (gp)	0	0x00000000	0b00000000000000000000000000000000
0	x4 (tp)	0	0x00000000	0b00000000000000000000000000000000
0	x5 (t0)	0	0x00000000	0b00000000000000000000000000000000
0	x6 (t1)	0	0x00000000	0b00000000000000000000000000000000
0	x7 (t2)	0	0x00000000	0b00000000000000000000000000000000
0	x8 (s0/fp)	0	0x00000000	0b00000000000000000000000000000000
0	x9 (s1)	0	0x00000000	0b00000000000000000000000000000000
0	x10 (a0)	0	0x00000000	0b00000000000000000000000000000000
0	x11 (a1)	-1803	0xfffff8f5	0b11111111111111111111100011110101
0	x12 (a2)	-8	0xfffffff8	0b11111111111111111111111111111000
0	x13 (a3)	0	0x00000000	0b00000000000000000000000000000000
0	x14 (a4)	0	0x00000000	0b00000000000000000000000000000000
0	x15 (a5)	0	0x00000000	0b00000000000000000000000000000000
0	x16 (a6)	0	0x00000000	0b00000000000000000000000000000000
0	x17 (a7)	0	0x00000000	0b00000000000000000000000000000000
0	x18 (s2)	0	0x00000000	0b00000000000000000000000000000000
0	x19 (s3)	0	0x00000000	0b00000000000000000000000000000000
0	x20 (s4)	0	0x00000000	0b00000000000000000000000000000000
0	x21 (s5)	0	0x00000000	0b00000000000000000000000000000000
0	x22 (s6)	0	0x00000000	0b00000000000000000000000000000000
0	x23 (s7)	0	0x00000000	0b00000000000000000000000000000000
0	x24 (s8)	0	0x00000000	0b00000000000000000000000000000000
0	x25 (s9)	0	0x00000000	0b00000000000000000000000000000000
0	x26 (s10)	0	0x00000000	0b00000000000000000000000000000000
0	x27 (s11)	0	0x00000000	0b00000000000000000000000000000000
0	x28 (t3)	0	0x00000000	0b00000000000000000000000000000000
0	x29 (t4)	0	0x00000000	0b00000000000000000000000000000000
0	x30 (t5)	0	0x00000000	0b00000000000000000000000000000000
0	x31 (t6)	0	0x00000000	0b00000000000000000000000000000000

Figure 4: Register

3.2 Part 2, Loops

The second study looked at iterating loops, this was done through a fixed number of iteration

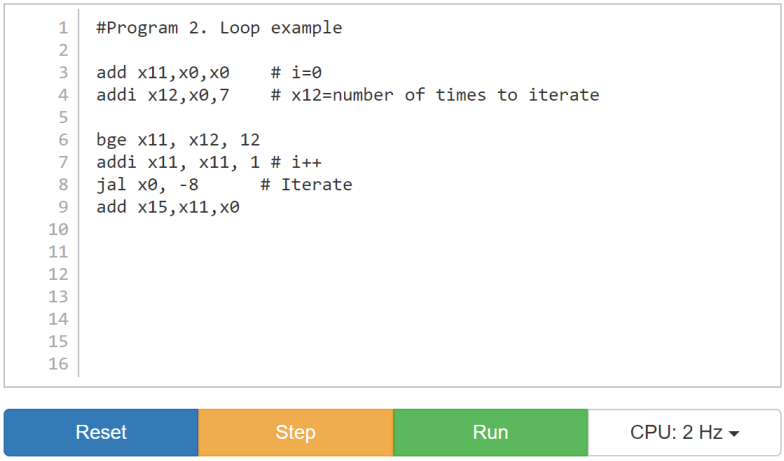

Figure 5: Looping Code

Memory Address	Decimal	Hex	Binary
0x00000000	0	0x00000000	0b00000000000000000000000000000000
0x00000004	0	0x00000000	0b00000000000000000000000000000000
0x00000008	0	0x00000000	0b00000000000000000000000000000000
0x0000000c	0	0x00000000	0b00000000000000000000000000000000
0x00000010	0	0x00000000	0b00000000000000000000000000000000
0x00000014	0	0x00000000	0b00000000000000000000000000000000
0x00000018	0	0x00000000	0b00000000000000000000000000000000
0x0000001c	0	0x00000000	0b00000000000000000000000000000000
0x00000020	0	0x00000000	0b00000000000000000000000000000000
0x00000024	0	0x00000000	0b00000000000000000000000000000000

Figure 6: Memory Address

Init Value	Register	Decimal	Hex	Binary
0	x0 (zero)	0	0x00000000	0b00000000000000000000000000000000
0	x1 (ra)	0	0x00000000	0b00000000000000000000000000000000
0	x2 (sp)	0	0x00000000	0b00000000000000000000000000000000
0	x3 (gp)	0	0x00000000	0b00000000000000000000000000000000
0	x4 (tp)	0	0x00000000	0b00000000000000000000000000000000
0	x5 (t0)	0	0x00000000	0b00000000000000000000000000000000
0	x6 (t1)	0	0x00000000	0b00000000000000000000000000000000
0	x7 (t2)	0	0x00000000	0b00000000000000000000000000000000
0	x8 (s0/fp)	0	0x00000000	0b00000000000000000000000000000000
0	x9 (s1)	0	0x00000000	0b00000000000000000000000000000000
0	x10 (a0)	0	0x00000000	0b00000000000000000000000000000000
0	x11 (a1)	-1803	0xfffff8f5	0b11111111111111111111100011110101
0	x12 (a2)	-8	0xfffffff8	0b11111111111111111111111111111000
0	x13 (a3)	0	0x00000000	0b00000000000000000000000000000000
0	x14 (a4)	0	0x00000000	0b00000000000000000000000000000000
0	x15 (a5)	0	0x00000000	0b00000000000000000000000000000000
0	x16 (a6)	0	0x00000000	0b00000000000000000000000000000000
0	x17 (a7)	0	0x00000000	0b00000000000000000000000000000000
0	x18 (s2)	0	0x00000000	0b00000000000000000000000000000000
0	x19 (s3)	0	0x00000000	0b00000000000000000000000000000000
0	x20 (s4)	0	0x00000000	0b00000000000000000000000000000000
0	x21 (s5)	0	0x00000000	0b00000000000000000000000000000000
0	x22 (s6)	0	0x00000000	0b00000000000000000000000000000000
0	x23 (s7)	0	0x00000000	0b00000000000000000000000000000000
0	x24 (s8)	0	0x00000000	0b00000000000000000000000000000000
0	x25 (s9)	0	0x00000000	0b00000000000000000000000000000000
0	x26 (s10)	0	0x00000000	0b00000000000000000000000000000000
0	x27 (s11)	0	0x00000000	0b00000000000000000000000000000000
0	x28 (t3)	0	0x00000000	0b00000000000000000000000000000000
0	x29 (t4)	0	0x00000000	0b00000000000000000000000000000000
0	x30 (t5)	0	0x00000000	0b00000000000000000000000000000000
0	x31 (t6)	0	0x00000000	0b00000000000000000000000000000000

Figure 7: Register

3.3 Part 3, Arrays

Arrays are a common way of storing a fixed number of elements that can be accessed using the address code of each element. In this example, an array of 12 numbers is created and the average is calculated using a loop/iteration.

```
1  #Program 3. Sum the elements of an array
2
3  #x8 holds pointer to the array A
4  addi x8,x0,0x00  #starting array at data memory location 0
5
6  #fill array with data
7  xor x7,x7,x7  #clear x7 register
8  addi x7,x0,4  #load x7 with data
9  sw x7,0(x8)   #store data pointed x8+offset
10 xor x7,x7,x7  #clear x7 register
11 addi x7,x0,5  #load x7 with data
12 sw x7,4(x8)   #store data pointed x8+offset
13 xor x7,x7,x7  #clear x7 register
14 addi x7,x0,3  #load x7 with data
15 sw x7,8(x8)   #store data pointed x8+offset
16 xor x7,x7,x7  #clear x7 register
```

Reset | Step | Run | CPU: 2 Hz

Figure 8: Arrays Code

```
16 xor x7,x7,x7  #clear x7 register
17 addi x7,x0,2  #load x7 with data
18 sw x7,12(x8)  #store data pointed x8+offset
19
20 #Assign x10=sum, x11=i
21 add x10, x0, x0 # sum=0
22 add x11, x0, x0 # i=0
23 addi x12,x0,4   # x12=number of elements in array
24
25 bge x11, x12, 28
26 slli x13, x11, 2  # i * 4
27 add x13, x13, x8  # & of A + i
28 lw x13, 0(x13)    # *(A + i)
29 add x10, x10, x13 # increment sum
30 addi x11, x11, 1  # i++
31 jal x0, -24      # Iterate
```

Reset | Step | Run | CPU: 2 Hz

Memory Address	Decimal	Hex	Binary
0x00000000	4	0x00000004	0b00000000000000000000000000000100
0x00000004	5	0x00000005	0b00000000000000000000000000000101
0x00000008	3	0x00000003	0b00000000000000000000000000000011
0x0000000c	2	0x00000002	0b00000000000000000000000000000010
0x00000010	0	0x00000000	0b00000000000000000000000000000000
0x00000014	0	0x00000000	0b00000000000000000000000000000000
0x00000018	0	0x00000000	0b00000000000000000000000000000000
0x0000001c	0	0x00000000	0b00000000000000000000000000000000
0x00000020	0	0x00000000	0b00000000000000000000000000000000
0x00000024	0	0x00000000	0b00000000000000000000000000000000

Figure 9: Memory Address

Init Value	Register	Decimal	Hex	Binary
0	x0 (zero)	0	0x00000000	0b00000000000000000000000000000000
0	x1 (ra)	0	0x00000000	0b00000000000000000000000000000000
0	x2 (sp)	0	0x00000000	0b00000000000000000000000000000000
0	x3 (gp)	0	0x00000000	0b00000000000000000000000000000000
0	x4 (tp)	0	0x00000000	0b00000000000000000000000000000000
0	x5 (t0)	0	0x00000000	0b00000000000000000000000000000000
0	x6 (t1)	0	0x00000000	0b00000000000000000000000000000000
0	x7 (t2)	0	0x00000000	0b00000000000000000000000000000000
0	x8 (s0/fp)	0	0x00000000	0b00000000000000000000000000000000
0	x9 (s1)	0	0x00000000	0b00000000000000000000000000000000
0	x10 (a0)	0	0x00000000	0b00000000000000000000000000000000
0	x11 (a1)	-1803	0xfffff8f5	0b11111111111111111111100011110101
0	x12 (a2)	-8	0xfffffff8	0b11111111111111111111111111111000
0	x13 (a3)	0	0x00000000	0b00000000000000000000000000000000
0	x14 (a4)	0	0x00000000	0b00000000000000000000000000000000
0	x15 (a5)	0	0x00000000	0b00000000000000000000000000000000
0	x16 (a6)	0	0x00000000	0b00000000000000000000000000000000
0	x17 (a7)	0	0x00000000	0b00000000000000000000000000000000
0	x18 (s2)	0	0x00000000	0b00000000000000000000000000000000
0	x19 (s3)	0	0x00000000	0b00000000000000000000000000000000
0	x20 (s4)	0	0x00000000	0b00000000000000000000000000000000
0	x21 (s5)	0	0x00000000	0b00000000000000000000000000000000
0	x22 (s6)	0	0x00000000	0b00000000000000000000000000000000
0	x23 (s7)	0	0x00000000	0b00000000000000000000000000000000
0	x24 (s8)	0	0x00000000	0b00000000000000000000000000000000
0	x25 (s9)	0	0x00000000	0b00000000000000000000000000000000
0	x26 (s10)	0	0x00000000	0b00000000000000000000000000000000
0	x27 (s11)	0	0x00000000	0b00000000000000000000000000000000
0	x28 (t3)	0	0x00000000	0b00000000000000000000000000000000
0	x29 (t4)	0	0x00000000	0b00000000000000000000000000000000
0	x30 (t5)	0	0x00000000	0b00000000000000000000000000000000
0	x31 (t6)	0	0x00000000	0b00000000000000000000000000000000

Figure 10: Register

4 MOSFET

A MOSFET, Metal Oxide Semiconductor Field Effect Transistor, is a gate operated switch or amplifier that is controlled using voltages. There are three terminals to be aware of: the source, drain and gate. Depending on whether an NMOS or PMOS device is used, the source and drain are both heavily doped as n-type or p-type regions respectively.

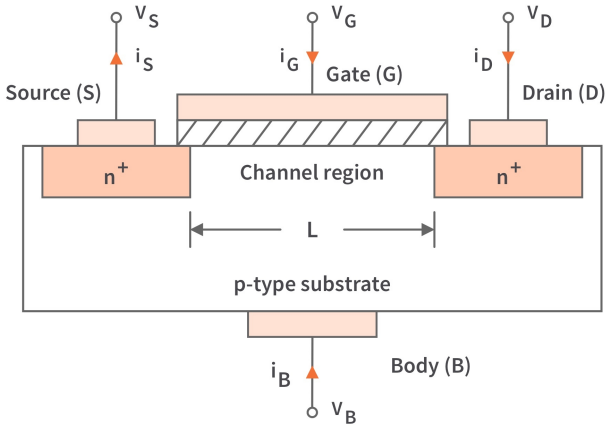

Figure 11: NMOS diagram[3]

From the previous section, a depletion region forms between the p-type substrate and n-type source and drain regions. Above the substrate is a layer of oxide acting as an insulator with an additional layer of metal on top. This acts as the gate and completes the MOS part of MOSFET. There are two modes that determine the voltage characteristics of the gate [4]:

1. **Enhancement-mode** - Requires a gate voltage above a voltage threshold to turn 'ON' ($V_g \geq V_s + V_{th}$)
2. **Depletion-mode** - Requires a gate voltage below a voltage threshold to turn 'OFF' ($V_g \leq V_s - V_{th}$)

For the purposes of the investigation the paper will focus only on enhancement-mode NMOS devices.

4.1 Field Effect and Operation of EM NMOS

The FET, Field Effect Transistor, is controlled by the gate using voltages. At 0V the MOSFET is in a cutoff region ($V_{gs} \leq V_{th}$[1]) meaning no charge carriers can flow through the channel region ($I_{ds} = 0$)[5].

[1] $V_{gs} = V_g - V_s$

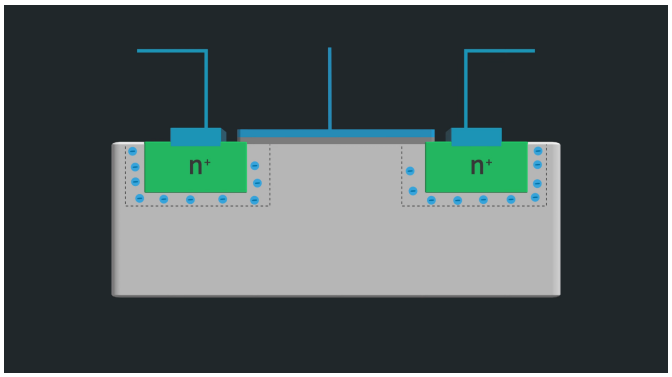

Figure 12: MOSFET when $V_{gs} = 0$[5]

4.1.1 Saturation Region

When a positive voltage is applied, both a depletion region and inversion layer are created around the channel region. The inversion layer expands from the source to drain and connects both regions when when $V_{gs} \geq V_{th}$. However, if $V_{gs} \leq V_{th}$ then it operates at the saturation region since the charge carriers are not fully connected to the drain.

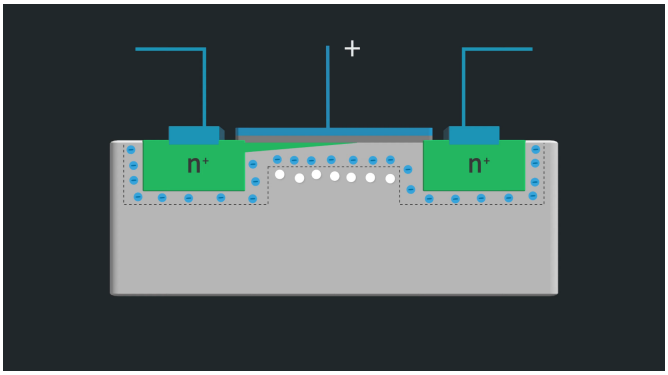

Figure 13: MOSFET when $V_{gs} \leq V_{th}$[5]

4.1.2 Triode Region

When $V_{gs} \geq V_{th}$ electrons will move from source and drain to complete the inversion layer, fully connecting it between both regions. Although if the V_{gs} is too high the MOSFET may operate in the saturation region again since $V_{gs} \leq V_{ds}$[2] to remain in the saturation region [6].

[2] V_{ds} - voltage across drain and source

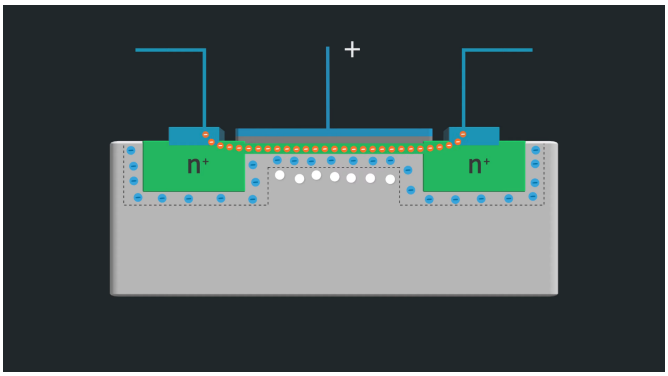

Figure 14: MOSFET when $V_{gs} \geq V_{th}$[5]

4.2 IV Characteristics

As expected, increasing the value of V_{ds} increases the current from drain to source, I_{ds}, since there is a higher negative-charge carrier density making the inversion layer more conductive. In the cutoff region there is no inversion layer, so $I_{ds} = 0$. Plotting the I_{ds} against V_{ds} allows us to see the various dependencies on the values of V_{ds} and V_{gs}.

4.2.1 Triode Region and Ideal IV Characteristics

In the triode region, the MOSFET exhibits properties of an ohmic device as I_{ds} is proportional to V_{ds}. The IV relation for this can be derived as follows [7]:

Given Q(x) as the charge per unit length along the channel length:

$$Q(x) = WC_{ox}\{(V_{gs} - V_{th}) - V(x)\}$$

Where V(x)[3] $= 0$ at the edge point of the source, simplifying to:

$$Q(x) = WC_{ox}(V_{gs} - V_{th})$$

Velocity of electrons in the inversion layers is given by:

$$v = -\mu_n{}^{4}E^{5} = -\mu_n \frac{dV}{dx}$$

Current in the inversion layer is given by:

$$-I_{DS}{}^{6} = vQ(x)$$

[3] V(x) is the voltage at distance x on the channel from the drain
[4] Recall that μ_e is the electron mobility
[5] E can be rewritten as $\frac{dV}{dx}$
[6] Negative because drain current is opposite conventional flow

Combining these equations:

$$-I_{DS} = -\mu_n \frac{dV}{dx} \times WC_{ox}\{(V_{gs} - V_{th}) - V(x)\}$$
$$I_{DS} = \mu_n WC_{ox}\{(V_{gs} - V_{th}) - V(x)\}\frac{dV}{dx}$$

This gives us a differential equation solved using separation of variables:

$$I_{DS}\,dx = \mu_n WC_{ox}\{(V_{gs} - V_{th}) - V(x)\}dV$$

Integrating using the defined boundaries of $x = (0^7, L^8)$ and $V(x) = (0, V_{DS})$:

$$I_{DS}\int_0^L dx = \mu_n WC_{ox}\int_0^{V_{DS}} \{(V_{gs} - V_{th}) - V(x)\}dV(x)$$

$$I_{DS}\,[x]_0^L = \mu_n WC_{ox}\,[(V_{gs} - \tfrac{V}{2} - V_{th})V]_0^{V_{DS}}$$

$$I_{DS}L = \mu_n WC_{ox}\,[(V_{gs} - \tfrac{V_{DS}}{2} - V_{th}) \cdot V_{DS}]$$

Rearrange to give us the final equation:

$$I_{DS} = \frac{\mu_n WC_{ox}}{L}\,[(V_{gs} - V_{th})V_{DS} - \tfrac{1}{2}{V_{DS}}^2]$$

Note: Worst point, x = L and V(x) = V_{DS}. Equation valid when $V_{gs} - V(x) \geq V_{th}$ at every x, so $V_{DS} \leq V_{gs} - V_{th}$ (V_{og}/overdrive voltage)[8].

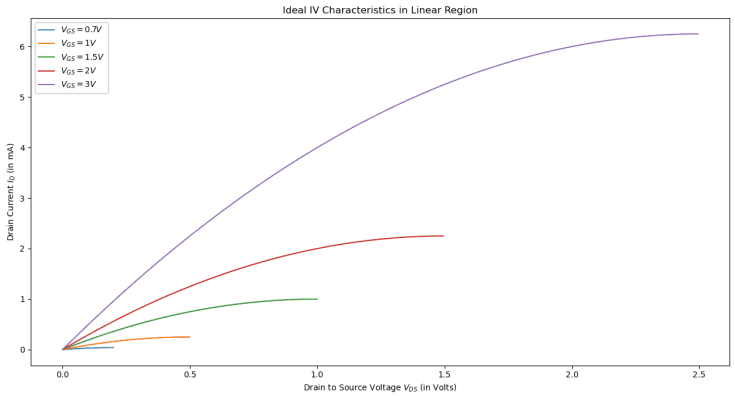

Figure 15: IV graph when V_{DS} is below overdrive voltage[9]

[7]Lower boundary defined as the value at source
[8]Upper boundary defined as the value at drain

Therefore by taking V_{DS} as close to zero we can ignore $\frac{1}{2}{V_{DS}}^2$ and get:

$$I_{DS} = \frac{\mu_n W C_{ox}}{L} \left[(V_{gs} - V_{th})V_{DS}\right]$$

This shows that the current varies proportionally to V_{DS} and is linear when the condition $V_{DS} << V_{gs} - V_{th}$ is fulfilled.

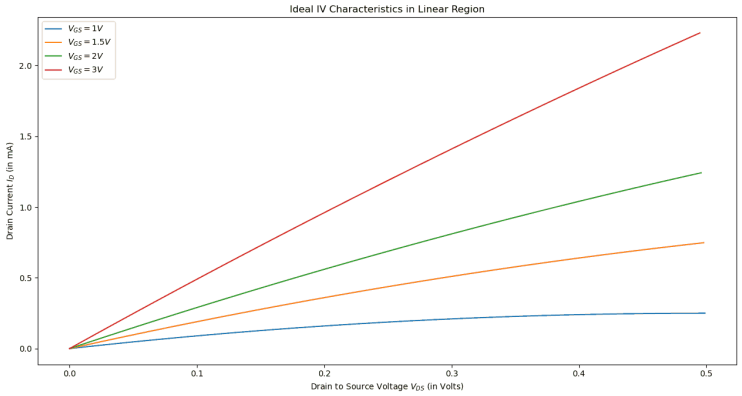

Figure 16: IV graph when $V_{GS} << V_{gs} - V_{th}$[10]

4.2.2 Saturation Region

The equation for an inversion layer in saturation can be done by making changes to the differential equation. At the point x = L'[9], V(L') = $V_{DSsat} = V_{gs} - V_{th}$ substituting that into the equation gives us a new equation:

$$I_{DS} \int_0^{L'} dx = \mu_n W C_{ox} \int_0^{V_{gs}-V_{th}} \{(V_{gs} - V_{th}) - V(x)\} dV(x)$$

Then:

$$I_{DSSat} = \frac{\mu_n W C_{ox}}{2L'} \left[(V_{gs} - V_{th})^2\right]$$

4.2.3 Final IV Characteristics

The final IV characteristics is summarised by the graph below. When V_{DS} exceeds V_{og} the I_{DS} reaches a constant value when the transistor is saturated.

[9] L' is the length of the pinch-off point

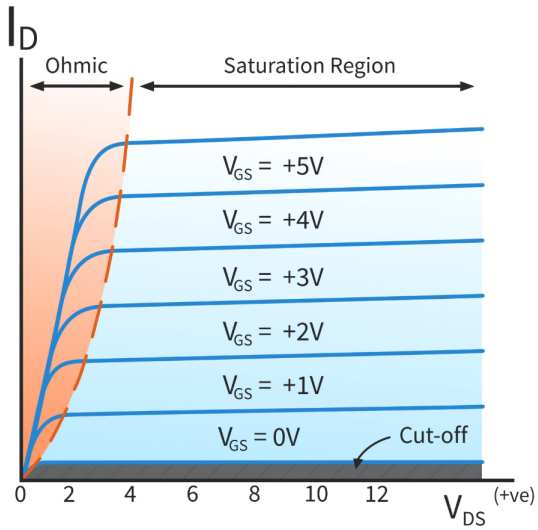

Figure 17: Final IV Characteristic[11]

5 Simulation

By far the most ambitious and challenging part of this paper is the real time simulations of CMOS circuitry in order to recreate the results from the Stanford and MIT models. Unfortunately, the inability to access HSPice for personal use meant that I had to use an alternative.

5.1 CMOS/Spice

LTSpice was used for the simulations and had a built in library of basic PFET and NFET gates so building each circuit was relatively easy. A series of basic logic gates were used to run some basic analysis: NAND, NOR and XOR.[10]

The parameters used in these initial tests are as follows: +5V signaled a digital on, and 0v signaled a digital off, the rise and fall times for each gate were set to 5ns, the cycle was either 5ms and 10ms for input A and input B and the total transient time was 20ms.

The schematics and results of each gate are summarised below:

5.1.1 Inverter

The simplest circuit composed of only a single P-FET and N-FET gate. Using inverse logic, the inverter converts an input of logical 1 (also referred to as a HIGH) to an output of logical 0 (LOW) and vice-versa. This can be displayed through a truth-table[11]:

Input	Output
0	1
1	0

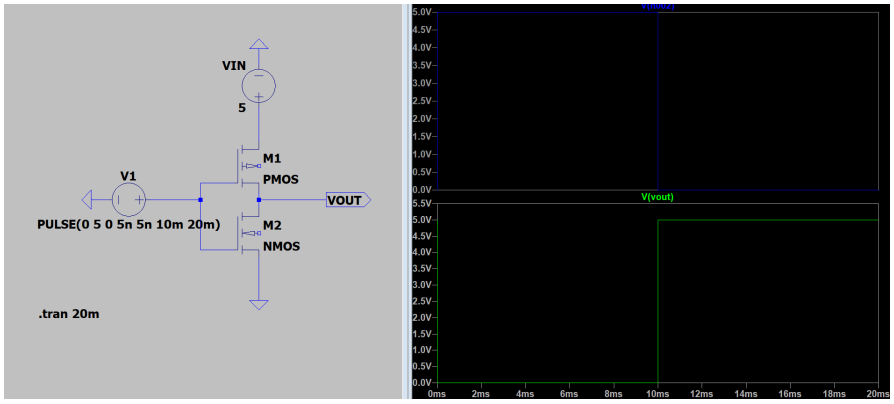

Figure 18: Inverter gate

[10] Figures 2, 3, 4 and 5 show the respective logic gates in LTSpice
[11] A table showing all the combinations of outputs from a logic circuit

5.1.2 NAND Device

A NAND gate outputs a HIGH when either of both inputs are a LOW. The opposite is true when both inputs are a logical 1 or a HIGH, in which case the outputs produces a logical 0 or a LOW. The following truth table for this gate is displayed below:

Input A	Input B	Output
0	0	1
1	0	1
0	1	1
1	1	0

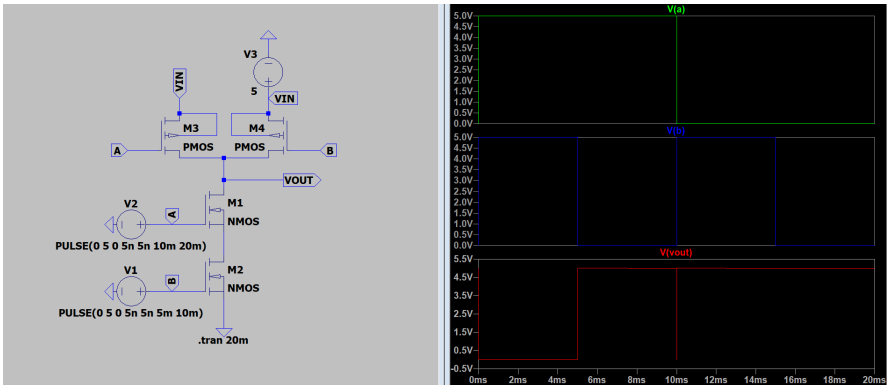

Figure 19: NAND gate

5.1.3 NOR Device

A NOR gate outputs a HIGH only when both inputs are a LOW, otherwise the output remains a LOW irrespective of inputs. Again the truth table for this logic gate is shown below:

Input A	Input B	Output
0	0	1
1	0	0
0	1	0
1	1	0

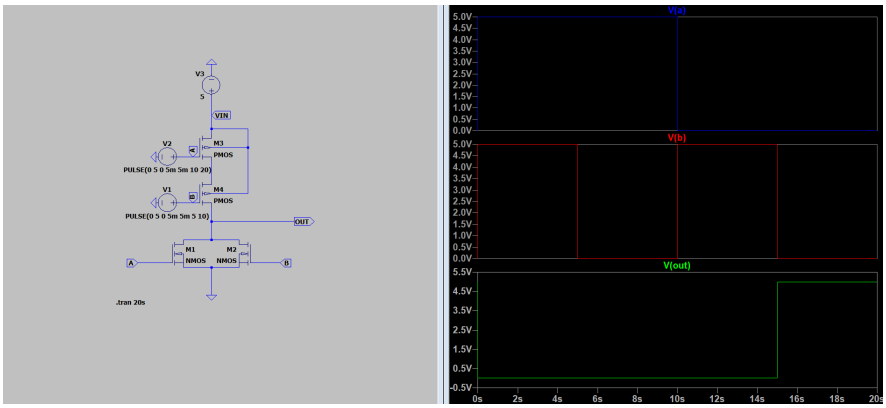

Figure 20: NOR gate

5.1.4 XOR Device

This is a slightly more complex gate requiring the use of multiple NAND and inverter gates. The final design included a more streamlined circuit shown in Figure 13

Input A	Input B	Output
0	0	1
1	0	1
0	1	1
1	1	0

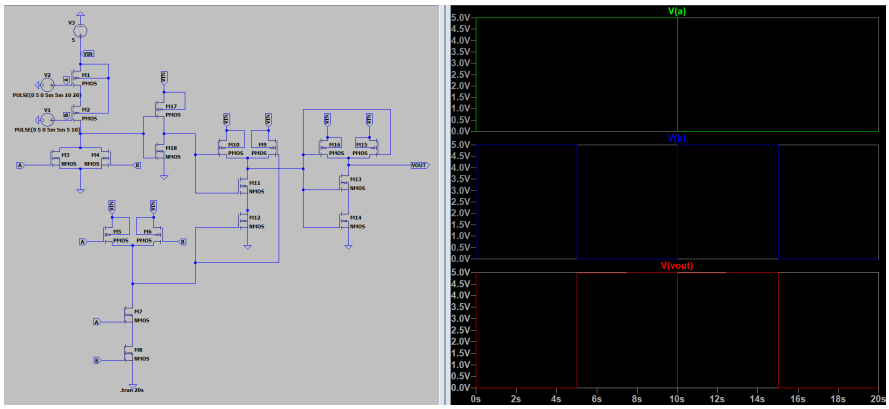

Figure 21: XOR gate

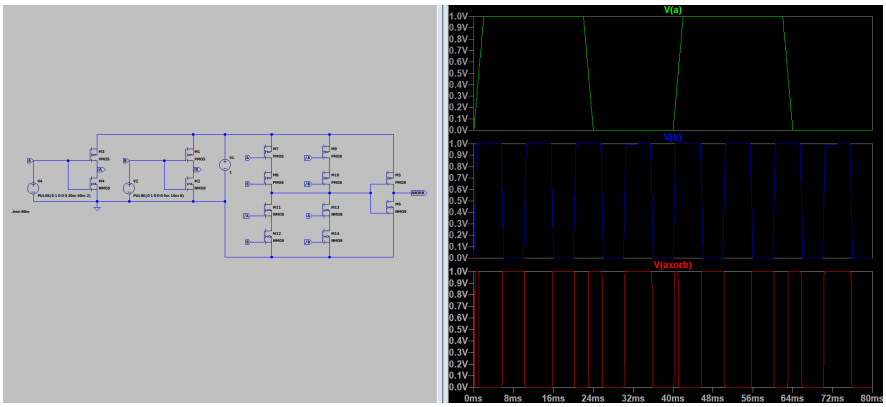

Figure 22: Simplified XOR gate

5.1.5 Full Adder

5.2 IV-Characteristics

6 Analysis

6.1 Issues

Initial tests with single logic gates proved straightforward as I did not need to keep a strict control over the rise and fall times for each gate. However as inputs became more varied and more gates were used to create more complex schematics such as full adders, I needed to do more tweaking on each parameter in order to reduce the run time for each iteration. Without making necessary adjustments, the voltages produced at the end of the circuit would be split into multiple steps instead of clearly defined boundaries. This was in part due to the length of the run time being too long for the program to simulate causing the program to run short

Although the creation of each schematic was simple, there were a few problems when implementing each logic gate in LTSpice. One main issue was syncing each voltage source so that the final output would be in phase instead of being split up into steps as shown below. This was resolved by using a set variable so that each cycle would be a multiple of 5, therefore ensuring that each input was at least half a phase apart.

6.2 Results

The basic single logic gates all produced expected values from inputs A and B, although the results varied when combining multiple logic gates together

7 Future Investigations

Although many of the initial goals of this paper have been achieved in some capacity, there is still areas where I'd like to spend more time delving through.

7.1 World of Quantum

As stated in the introduction, however due to the restraint in time, I was unable to explore the potential use of CNTs in quantum computing mainly to reduce the operating temperatures which these systems relied on in order to function

7.2 Micro Devices

8 Conclusion

References

[1] Rambus. Press. *Understanding Dennard scaling*. 2016.

[2] Matthew Hutson. *Scientists use carbon nanotubes to make the world's smallest transistors*. AAAS, 2017. URL `https://www.researchgate.net/publication/321660078_Scientists_use_carbon_nanotubes_to_make_the_world's_smallest_transistors`.

[3] Circuit Bread. *Nmos-transistor-structure*. `https://dwma4bz18k1bd.cloudfront.net/tutorials/NMOS-Transistor-Structure.jpg`, last accessed: Aug 2021.

[4] Wikipedia. *Depletion and enhancement modes*. `https://en.wikipedia.org/wiki/Depletion_and_enhancement_modes`, last accessed: Aug 2021.

[5] Circuit Bread. *How a mosfet works at the semiconductor level*. `https://www.circuitbread.com/tutorials/how-a-mosfet-works-at-the-semiconductor-level`, last accessed: Aug 2021.

[6] Adam Huan. *Relationship between vds and vgs - mosfet*. `https://electronics.stackexchange.com/questions/222863/relationship-between-vds-and-vgs-mosfet`, last edited: Nov 2017.

[7] Archishman Biswas. *Working of mos transistors – ideal iv characteristics of a mosfet*. `https://technobyte.org/working-mos-transistors-ideal-iv-characteristics/`, last edited: Nov 2017.

[8] Nickolas Thomas. *Lecture 8 mosfet(i) mosfet i-v characteristics*. `https://docplayer.net/20611391-Lecture-8-mosfet-i-mosfet-i-v-characteristics.html`, last edited: 2016.

[9] Technobyte. *Triode region iv characteristics diagram.* https://i0.wp.com/technobyte.org/wp-content/uploads/2020/04/Triode_Region.png?ssl=1, last edited: Nov 2017.

[10] Technobyte. *Linear iv characteristics diagram.* https://i0.wp.com/technobyte.org/wp-content/uploads/2020/04/Linear.png?ssl=1, last edited: Nov 2017.

[11] Circuit Bread. *Iv characteristics graph.* https://dwma4bz18k1bd.cloudfront.net/tutorials/FET-Different-Operating-Regions-Graph.png, last accessed: Aug 2021.