DEPARTMENT OF PHYSICS, UNIVERSITY OF CALIFORNIA BERKELEY

Instrumentation Lab Final Project: 4-Bit Adder/Subtractor with LabVIEW Readout

Oliver Gorton

Partner: Oliver Wang

December 11, 2015

Introduction		20
Diagrams		20
Description		40
Level of Functioning		40
Design Creativity		40
Measurements		10
Conclusions		10
References		5
Instructor Arb Adjust		
Total		**185**

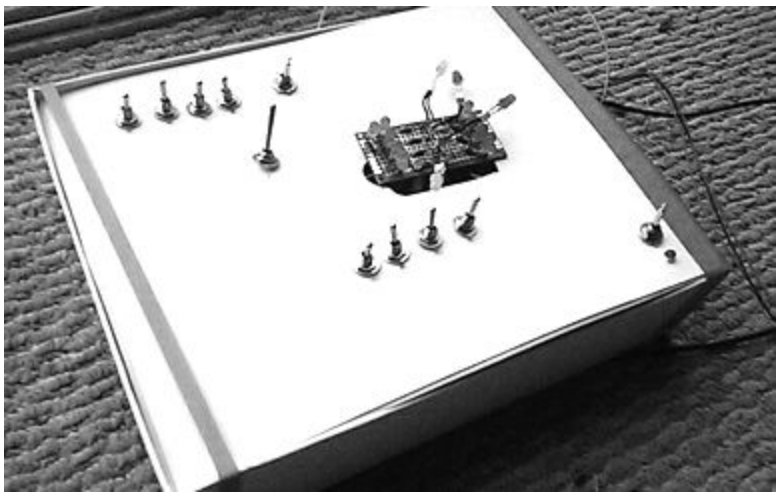

Image: Exterior of 4-Bit Adder Subtractor

DEPARTMENT OF PHYSICS, UNIVERSITY OF CALIFORNIA BERKELEY

Introduction

Today everything relies on some sort of digital computation. Personal computers, household appliances, public transportation, everything. The project which I and my partner Oliver Wang have planned, designed, and constructed is a 4-Bit Adder and Subtractor. See Diagram 10. It is capable of adding or subtracting two 4-Bit numbers i.e. any two numbers between 0000 and 1111 (0 and 15 in decimal). In order to do this computation the circuit must complete 52 to 56 logical operations. Physically the calculator is a box containing a full soldered circuit board with 11 switches, 14 TTL logic chips, 13 LEDs, a DAC circuit, and a USB power supply, with somewhere around 250 soldered joints. The calculator is operated with three sets of switches, two sets which control the input values for the calculation, and one set which selects the function to be calculated. Once the two 4-bit numbers are entered selected using 8 switches (one switch per bit) the result of the calculation it displayed on 4 LEDs. (LED on = 1, LED off = 0). The output of the circuit can also be read off using a LabVIEW VI which we programmed to display the result with both boolean indicators, binary numbers, and decimal numbers. This is accomplished with a DAC circuit that was integrated into the calculator. Its design is similar to that of the one used in Lab 10[11]. I personally put over 60 hours into this project including research, design, and construction. This project is an ode to the days of macroscopic computation. It is also a demonstration of technical skills in circuit design and construction.

With any type or sophistication of digital circuit but a click away at our disposal, we had to make a decision as to what circuit components we would allow ourselves to use. Left otherwise unconstrained we could have easily acquired a 7483 logic chip: a 4 Bit Binary Full Adder [1]. Our task would then be reduced to simply soldering switches and LEDs to the chips terminals. So clearly some constraints had to be made, as has always been the case in this course. The decision we came to was to use exclusively NAND, AND, and NOT logic: three logic gates available in the lab. The most obvious choice might have been to avoid composite logic gates and implement directly the logic gates found in the logic diagram (Diagram 3): XOR, AND, OR, and NOT. But our choice to construct XOR gates out of NAND gates, while perhaps an unnecessary extra step, was well founded in the theory of logic operations. NAND gates form a functionally complete set[2], i.e. can be used to create all possible truth tables. (Other such complete sets are {NOR} and {AND, NOT}. These are to digital logic as sines and cosines are to analytic functions). Its then no surprise that NAND, AND, and NOT logic chips could be found in the lab while XOR gates could not. Our assigned task, therefore, would be to use the logic gates made available to us in the lab: NAND, AND, and NOT.

Theory

TTL Physics

Transistor-transistor logic (TTL) is what makes up digital circuitry [6]. If you could open up a logic chip and look inside you would find a simple network of bipolar junction transistors and resistors. The core function of any digital circuitry is to perform calculations using binary logic operations. This is done by assigning the value 1, or true, to high voltage, and assigning the

value 0, or false, to low voltage (usually ground). The nature of transistors is to allow current to flow from terminal (the collector) to another (the emitter) when a voltage is applied between a third terminal (the base) and the emitter. Each of the three logic gates: AND, NOT, and NAND, is used in this project. Below are the circuit diagrams depicting the construction of an AND gate and NAND gate (See diagrams 1-2). However, we will use pre-built logic chips which contain several (4-6) logic gates each.

By combining individual logic gates in ever more complex ways, one can create algorithms which can compute ever more complex functions. The full explanation of this process applied to our project is described in the following section.

Fundamental Logic Gates

AND

The AND logic gate has two inputs and one output. AND outputs 1 if both of its inputs are 1, and 0 otherwise. AND logic gates are available in compact packaging in the 7408 logic chip[4]. .

AND(A,B) = A ∧ B = Y

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

NAND

The NAND logic gate has two inputs and one output. NAND outputs 1 if both of its inputs and 0, and 1 otherwise. NAND logic gates are available in compact packaging in the 7400 logic chips [5]. NAND logic can be thought of as a composition of NOT and AND logic: NOT(AND(A, B)).

NAND(A,B) = ¬(A ∧ B) = Y

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOT

The NOT logic gate, or inverter, is different from the AND and NAND logics in that it has only one input and one output. The NOT output is 1 if its input is 0 and 0 if its input is 1. NOT logic

gates are available in compact packaging in the 7404 logic chips.

NOT(A) = ¬A = Y

A	Y
0	1
1	0

Construction of composite logic gates

Our full adder logic gate requires both XOR and OR logic gates, which must be constructed from the fundamental logic gates defined above. While these logic gates are manufactured in prepackaged units, we will be constructing them ourselves out of other logic gates, namely NAND. See Introduction.

OR [8]

The OR logic gate has two inputs and one output; it outputs 1 if either one or both of its inputs are 1, and 0 if both of its inputs are 0.

OR(A,B) = A v B = Y

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

In order to produce this truth table, three NAND gates can be combined in the following algebraic formulation:

F(A,B) = **NAND(** NAND(A,A), NAND(B,B) **)**

The bold letters here are simply to emphasize the the order of the functional dependence. See logic diagram 4.

To see that this composite function is indeed equivalent to OR, let's compute the composite truth table using the known truth table of the NAND function defined above.

First, let X = X(A) = NAND(A,A) and Z = Z(B)= NAND(B,B). Then reading off the NAND truth table:

A	X	B	Z
0	1	0	1
1	0	1	0

(Notice this is equivalent to a NOT gate!)

X	Z	NAND(X,Z)
0	0	1
0	1	1
1	0	1
1	1	0

But now augment in the corresponding truth tables for X and Z:

A	B	X	Z	NAND(X(A), Z(B))
1	1	0	0	1
1	0	0	1	1
0	1	1	0	1
0	0	1	1	0

Removing the intermediate variables X and Z and reordering the table into the standard order we get:

A	B	F(A,B)
0	0	0
0	1	1
1	0	1
1	1	1

Which is exactly what we would expect from the OR logic gate! So indeed

OR(A,B) = **NAND(** NAND(A,A), NAND(B,B) **)**.

See Diagram 4.

To actually build such a logic gate is simple. Since we need to connect three NAND gates together, we can simply connect the appropriate terminals of a 7400 logic chip together, thus turning a single 7400 logic chip into a dedicated OR logic gate. (This convenient circuit design is the main reason that we use three NAND gates rather than one NAND gate and two NOT gates, as we showed above would have also worked.) See Diagram 5.

XOR [9,10]

The XOR logic gate has two inputs and one input. XOR stands for exclusive-or to it shouldn't be a surprise that its function is similar to that of the OR gate. An XOR gate outputs a 1 only if just one of its inputs are 1, and 0 otherwise.

XOR(A,B) = A $\oplus$ B = Y

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

In order to produce this truth table, four NAND gates can be combined in the following logical formulation:

F(A,B) = **NAND(** NAND(A, NAND(A,B)), NAND(B, NAND(A,B))**)**

See logic diagram 6.

To show that this function is equivalent to the XOR function, we will show that is produces the correct truth table. As before, lets define some intermediate variables to aid our analysis. Let X = X(A,B) = NAND(A,B). Furthermore, let P = NAND(A, NAND(A, B)) and Q = NAND(B, NAND(A, B)). From these three definitions we also get that

P = NAND(A, X) = P(A, X);

Q = NAND(B, X) = Q(B, X); and,

F = NAND(P, Q) = F(P,Q).

This time let's begin with the highest order truth table and move down from there. With our newly defined intermediate variables we have that:

F(A,B) = NAND(P,Q) with truth table:

P	Q	F(P, Q)
0	0	1
0	1	1
1	0	1
1	1	0

(The usual NAND truth table.) Similarly, the truth tables for P, Q, and X are:

A	X	P(A,X)
0	0	1
0	1	1
1	0	1
1	1	0

B	X	Q(B,X)
0	0	1
0	1	1
1	0	1
1	1	0

A	B	X(A, B)
0	0	1
0	1	1
1	0	1
1	1	0

All NAND tables. Now let's augment all of these truth tables with increasing dependence

moving from left to right, i.e. (A,B)|(X)|(P,Q)|F:

A	B	X	P	Q	F
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	0

Omitting the intermediate variables, we are left with the following truth table giving F as a function of A and B:

A	B	F(A,B)
0	0	0
0	1	1
1	0	1
1	1	0

Which exactly matches the truth table for the XOR gate, thus we have shown that the logical function F(A,B) = **NAND(** NAND(A, NAND(A,B)), NAND(B, NAND(A,B))**)** is congruent to XOR(A,B).

XOR(A,B) = **NAND(** NAND(A, NAND(A,B)), NAND(B, NAND(A,B))**)**

As with the OR logic gate, the XOR logic gate is constructed entirely out of NAND gates and so can be made by connecting the terminals of a 7400 logic chip in a configuration corresponding to the formula above. See diagram 7.

1-Bit Full Adder

A full adder three inputs and two outputs. The inputs are: Input A, Input B, and Carry-in. The outputs are: Sum, and Carry-out. Inputs A and B are the bits which we set out to add. The purpose and function of Carry-in will become clear in a moment. The Sum output is the 1-bit sum of A and B. However, suppose A = 1 and B = 1, then A+B = 10. In this case Sum = 1 and Carry-out = 1; we have added two digits whose sum exceeds the modulo of its place, and so the result involves two bits: the ones' place (Sum) and the twos' place (Carry-out). If you want to construct an adder with more bits, you feed the Carry-out result of the n-th bit into the Carry-in input of the n+1-th bit. This process is called ripple-carry (ref.), and is used to construct the 4-bit adder. For brevity let's assign a letter to each of the inputs and outputs of the 1-bit full adder. Let A = input A, B = Input B, Cin = Carry-in, S = Sum, and Cout = Carry-out.

Full adder truth table:

{A	B	Cin}	{S	Cout}

0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

The composition of logical operations that yields this truth table is

F(A, B, Cin) = { **XOR(** XOR(A,B), Cin**)**, **OR(** AND(A, B), AND(XOR(A,B), Cin) **)}**

Where, as I shall show, the first output in the curly brackets is Sum and the second output in the curly brackets is Carry-out. Again, if we can show that this function produces the same truth table as the 1-bit full adder, then it is equivalent to it, and we can construct a full adder using the logic gates suggested by the formulation.

Let X = XOR(A ,B), and Z = AND(A, B). Then

F(A, B, Cin) = {**XOR(**X, Cin**)**, **OR(** Z, AND(X, Cin)**)**}.

Let W = AND(X, Cin). Then

F(A, B, Cin) = {**XOR(**X, Cin**)**, **OR(**Z, W**)}**.

Here is a table summarizing the variables we have to deal with:

Original Variables		Intermediate variables
Input variables	Output Variables	
A	S	X
B	Cout	Z
Cin		W

From here our goal will be to express the truth table for the outputs XOR(X, Cin) and OR(Z, W) in terms of the input variables A and B. Eventually we wish to show that XOR(X, Cin) = S and OR(Z, W) = Cout. The truth tables for XOR and OR in terms of the relevant intermediate variables are:

X	Cin	XOR(X, Cin)	Z	W	OR(Z, W)
0	0	0	0	0	0
0	1	1	0	1	1

1	0	1	1	0	1
1	1	0	1	1	1

The truth tables for Z = AND(A, B), X = XOR(A, B), and W = AND(X, Cin) are, respectively,

A	B	Z = AND(A, B)	X = XOR(A, B)	Cin	W = AND(X, Cin)
0	0	0	0	0	0
0	1	0	1	0	0
1	0	0	1	0	0
1	1	1	0	0	0
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	1	1
1	1	1	0	1	0

Now we augment the two sets of truth tables:

A	B	Z	X	Cin	W	XOR(X, Cin)	OR(Z, W)
0	0	0	0	0	0	0	0
0	1	0	1	0	0	1	0
1	0	0	1	0	0	1	0
1	1	1	0	0	0	0	1
0	0	0	0	1	0	1	0
0	1	0	1	1	1	0	1
1	0	0	1	1	1	0	1
1	1	1	0	1	0	1	1

Omitting the intermediate variables Z, X, and W, and labeling P = XOR(X, Cin) and Q = OR(Z, W)

A	B	Cin	P	Q
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0

0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

This is exactly the truth table expected from the 1-bit adder, and thus we have shown that P = S and Q = Cout. We have shown that,

F(A, B, Cin) = { **XOR(** XOR(A,B), Cin**)**, **OR(** AND(A, B), AND(XOR(A,B), Cin) **)**} = {S, Cout},

where F(A, B, Cin is the logical operation enacted by the 1-bit full adder.

Or equivalently,

Sum(A, B, Cin) = **XOR(** XOR(A,B), Cin**)**, and

Cout(A, B, Cin) = **OR(** AND(A, B), AND(XOR(A,B), Cin) **)**.

See Diagram 3.

4-Bit Full Adder.

If the 4-bit adder is to perform as it should and add any two integers between 0000 and 1111 (0 and 15), then is must be shown that the logic diagram for the 4-bit adder produces the correct truth table.

The function of a 4-bit adder is to perform the following operation:

Given two 4-bit numbers in binary representation, denoted A = a4.a3.a2.a1 and B = b4.b3.b2.b1 where xN represents the N-th bit of the x-th number (ref.), output the sum A+B = c4.S4.S3.S2.S1.

The structure of the 4-bit adder is as follows:

A + B = c4.S4.S3.S2.S1

Where each place in the binary number is separated by a period "." and where

S4 = S4(a4, b4, c3)

S3 = S3(a3, b3, c2)

S2 = S2(a2, b2, c1)

S1 = S1(a1, b1, c0)

are the four 1-bit Sum outputs of each 1-bit adder, and c1, c2, c3, and c4 are the Carry-out outputs of the first, second, third, and fourth 1-bit adder, respectively. Notice that while the maximum sum of two 4-bit digits is 11110 (i.e. 30,) by using the Carry-out terminal of the fourth 1-bit adder we can effectively produce a 5-bit output. Therefore increasing the maximum representable sum to 11111 (i.e. 31). Using the logical equivalence proven in the previous section, we can conclude that the logic diagram of a 4-bit adder is as shown in

Diagram 8.

Subtracter

I will take to be true without proof that in order to add two binary numbers one must compute:

A – B = A + $\bar{B}$ + 1

Where $\bar{B}$ is the complement of B. In this context be mean the inverted value of B. If B is 1 then its inverse is 0, and vice versa. This fact comes from the theory of Two's Complement[7]. Thus if we want to subtract B from A, all we need to do is first invert B, add it to A, and then add one. The logic operation which yields the inverse of its input it the NOT gate. Therefore, if we need to compute:

A – B = A + NOT(B) + 1.

Fortunately we do not need to construct an entirely new circuit in order to get subtraction. We can use the existing 4-Bit Adder as a 4-Bit subtractor by first feeding each bit of input B through a NOT gate, and adding 1. This can be done in a single operation by connecting the first 1-bit adder's Carry-in input to high-voltage. See Diagram 9.

LabVIEW Readout

Our circuit also includes a DAC which can be connected to a computer with a DAQ card and LabVIEW. The DAC which we implemented is a scaled resistor DAC. It works by scaling the voltage output of each bit of the calculator by some factor. Each of these scaled value is then send through the same unity gain amplifier, which outputs a single voltage value. Each of the 5 bits of data is scaled by an increasing value of two using voltage dividers, so that the linear combination of the outputs is unique up to the smallest difference in the voltage. Once the DAQ receives the analog output of the DAC, our LabVIEW program decodes it back into its binary form and displays the 5-bit boolean array, the binary representation, and the decimal representation.

The program looks something like this:

The DAQ measures the output voltage of the DAC and we compare that value to the first voltage value correspond to the first power of two. If the voltage is greater than that value then the first bit is 1. We then subtract that value from the voltage and compare the remaining value to the next order value of two. If the voltage is less than that value then we assign a 0 to the current bit value and move on to the next value. This continues for 5 iterations: the maximum number of bits contained in the signal. The program involves a scaling factor which must be entered by the user: the output voltage of the unaltered/uncombined single bit signal.

We had originally planned to include ADC functionality where the circuit could also be controlled from LabVIEW. With this addition to the program we would have been able to both control the inputs and read the output of the circuit, thus having a fully functional 4-bit calculator program that used our circuit to do the actual computation. The DAQ interface available to us, however, only has 7 output bits, so we were not able to use this option.

Supplemental Description

So far I have described many of the parts that make up this circuit individually in great detail. In this section I will give a brief overview of how all of these parts come together to make the final product, and also explain some finer details that were previously omitted.

In the section above on the 4-Bit adder and in the sections leading up to it, the logic diagram for the 4-Bit adder is explained. However, this project involves real circuits, which are not a series of elliptic triangles but digital logic chips. The circuit construction of OR and XOR gates from NAND chips is also described, and it is with these logic circuits along with the digital logic chips containing AND and NOT gates that the circuit is made.

All of the digital logic gates in this circuit require a +5V power source to operate. The power to the entire circuit is supplied by a USB power cord that has been modified to supply power to a power strip which is mounted inside the device. The power supply is controlled by a single SPDT switch. This strip also supplies ground to the circuit which is crucial to the operation of the logic chips. Similarly all of the voltage inputs carrying the digital information is ran off of this power source. Therefore a signal around +5V corresponds to a 1, and a signal close to 0V corresponds to a 0. See introduction.

The are 14 LED indicators on the device: 4 for each of the 4-bit inputs, 5 for the 5-bit output, and one for a main power indicator. Each LED is terminated to ground with a 100 ohm resistor in order to reduce the voltage applied. We found that a voltage of around 3V is ideal for these LEDs. The output of the switches is 5V, so the resistors prevent damage to the LEDs, which will go out after a few minutes with 5V supply. The output of the Adder/Subtractor circuit is closer to 3V however. Because the LED indicators for these outputs were mistakenly also attached to 100 ohm resistors, they are noticeably dimmer than the input bit LEDs.

The method for augmenting the subtractor function into the existing adder also deserves special attention. To compute a subtraction, the second 4-bit number is first inverted, and then 1 is added to the sum of the first 4-bit number and the now inverted second number. To accomplish this on the circuit we have two sets of wires leaving the switches governing the second bit number. The first set goes to the LEDs to display the selection. The second set is then send to a dedicated circuit board containing a 7404 logic chip: a hex inverter. (A chip with six NOT gates). But before the signal is sent into the inverter, it is split again. This secondary signal along with the output of the inverter are not sent to a 4PDT switch, with each signal occupying one set of the poles. This means that when the switch is thrown one way, it will output the signal set from the inverter, and when thrown the opposite way will output the uninverted signal. The output of this switch is then sent to the Adder. See Diagram 10. Now we are able to compute A + NOT(B) but we still need to add 1. This is done by setting to 1 the input of the Carry-in input of the first 1-bit adder. A SPDT switch is attached to this input with inputs of ground and power. In order to select the function that the calculator computes, one has to flip both of these two switches (the 4PDT and SPDT) to the correct setting. To add select the uninverted input and Cin1 to ground. To subtract select the inverted

input and Cin1 to power.

Functionality/Conclusion

The design of this circuit was a success. It correctly computes any two sum of integers between 0 and 15 (thanks to the 5-bit output) and correctly computes the difference between any two integers as long as A>B where D = A – B is the difference you wish to compute. We verified these computations using both the LED indicators and the LabVIEW software (for which the binary to decimal converter was particularly useful). Physically however our circuit is less than ideal. With over 250 soldered joints, it no surprise that mistakes were made, and faulty joints were soldered. While the circuit works some of the time, there are apparently some faulty connections which cause false results and inputs to be displayed for subtraction. I would like to emphasize however that this is not a flaw in the design of the circuit but in the skills of its manufacturer's: myself and my partner, whom had no significant experience soldering before this project. We are now reasonably proficient, however, having learned and improved along the way. The LabVIEW implementation also functioned correctly, however not at first. Due to imperfections in the components and construction, and also due to the design of the circuit, each output voltage of each bit varied slightly from the others. The power source for the entire circuit is +5V. The voltage output of the 4-bit adder varied from 3.15V to 3.4V. This range, while small, was enough to cause sufficient distortion of the DAC output that the program was unable to accurately determine the calculator's output bit pattern outside of a certain range of values. We were eventually able to remedy this problem by changing the gain of the op amp in the DAC circuit and adjusting the scaling factors in the program itself.

Acknowledgements

Having not done the digital circuit labs taught in previous years, and having no previous experience with digital circuitry, all of my knowledge concerning this project was acquired 'in-flight'. I do not claim to have reinvented the wheel. The logical construction of the adder used in this project Diagram 6 comes from Wikipedia.org[3]. However, the proofs contained in this report were written by myself. The circuit design is also of my own creation and it underwent many transformations throughout the design and construction process to become what it is now. I would like to thank my lab partner Oliver Wang who designed the LabVIEW software and DAQ circuit, and constructed a considerable amount of the adder/subtractor circuit, and helped to test and troubleshoot the whole thing during construction. I would also like to thank Joel Fajans, who wrote the digital lab manuals, which provided crucial technical information for dealing with logic circuits, and my roommate Owen Jow who lent me his soldering iron for several days. See references page.

I used Multisim to create all (but one) of the circuit diagrams presented in this report.

References

[2] Functional completeness
https://en.wikipedia.org/wiki/Functional_completeness

[3] Adder
https://en.wikipedia.org/wiki/Adder_%28electronics%29#Full_adder

[9,10] XOR gate
http://instrumentationlab.berkeley.edu/Digital1
https://en.wikipedia.org/wiki/XOR_gate

[8] OR gate
https://en.wikipedia.org/wiki/OR_gate

[7] Method of Complements
https://en.wikipedia.org/wiki/Method_of_complements

[7] Two's Complements
https://en.wikipedia.org/wiki/Two's_complement

[1]4-Bit Adder
http://www.jameco.com/1/1/848-7483-4-bit-binary-full-adder-packaging-dip-16-74-series.html

[6] TTL
https://en.wikipedia.org/wiki/Transistor%E2%80%93transistor_logic

[5]7400 Logic
http://instrumentationlab.berkeley.edu/Digital1
http://www.ti.com/lit/ds/symlink/sn74ls00.pdf

[4]7408 Logic
http://ecee.colorado.edu/~mcclurel/dm74ls08.pdf

[1 1]DAC
http://instrumentationlab.berkeley.edu/Lab10

[12] NAND Logic Construction
https://en.wikipedia.org/wiki/NAND_gate

Logic gates, Symbols, Chip IDs
Horowitz and Hill, “The Art of Electronics” 2nd Ed. Pp478-484

Diagrams

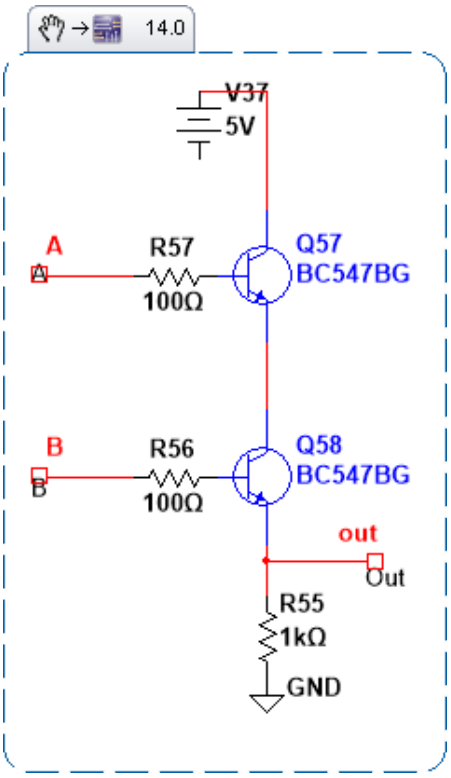

Diagram 1: Transistor construction of AND logic gate

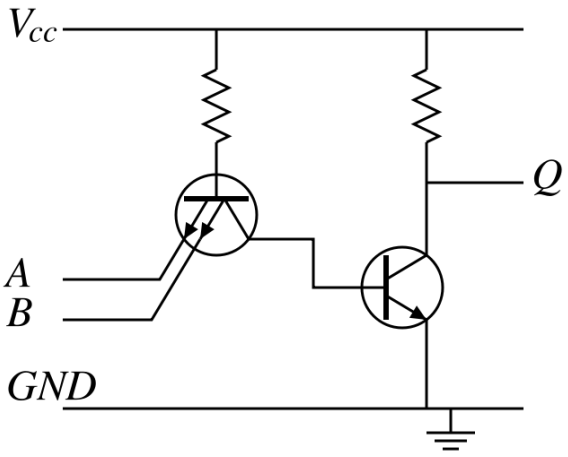

Diagram 2: Transistor construction of NAND logic gate, credit: Wikipedia.org [12]

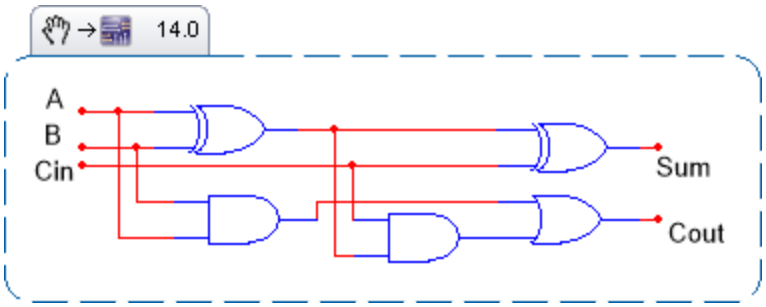

Diagram 3: Logic of 1-Bit Full Adder

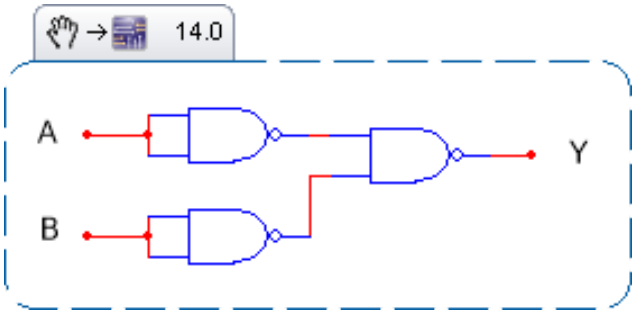

Diagram 4: OR Gate From Three NAND Gates

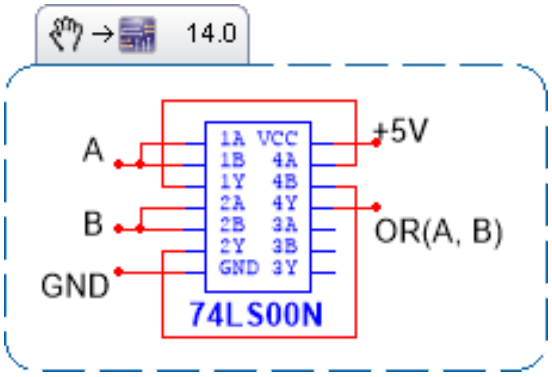

Diagram 5: OR Circuit Construction on 7400 Chip.

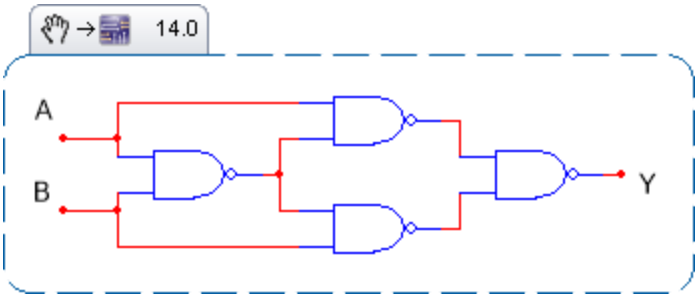

Diagram 6: XOR Gate from Four NAND Gates

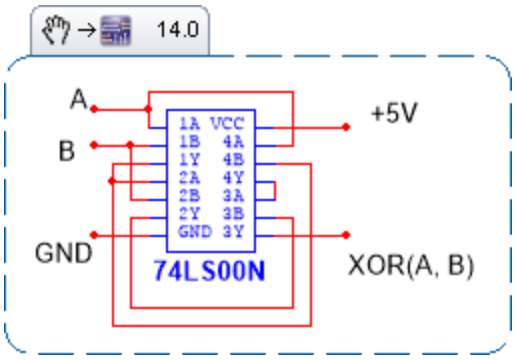

Diagram 7: XOR Circuit Construction on 7400 Chip

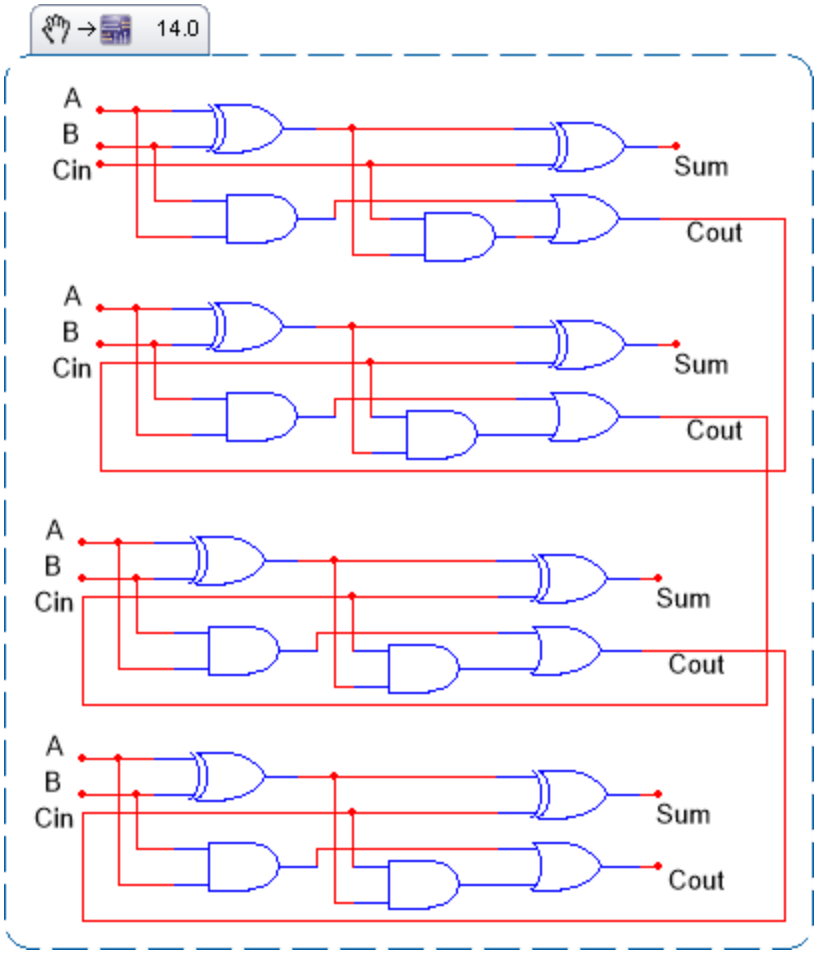

Diagram 8: 4-Bit Adder Logic

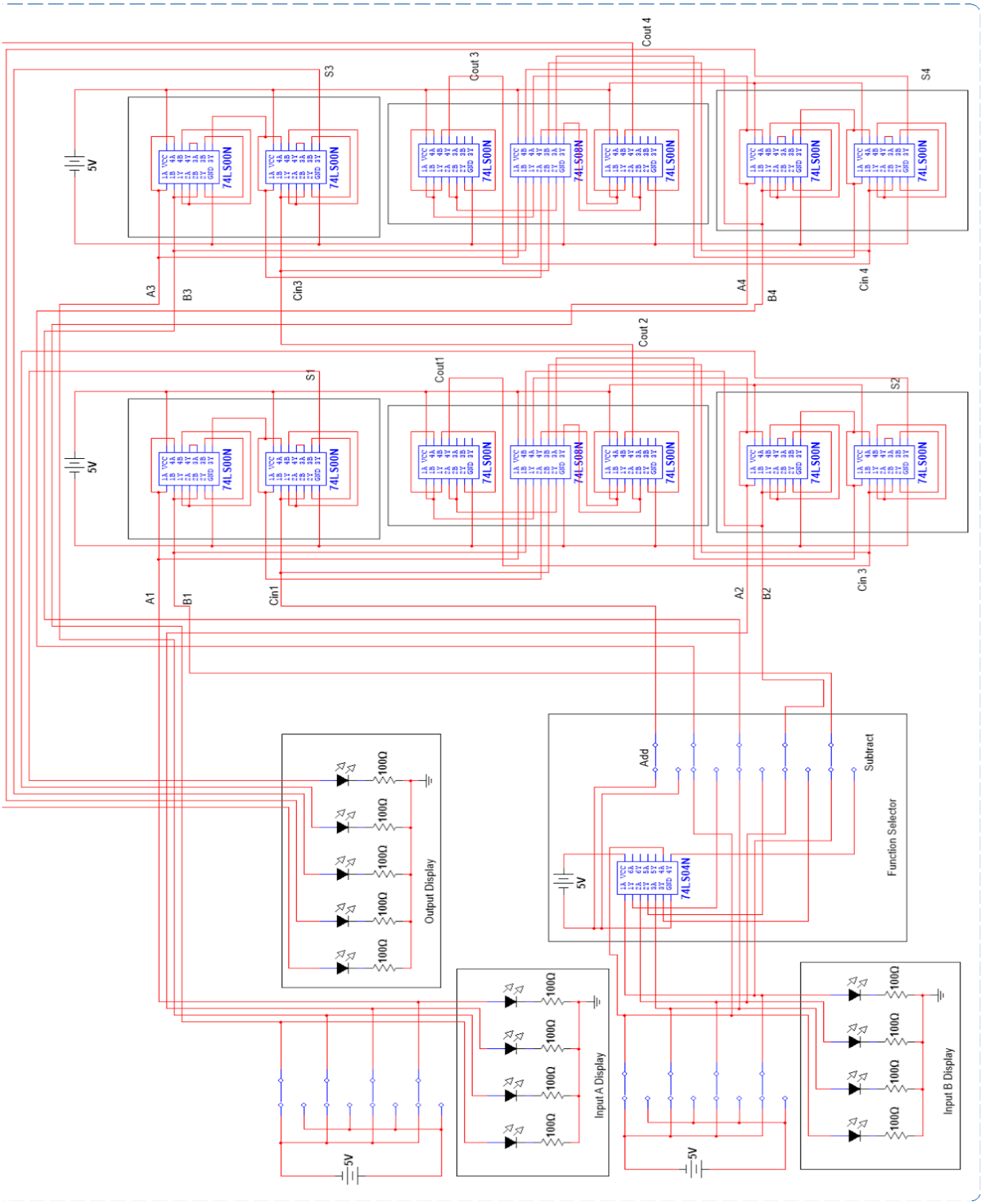

Diagram 9: Full Circuit Diagram of 4-Bit Adder/Subtractor

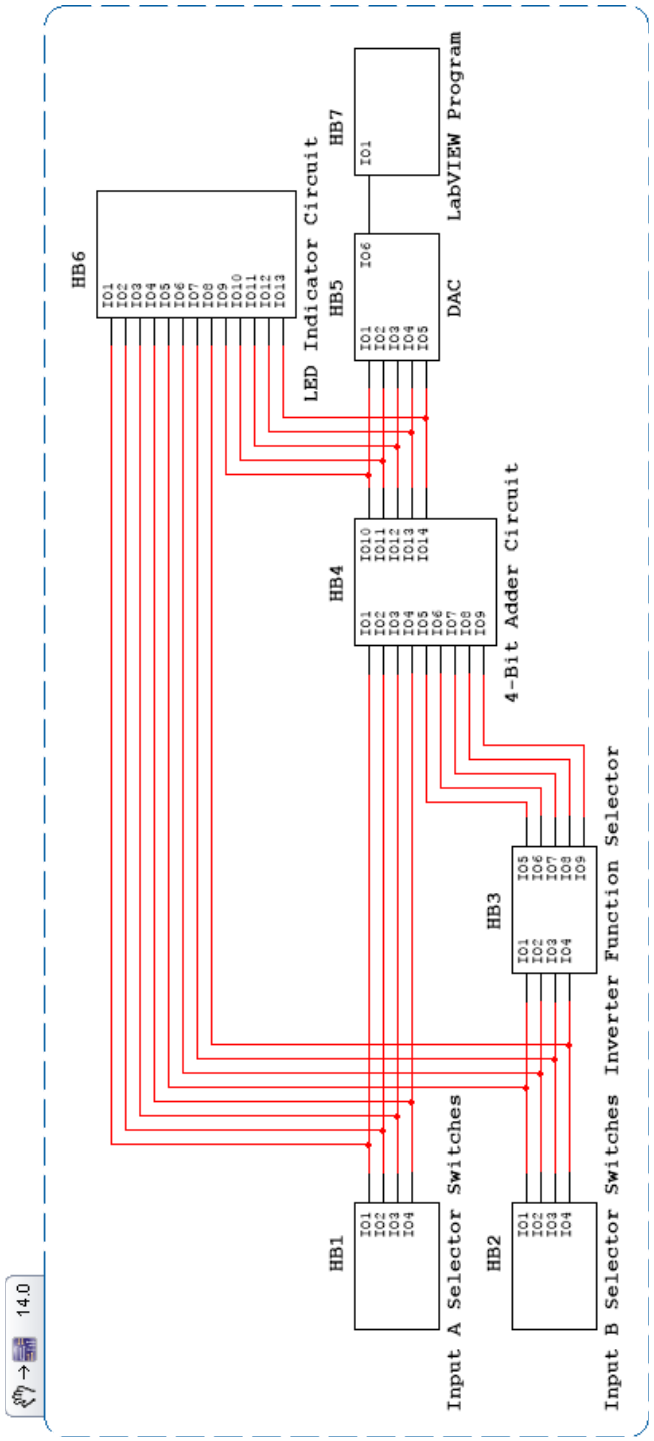

Diagram 10: Block Diagram of 4-Bit Adder/Subtractor