Riley Guidry
Alberto Rosas
Dang Tran
CMPE 480
8 May 2024

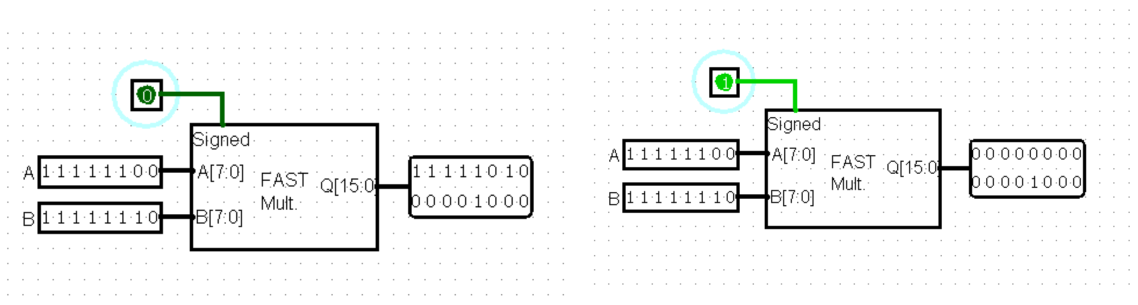
Fast Multiplier

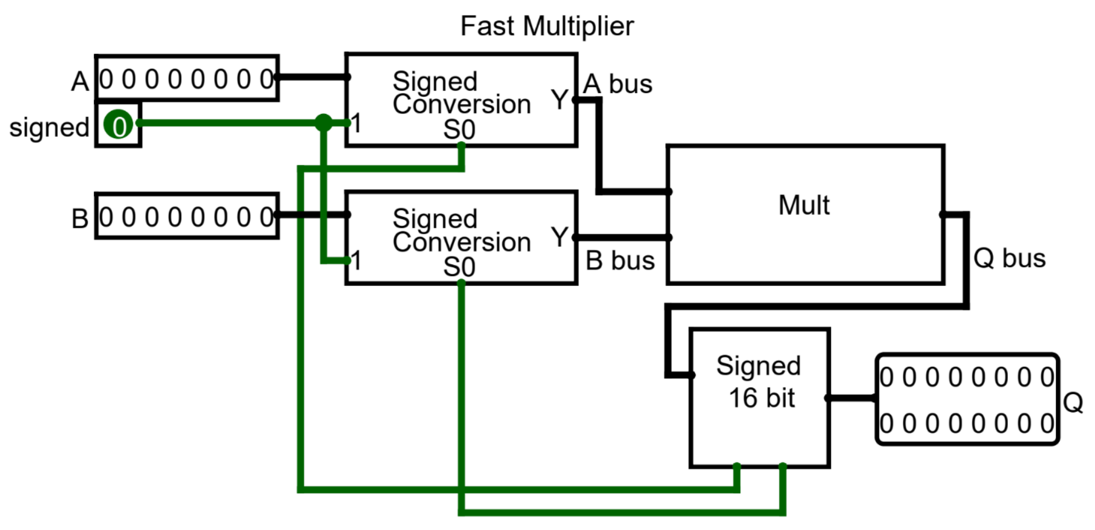
Introduction

Computer chips were made to perform mathematical calculations for the ease of humans, and they were built to do it fast. So it is an area of intrigue for Computer Engineers to consider how best to do an operation like multiplication. Because multiplication is repeated addition, it can be considerably slower than any of the other operations a CPU may perform, up to 32 times for a standard MIPS processor if one addition is performed every clock cycle. A much faster method of multiplication is the Wallace tree, or a flow-through multiplier, which completes the whole process in one clock cycle, "flowing through" the whole system like an unmitigated waterfall.

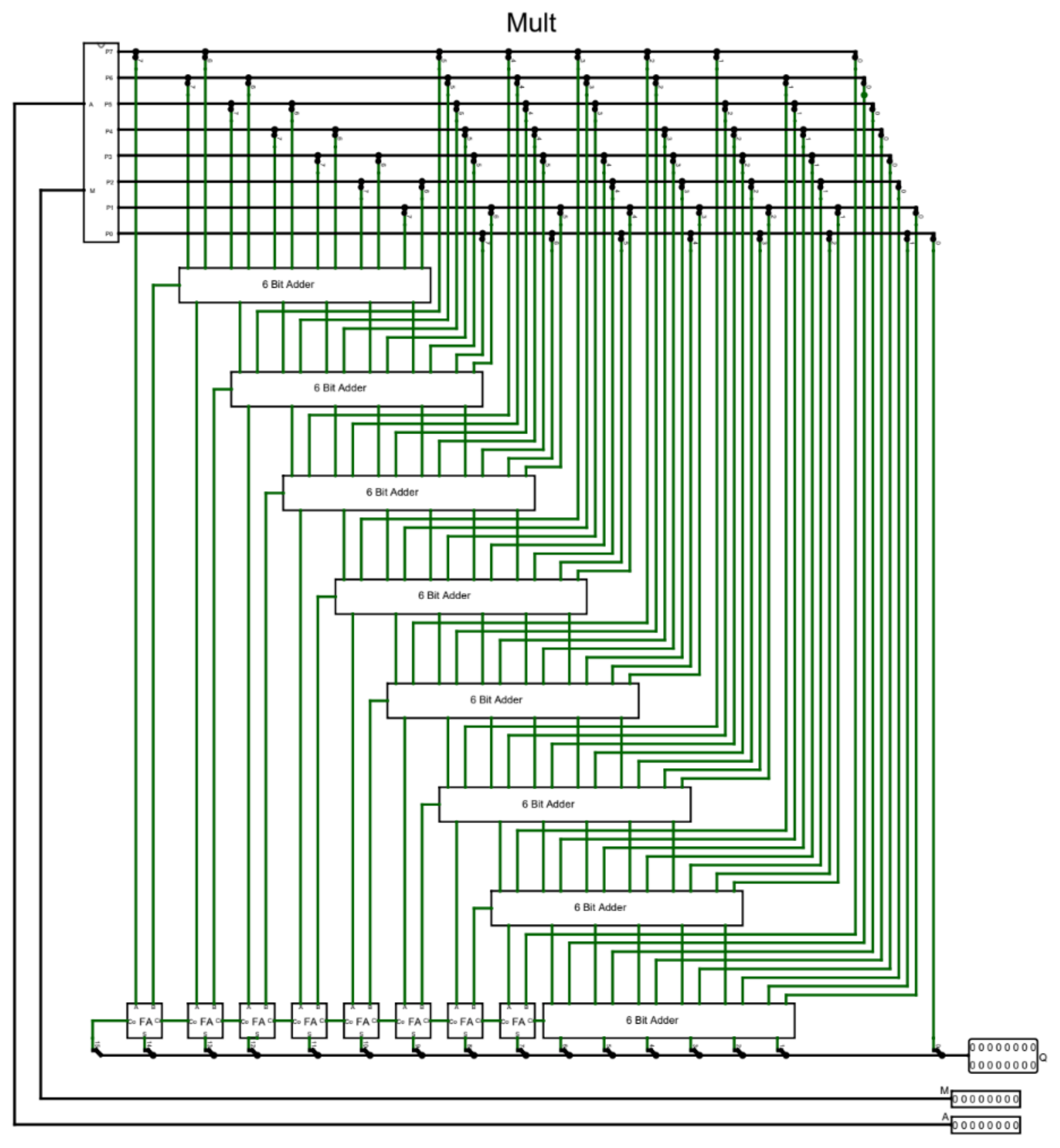
Specification

Our goal was to implement a Wallace tree multiplier in Magic, the VLSI layout software. For simplicity, we wanted our multiplier to multiply two eight-bit numbers together, to have an output of 16 bits. We also wanted to control whether the multiplier understood the sign of an input, in two's complement form, or if it treated the two inputs as unsigned integers. Because a Wallace tree performs solely unsigned multiplication, we need to manipulate the data before it enters the tree, and then manipulate the data that comes out of it depending on the state of the inputs and if we are doing signed/unsigned multiplication.

Top-Level Block
 0 Signed A 11111100 B 11111110 A[7:0] B[7:0] FAST Mult. Q[15:0] 11111010 00001000 1 Signed A 11111100 B 11111110 A[7:0] B[7:0] FAST Mult. Q[15:0] 00000000 00001000
A: 8-bit input (Multiplicand) *B: 8-bit input (Multiplier)* *Signed: 1-bit input where 0 represents unsigned multiplication and 1 is signed multiplication* *Q: 16-bit Product*

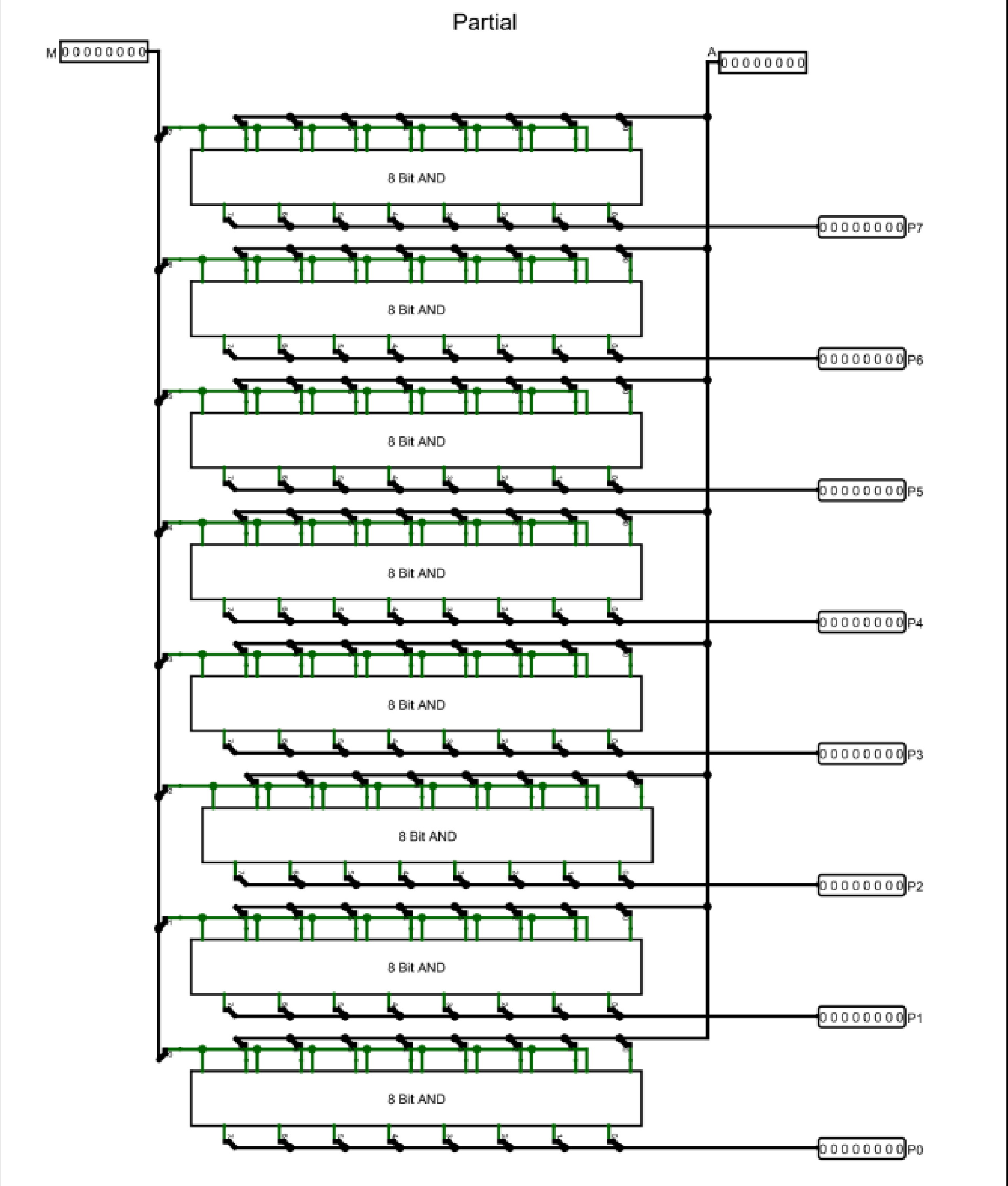
Logisim Schematic
 Fast Multiplier A 00000000 signed 0 B 00000000 Signed Conversion 1 S0 Y A bus B bus Mult Q bus Signed 16 bit 0000000000000000 Q
A bus: 8-bit bus(Multiplicand) *B bus: 8-bit bus(Multiplier)* *Signed: 1-bit input where 0 represents unsigned multiplication and 1 is signed multiplication* *Signed Conversion: Chooses if the input is signed or unsigned using 2s Complement* *Mult: Computes product of A bus and B bus resulting in Q bus by using a Wallace tree approach.* *Signed 16-bit: Takes in the 16 bit Q bus and converts it to unsigned or signed* *Q: 16-bit Product*

Looking down one layer into the hierarchy the 4 main parts of the fast multiplier are depicted. As mentioned previously Wallace trees do not do Signed multiplication natively so the input is manipulated beforehand using the Signed Conversion blocks. These blocks take in the input and will only output a negative signed number if the Signed input is 1 and the leading bit of the 8-bit input is 1, as this would signal a negative number in the 2's Complement standard. Then the Mult block computes the multiplication using a Wallace tree structure, using the A and B inputs from the Signed Conversion block. After the Product is calculated we check to see if one of the inputs is negative by looking at the selector bit of the mux inside the Signed conversion block, 1 is negative 0 is positive. Using a 2-input XOR gate we can determine if the result will be negative or positive. Like in standard multiplication, the only way to get a negative number is to have a negative multiplied by a positive number. Any other signed combination results in a positive product. That is why the XOR gate checks if exclusively one or the other inputs are negative to result in a negative product.

Multiplier


Above is the schematic for the Wallace tree. A Wallace tree takes advantage of the nature of binary multiplication, in that a multiplication between two bits is merely an ANDing of them, and the results of those ANDings need to be added according to their "weight." The weight of a product is the addition of the position of the bits that produced it. For example, P0_1 is the product of (A0 AND B1) and its weight is (1 + 0 = 1). And P3_2 is the product of (A3 and B2) and its weight is (3 + 2 = 5). Products of the same weight are added to produce a sum bit, which is considered the same weight as its parents, and a carry bit, which is one weight higher than its parents. The carry bit gets added to the weight that is higher than its parents.

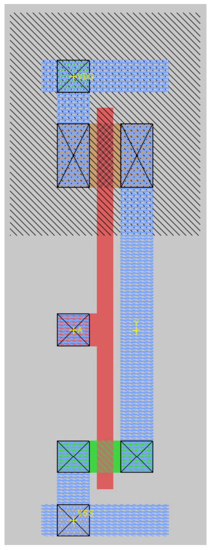
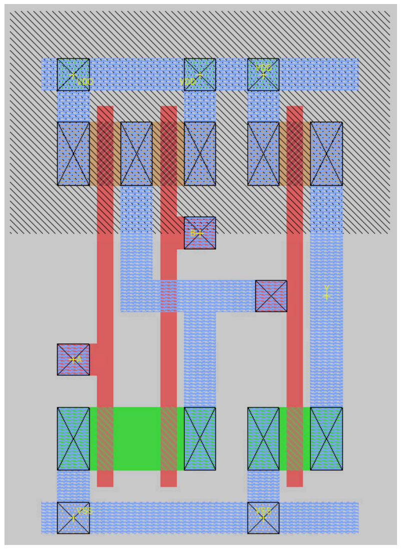
Partial Product

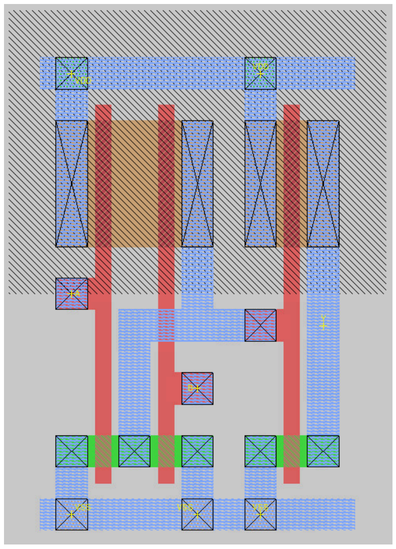
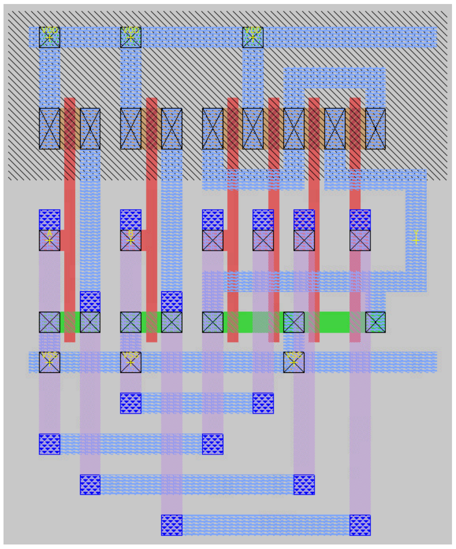


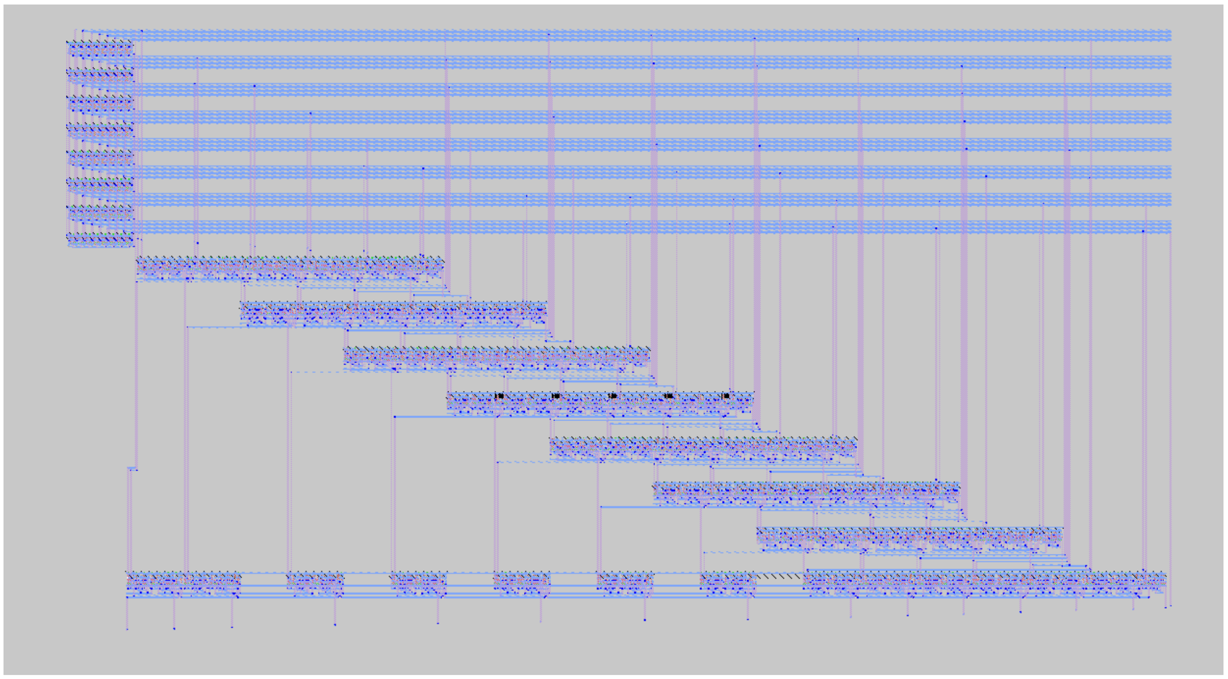
This is the array of AND gates mentioned previously.

Implementation

To realize this idea fully, we used Magic, the VLSI layout software, to lay out the chemical makeup of a microchip that operated this way. We start by making the basic gates that constitute the fundamental building blocks of the massive schematic this will eventually turn into. The list of parts made is extensive and is featured later on in this report. Certain layouts had different variations as to fulfill certain purposes in different cases when needed. First we started with the basic gates then created half and full adders. After an 8 bit Inverted and Adder were made along with an 8 bit 2x1 mix to create the 2s Complement layout that converts inputs from signed to unsigned. It covers an unsigned integer to signed if both the Signed input is on and the number has a leading 1 (negative) by outputting high from an AND gate choosing the second input from the mux(signed output). Then 8 bit ANDer was created to make the Partial Product component that is used in the multiplier. The multiplier uses a 6 Adder with no Cin and Full adders. Finally the 16 bit 2s Complement lets us choose between an unsigned or signed product. This was built from the 2 complement cells and an xor2 that chooses the input based on if either one of the inputs was negative.

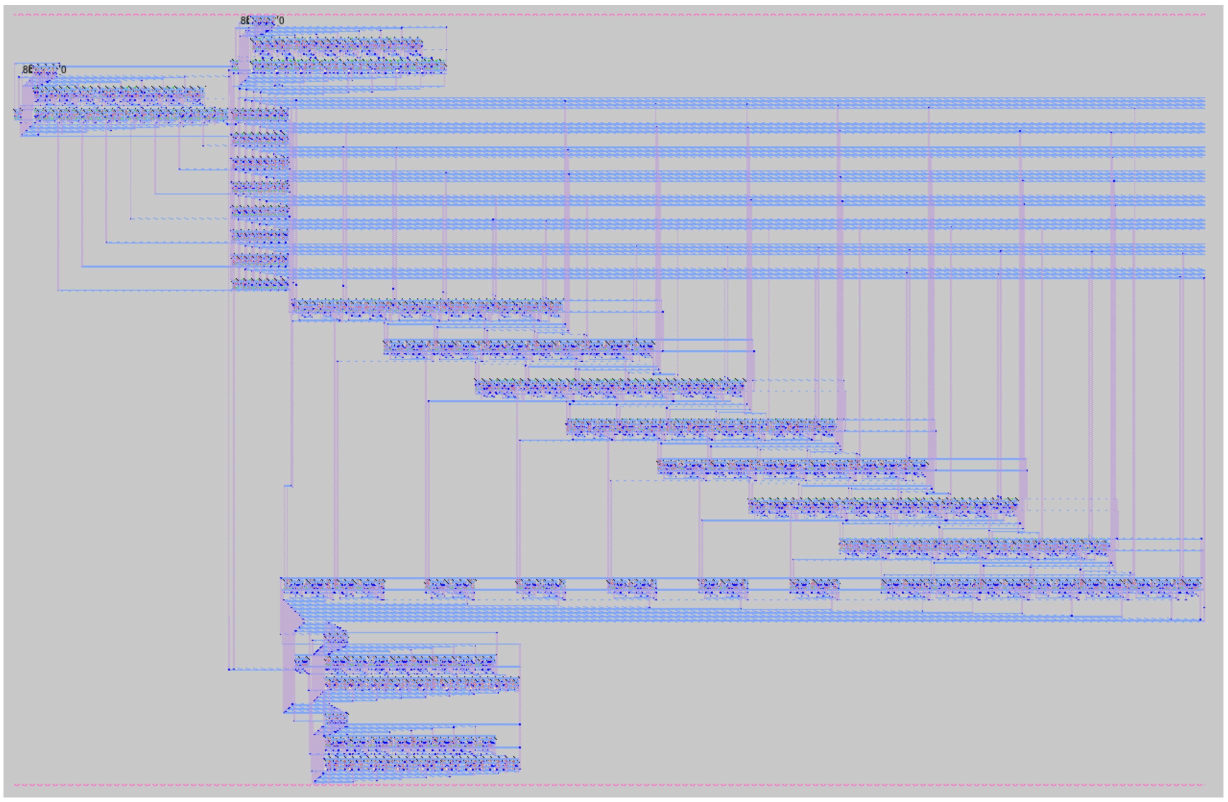
Basic Logic Gates in Magic	
 INV	 AND-2

 OR-2	 XOR-2

Multiplier


This is what it looks like when the AND signals are pulled out into the ADDers according to their weights. If you look above at Logisim schematics, the similarities in structure are obvious. There was a naming error where the 6 Adder was called Add5 in the Magic layout tool, a misnomer if you will.

Final Multiplier



Conclusion

We have successfully implemented the schematic for 8-bit fast multiplication into layout using Magic. The layout takes a step size of 20 to complete a multiplication. The suspected reason for this is the resistance and capacitance of the VDD and VSS wires to join everything together. With an additional metal 3 layer we can cut down on both resistance and capacitance, this layer was implemented but the functionality is unknown. There are some ways we can improve our layout including beta balancing, cutting down on area used, and fully implementing the metal 3 layer. All in all, implementing an 8 bit fast multiplier has taught us the basics of layout design and given us new skills to use in industry.