Final Project Report: Four Bit Calculator

Ryan Barker & David Saxton

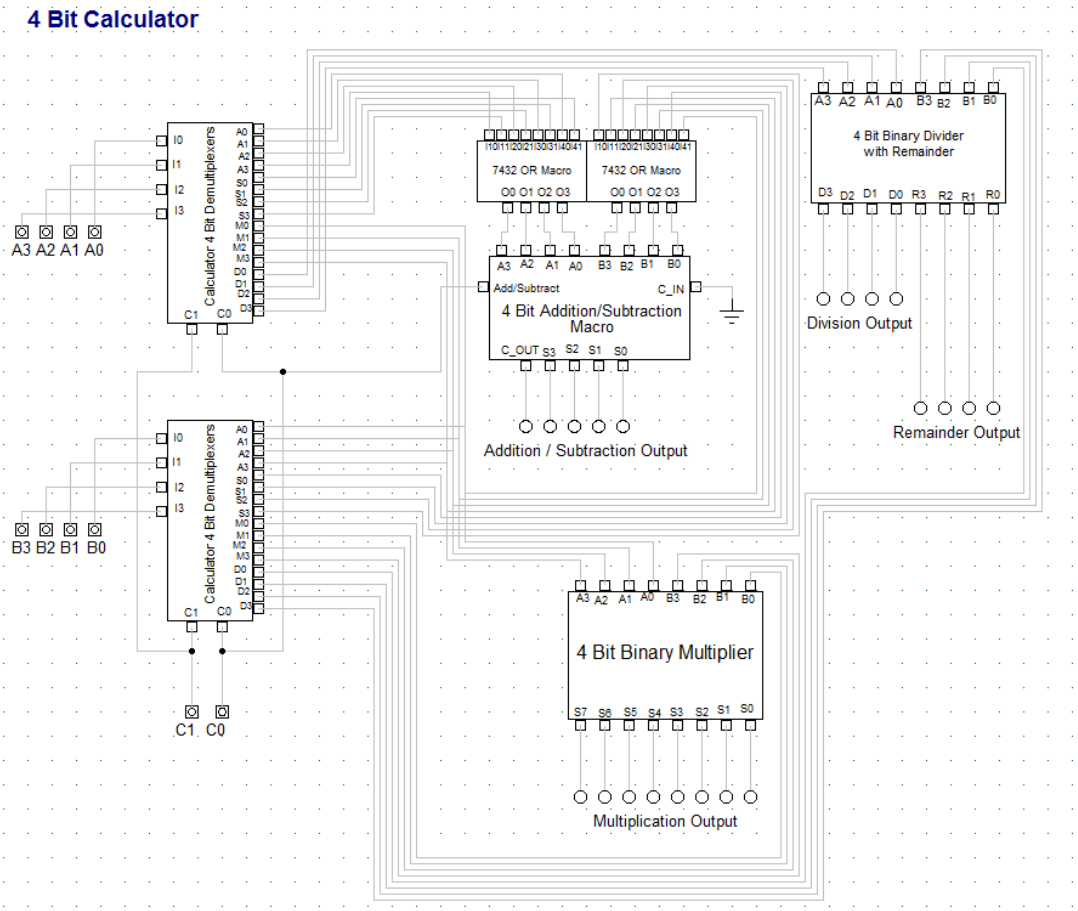
Ranajeet Anand

ECE 209 Lab, Section 014

11-12-2013

Overall Function and Design

Figure 1



The 4-bit four function calculator circuit was designed to take two 4-bit numbers and two control bits and then performs one of four operations (addition, subtraction, multiplication, or division) based on the "decision" made by the control bits inputted. This decision follows the scheme of:

De-Multiplexer Control Bit Truth Table		
C1	**C0**	**Operation**
0	0	Addition
0	1	Subtraction
1	0	Multiplication
1	1	Division

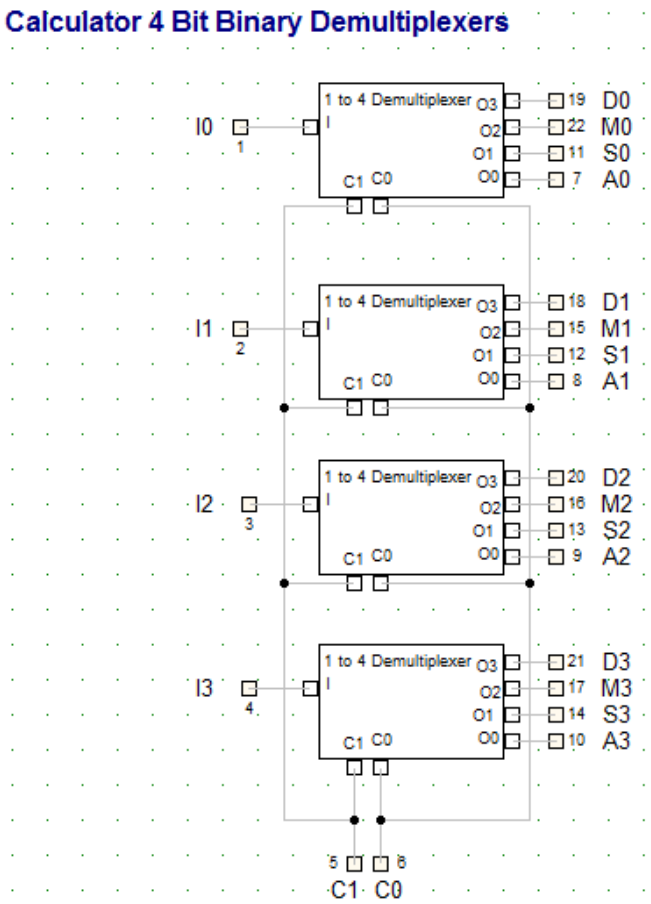
After the de-multiplexers make a "decision", they allow the circuit to perform that operation by routing the two 4-bit inputs to one of three function macros that models the selected operation. These function

macros are a 4 bit addition/subtraction macro, a 4 bit multiplier macro, and a 4 bit division with remainder macro. The overall circuit is shown in Fig. 1.

De-Multiplexers – Function Selection

The calculator circuit has two de-multiplexers that take 4 inputs, 2 control bits, and have 16 outputs each. The upper de-multiplexer chip takes the 4-bits of the first input number and the lower de-multiplexer takes the 4-bits of the second input number. These de-multiplexers send the inputs out over 4 of 16 output lines at a time based on their control bits. This is accomplished using four 1x4 de-multiplexers inside each of the larger de-multiplexer chips. The contents of the larger de-multiplexers are shown in Fig. 2.

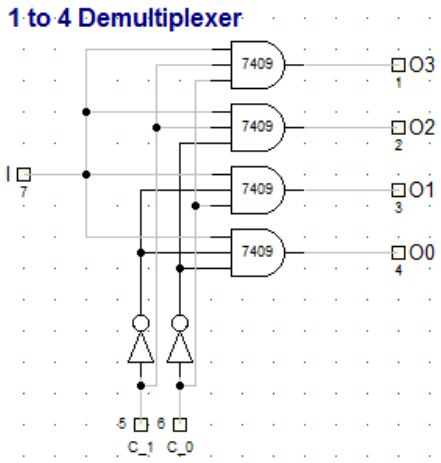
Figure 2



Each of the 1x4 de-multiplexers take a single bit of the input number and send it out over one of four lines. Each of these lines corresponds to a different operation. For example, if the larger de-multiplexer received C1 = 0 and C0 = 0 then the de-multiplexers would pass the I0, I1, I2, and I3 inputs through A0, A1, A2, and A3, respectively. The outputs on the circuit diagrams are labeled according to their corresponding operations: A for addition, S for subtraction, M for multiplication, and D for division.

Within each of the 4x1 de-multiplexers is a row of 4 AND gates. These AND gates and two NOT gates each form a different minterm of the two bit binary function F = 1 with the two control bits, which only allows one of the outputs to be active at a time dependent on the control bits. In the larger de-multiplexer chip both control bits go to each of these 1x4 de-multiplexers, which each handle a different bit of the four bit input number. Fig. 3 shows the inside of each 1x4 de-multiplexer.

Figure 3

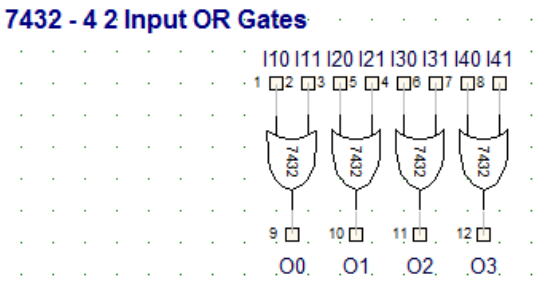


OR Macro – Combining Inputs

The simulation of the 7432 OR Gate macro only exists in the circuit due to a limitation of the Digital Works software: In Digital Works, it is impossible to tie multiple outputs from the same chip to the same input of another chip (Which is bad practice on real breadboards). This poses problems because, as discussed earlier, the 4 bit de-multiplexers route addition and subtraction over different outputs, but they need to go to the same chip, since one of the function macros is responsible for both the addition and subtraction operators.

To solve this problem, the 7432 macros were created to OR each respective addition and subtraction bit into the same input of the 4 bit addition/subtraction macro. For instance, A0 and S0 from the first 4 bit de-multiplexer are OR'ed together into A0 in the 4 bit addition/subtraction macro. Note that there is one 7432 chip for each 4 bit de-multiplexer chip. The 7432 OR Gate macro is shown in Fig. 4.

Figure 4



Addition / Subtraction Function Macro

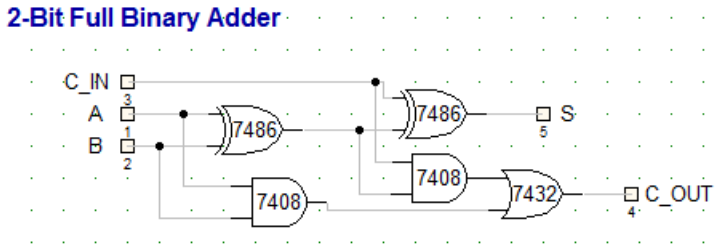
The 4 bit addition/subtraction function macro works very similairly to the addition / subtraction circuit designed in lab 5. It works addition utilizing full adders and works subtraction using the idea of double-complements, since the double-complement of addition is subtraction.

Inside, there are four 2 bit full adders chained together. These work by doing two operations at once:

- Firstly, the inputs of each adder, "A" and "B", are summed together with a carry input bit, "C_IN", from the previous adder in the chain to result in a "S" output bit. The carry input of the first adder is tied to ground. This allows the 2 bit full adders to work in tandem as one four bit full adder. XOR gates are used to sum each set of three inputs together, since the truth table for the XOR operation matches that of the binary addition identities. Essentially, the XOR gates take the two four bit inputs (A3 through A0 and B3 through B0) and sums them together into one four bit output (S3 through S0).
- Secondly, since binary 1 + 1 = 0 carry 1, the full adders also have an another function to perform: To determine if a value needs to be carried out of the current adder into the next adder in the chain. A carry out of one will result when either both of the inputs is one or the sum of the sum of the inputs and the carried in value is one, so both of these combinations are AND'ed together in the macro design. These AND gates are then taken to an OR gate which is tied to the carry output bit, "C_OUT".

Fig. 5. shows the inside of each 2 bit full adder.

Figure 5



In summary, the 4 bit full adder part of the addition / subtraction macro takes four inputs each for two four bit numbers (A3 through A0 and B3 through B0) and a carry input for the first 2 bit full adder in the chain (C_IN). It then produces four sum outputs (S3 through S0) and a carry output for the last adder (C_OUT). C_OUT was used as the most significant bit of the resultant sum since adding two binary four bit numbers can produce 5 bit numbers. For instance, decimal 15 + decimal 15, two binary four bit numbers, results in decimal 30, a binary 5 bit number.

To accomplish subtraction with the same chip, the "S" outputs and the "A" inputs of the 2-bit full adders are also all connected to XOR gates. The other input of all of these XOR gates is to a control bit, called

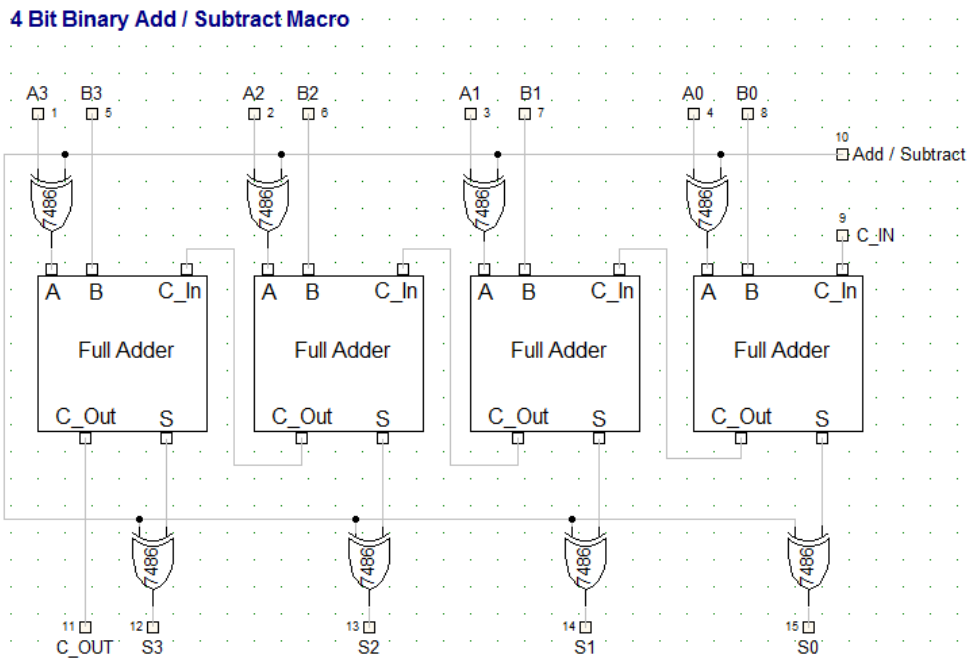
"Add / Subtract". The combination of these XOR gates and the control bit implement the idea of double-complementation to what would normally just be a 4 bit full adder, by utilizing the following XOR identities:

$$X \oplus 0 = X$$

$$X \oplus 1 = X'$$

This allows for the "A" input and "S" output to be inverted when the control bit is high, causing double-complement addition to occur. In other words, it allows the chip to enter "addition mode" when the control bit is low and "subtraction mode" when the control bit is high, so the control bit needed to be tied to an input that follows this scheme. Recall from the "Overall Function and Design" section that C0, one of the control bits for the de-multiplexers, is low in addition mode and high in subtraction mode, so it is also used as the control bit for this macro. Finally, note that the carry output in the Addition / Subtraction macro is left untouched because the subtraction of two four bit numbers will never result in a 5 bit number. Fig. 6 shows the complete Addition / Subtraction Macro.

Figure 6



Multiplier Function Macro

The 4-bit multiplier function macro contains four separate 4x1 bit binary multiplier circuits and three 4-bit binary adders. Each of the 4x1 multiplier circuits take a different single digit of B and multiplies it across each of the digits of A. Then each of these products are shifted to the left (appropriately) and summed. The product of the least significant digit is not shifted, the product of the digit to the left is

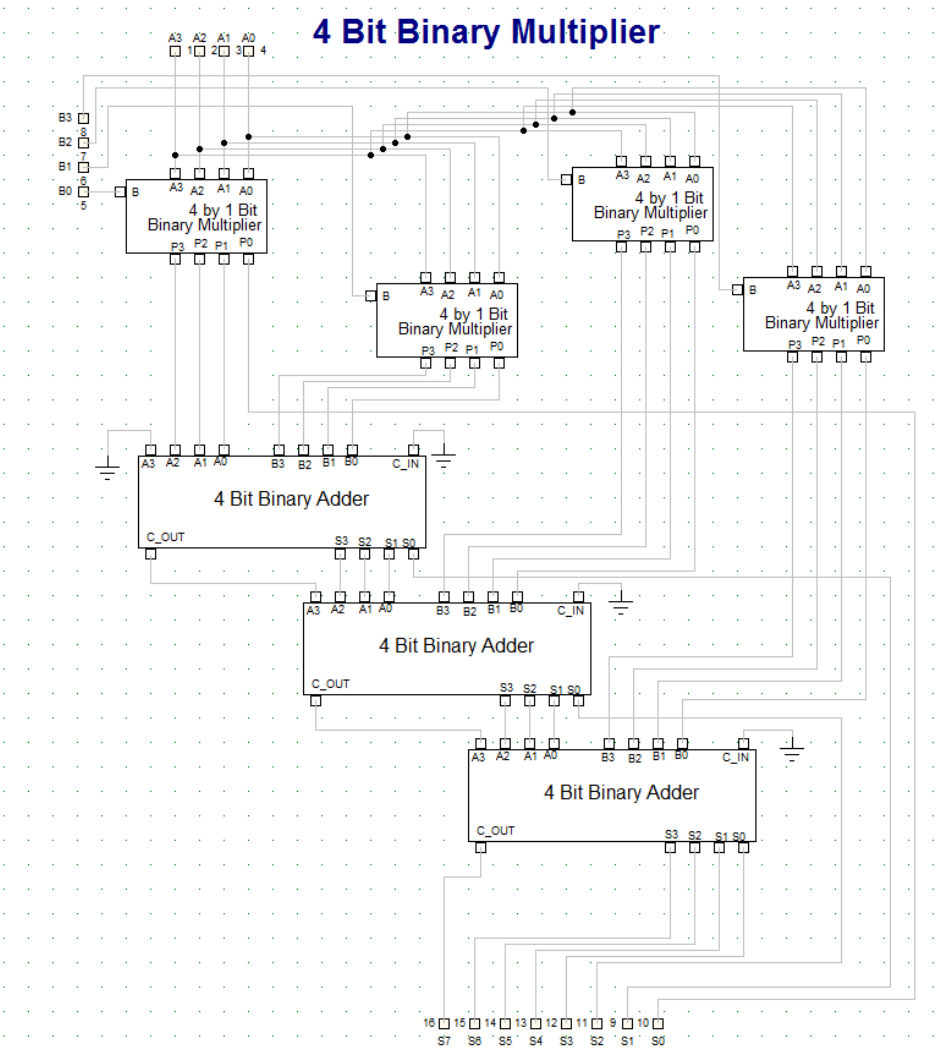
shifted one place to the left, and the product of the digit two places to the left is shifted two places to the left, and so on (see example below).

$$1110 \times 0101 = 1 \times (1110) + 00 \times (1110) + 100 \times (1110) + 0000 \times (1110) =$$

$$1110 + 00000 + 111000 + 00000000 = 01000110$$

In the circuit this shifting is accomplished by tying the least significant sum of the first multiplier (P0) directly to the least significant output (S0) and tying its sum and carry outputs into the inputs of the next 4-bit adder. The least significant sum out of the first adder (S0) is tied to the second least significant digit output (S1) while the rest of its sum and carry outs are tied to the second 4-bit adder. Then the third shift is accomplished by tying the least significant sum out of the second adder (S0) to the third least significant output (S2). The final outputs from the third 4-bit adder are all connected directly to the five most significant outputs (S7-S3). The multiplier function macro is shown in Fig. 7.

Figure 7



The 4x1 bit multiplier circuit works on the idea that the operation of AND gates is identical to one bit binary multiplication (1×1=1, 1×0=0, 0×0=0). Inside each of the 4x1 binary multipliers is a row of 4 AND gates (shown in Fig. 8). One input of the AND gates is tied to a single digit of number B while the other four inputs are each of the 4 digits in number A. The 4-bit binary adders used in the multiplication function macro perform addition in the same manner as the addition function macro discussed above. The 4-bit binary adders used in the multiplication macro are shown in Fig. 9.

Figure 8

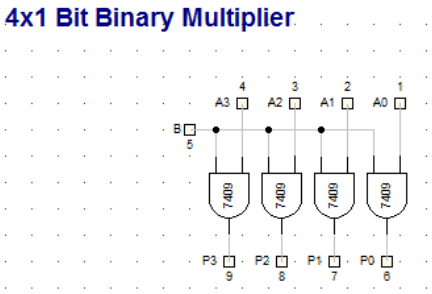
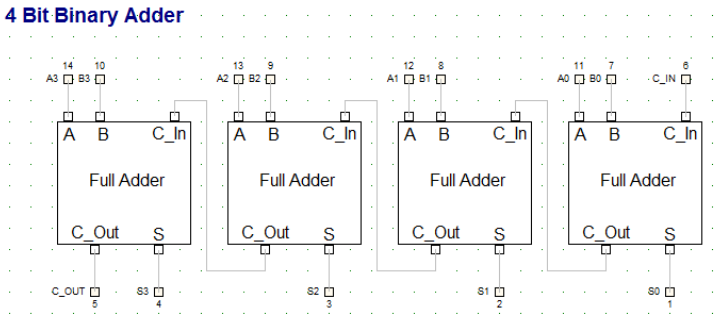


Figure 9



Divider with Remainder Function Macro

Multiplication is very frequently called “repeated addition”. That is to say that:

$$2 \cdot 3 = 2 + 2 + 2 = 6$$

Though it is not often thought of like this, in a similar manner, division is actually the number of times you can repeatedly subtract the divisor from the dividend while still obtaining a number greater than or equal to zero. The remainder of this process is then whatever is left over. For instance:

$$6 \div 2$$

$$R = 6 - (2 + 2 + 2) = 0$$

$$6 \div 2 = 1 + 1 + 1 = 3$$

Notice how two can be subtracted from six an even three times, yielding a result of zero. Six divided by two then becomes the number of times one subtracted two, or 1 + 1 + 1, which is 3. Since nothing was

left over, the final result is six divided by two equals three remainder zero. In the case of an example with remainders:

$$7 \div 3$$

$$R = 7 - (3 + 3) = 1$$

$$7 \div 3 = 1 + 1 = 2$$

Here, three will go into seven twice and still yield a positive number; one. Seven divided by three, like before, then becomes the number of times one subtracted three, or 1 + 1, which is 2. This yields a final result of two remainder one. One will find this result if they try seven divided by three using the traditional, long-hand method as well.

The reason such basic mathematical methods for division must be established is that division as normally known is a *complex decision-based process*. That is to say that it is a process that involves non-binary decisions, or decisions that can result in more than two possibilities (In this case, greater than, less than, or equal to). Furthermore, it also involves conditional looping, or going back to a certain step until a condition is satisfied. Neither of these things are easy to implement into a circuit, so building a function macro for a binary divider that behaves in this manner is very difficult. On the other hand, the repeated subtraction method only involves one decision: Stop when the remainder will become negative. This is much more feasible to implement with logic based chips, so it is the easiest way to build this function macro.

As stated above, the divider function macro must subtract the divisor from the dividend until the remainder becomes negative to work correctly (At which point it will stop subtracting), because the result will be the number of times that the divider subtracted the dividend. This can be implemented in a macro with a sequence of 4 bit subtractors and adders, but the question then becomes how many of each to include, since there is no easy way to create a logic loop using simple logic-based chips.

Section A: The Subtractors

To answer this problem, it is necessary to examine the subtractors first. Specifically, the maximum number of subtractions that can be done when dividing four bit numbers needs to be determined. Common sense dictates that this will occur for the largest value in the four bit system divided by the smallest value capable of being a divisor in a four bit system, or binary fifteen (1111) divided by binary one (0001). Doing this in decimal yields:

$$15 \div 1$$

$$R = 15 - (1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1) = 0$$

$$15 \div 1 = (1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1) = 15$$

The result is a fifteen. However, remember that the divider must subtract until it “sees” a negative so it knows to stop subtracting, so the maximum number of subtractor chips necessary for a four bit system

is sixteen. In fact, the maximum number of subtractors necessary for a divider of a n bit system utilizing this method is two raised to the n: Notice that two to the fourth is sixteen.

Regardless, the only part of the first step in this process left to determine is how to design each individual subtractor chip. The actual subtraction part of the chip can be designed in the same method described in the "Addition / Subtraction Macro" section of this report, but a problem does arise: The divider still has to stop subtracting as soon as it sees the remainder become negative.

Addressing this problem requires an additional output and input to the subtractor chip that has already designed: The chip needs an input to tell it if a previous subtraction resulted in a negative number and an output to tell other subtractors down the line if its subtraction resulted in a negative number. Before either of these things can be done, however, the big challenging question becomes:

How can it be determined if subtraction of two, positive, four bit numbers yielded a negative result?

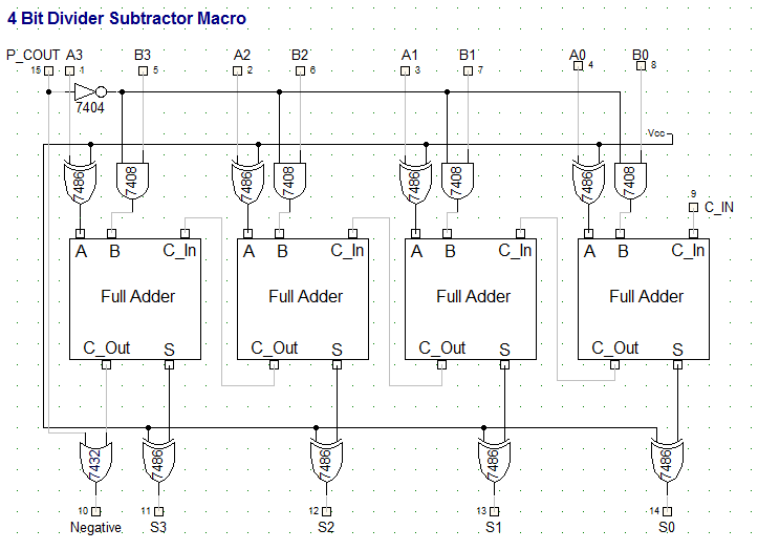
Thankfully, since the system of numbers is only four bits, this can in fact be determined by the carry out bit, C_OUT, of the convention subtractor. If this bit outputs a one, then the subtractor "knows" that its subtraction resulted in a negative number. In other words, the name of the carry output on the convention subtractor needs to be changed to "Negative" for the divider's subtractors, since it is what determines if the subtraction done by the subtractor was negative.

However, this is not the only change that needs to be made to the subtractor. As mentioned previously, an addition input, called previous carry out, or "P_COUT", which takes the "Negative" output from the previous subtractor in the chain needs to be added into all of the subtractor macros (This input will be grounded for the first subtractor in the line). Not only that, but a one on this input must pass down to the next subtractor in line and stop the subtraction of the divisor from the remainder.

To pass the one down to each "Negative" output on each subtractor, the P_COUT input can simply be joined to the "Negative" output of each subtractor by means of an OR gate. This means that if a one comes on the previous carry out input, a one will come out on all of the remaining "Negative" outputs.

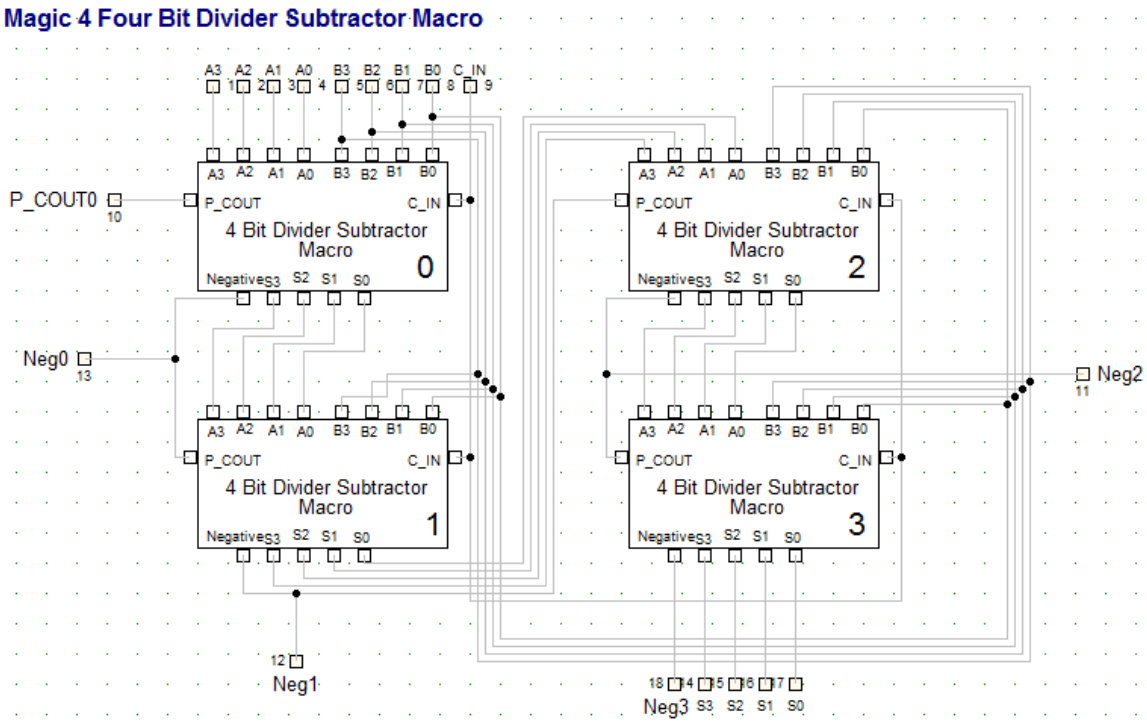
To stop the subtraction of the divisor from the remainder, the divisor input can be AND'ed with this previous carry out input NOT'ed. This says logically, "If the last subtraction was negative, subtract a zero from the remainder, not the divisor", which leaves the remainder unchanged as it passes through the subtractor. As a result, the inside of the subtractor macro needed for this operation is shown in Fig. 10.

Figure 10



However, since 16 divider subtractor macros on the base level of the divider function macro would be very disorganized, a more efficient way to design the chip is to group the subtractor macros in their own macros in groups of four. They will be chained together appropriately and all of the necessary outputs (Namely, the negative outputs), will be present on the chip containing all 4. Fig. 11 shows the inside of this chip.

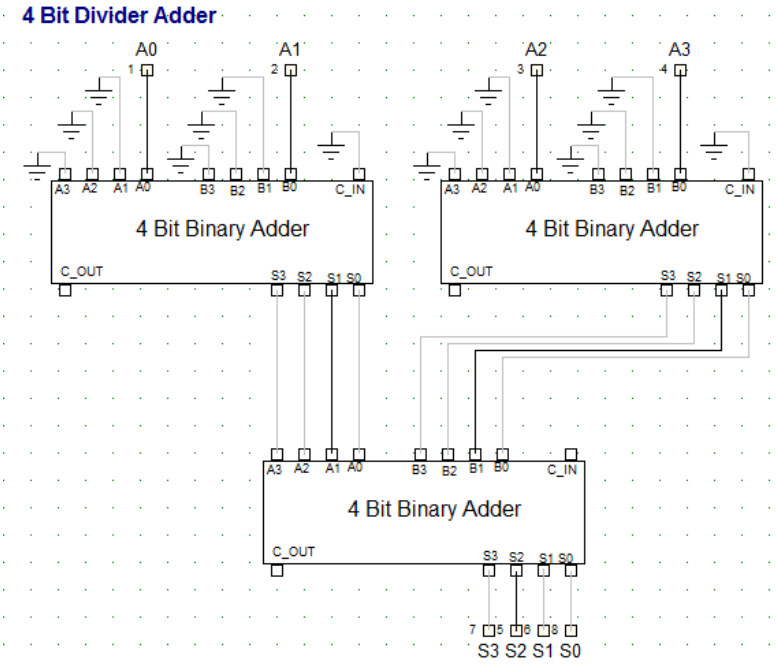
Figure 11



Section B: The Adders

Thankfully, the adders in this chip function in a much less complicated manner. The first set of adders is needed to sum the number of times a subtraction yielded a positive number. The "Negative" outputs of all of the subtractors output a zero if the subtraction inside of that subtractor chip yielded a positive number, so if these "Negative" outputs are NOT'ed and then inputted into special adders that only add ones together, and then the result of these additions are added together, it is easy to obtain a count of the number of positive subtractions. This "count" is the correct result for the division. The inside of "special adders" needed for this is shown in Fig. 12, and the adders chained after them are the same ones shown in Fig. 9.

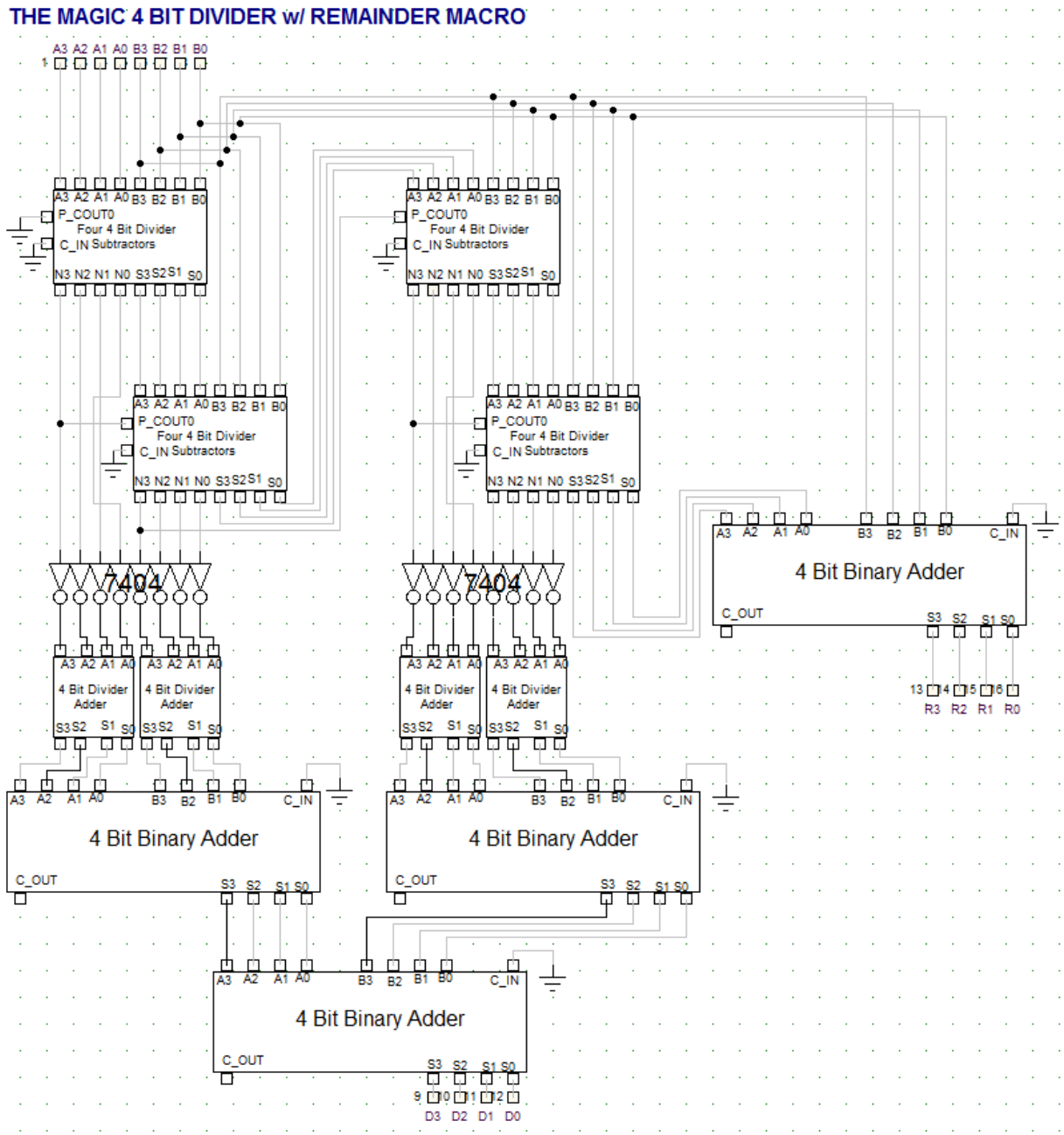
Figure 12



The second adder that is needed is necessary for the remainder. Recall that the subtractors subtract until they "see" a negative output, but the still do that subtraction, making the remainder negative. The value for the remainder required will be its value just before this occurs, so to obtain the remainder, all that is necessary is to use the adder in Fig. 9 to add the divisor back to the output of the subtractors once. This will produce the correct remainder.

Putting all of this together, the inside of the correct function macro for the divider will be as shown in Fig. 13.

Figure 13


THE MAGIC 4 BIT DIVIDER w/ REMAINDER MACRO
A3 A2 A1 A0 B3 B2 B1 B0
P_COUT0
Four 4 Bit Divider
C_IN Subtractors
N3 N2 N1 N0 S3 S2 S1 S0
7404
4 Bit Divider Adder
4 Bit Binary Adder
C_IN
C_OUT
S3 S2 S1 S0
R3 R2 R1 R0
D3 D2 D1 D0