This is the design of the 8-bit ALU by using 8 1-bit ALUs.

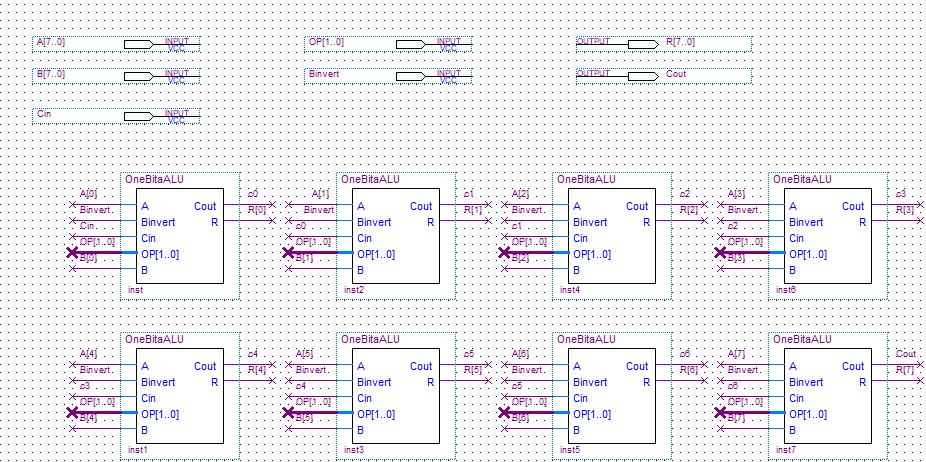


Figure 1: The whole 8-Bit ALU Circuit Design

Major Inputs

* A[7..0] (A[0] – A[7]) – First 8-bit binary number input for the circuit. Each index corresponds to each digit of the number.
* B[7..0] (B[0] – B[7]) – Second 8-bit binary number input for the circuit. Each index corresponds to each digit of the number.
* Cin – The 1-bit binary number as “carry input”, needed for the 8-bit adder inside to work properly.
* OP[1..0] – The “Operation control” input. A 2-bit binary number input to control the circuit’s function.
* Binvert – The “Binvert” is a 1-bit binary that provides the subtraction function to the ALU as it inverts the second input as the name suggests.

Major Outputs

* Cout – The “Carry Output” bit. A component of the adder circuit. Similar to Cin but this is an output.
* R[7..0] (R[0] – R[7]) – The main 8-bit binary number output of the circuit.

Circuit connections

* Similar to the in-class work that was done, the four-bit adder from one bit adder. The 8-bit adder circuit was connected in a similar fashion as the one mentioned but with a few key differences as there are more inputs than a traditional binary adder.

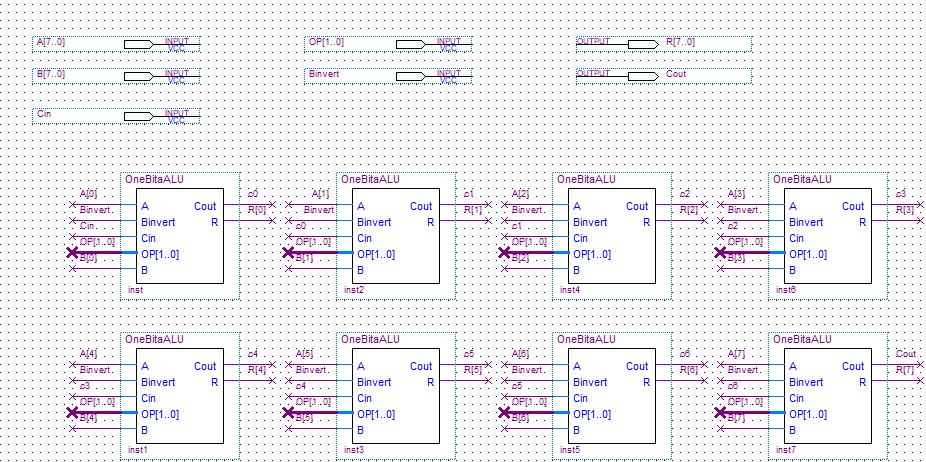


Figure 2: Individual Circuit Connections

* From the picture above, the “Cout” of the left circuit block becomes the “Cin” of the right circuit, similar connections to the 4-bit adder.
* The Binvert and OP inputs are the same for all 8 circuit blocks because the circuit there can only be one operation and one binvert control for each input.

Simulation

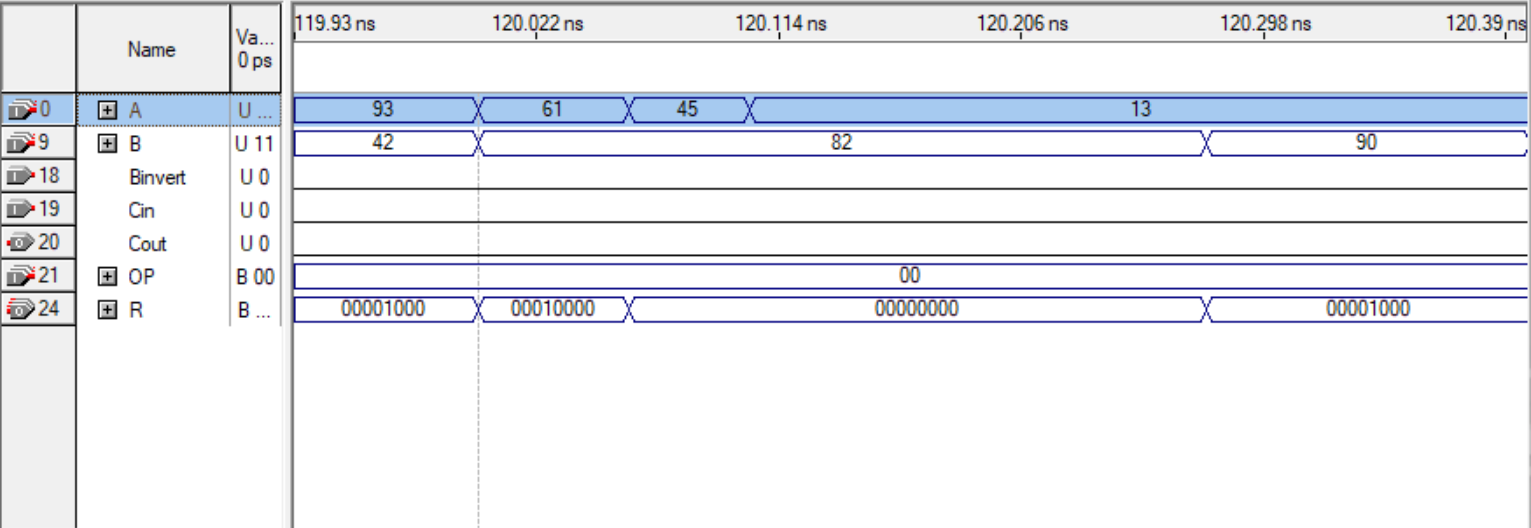


Figure 3: The result of the simulation when "AND" is the operation.

* The “AND” operation compares each digit of the input values. This is best explained by showing the input and output below:
  + Number 01011101 (93) and 00101010 (42) outputs the result of 00001000 (8).
  + Number 00111101 (61) and 01010010 (82) outputs the result of 00010000 (16).
  + Number 00101101 (45) and 01010010 (82) outputs the result of 00000000 (0).
* From the three examples above, it is obvious that when the digit of a given number is the same as another, output one as the result. (Look at the red highlights from the examples). If the digit is the same, output one and if not or both is zero, output zero. This behavior is the same as the “AND” truth table.

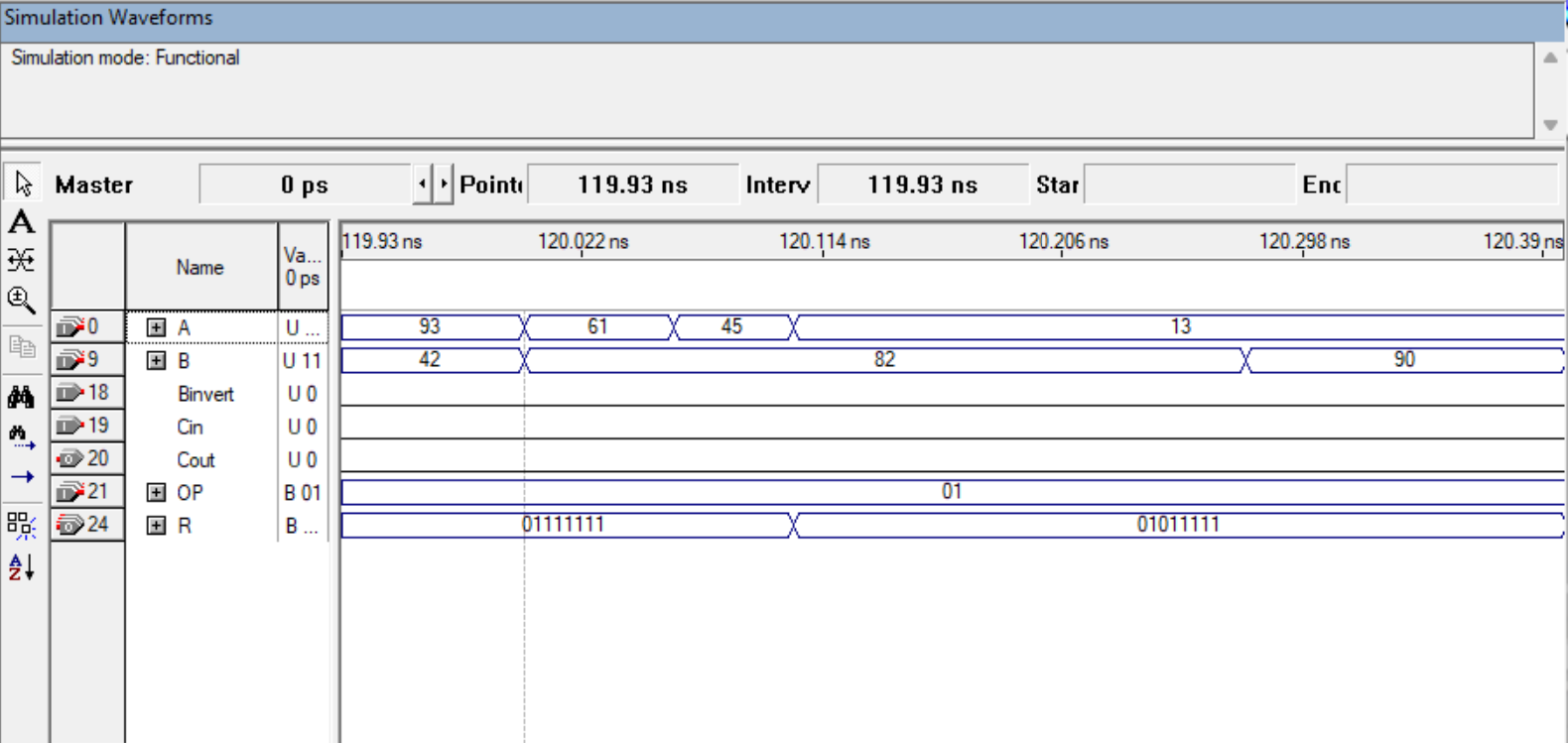


Figure 4: The result of the simulation when "OR" is the operation.

* The “OR” operation compares each digit of the input values. This is best explained by showing the input and output below:
  + Number 01011101 (93) and 00101010 (42) outputs the result of 01111111 (127).
  + Number 00111101 (61) and 01010010 (82) outputs the result of 01111111 (127).
  + Number 00101101 (45) and 01010010 (82) outputs the result of 01011111 (95).
* From the three examples above, it is obvious that when the digit of a given number is the same as another, output one as the result. (Look at the red highlights from the examples). If the digit any of the two digit is 1 (both also counted), the output is one. Otherwise, the output is zero. This behavior is the same as the “OR” truth table.

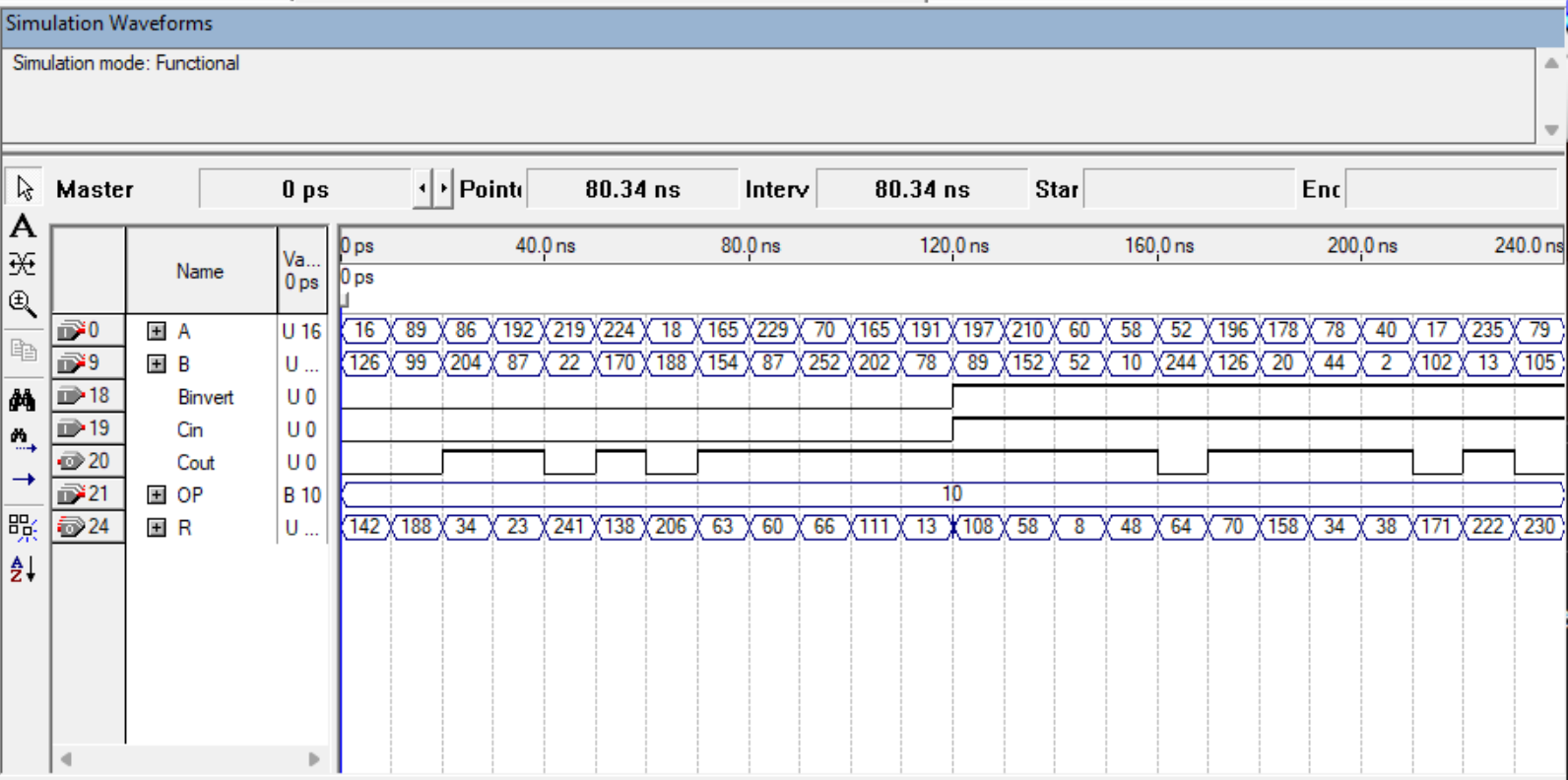


Figure 5: The result of the simulation when the operation is "ADD" and "SUB".

* This simulation is different from the other since it contains two operations in one simulation waveform. Since the ADD and SUB operation is technically the same, the differences is on the “Binvert” input. If the operation is “SUB”, the Binvert is one. If not, it is zero.
* For the “ADD” part, there are some cases that the output isn’t the expected value since there is an integer overflow. The indicator being the signal of the “Cout”. If the Cout signal is present, there is an integer overflow.
  + The integer overflow occurs when the two input adds up and exceeds the limits of (in this case) an 8-bit binary. The limit is 256.
  + If the result goes over 256, the result will start counting from zero again. The number can be calculated by using the actual result number and minus 256.
  + For an example, 224 + 170 results in 138. 224 +170 equals 394, subtracting 256 equals 138.
  + Other than cases that integer overflow happen, the addition result is what expected.
* The “SUB” part is a little different than the “ADD” part and the result could be seen when the “Binvert” signal is present. The Cin is also present on this particular operation.
  + The operation is similar to the “ADD” part since the only differences are that the second input is inverted therefore, making it a “SUB” operation.
  + Usually, integer overflow occurs on this operation more than the “ADD” operation which was the basis of how the operation works.
  + The result could simply be calculated by taking two input numbers and subtract them. This isn’t the same thing as what the ALU is doing.
  + The actual operation goes by the following: For an example, 197 – 89. When the number 89 is inverted (256 – 89) which is 167 then, 197 + 167 which is 364. Since 364 is over the limit, subtract 256 from that number and then the result is 108.