**Decoder**

* Decoder.S0, lowest bit from opcode
* Decoder.S1, 3rd highest bit from opcode
* Decoder.S2, 2nd highest bit from opcode
* Decoder.S3, highest bit from opcode

**Mux**

* MUX1.Ch0, 1 bit, result of adder into multiplexer
* MUX1.Ch1, 1 bit, result of subtracter into multiplexer
* MUX1.Ch2, 1 bit, result of divider into multiplexer
* MUX1.Ch3, 1 bit, result of multiplier into multiplexer
* MUX1.Ch4, 1 bit, result of AND into multiplexer
* MUX1.Ch5, 1 bit, result of OR into multiplexer
* MUX1.CH6, 1 bit, result of XOR into multiplexer
* MUX1.CH7, ?
* MUX1.b, 1 bit, output of MUX1 into MUX3
* MUX1.s, 8 bits, result of decoder fed into MUX1, MUX2, and MUX3 simultaneously
* MUX2.Ch0, 4 bits, 1111
* MUX2.Ch1, 4 bits, 1110
* MUX2.Ch2, 4 bits, 1101
* MUX2.Ch3, 4 bits, 1100
* MUX2.Ch4, 4 bits, 1011
* MUX2.Ch5, 1 bit, result of NOR into multiplexer
* MUX2.CH6, 1 bit, result of NAND into multiplexer
* MUX2.CH7, 1 bit, result of NOT into multiplexer
* MUX2.b, 1 bit, output of MUX2 into MUX3
* MUX2.s, 8 bits, result of decoder fed into MUX1, MUX2, and MUX3 simultaneously
* MUX3.Ch0, 8 bits, result of MUX1 into MUX3
* MUX3.Ch1, 8 bits, result of MUX2 into MUX3
* Mux8.k, parameter used for amount of bits (8 bits)
* Mux3.k, parameter used for amount of bits (3 bits)

**Adder**

* Adder2.p, 16 bits, a XOR b
* Adder2.g, 16 bits, a AND b
* Adder2.c, 15 bits, combination of wires p and g with cin
* Adder2.s
* Adder2.cout, 16 bits, used to get result from adder

**Subtractor**

* AddSub.c1, 16 bits, carry out last two bits
* AddSub.c2, 16 bits, carry out last two bits
* Adder1.cout, 16 bits, used to get result from AddSub
* AddSub.ovf, 16 bits overflow if signs don’t match

**Multiplier**

* Mul4.pp0, 16 bits, a AND b[0]
* Mul4.pp1, 16 bits, a AND b[1]
* Mul4.pp2, 16 bits, a AND b[2]
* Mul4.pp3, 16 bits, a AND b[3]
* Mul4.pp4, 16 bits, a AND b[4]
* Mul4.pp5, 16 bits, a AND b[5]
* Mul4.pp6, 16 bits, a AND b[6]
* Mul4.pp7, 16 bits, a AND b[7]
* Mul4.pp8, 16 bits, a AND b[8]
* Mul4.pp9, 16 bits, a AND b[9]
* Mul4.pp10, 16 bits, a AND b[10]
* Mul4.pp11, 16 bits, a AND b[11]
* Mul4.pp12, 16 bits, a AND b[12]
* Mul4.pp13, 16 bits, a AND b[13]
* Mul4.pp14, 16 bits, a AND b[14]
* Mul4.pp15, 16 bits, a AND b[15]
* Mul4.cout1, 15 bits, used for carry
* Mul4.cout2, 15 bits, used for carry
* Mul4.cout3, 15 bits, used for carry
* Mul4.cout4, 15 bits, used for carry
* Mul4.cout5, 15 bits, used for carry
* Mul4.cout6, 15 bits, used for carry
* Mul4.cout7, 15 bits, used for carry
* Mul4.cout8, 15 bits, used for carry
* Mul4.cout9, 15 bits, used for carry
* Mul4.cout10, 15 bits, used for carry
* Mul4.cout11, 15 bits, used for carry
* Mul4.cout12, 15 bits, used for carry
* Mul4.s1, 16 bits, sum of each partial product
* Mul4.s2, 16 bits, sum of each partial product
* Mul4.s3, 16 bits, sum of each partial product
* Mul4.s4, 16 bits, sum of each partial product
* Mul4.s5, 16 bits, sum of each partial product
* Mul4.s6, 16 bits, sum of each partial product
* Mul4.s7, 16 bits, sum of each partial product
* Mul4.s8, 16 bits, sum of each partial product
* Mul4.s9, 16 bits, sum of each partial product
* Mul4.s10, 16 bits, sum of each partial product
* Mul4.s11, 16 bits, sum of each partial product
* Mul4.s12, 16 bits, sum of each partial product
* Mul4.p, 32 bits, used to get result from Mul4

**Divider**

* Div.quotient0 16 bits To store quotient from the step by step long division process
* Div.quotient1 16 bits To store quotient from the step by step long division process
* Div.quotient2 16 bits To store quotient from the step by step long division process
* Div.quotient3 16 bits To store quotient from the step by step long division process
* Div.quotient4 16 bits To store quotient from the step by step long division process
* Div.quotient5 16 bits To store quotient from the step by step long division process
* Div.quotient6 16 bits To store quotient from the step by step long division process
* Div.quotient7 16 bits To store quotient from the step by step long division process
* Div.quotient8 16 bits To store quotient from the step by step long division process
* Div.quotient9 16 bits To store quotient from the step by step long division process
* Div.quotient10 16 bits To store quotient from the step by step long division process
* Div.quotient11 16 bits To store quotient from the step by step long division process
* Div.quotient12 16 bits To store quotient from the step by step long division process
* Div.quotient13 16 bits To store quotient from the step by step long division process
* Div.quotient14 16 bits To store quotient from the step by step long division process
* Div.quotient15 16 bits To store quotient from the step by step long division process
* Div.rem0 16 bits To store remainder from the step by step long division process
* Div.rem1 16 bits To store remainder from the step by step long division process
* Div.rem2 16 bits To store remainder from the step by step long division process
* Div.rem3 16 bits To store remainder from the step by step long division process
* Div.rem4 16 bits To store remainder from the step by step long division process
* Div.rem5 16 bits To store remainder from the step by step long division process
* Div.rem6 16 bits To store remainder from the step by step long division process
* Div.rem7 16 bits To store remainder from the step by step long division process
* Div.rem8 16 bits To store remainder from the step by step long division process
* Div.rem9 16 bits To store remainder from the step by step long division process
* Div.rem10 16 bits To store remainder from the step by step long division process
* Div.rem11 16 bits To store remainder from the step by step long division process
* Div.rem12 16 bits To store remainder from the step by step long division process
* Div.rem13 16 bits To store remainder from the step by step long division process
* Div.rem14 16 bits To store remainder from the step by step long division process
* Div.rem15 16 bits To store remainder from the step by step long division process
* Div.ovf 16 bits Handles overflow
* Div.valid0 16 bits
* Div.valid1 16 bits
* Div.valid2 16 bits
* Div.dendSize 4 bits
* Div.sorSize 4 bits
* Div.diffSize 4 bits
* Div.tempDiff 4 bits
* Div.subOverFlow 1 bit Handles overflow on subtraction
* Div.sign 1 bit sign bit for divident^divisor
* Div.doubleNegative 1 bit
* Div.dividendFixed 16 bits
* Div.divisorFixed 16 bits
* Div.divisorOneFlipped 16 bits Flipping negative numbers
* Div.dividendOneFlipped 16 bits Flipping negative numbers
* Div.divisorFlipped 16 bits Flipping negative numbers
* Div.dividendFlipped 16 bits Flipping negative numbers
* Div.quotientOut, remOut 16 bits This is a quotient and reminder variable used to store the long division mux output to combine the quotients together into one quotient answer, and remainders as well.
* Div.eq 1 bit
* Div.gt 1 bit
* Div.lt 1 bit
* Div.gteq 1 bit

**Encoder**

* Enc164.c, 8 bits, used as intermediate result for first stage of encoding
* Enc164.d, 4 bits, used as Boolean for groups of four

**Decoder**

* Dec4to16.x, 4 bits, used for intermediate decoder values along with y; values used for subsequent and gates
* Dec4to16.y, 4 bits, used for intermediate decoder values along with x; values used for subsequent and gates