CLJD

System Design

<2019-11-22>

Parts List

\*The following part names match their module names from the Verilog code

Mux2: The mux two accepts an input signal, an in1, and an in2. The Mux2 outputs an out, and is used for the shift left module, the shift right module, the change sign module, and for flipping a negative number in a later module for error handling purposes.

Mux4: The mux four accepts a0-a3 as inputs and ‘s’ as a one-hot select output, which assign the variable ‘b’ accordingly.

binaryMux4: The binary mux four inputs am a,b,c,d,and select. It then outputs an out for the sixteen bit multiplexer and sixteen bit priority encoder. The section used determines the use of the Mux four. The divide module uses the binary mux four.

Mux8: The mux eight is primarily used for defining the logical decisions like NOT, XOR, AND, etc. As it is used for deciding the arithmetic decision for shifting right, left, adding, subtracting, multiplying, and dividing.

FourBitPriorityEncoder: Accepts an in and outputs ‘valid’ and ‘out’ which are needed for the sixteen bit priority encoder. The four bit priority encoder is called from the sixteen bit priority encoder which encodes ‘con1’ to use the out as a selector for the mux.

SixteenBitPriorityEncoder: Accepts an in and outputs a ‘valid’ and ‘out’ which are needed for the divide module. The variable ‘con1’ is encoded to use as an output for the selector for the mux.

Input List

\*The following inputs are organized by module:

AddHalf:

a: Used for XORing with b, then ADDing ultimately used for the AddFull module.

b: Used for XORing with a, then ADDing ultimately used for the AddFull module.

AddFull:

a: Used for passing to Add half, ultimately used for the Add module.

b: Used for passing to Add half, ultimately used for the Add module.

c\_in: Used for passing to Add half, ultimately used for the Add module.

Add:

a: Inputs as 16-bits and calculated with the AddFull/ AddHalf modules.

b: Inputs as 16-bits and calculated with the AddFull/ AddHalf modules.

Partial:

a: Inputs as 16-bits, and used for producing 16 partial multiplicands.

b: Inputs as 16-bits, and used for producing 16 partial multipliers.

Mult:

a & b: Inputs as 16-bits, and used for defining the partial product of the multiplicand and multiplier used for determining the upper 16 and lower 16 bits for the final product.

Sub:

a & b: The subtractor module uses these inputs to XOR the two unless there is a carry condition, in which the cin is used.

cin: Used for the XOR wire, where it is filled n times so it can be XORed accordingly.

ShiftLeft:

num & shift: Both of these inputs are used for the Mux2 in order to shift the bits left accordingly.

ShiftRight:

num & shift: Both of these inputs are used for the Mux2 in order to shift the bits right accordingly.

changeSign:

sign & num: These inputs are used for flipping the sign which is ultimately called by the divide module.

flipNegativeNum:

sign: The sign input is used for calling the changeSign module, where the flipped number defined in the wire is passed with it to perform the necessary operations in order to flip.

num1 & num2: Num1 and num2 are given with the changeSign module call. Nums 1 and 2 are the numbers needed to be flipped.

equalBitsDivide:

dividend & divisor: Both of these are defined as 16-bits and used for the divide module accordingly.

oneShiftDivide through fifteenShiftDivide:

dividend & divisor: Both of these inputs are defined as 16-bits and used for passing into the ShiftLeft module n times, where n is in ‘n’ shift divide for the appropriate modules shifting one through fifteen times, then calling the divide module accordingly.

sixteenBitComparator:

a & b: Both of these inputs are passed to Sub #(16), which is necessary for handling potential errors with the divide module.

divideModule:

Dividend: Defined as 16-bits, and used as parameters when calling the Sub module, and sixteenBitComparator, which will calculate the quotient accordingly.

Divisor: Defined as 16-bits, and used as parameters when calling the Sub module, and sixteenBitComparator, which will calculate the quotient accordingly.

Div:

Dividend & Divisor: These inputs are given to the above modules which are used to calculate the quotient for the ALU.

ALU:

Opcode: The opcode determines which operation to perform. The below chart determines which opcode determines which function:

| code | operation |

|------+-----------|

| 0000 | no-op |

| 0001 | and |

| 0010 | nand |

| 0011 | or |

| 0100 | nor |

| 0101 | xor |

| 0110 | xnor |

| 0111 | not |

| 1000 | add |

| 1001 | subtract |

| 1010 | multiply |

| 1011 | divide |

| 1100 | shift <- |

| 1101 | shift -> |

Operand1 & Operand 2: The two operands are passed to the appropriate operation to calculate the desired arithmetic or logic function.

Output List

AddHalf:

c\_out: Output when the inputs a and b are ANDed together.

sum: Output when inputs a and b are XORed together.

AddFull:

c\_out: Output when the outputs from AddHalf are ORed together.

sum: Output when the outputs from AddHalf are calculated accordingly.

Add:

sum: Outputs a 16 bit sum from inputs a and b for AddFull

cout: Outputs for the AddFull module to calculate accordingly

Partial:

p: The p output is intended as a the result from ANDing a & b for the Mult module.

Mult:

Upper: The upper output is calculated from the partial module which is passed to the shiftRight module. The upper stores the first 16 bits of multiplying two 16 bit numbers.

Lower: The lower output is calculated from the partial module which is passed to the shiftLeft module. The lower stores the last 16 bits of multiplying two 16 bit numbers.

Sub:

Diff: Holds the difference between inputs a and b that were calculated accordingly.

ShiftLeft:

Shifted: Holds the 16-bit shifted number.

ShiftRight:

Shifted: Holds the 16-bit shifted number.

changeSign:

out: Holds the result of changing the sign on the num inputs.

flipNegativeNum:

out 1 & 2: The output from changing the sign of the number used for the divide module.

equalBitsDivide:

Quotient: Holds the quotient calculated from the divideModule.

Remainder: In case there is a remainder when dividing, the output holds this result.

oneShiftDivide through fifteenShiftDivide:

Quotient: Holds the quotient calculated from the divide Module.

Remainder: In case there is a remainder when dividing, the output holds this result.

sixteenBitComparator:

eq: Holds the output which is assigned from the a and b input calculations.

eq: Holds the output which is assigned from the a and b input calculations.

eq: Holds the output which is assigned from the a and b input calculations.

divideModule:

QuotientBit: Holds if there is a quotient or not when dividing.

Result: Holds the result of the quotient.

Div:

Quotient: Holds the quotient of the dividend and divisor.

Remainder: If there is a remainder, it is held in this output.

ALU:

Result: Holds the result of the operand1 and operand2 that were calculated.

High: Determines if there is an arithmeticHigh needed to be held, if so, it holds this value.

statusOut: Holds the status the operand/ opcode from the ALU.

Interface List

wire and1, e1o, e2o, e3o, e4o:

Wires e1o…e4o connect to the and1 in the 4 bit priority encoder which runs through the entire 16 bit priority encoder.

wire v0, v1, v2, v3, v5, v5:

These wires are also a part of the 16 bit priority encoder, they connect to the 4 bit priority encoders via the e1o…e4o wires which are defined above.

wire w1, w2, w3:

These wires in AddFull contain the output values c\_out and sum which are XORed and ANDed respectively out of AddHalf. The AddFull wires convert to its own cout and sum after finding the value from the AddHalf module.

wire carry:

The carry wire contains the values which were taken from the AddFull module and summed in the Add module via a and b. The wires continue through until it reaches a final sum.

wire p0…p15 (In the Partial and Mult Modules):

These wires are values which are ANDed together in the partial module, the wires are used for storing the values found in the Partial module which are used in the Mult module. These wires pass through the ShiftLeft, ShiftRight, and Add modules once Partial has reached the Mult module.

wire carry, w, xorWire:

These wires are used in the Sub module, as two flow through the AddFull module and the appropriate logic gates to calculate the carry.

wire layer0, layer1, layer2:

These wires circulate through the 16 different num inputs which create different “layers” that allow the bits to all shift left one in the ShiftLeft module. These layers apply to the ShiftRight module as well, but the bits are shifting to the right instead of left.

The following wires are for calculating the divide module:

wire valid

flippedNum1

flippedNum2

tempIn1

tempIn2

remainder13

shiftedDivisor1…shiftedDivisor5

remainder1…remainder13

These wires circulate through the Div module and calculate the quotient through finding a remainder and a main quotient.

wire MultHigh, DivRem, arithmetic, logical, arithmeticHigh, op1hot:

These wires are used for the ALU itself. They pass through the necessary parts to decode, hold opcodes, operands, and other necessary values to calculate all of the arithmetic and logic operands for the entire ALU.

Mode List

0000 : no-op

- No operation.

0001 : and

- The Alu will call the ‘and’ operation and and the two 16 bit integers providing a result into resultAnd.

0010 : nand

- The Alu will call the ‘nand’ operation and nand the two 16 bit integers providing a result into resultNand.

0011 : or

- The Alu will call the ‘or’ operation and or the two 16 bit integers providing a result into resultOr.

0100 : nor

- The Alu will call the ‘nor’ operation and nor the two 16 bit integers providing a result into resultNor.

0101 : xor

- The Alu will call the ‘xor’ operation and xor the two 16 bit integers providing a result into resultXor.

0110 : xnor

- The Alu will call the ‘xnor’ operation and xnor the two 16 bit integers providing a result into resultXnor.

0111 : not

- The Alu will call the ‘not’ operation and not the two 16 bit integers providing a result into resultNot.

1000 : add

- For the arithmetic operation of addition, the Add module will be called and the two 16 bit integers will be added together, providing a result into resultAdd.

1001 : subtract

- For the arithmetic operation of subtraction, the Sub module will be called and the second 16 bit integer will be subtracted from the first, providing a result into resultSub.

1010 : multiply

- For the arithmetic operation of multiplication, the Mult module will be called and the two 16 bit integers will be multiplied by each other, providing a result into resultMult.

1011 : divide

- For the arithmetic operation of division, the Div module will be called and the 16 bit integers will be divided, providing a result into resultDiv.

1100 : shift left

- Shifting left by n bits means the integer is being multiplied by 2n, providing a result into resultSL.

1101 : shift right

- Shifting right by n bits means the integer is being divided by 2n , providing a result into resultSR.

If statusOut is non-zero, there is an error:

00 : no error

01 : carry-over

10 : divide by zero

11 : overflow