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

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