PETEK KARAGEDİK 34042
CS 303 LAB #3 – 15 BIT CLA LAB REPORT

Our aim for this lab is to design an 15-bit Carry Lookahead Adder using Verilog on VS Code. This CLA is capable of doing both addition and substraction thanks to the mode input we get. If the user gives 0 as mode, the CLA will do an addition operation; if the user gives 1 as mode, the CLA will do substraction operation. Also, it is stated in the lab pdf file that the CLA should do overflow detection as well.

There are 2 modules to be implemented and a testbench module to test our CLA modules' functionality:

1. **3-bit CLA module to form the bigger 15-bit one: CLA_3bit module**

This is a 3-bit CLA that performs both addition and substraction according to the mode signal. Does carry propogation generation for these operations.

2. **15-bit CLA module using 3-bit ones: CLA_15bit_top module**

This is a "top" module made from smaller 3-bit CLAs. Uses 3-bit CLAs and chains carries between them to form a 15-bit CLA.

Now, I will present the details of these 2:

CLA_3bit module:
inputs: C, D, Cin (Carry-in → We will get this from the previous 3-bit CLA, fort he first one it will just be the mode) , mode (0 for addition and 1 for substraction)
outputs: RES(3-bit result), Carry (carry – out → will go to the next one, for the last one, will give final Cout)

- First we have to calculate the Propogation (P) and Generation (G) signals as if in a typical CLA implementation.

$$P = C \wedge D$$
$$G = C \,\&\, D$$

- And we also have to calculate carry signals for every bit, again just like a typical CLA logic.

$$carry[1]=G[0] \mid (P[0]carrie[0])$$
$$carry[2]=G[1] \mid (P[1]carrie[1])$$

- And finally, the result is defined as

$$RES = P \wedge carrie$$

given by XOR of the Propogate signal and carry. (This is calculated for each bit of the 3-bit input thanks to carrie)

CLA_15bit_top module:
inputs: A, B, mode (to perform addition or substraction)
output: S (result/sum), Cout (final last carry out), OVF(flag that detects overflow)

- For the substraction operations, we have to make sure that B (the rhs operand) is in a form that's suitable for substraction. So we XOR B to flip its bits if mode is 1. (We do it 15 times because B has 15 bits, that's why we have 15 there) And assiging this XORed version to B_forsubstraction
- Five CLA_3bit modules are added to process 15 bits in pieces of 3 bits
- Carry signals are chained between these CLA_3bit modules.
    The initial carry a.k.a. carry [0] is set to mode
    Carry-out from one CLA_3bit becomes the carry-in for the next one
- Overflow is calcualted fort he most significant bit using the following expression:
  OVF= (A[14]∧B_forsubstraction[14]∧∼S[14])∨(∼A[14]∧∼B_forsubstraction [14]∧S[14])

Overflow occurs when the sign of the result is incorrect due to the addition or substraction exceeding the representable range of the 15-bit signed integers

Tests

I have given random numbers with respect to the 8 cases below. And confirmed that my modules are working as expected by observing the waveform.

1. addition without a carry and without an overflow,
2. addition without a carry and with an overflow,
3. addition with a carry and without an overflow,
4. addition with a carry and with an overflow.
5. substraction without a carry and without an overflow,
6. substraction without a carry and with an overflow,
7. substraction with a carry and without an overflow,
8. substraction with a carry and with an overflow.

(also there was an additional case given in the testbench file initially which is setting all kinputs to zero including the mode)

Conclusion

This implementation demonstrates the modular design of a 15-bit CLA using 3-bit units. The design efficiently handles addition and substraction while ensuring proper overflow detection. The modular approach makes the system scalable and easy to debug.