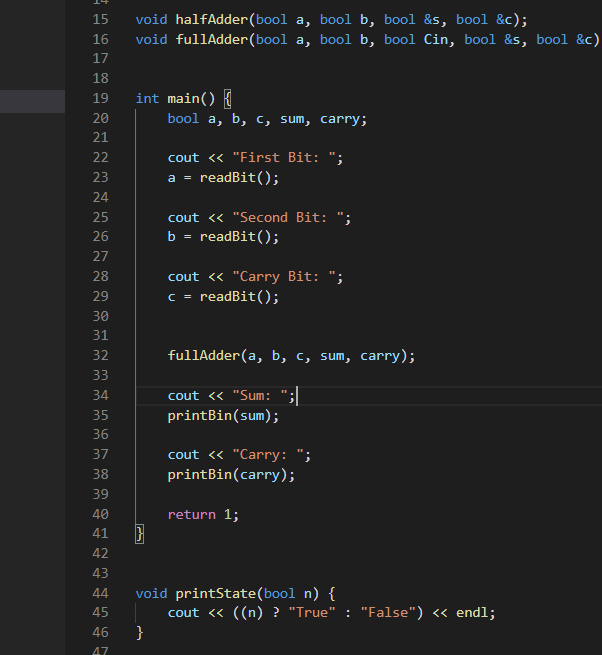
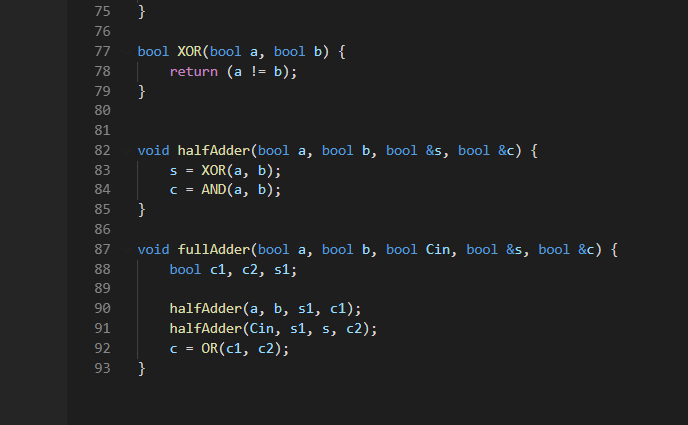
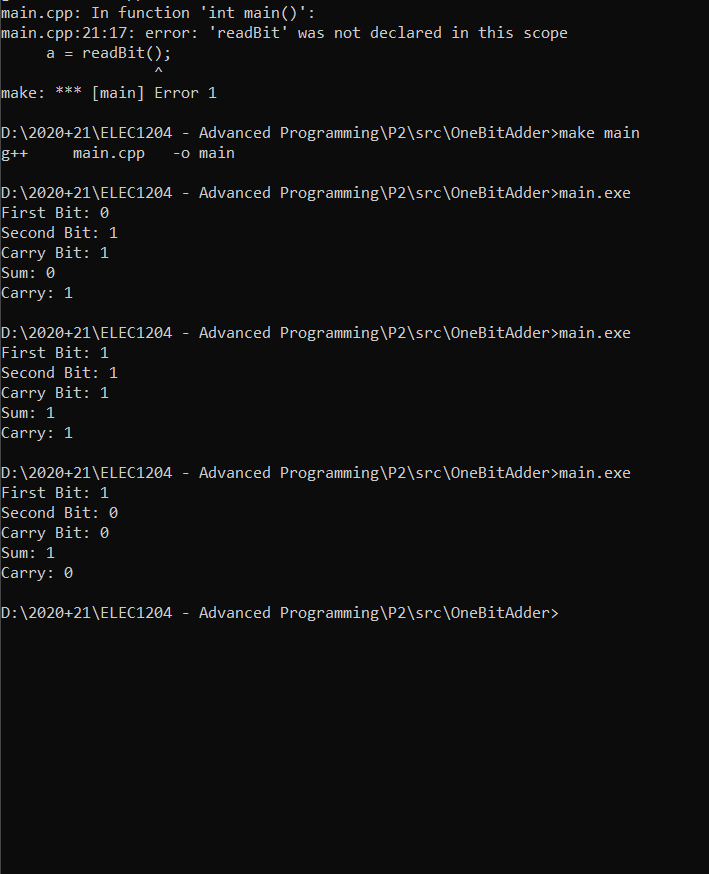
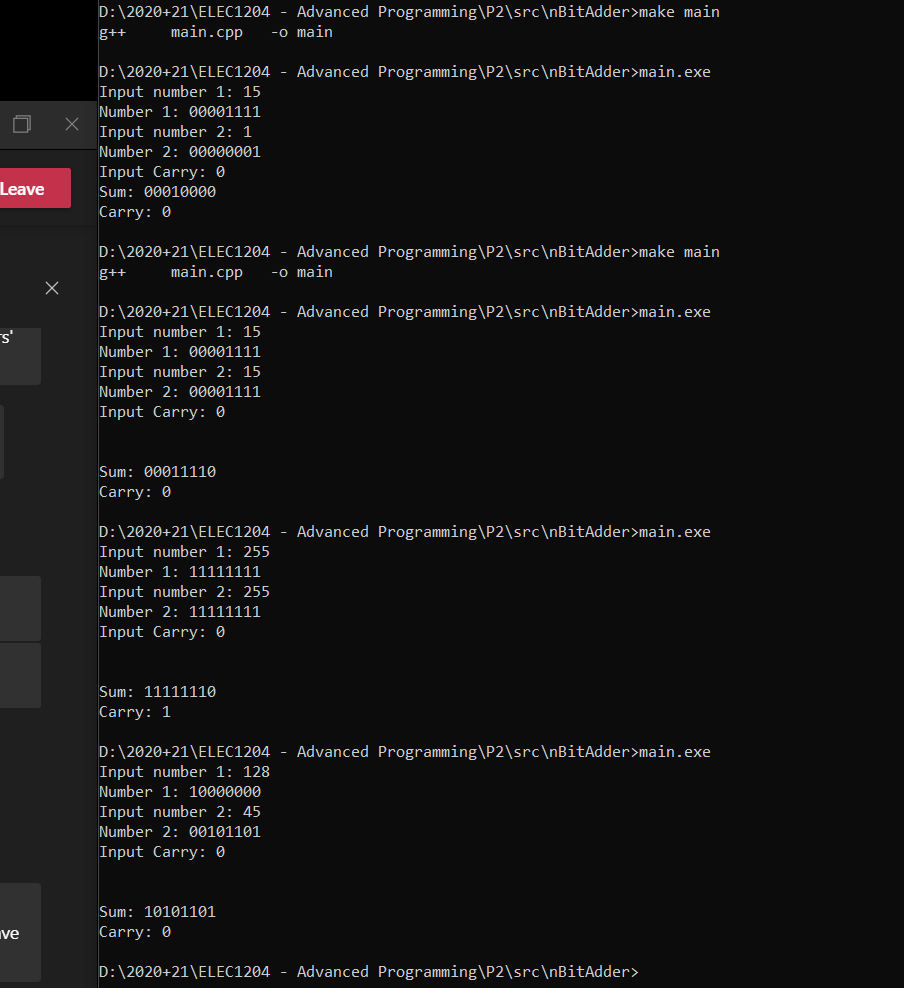
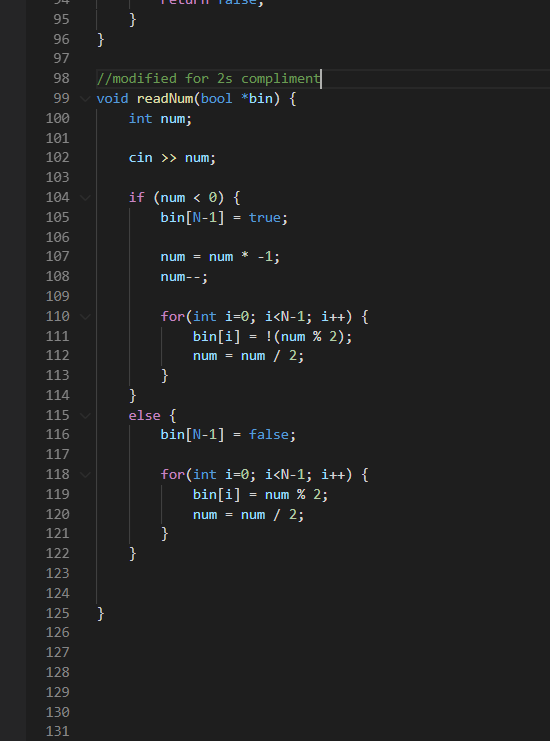
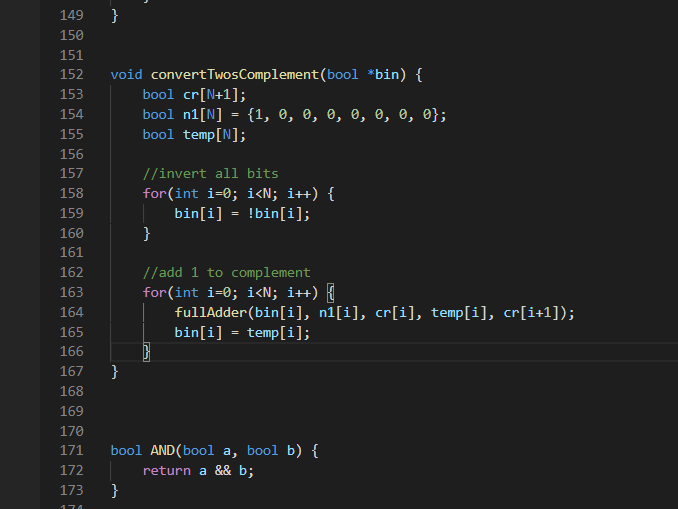
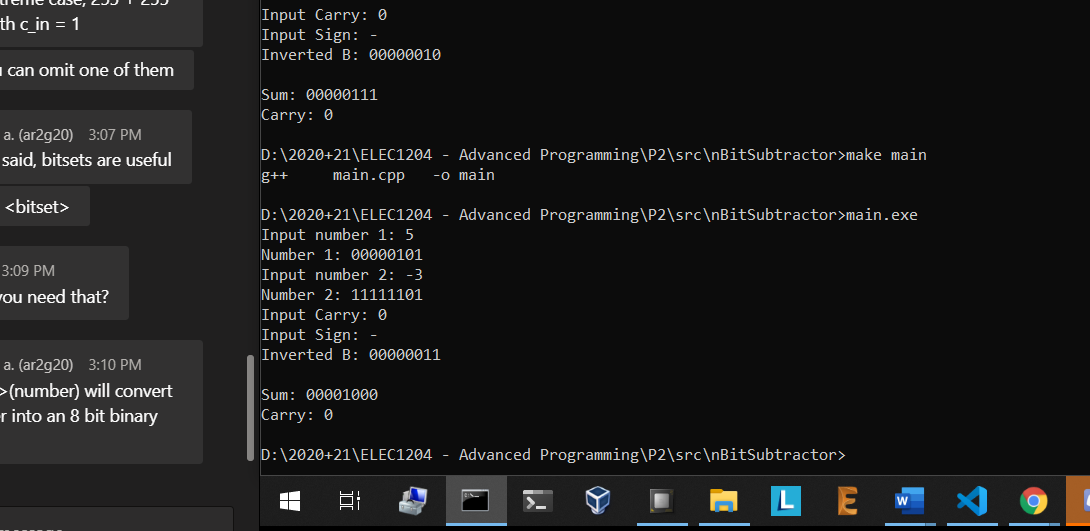
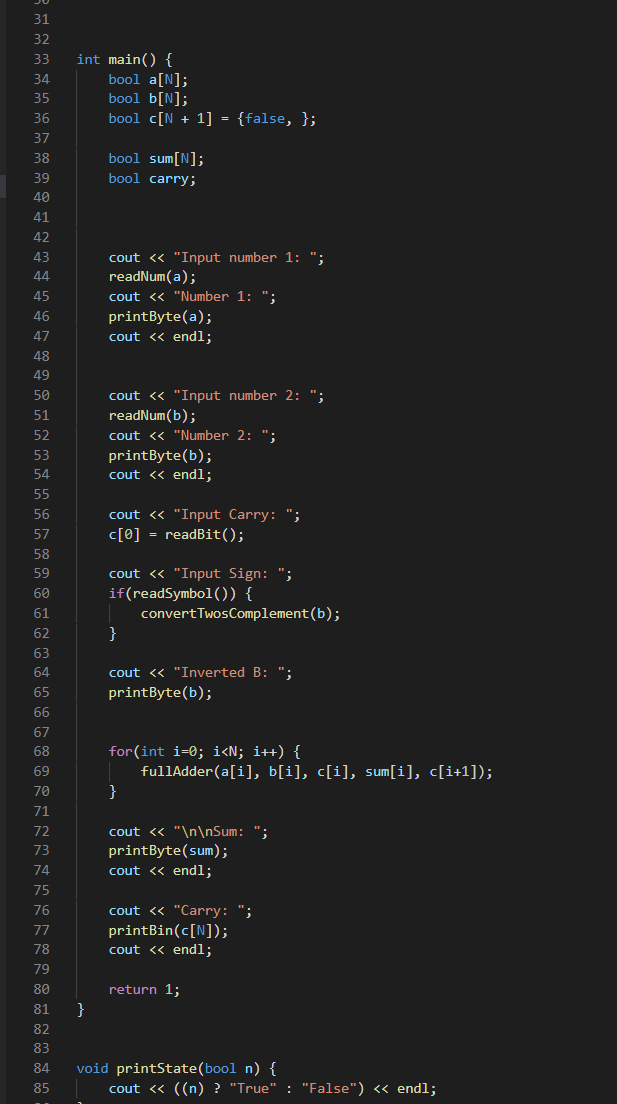
**P2 – Adding and Subtracting**

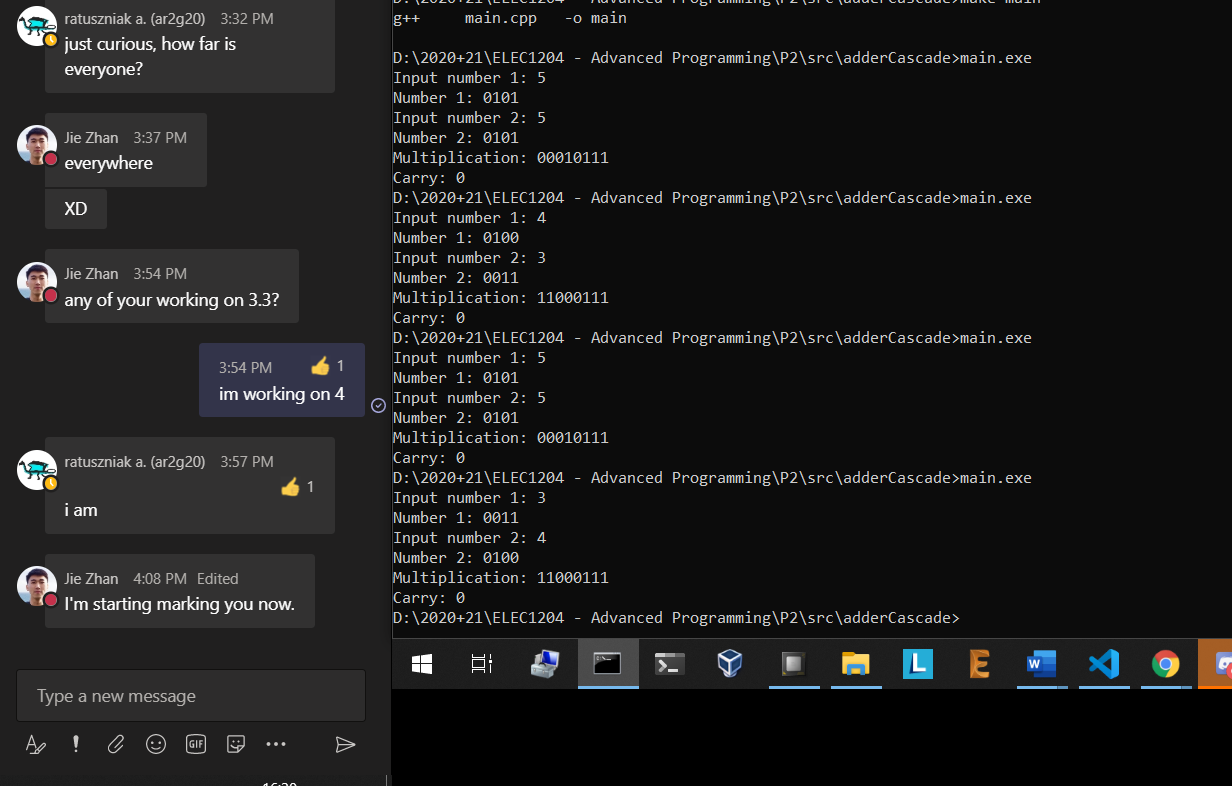
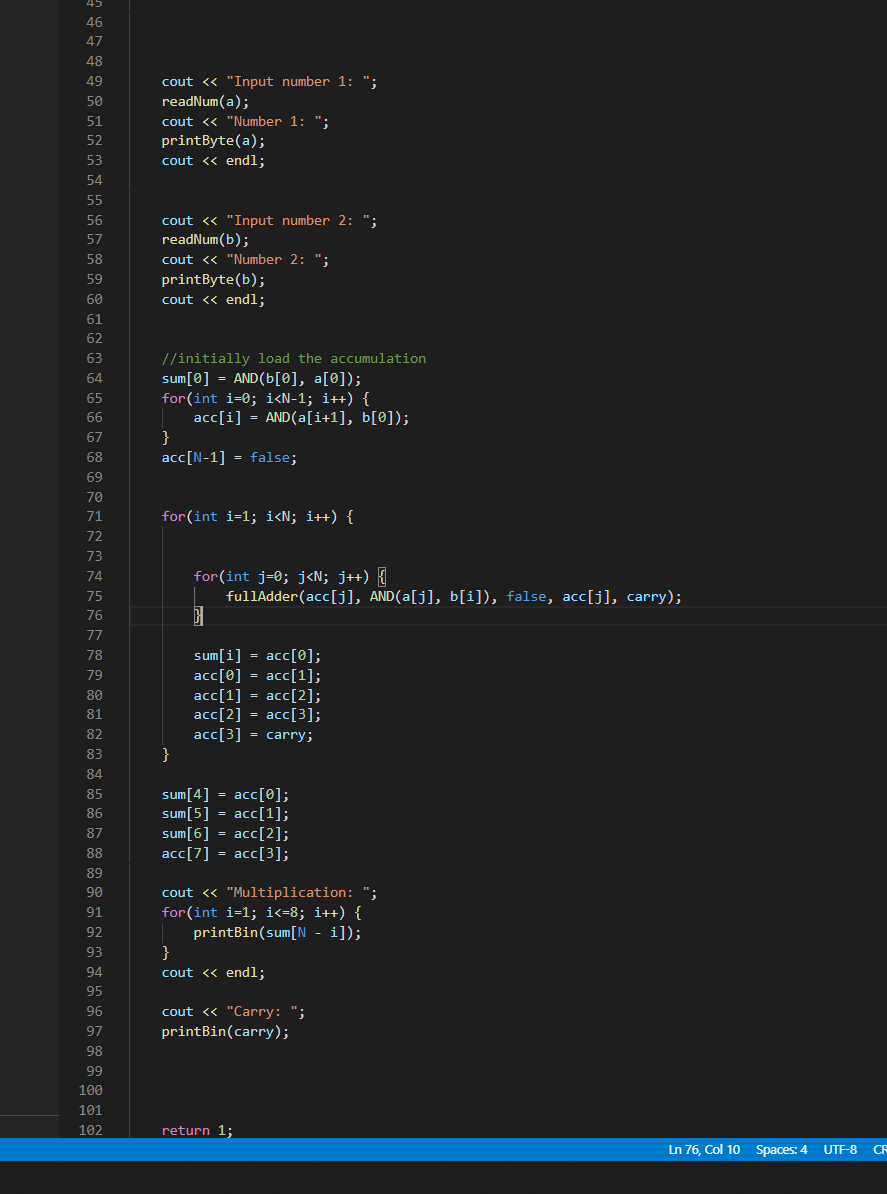
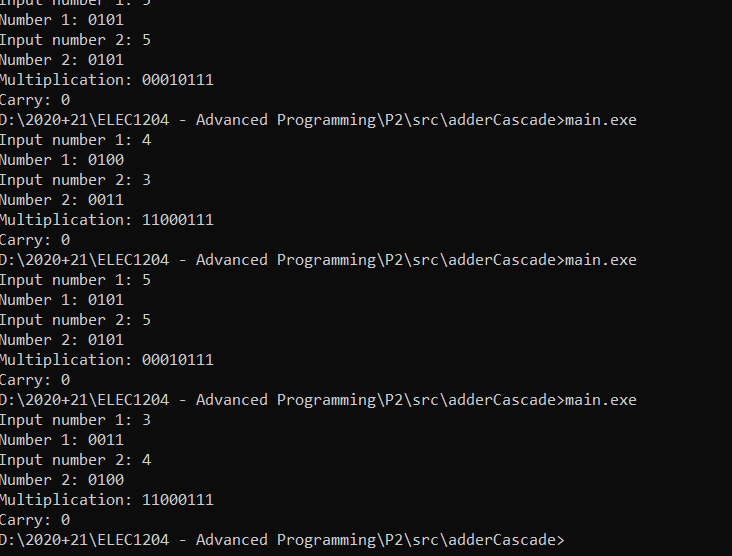
3.1 Implementing Single Bit Adder



3.2 Implementing an 8-Bit Adder

3.3 Subtraction

4 Additional Work

This code only works for certain values, I was unable to debug it in time before the end of the lab.

Full 3.3 Code

#include <iostream>

#include <math.h>

using namespace std;

#define N 8

bool AND(bool a, bool b);

bool OR(bool a, bool b);

bool XOR(bool a, bool b);

void printState(bool n);

void printBin(bool n);

void printByte(bool \*n);

bool readBit(void);

void readNum(bool \*bin);

bool readSymbol(void);

void halfAdder(bool a, bool b, bool &s, bool &c);

void fullAdder(bool a, bool b, bool Cin, bool &s, bool &c);

void convertTwosComplement(bool \*bin);

int main() {

    bool a[N];

    bool b[N];

    bool c[N + 1] = {false, };

    bool sum[N];

    bool carry;

    cout << "Input number 1: ";

    readNum(a);

    cout << "Number 1: ";

    printByte(a);

    cout << endl;

    cout << "Input number 2: ";

    readNum(b);

    cout << "Number 2: ";

    printByte(b);

    cout << endl;

    cout << "Input Carry: ";

    c[0] = readBit();

    cout << "Input Sign: ";

    if(readSymbol()) {

        convertTwosComplement(b);

    }

    cout << "Inverted B: ";

    printByte(b);

    for(int i=0; i<N; i++) {

        fullAdder(a[i], b[i], c[i], sum[i], c[i+1]);

    }

    cout << "\n\nSum: ";

    printByte(sum);

    cout << endl;

    cout << "Carry: ";

    printBin(c[N]);

    cout << endl;

    return 1;

}

void printState(bool n) {

    cout << ((n) ? "True" : "False") << endl;

}

void printBin(bool n) {

    cout << ((n) ? "1" : "0");

}

void printByte(bool \*n) {

    for(int i=1; i<=N; i++) {

        printBin(n[N - i]);

    }

}

bool readBit() {

    char buf;

    cin >> buf;

    if(buf == '1') {

        return true;

    }

    else {

        return false;

    }

}

//modified for 2s complement

void readNum(bool \*bin) {

    int num;

    cin >> num;

    if (num < 0) {

        bin[N-1] = true;

        num = num \* -1;

        num--;

        for(int i=0; i<N-1; i++) {

            bin[i] = !(num % 2);

            num = num / 2;

        }

    }

    else {

        bin[N-1] = false;

        for(int i=0; i<N-1; i++) {

            bin[i] = num % 2;

            num = num / 2;

        }

    }

}

bool readSymbol() {

    char sym;

    cin >> sym;

    if(sym == '-') {

        return true;

    }

    else {

        return false;

    }

}

void convertTwosComplement(bool \*bin) {

    bool cr[N+1];

    bool n1[N] = {1, 0, 0, 0, 0, 0, 0, 0};

    bool temp[N];

    //invert all bits

    for(int i=0; i<N; i++) {

        bin[i] = !bin[i];

    }

    //add 1 to complement

    for(int i=0; i<N; i++) {

        fullAdder(bin[i], n1[i], cr[i], temp[i], cr[i+1]);

        bin[i] = temp[i];

    }

}

bool AND(bool a, bool b) {

    return a && b;

}

bool OR(bool a, bool b) {

    return a || b;

}

bool XOR(bool a, bool b) {

    return (a != b);

}

void halfAdder(bool a, bool b, bool &s, bool &c) {

    s = XOR(a, b);

    c = AND(a, b);

}

void fullAdder(bool a, bool b, bool Cin, bool &s, bool &c) {

    bool c1, c2, s1;

    halfAdder(a, b, s1, c1);

    halfAdder(Cin, s1, s, c2);

    c = OR(c1, c2);

}