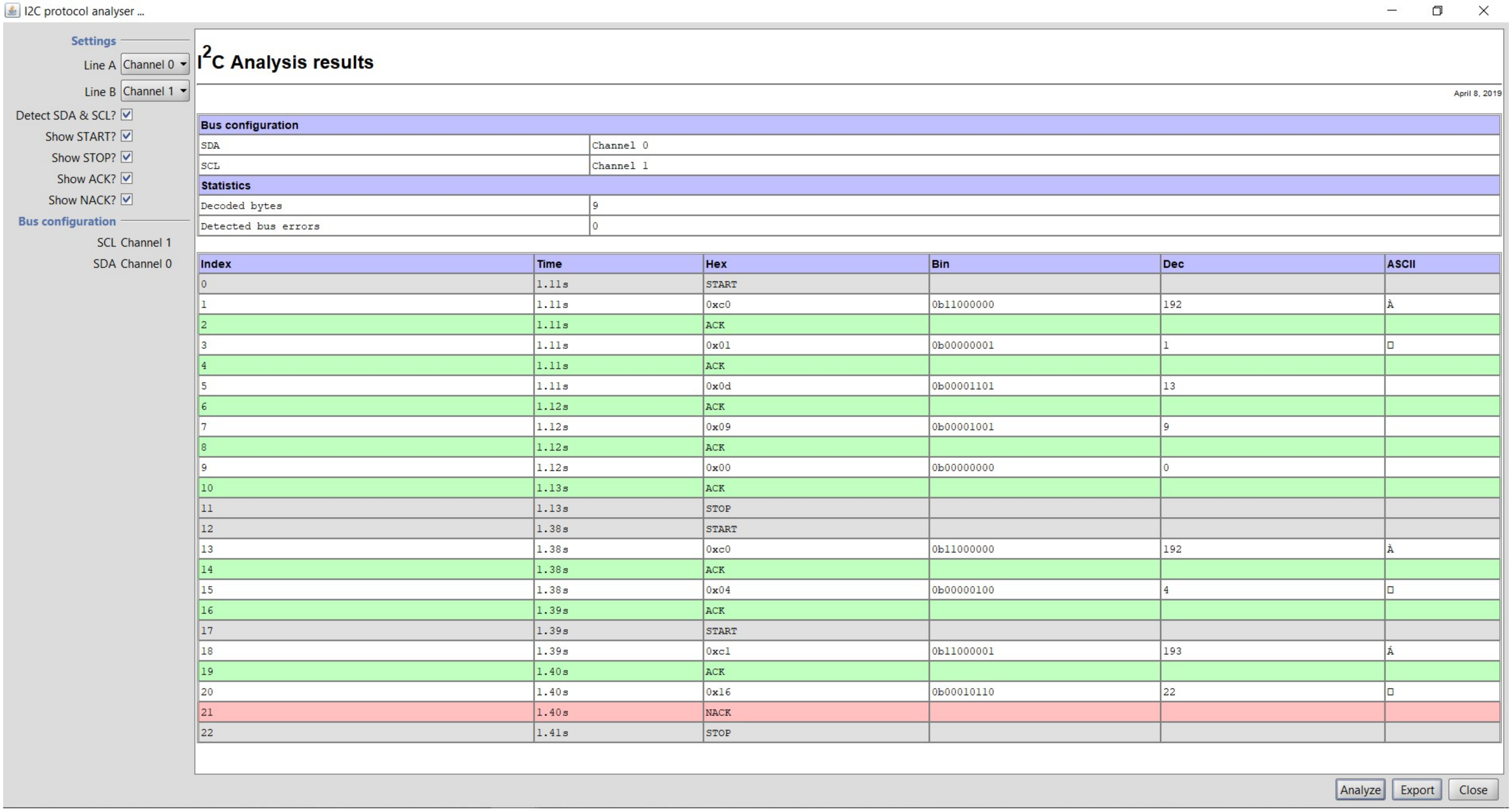
```cpp
/*
 * CMPE 146: I2C Lab main_master.cpp
 */

/**
 * @file
 * @brief This is the application entry point.
 */
#include <stdio.h>
#include "utilities.h"
#include "io.hpp"
#include <tasks.hpp>
#include "i2c2.hpp"
#include "time.h"
void vCalculate(void *pvParameters){
    uint8_t op_1, op_2, opr, result;
    I2C2& i2c = I2C2::getInstance(); // Get I2C driver instance
    const uint8_t slaveAddr = 0xC0;  // Pick any address other than an existing one at i2c2.hpp
    while (1){
    uint8_t arr[3] = { 0 };
        op_1 = rand() % 16;
        op_2 = rand() % 16;
        opr  = rand() % 3;
        arr[0] = op_1;
        arr[1] = op_2;
        arr[2] = opr;
        i2c.writeRegisters(slaveAddr, 0x01, arr, 3);
        vTaskDelay(500);
        result = i2c.readReg(0xc0, 0x04);
        switch (opr){
            case 0:
                if ((op_1 + op_2) == result){
                    printf("%u + %u = %u\n", op_1, op_2, result);
                }
                else {
                    printf("error: got %u + %u = %u\nexpected %u + %u = %u\n", op_1, op_2, result, op_1, op_2, op_1 + op_2);
                }
                break;
            case 1:
                if (op_1 > op_2){
                    if (result == (op_1 - op_2)){
                        printf("%u - %u = %u\n", op_1, op_2, result);
                    }
                    else {
                        printf("error: got %u - %u = %u\nexpected %u - %u =
%u\n", op_1, op_2, result, op_1, op_2, op_1 - op_2);
                    }
                }
                break;
            case 2:
                if ((op_1 * op_2) == result){
                    printf("%u * %u = %u\n", op_1, op_2, result);
                }
                else {
                    printf("error: got %u * %u = %u\nexpected %u * %u = %u\n", op_1, op_2, result, op_1, op_2, op_1 * op_2);
                }
                break;
        }
    }
}
int main(void)
{
    srand(time(NULL));
    xTaskCreate(vCalculate, "Calc", 1024, NULL, PRIORITY_LOW, NULL);
    scheduler_add_task(new terminalTask(PRIORITY_HIGH));
    scheduler_start();
    return -1;
}
```

```cpp
/*
 * CMPE 146: I2C Lab Main_Slave
 */

/**
 * @file
 * @brief This is the application entry point.
 */

#include <stdio.h>
#include "utilities.h"
#include "io.hpp"
#include <i2c2.hpp>
#include <tasks.hpp>
#include <GPIO/GPIOInterrupt.hpp>
#include <printf_lib.h>
#include <uart0_min.h>

volatile uint8_t buffer[256] = { 0 };

typedef enum {
    addition,
    subtraction,
    multiplication
} operation;

uint8_t operand_1 = 0, operand_2 = 0;

uint8_t result = 0;




void vReadBuffer(void *pvParameters){
    while(1){
        for (uint8_t i = 0; i < 10; i++){
            printf("Buffer %u: %X\n", i, buffer[i]);
        }
        puts("\n");
        vTaskDelay(1000);
    }
}


void vCalculate(void *pvParameters){
    while (1){
        //...do stuff
        operand_1 = buffer[1];
        operand_2 = buffer[2];

        switch (buffer[3]){
            case addition:
                result = operand_1 + operand_2;
                break;
            case subtraction:
                if (operand_1 >= operand_2){
                    result = operand_1 - operand_2;
                }
                else result = 0;
                break;
            case multiplication:
                if ((operand_1 < 16) && (operand_2 < 16)){
                    result = operand_1 * operand_2;
                }
                else result = 0;
                break;
            default:
                result = 0;
                break;
        }


        buffer[4] = result;

        //printf("op1: %u\nop2: %u\noperation: %u\nResult: %u\n\n\n",operand_1, operand_2, buffer[3], result);

        vTaskDelay(10);
    }
}



int main(void)
{


    I2C2& i2c = I2C2::getInstance();
    const uint8_t slaveAddr = 0xC0;

    i2c.initSlave(slaveAddr, &buffer[0], (size_t)sizeof(buffer));


    //xTaskCreate(vReadBuffer, "ReadBuf", 1024, NULL, PRIORITY_LOW, NULL);
    xTaskCreate(vCalculate, "Calc", 1024, NULL, PRIORITY_LOW, NULL);

    scheduler_add_task(new terminalTask(PRIORITY_HIGH));

    scheduler_start();

    return -1;
}
```

```cpp
bool I2C_Base::initSlave(const uint8_t slaveAddr, volatile uint8_t *bufferAddr, size_t bufferSize)
{
    /*
     * Slave Sender/Receiver Mode (19.6.3/4 in manual)
     */
    LPC_I2C2->I2CONSET = 0x44;

    /*
     * Make sure requested address is not reserved.
     */
    switch ((int)slaveAddr){
        case 0x38:
            return false;
        case 0x90:
            return false;
        case 0x40:
            return false;
        default:
            break;
    }
    /*
     * Set Slave Address from parameter (19.8.7 in manual)
     */
    LPC_I2C2->I2ADR2 = slaveAddr;

    /*
     * Save buffer location
     */
    mTransaction.pMasterData = (uint8_t*) bufferAddr;
    mTransaction.trxSize = (uint32_t) bufferSize;

    return true;


}
```

```
I2C_Base::mStateMachineStatus_t I2C_Base::i2cStateMachine()
{
    enum {
        // General states :
        busError        = 0x00,
        start           = 0x08,
        repeatStart     = 0x10,
        arbitrationLost = 0x38,

        // Master Transmitter States:
        slaveAddressAcked  = 0x18,
        slaveAddressNacked = 0x20,
        dataAckedBySlave   = 0x28,
        dataNackedBySlave  = 0x30,

        // Master Receiver States:
        readAckedBySlave      = 0x40,
        readModeNackedBySlave = 0x48,
        dataAvailableAckSent  = 0x50,
        dataAvailableNackSent = 0x58,

        // Slave Receiver States
        slaveAddressReceived    = 0x60,
        slaveDataReceived       = 0x80,
        slaveStoporRptStartRecv = 0xA0,

        // Slave Transmitter States
        slaveDataSend           = 0xA8,
        dataAckedByMaster       = 0xB8,
        masterNackRecv          = 0xC0

    };
```

```
        /*
         * I2C Slave RX States
         */

        case slaveAddressReceived: {
            uart0_puts("Entered state 0x60");
            mpI2CRegs->I2CONSET = 0x04;
            clearSIFlag();
            break;
        }

        case slaveDataReceived: {
            uart0_puts("Entered state 0x80");
            if (isFirst80) { //Register number is received
                isFirst80 = false;
                mTransaction.firstReg = mpI2CRegs->I2DAT;
            }
            else {
//                if ((mTransaction.firstReg - *mTransaction.pMasterData) + write_counter + 1 <= mTransaction.trxSize){
//                    *(mTransaction.pMasterData + mTransaction.firstReg + write_counter++) = mpI2CRegs->I2DAT;
//                }
//                else {
//                    uart0_puts("buffsploit prevented");
//                }
                *(mTransaction.pMasterData + mTransaction.firstReg + write_counter++) = mpI2CRegs->I2DAT;
            }
            clearSIFlag();
            break;
        }

        case slaveStoporRptStartRecv: {
            uart0_puts("Entered state 0xA0");
            isFirst80 = true;
            write_counter = 0;
            mpI2CRegs->I2CONSET = 0x04;
            clearSIFlag();
            break;
        }

        /*
         * I2C Slave TX States
         */
        case slaveDataSend: {
            uart0_puts("Entered State 0xA8");
            if (read_counter + 1 <= mTransaction.trxSize){
                mpI2CRegs->I2DAT = *(mTransaction.pMasterData + mTransaction.firstReg + read_counter++);
            }
            else {
                uart0_puts("Read too far");
            }
            mpI2CRegs->I2CONSET = 0x04;
            clearSIFlag();
            break;
        }

        case dataAckedByMaster: {
            uart0_puts("Entered State 0xB8");
            if (read_counter + 1 <= mTransaction.trxSize){
                mpI2CRegs->I2DAT = *(mTransaction.pMasterData + mTransaction.firstReg + read_counter++);
            }
            mpI2CRegs->I2CONSET = 0x04;
            clearSIFlag();
            break;
        }

        case masterNackRecv: {
            uart0_puts("Entered State 0xC0");
            read_counter = 0;
            mpI2CRegs->I2CONSET = 0x04;
            clearSIFlag();
            break;
        }
```



Settings

Line A Channel 0

Line B Channel 1

Detect SDA & SCL? ☑

Show START? ☑

Show STOP? ☑

Show ACK? ☑

Show NACK? ☑

Bus configuration

SCL Channel 1

SDA Channel 0

I^{2}C Analysis results

April 8, 2019

Bus configuration	
SDA	Channel 0
SCL	Channel 1
Statistics	
Decoded bytes	9
Detected bus errors	0

Index	Time	Hex	Bin	Dec	ASCII
0	1.11s	START			
1	1.11s	0xc0	0b11000000	192	À
2	1.11s	ACK			
3	1.11s	0x01	0b00000001	1	□
4	1.11s	ACK			
5	1.11s	0x0d	0b00001101	13	
6	1.12s	ACK			
7	1.12s	0x09	0b00001001	9	
8	1.12s	ACK			
9	1.12s	0x00	0b00000000	0	
10	1.13s	ACK			
11	1.13s	STOP			
12	1.38s	START			
13	1.38s	0xc0	0b11000000	192	À
14	1.38s	ACK			
15	1.38s	0x04	0b00000100	4	□
16	1.39s	ACK			
17	1.39s	START			
18	1.39s	0xc1	0b11000001	193	Á
19	1.40s	ACK			
20	1.40s	0x16	0b00010110	22	□
21	1.40s	NACK			
22	1.41s	STOP			