```
/*
 * 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);
46                      }
47                  }
48                  break;
49              case 2:
50                  if ((op_1 * op_2) == result){
51                      printf("%u * %u = %u\n", op_1, op_2, result);
52                  }
53                  else {
54                      printf("error: got %u * %u = %u\nexpected %u * %u = %u\n",
  op_1, op_2, result, op_1, op_2, op_1 * op_2);
55                  }
56                  break;
57          }
58      }
59  }
60  int main(void)
61  {
62      srand(time(NULL));
63      xTaskCreate(vCalculate, "Calc", 1024, NULL, PRIORITY_LOW, NULL);
64      scheduler_add_task(new terminalTask(PRIORITY_HIGH));
65      scheduler_start();
66      return -1;
67  }
```

```
 1 /*
 2  * CMPE 146: I2C Lab Main_Slave
 3  */
 4
 5 /**
 6  * @file
 7  * @brief This is the application entry point.
 8  */
 9
10 #include <stdio.h>
11 #include "utilities.h"
12 #include "io.hpp"
13 #include <i2c2.hpp>
14 #include <tasks.hpp>
15 #include <GPIO/GPIOInterrupt.hpp>
16 #include <printf_lib.h>
17 #include <uart0_min.h>
18
19 volatile uint8_t buffer[256] = { 0 };
20
21 typedef enum {
22     addition,
23     subtraction,
24     multiplication
25 } operation;
26
27 uint8_t operand_1 = 0, operand_2 = 0;
28
29 uint8_t result = 0;
30
31
32
33
34 void vReadBuffer(void *pvParameters){
35     while(1){
36         for (uint8_t i = 0; i < 10; i++){
37             printf("Buffer %u: %X\n", i, buffer[i]);
38         }
39         puts("\n");
40         vTaskDelay(1000);
41     }
42 }
43
44
45 void vCalculate(void *pvParameters){
46     while (1){
47         //...do stuff
```

```
48          operand_1 = buffer[1];
49          operand_2 = buffer[2];
50
51          switch (buffer[3]){
52              case addition:
53                  result = operand_1 + operand_2;
54                  break;
55              case subtraction:
56                  if (operand_1 >= operand_2){
57                      result = operand_1 - operand_2;
58                  }
59                  else result = 0;
60                  break;
61              case multiplication:
62                  if ((operand_1 < 16) && (operand_2 < 16)){
63                      result = operand_1 * operand_2;
64                  }
65                  else result = 0;
66                  break;
67              default:
68                  result = 0;
69                  break;
70          }
71
72
73          buffer[4] = result;
74
75          //printf("op1: %u\nop2: %u\noperation: %u\nResult:
  %u\n\n\n",operand_1, operand_2, buffer[3], result);
76
77          vTaskDelay(10);
78      }
79  }
80
81
82
83  int main(void)
84  {
85
86
87      I2C2& i2c = I2C2::getInstance();
88      const uint8_t slaveAddr = 0xC0;
89
90      i2c.initSlave(slaveAddr, &buffer[0], (size_t)sizeof(buffer));
91
92
93      //xTaskCreate(vReadBuffer, "ReadBuf", 1024, NULL, PRIORITY_LOW, NULL);
```

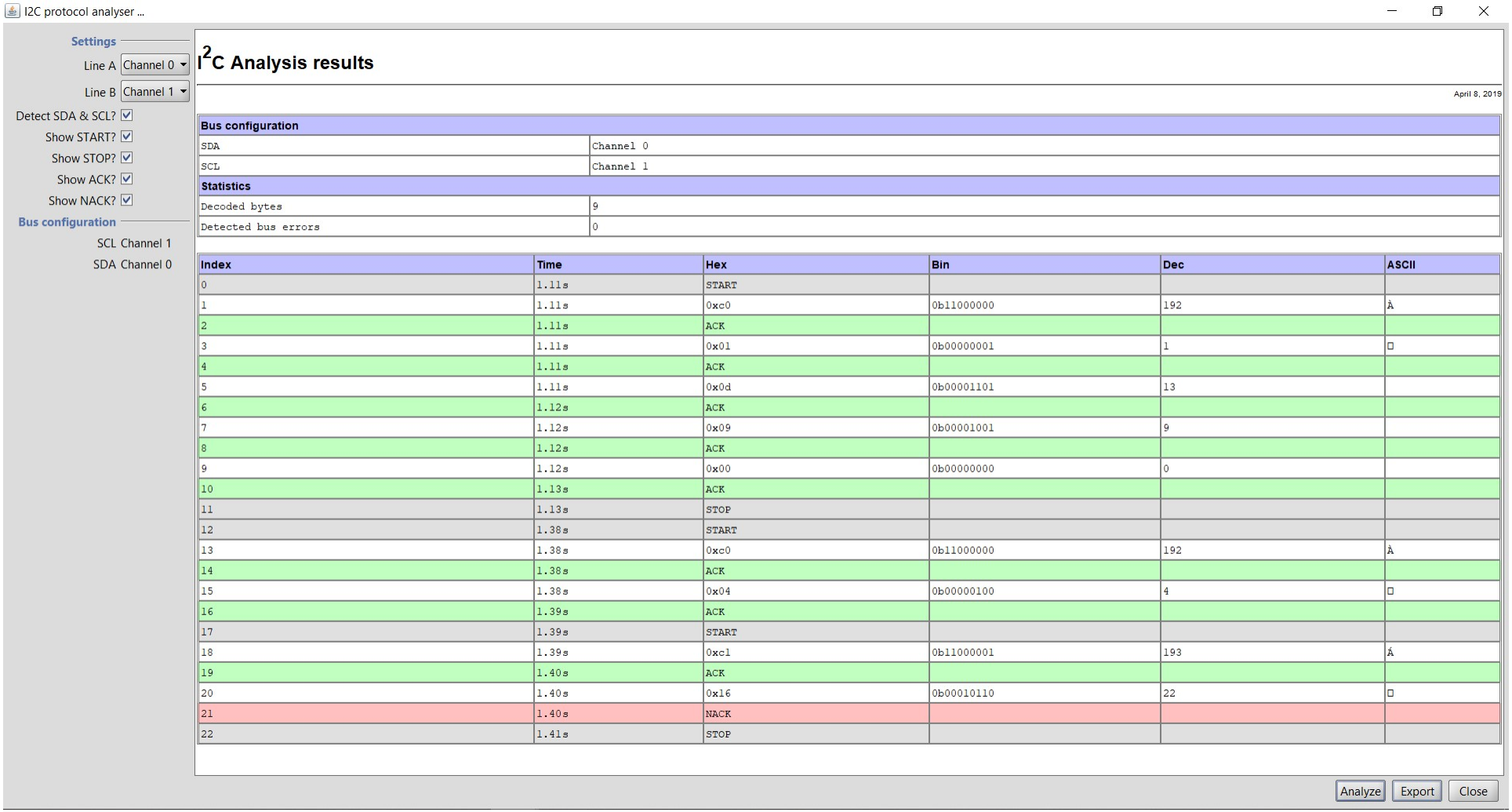
```
 94     xTaskCreate(vCalculate, "Calc", 1024, NULL, PRIORITY_LOW, NULL);
 95
 96     scheduler_add_task(new terminalTask(PRIORITY_HIGH));
 97
 98     scheduler_start();
 99
100     return -1;
101 }
```

```
195 bool I2C_Base::initSlave(const uint8_t slaveAddr, volatile uint8_t *bufferAddr,
    size_t bufferSize)
196 {
197     /*
198      * Slave Sender/Receiver Mode (19.6.3/4 in manual)
199      */
200     LPC_I2C2->I2CONSET = 0x44;
201 
202     /*
203      * Make sure requested address is not reserved.
204      */
205     switch ((int)slaveAddr){
206         case 0x38:
207             return false;
208         case 0x90:
209             return false;
210         case 0x40:
211             return false;
212         default:
213             break;
214     }
215     /*
216      * Set Slave Address from parameter (19.8.7 in manual)
217      */
218     LPC_I2C2->I2ADR2 = slaveAddr;
219 
220     /*
221      * Save buffer location
222      */
223     mTransaction.pMasterData = (uint8_t*) bufferAddr;
224     mTransaction.trxSize = (uint32_t) bufferSize;
225 
226     return true;
227 
228 
229 }
```

```
260 I2C_Base::mStateMachineStatus_t I2C_Base::i2cStateMachine()
261 {
262     enum {
263         // General states :
264         busError        = 0x00,
265         start           = 0x08,
266         repeatStart     = 0x10,
267         arbitrationLost = 0x38,
268 
269         // Master Transmitter States:
270         slaveAddressAcked  = 0x18,
271         slaveAddressNacked = 0x20,
272         dataAckedBySlave   = 0x28,
273         dataNackedBySlave  = 0x30,
274 
275         // Master Receiver States:
276         readAckedBySlave      = 0x40,
277         readModeNackedBySlave = 0x48,
278         dataAvailableAckSent  = 0x50,
279         dataAvailableNackSent = 0x58,
280 
281         // Slave Receiver States
282         slaveAddressReceived    = 0x60,
283         slaveDataReceived       = 0x80,
284         slaveStoporRptStartRecv = 0xA0,
285 
286         // Slave Transmitter States
287         slaveDataSend           = 0xA8,
288         dataAckedByMaster       = 0xB8,
289         masterNackRecv          = 0xC0
290 
291     };
```

```
404         /*
405          * I2C Slave RX States
406          */
407
408         case slaveAddressReceived: {
409             uart0_puts("Entered state 0x60");
410             mpI2CRegs->I2CONSET = 0x04;
411             clearSIFlag();
412             break;
413         }
414
415         case slaveDataReceived: {
416             uart0_puts("Entered state 0x80");
417             if (isFirst80) { //Register number is received
418                 isFirst80 = false;
419                 mTransaction.firstReg = mpI2CRegs->I2DAT;
420             }
421             else {
422 //                if ((mTransaction.firstReg - *mTransaction.pMasterData) +
    write_counter + 1 <= mTransaction.trxSize){
423 //                    *(mTransaction.pMasterData + mTransaction.firstReg +
    write_counter++) = mpI2CRegs->I2DAT;
424 //                }
425 //                else {
426 //                    uart0_puts("buffsploit prevented");
427 //                }
428                 *(mTransaction.pMasterData + mTransaction.firstReg +
    write_counter++) = mpI2CRegs->I2DAT;
429             }
430             clearSIFlag();
431             break;
432         }
433
434         case slaveStoporRptStartRecv: {
435             uart0_puts("Entered state 0xA0");
436             isFirst80 = true;
437             write_counter = 0;
438             mpI2CRegs->I2CONSET = 0x04;
439             clearSIFlag();
440             break;
441         }
442
443         /*
444          * I2C Slave TX States
445          */
446         case slaveDataSend: {
447             uart0_puts("Entered State 0xA8");
```

```
448             if (read_counter + 1 <= mTransaction.trxSize){
449                 mpI2CRegs->I2DAT = *(mTransaction.pMasterData +
    mTransaction.firstReg + read_counter++);
450             }
451             else {
452                 uart0_puts("Read too far");
453             }
454             mpI2CRegs->I2CONSET = 0x04;
455             clearSIFlag();
456             break;
457         }
458 
459         case dataAckedByMaster: {
460             uart0_puts("Entered State 0xB8");
461             if (read_counter + 1 <= mTransaction.trxSize){
462                 mpI2CRegs->I2DAT = *(mTransaction.pMasterData +
    mTransaction.firstReg + read_counter++);
463             }
464             mpI2CRegs->I2CONSET = 0x04;
465             clearSIFlag();
466             break;
467         }
468 
469         case masterNackRecv: {
470             uart0_puts("Entered State 0xC0");
471             read_counter = 0;
472             mpI2CRegs->I2CONSET = 0x04;
473             clearSIFlag();
474             break;
475         }
```


I2C protocol analyser ...
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
I2C 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
Analyze
Export
Close