Discovery and testing of the module using Linux-based i2c-tools.

**Listing 1: I2CDevice.cpp**

#include"I2CDevice.h"

#include<iostream>

#include<sstream>

#include<fcntl.h>

#include<stdio.h>

#include<iomanip>

#include<unistd.h>

#include<sys/ioctl.h>

#include<linux/i2c.h>

#include<linux/i2c-dev.h>

using namespace std;

#define HEX(x) setw(2) << setfill('0') << hex << (int)(x)

namespace EE513 {

/\*\*Constructor for the I2CDevice class.

\*It requires the bus number and device number.

\* The constructor opens a file handle to the I2C device, which is destroyed \*when the destructor is called

\* @param bus The bus number.

\* @param device The device ID on the bus.

\*/

I2CDevice::I2CDevice(unsigned int bus, unsigned int device) {

this->file=-1;

this->bus = bus;

this->device = device;

this->open();

}

/\*\*

\* Open a connection to an I2C device

\* @return 1 on failure to open to the bus or device, 0 on success.

\*/

int I2CDevice::open(){

string name;

if(this->bus==0) name = I2C\_0;

else name = I2C\_1;

else if(this->bus==1) name = I2C\_1;

else name = I2C\_2; /\*\* Error found and code fixed and \*uploaded to Loop by Aminuddin Mohammed so reference added. \*https://loop.dcu.ie/mod/forum/discuss.php?d=503899

\*accessed on 23rd Feb 2023\*/

if((this->file=::open(name.c\_str(), O\_RDWR)) < 0){

perror("I2C: failed to open the bus\n");

return 1;

}

if(ioctl(this->file, I2C\_SLAVE, this->device) < 0){

perror("I2C: Failed to connect to the device\n");

return 1;

}

return 0;

}

/\*\*

\* Write a single byte value to a single register.

\* @param registerAddress The register address

\* @param value The value to be written to the register

\* @return 1 on failure to write, 0 on success.

\*/

int I2CDevice::writeRegister(unsigned int registerAddress, unsigned char value){

unsigned char buffer[2];

buffer[0] = registerAddress;

buffer[1] = value;

if(::write(this->file, buffer, 2)!=2){

perror("I2C: Failed write to the device\n");

return 1;

}

return 0;

}

/\*\*

\* Write a single value to the I2C device. Used to set up the device to read from a

\* particular address.

\* @param value the value to write to the device

\* @return 1 on failure to write, 0 on success.

\*/

int I2CDevice::write(unsigned char value){

unsigned char buffer[1];

buffer[0]=value;

if (::write(this->file, buffer, 1)!=1){

perror("I2C: Failed to write to the device\n");

return 1;

}

return 0;

}

/\*\*

\* Read a single register value from the address on the device.

\* @param registerAddress the address to read from

\* @return the byte value at the register address.

\*/

unsigned char I2CDevice::readRegister(unsigned int registerAddress){

this->write(registerAddress);

unsigned char buffer[1];

if(::read(this->file, buffer, 1)!=1){

perror("I2C: Failed to read in the value.\n");

return 1;

}

return buffer[0];

}

/\*\*

\* Method to read a number of registers from a single device. This is much more

\* efficient than reading the registers individually. The from address is the

\* starting address to read from, which defaults to 0x00.

\* @param number the number of registers to read from the device

\* @param fromAddress the starting address to read from

\* @return a pointer of type unsigned char that points to the first element in the \*block of registers

\*/

unsigned char\* I2CDevice::readRegisters(unsigned int number, unsigned int fromAddress){

this->write(fromAddress);

unsigned char\* data = new unsigned char[number];

if(::read(this->file, data, number)!=(int)number){

perror("IC2: Failed to read in the full buffer.\n");

return NULL;

}

return data;

}

/\*\*

\* Method to dump the registers to the standard output. It inserts a return

\* character after every 16 values and displays the results in hexadecimal to \*give a standard output using the HEX() macro that is defined at the top of this \*file.

\* The standard output will stay in hexadecimal format, hence

\* the call on the last like.

\* @param number the total number of registers to dump, defaults to 0xff

\*/

void I2CDevice::debugDumpRegisters(unsigned int number){

cout << "Dumping Registers for Debug Purposes:" << endl;

unsigned char \*registers = this->readRegisters(number);

for(int i=0; i<(int)number; i++){

cout << HEX(\*(registers+i)) << " ";

if (i%16==15) cout << endl;

}

cout << dec;

}

/\*\*

\* Close the file handles and sets a temporary state to -1.

\*/

void I2CDevice::close(){

::close(this->file);

this->file = -1;

}

/\*\*

\* Closes the file on destruction, provided that it has not already been closed.

\*/

I2CDevice::~I2CDevice() {

if(file!=-1) this->close();

}

}

/\* namespace EE513\*/