Huy Hoang

Giao Phung

Modbus RTU Protocol-RS485

Light Dependent Sensor

04/2021

Information Technology-Embedded Systems Design

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Embedded Systems Design

ABSTRACT

The report was made for learning Modbus RTU RS485 protocol, master PC and slave node. Designing circuit board to communicate between NUCLEO-LR152RE, sensor, power supply and other components, after that using IoT TICKET software to show the results in the web interface.

The software was developed base on Ultra low power STM32L152RE device features and peripheral to identify and read sensor and the sensor measurement. RS485 bus standard used to send data to master PC and python API client was pushed data to IoT TICKET

Software communication from slave to master was completed parallel with PCBs design for RS 485.

CONTENTS

LIST OF FIGURES AND TABLES

1 TERMS AND ABBREVIATIONS

LDR   **Light dependent resistor**

SI   units of measurement derived from seven base units specified by the International System of Units

RTU   Remote Terminal Unit

USART   Universal Synchronous Asynchronous Receiver-Transmitter

2   INTRODUCTION

2.1 Scope of work

- Datasheet and calculate sensor measurement

- Software design for communication between master and slave node

- Using USART2 to test with Realterm
- Changing from USART2 to USART1 and test with PC
- PCBs design for RS485 and power supply
- Design dashboard on IoT TICKET to overview sensor data

A Light dependent sensor is NSL-06S53 Type 5 Cds photoconductive cell in a flat lens TO-18 package. Peak AC or DC voltage is 100V and soldering temperature is 260 Celsius degree.

RS485 bus standard was chosen because significant feature using widely in industry.

I would like to thank VAMK, University of Applied Sciences for supporting us the component and physical tools. Beside that, a big thank to Mr Jani Ahvonen, Who is our group supervisor and best teacher ever.

3 REQUIREMENTS SPECIFICATION

3.1 SOFTWARE AND MODBUS RTU

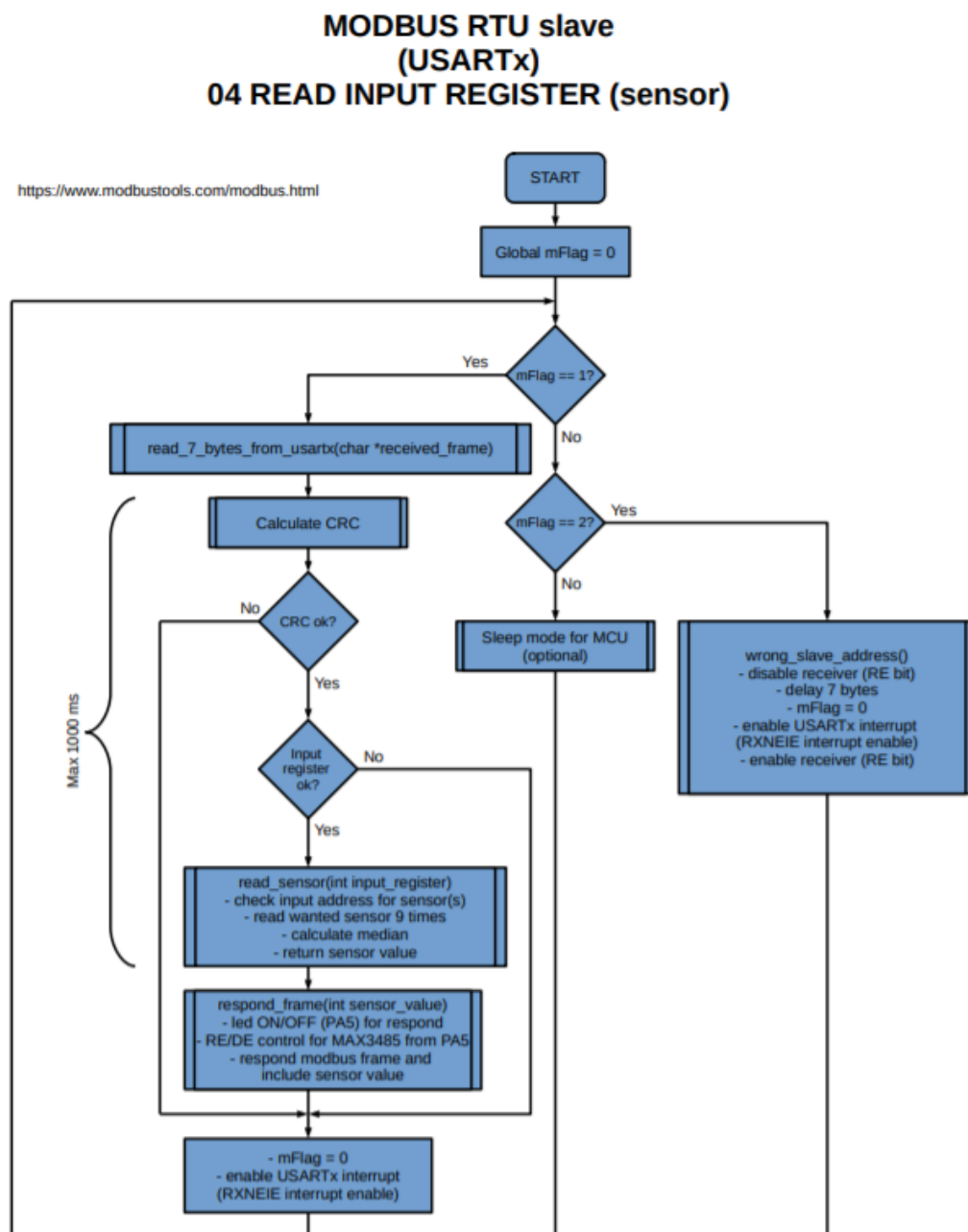The figure 1 is Modbus flow chart showed the following step to design for the software

Figure 1: Modbus Flow Chart

The requirement is: testing step by step functional follow flow chart (Figure 1)

Modbus protocol is a messaging structure, or master-slave communication included: the address of the slave, the register, the data and the check sum LRC or CRC. In this report, we gone through RS485.

The request: function code tells the slave action to perform

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	04	0 4
Function	01	0 1
Starting Address Hi	00	0 0
Starting Address Lo	0A	0 A
Quantity of Coils Hi	00	0 0
Quantity of Coils Lo	0D	0 D
Error Check Lo	DD	LRC (E 4)
Error Check Hi	98	
Trailer	None	CR LF
Total Bytes	8	17

Figure 2: Sample Request Frame

The response: function code answer to the master

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	04	0 4
Function	01	0 1
Byte Count	02	0 2
Data (Coils 7...10)	0A	0 A
Data (Coils 27...20)	11	1 1
Error Check Lo	B3	LRC (D E)
Error Check Hi	50	None
Trailer	None	CR LF
Total Bytes	7	15

Figure 3: Sample Response Frame

The obstacle of Modbus master-slave communication that the request and response could not perform at the same time, in case have much more than 1 slave

RTU Mode: when Modbus network using RTU mode, each eight-bit byte in a message contains two four bits hexadecimal characters. Each message must be transmitted in a continuous stream.

3.2 HARDWARE

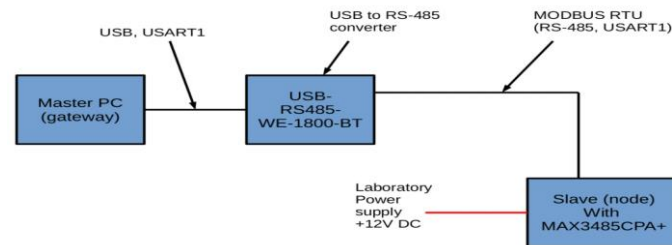### 3.2.1 Connection between Master PC and slave through USB RS485

Figure 4: Master-Slave Communicate RS485
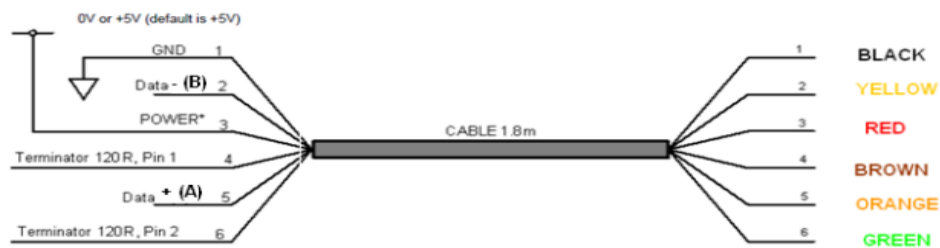
Figure 5: RS485 Connections Details
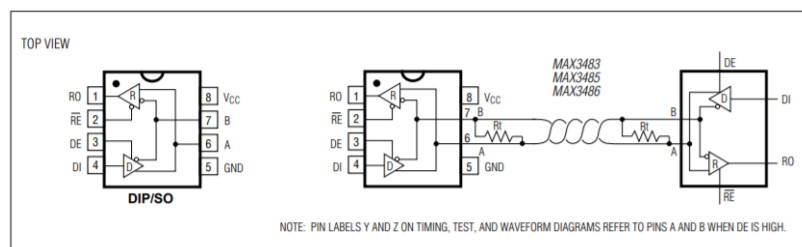
Figure 6: MAX3485 Pin Configuration
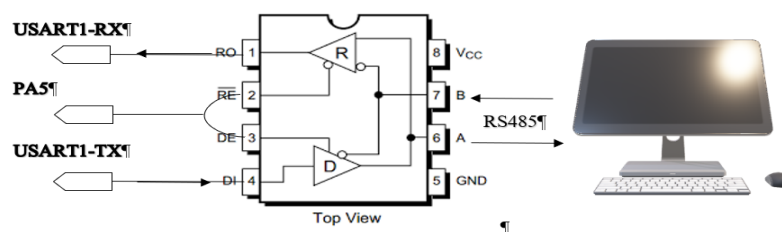
Figure 7: RS485 communicate with MAX3485 and slave

3.2.2   Nucleo STM32 Convert Power Supply

The board can use power supply from PC through USB cable or External source: Vin(7-12V), E5V (5V) or +3.3V power supply pins on CN6 or CN7 (Appendix 1: 6.3-UM1724 User manual).

The board was constructed a circuit using IC LD1117S50TR, LD39050PU33R, LD3985M33R to drop down the voltage as
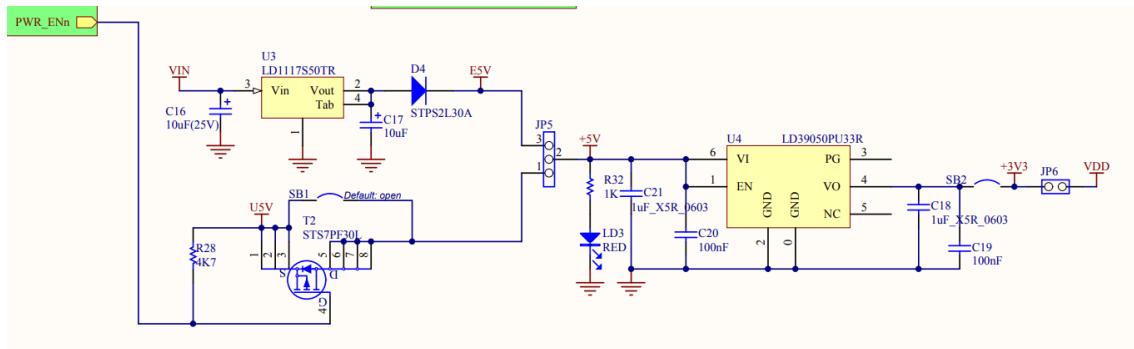
Figure 8: Build in circuit to generate 3.3V (1)
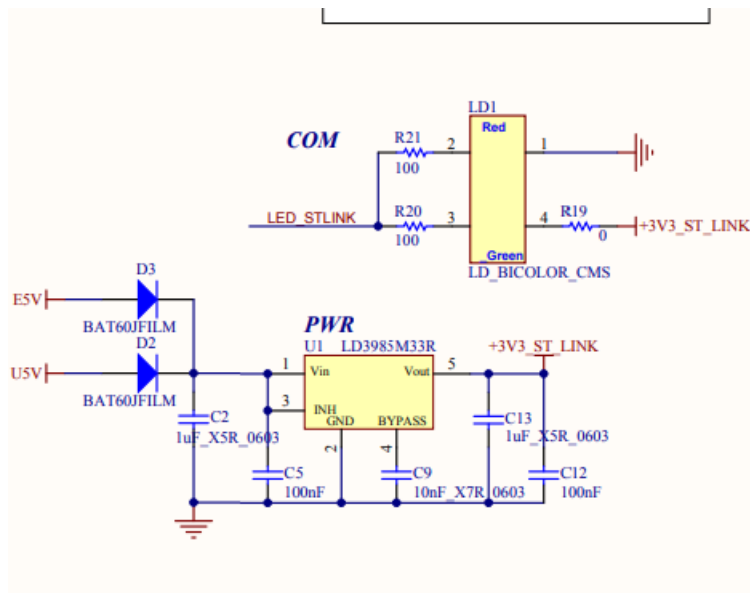
Figure 9: Build in circuit to generate 3.3V (2)

Requirement: following the user manual for external power supply (Appendix 1: UM1724 6.3 User manual)

4   SENSOR AND SENSOR MEASUREMENT

The figure 10 shows sensor design dimension
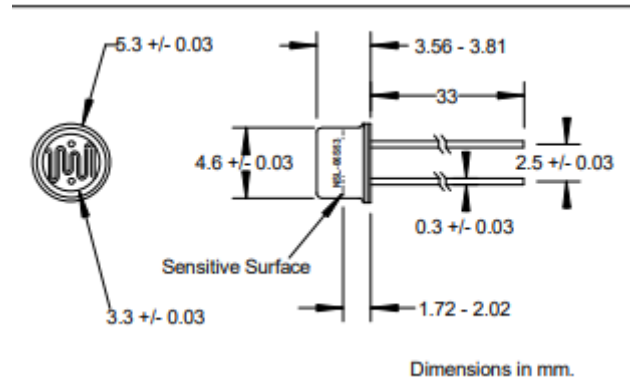
Figure 10: Light Dependent Sensor NSL-06S53
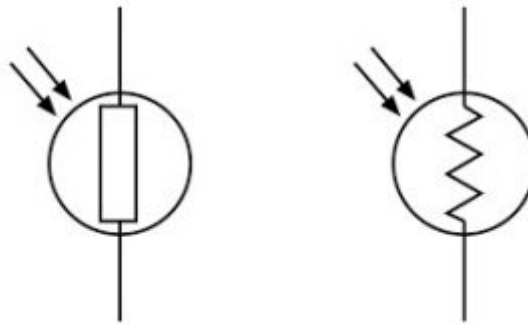
Figure 11: LDR symbols

A light-dependent resistor LDR or a photoresistor is a device made up of high resistance semiconductor material (Cadmium sulfide). LDR circuit and its working responsive to light, once light rays drop on it, the resistance will be changed. (Table 1)

The LUX (symbol: lx) is the SI derived unit of illuminance, measuring luminous flux per unit area.

Table 1: LUX value calculated

R(ref) Ω	10000		
R(Light sensor) Ω	LUX (lx)	Voltage (V)	ADC
1060	854	0,316274864	392
1570	531	0,447796024	556
2940	262	0,749768161	930
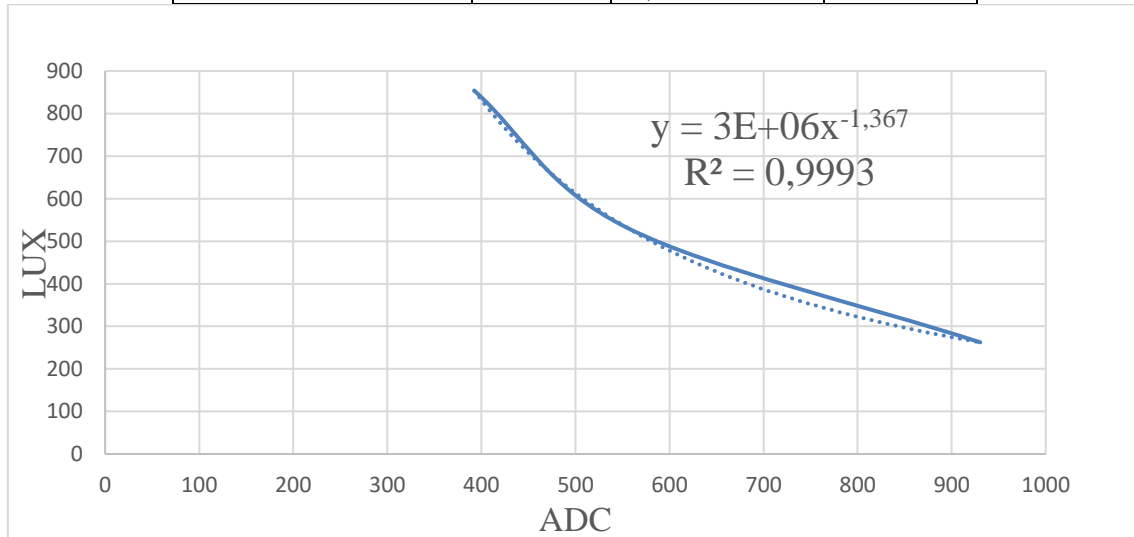
Figure 12: ADC & LUX relationship

The figure 12 shows related function between ADC and LUX value given by function

$$y = 3 * 10^6 * x^{-1,367}$$

With x is ADC value. It was shown that at the higher LUX value got the smaller ADC value, smaller voltage value, resistor value follows as well and opposite.

In theory, the famous application of LDR is Auto light in car, storage, school, mobile phone, street light ..etc..

5 SOFTWARE DEVELOPMENT

5.1 USART2, Realterm testing
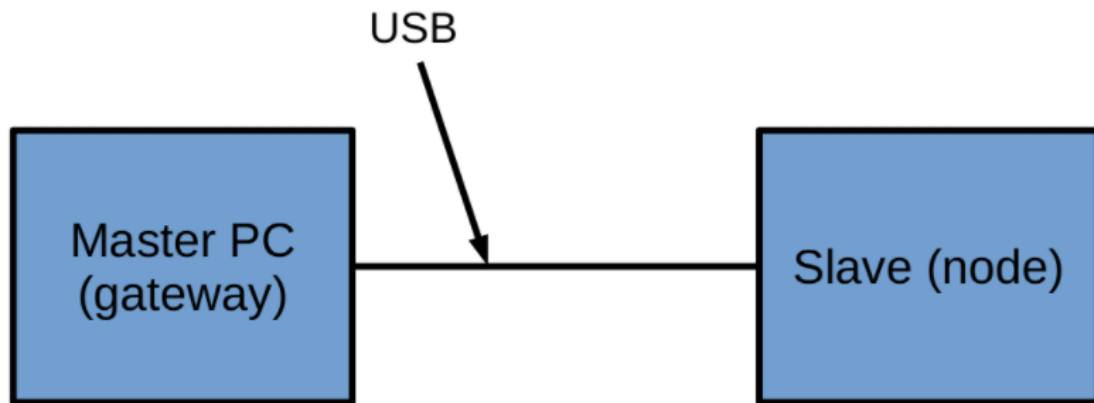
Figure 13: USART2 Master-Slave (USB)

To communicate with Master, the slave was registered a specific slave address

Request frame from Master formatted in 8 bytes in hexadecimal with structure

Table 2: RTU Request Frame

| Slave address 0b	Function 04	Starting Address Hi 00	Starting Address Lo 01	Quantity of Registers Hi 00	Quantity of Registers Lo 01	Error Check Hi 0A	Error Check Lo 60

Request Frame order: 0B 04 00 01 00 01 60 A0

LDR slave address 0b. When master send the request to the slaves, It triggered interrupt using function, It will wait inside the IRQHandler until receive enough 8 bytes then check the slave address

```
void USARTx_IRQHandler(void)
{
        char c=0;
        if(USART2->SR & 0x0020)          //if data available in DR register. p737
        {
                c = USART2->DR;          // save read value into c
        }
        if(c==SLAVE_ADDRESS)          // check slave address
```

```
        {
                mFlag = 1;
        }
        else
        {
                mFlag = 2;
        }
        USART2->CR1 &= ~0x0020;        // this is to clear bit no.5 of CR1
```

Then set mFlag value to 1 in case right address and 2 in case wrong address.

mFlag ==1: trigger function `void read_7_bytes_from_usartx(char *received_frame)`

mFlag ==2 reset mFlag=0, remain idle and response "Wrong address"
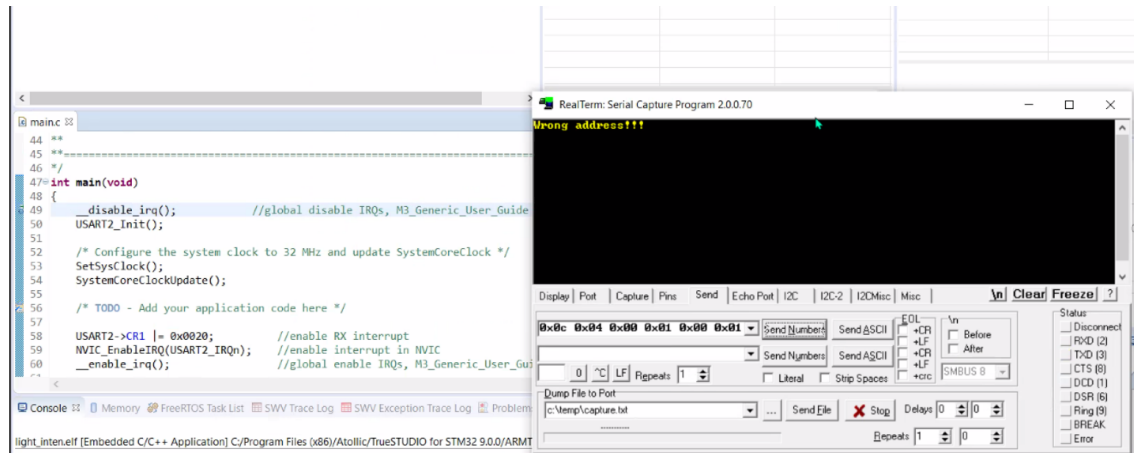
Figure 14: USART2 Wrong address response

The function `void read_7_bytes_from_usartx(char *received_frame)` read the next 7 bytes
of request frame to calculate check sum

With the wrong check sum set mFlag=0 remain idle

Right check sum we continued to check the input register (the number of sensor you
was used, in out case is 1) then function `int read_sensor(int input_register)` read sensor
9 times taken LUX (decimal part) value from LDR sensor calculate median of LUX
then response to the master with function `void return_to_master(int sensor_value)` after
calculate check sum again.

Table 3:RTU Response Frame

Slave address 0b	Function 04	Byte count 02	High Register Value 00	Low Register Value 00	Error Check Hi 00	Error Check Low 00

Response frame included 7 bytes: 0b 04 02 00 00 00 00

The figure shown the response frame from LDR slave to master contained last two bytes is check sum was calculated inside function `void return_to_master(int sensor_value)` the value " 39, E7, 3B" is median LUX decimal value from LDR sensor.
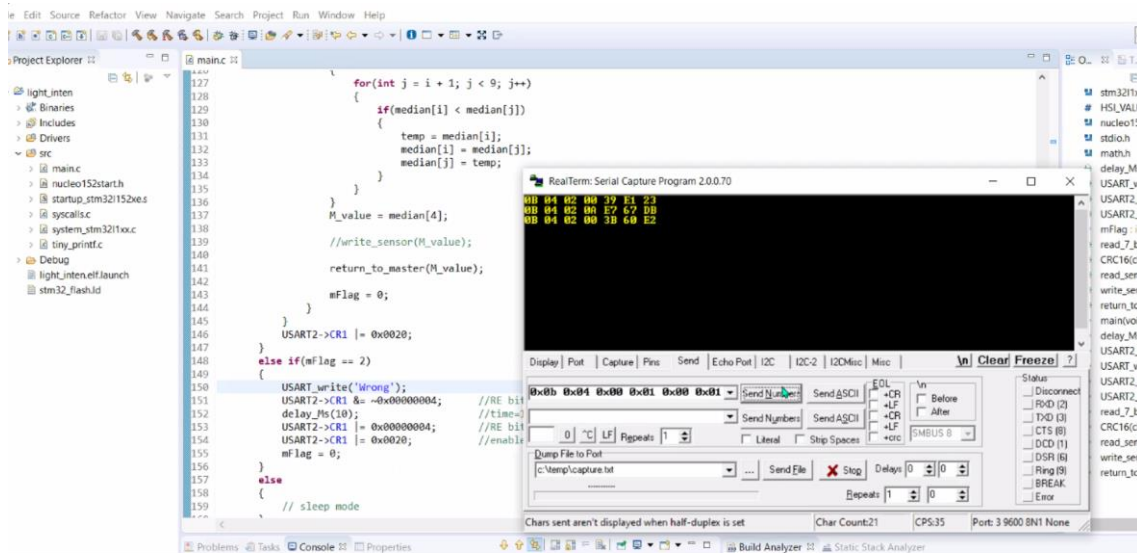
Figure 15: USART2 Right Address Response Frame

5.2 USART1 testing and sending data to IoT TICKET

We changed from USART2 to USART1 and tested with Realterm (complete code Appendix 2)
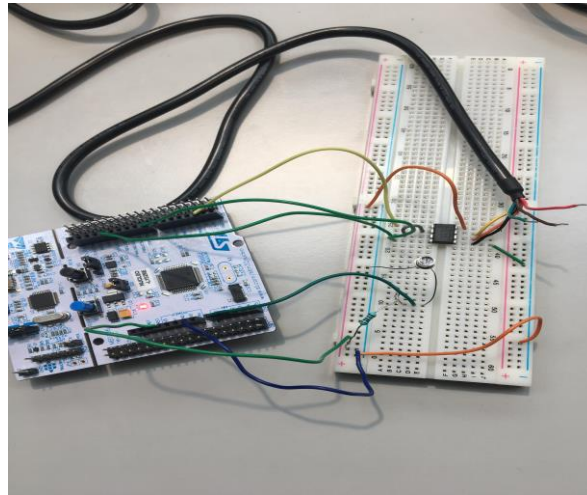
Figure 16: USART1 Breadboard Testing

Figure 17: USART1 Testing With Realterm

The response frame in figure 17 shows in two test cases:

The first case: normal light response 01 16 (HEX) is 278 -> LUX= 278

The second case: direct light from flash mobile phone response 1B 00 -> LUX= 6912

After testing with Realterm We send data to IoT TICKET with masterpython.py (Appendix 2) Figure 18 shows beautiful results
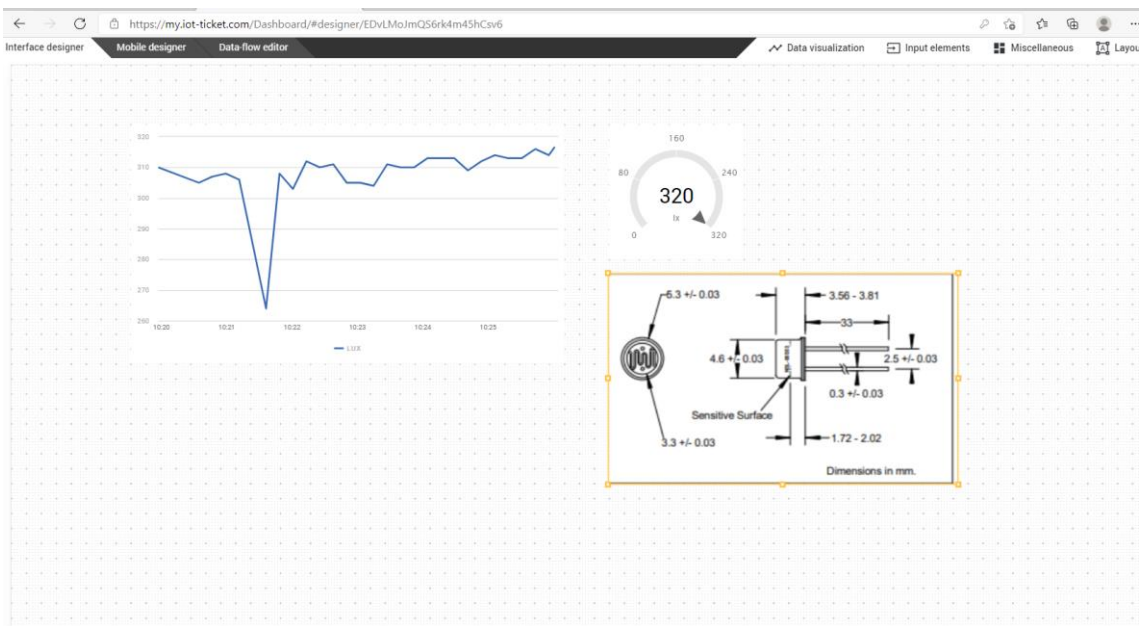
Figure 18: IoT Dashboard

6 HARDWARE DEVELOPMENT

6.1 PCB design for MAX3485 and RS485

Move to the schematic for the PCB; after hours of designing and learning, we have drawn this:
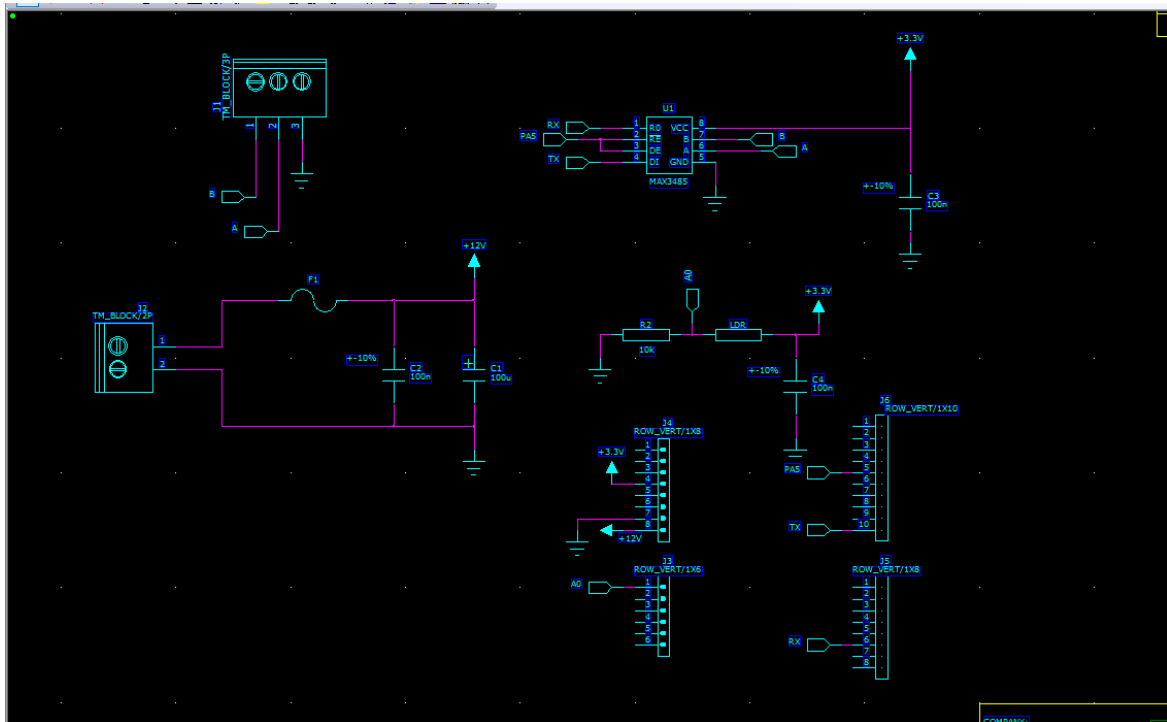
Figure 19: Schematic for the PCB

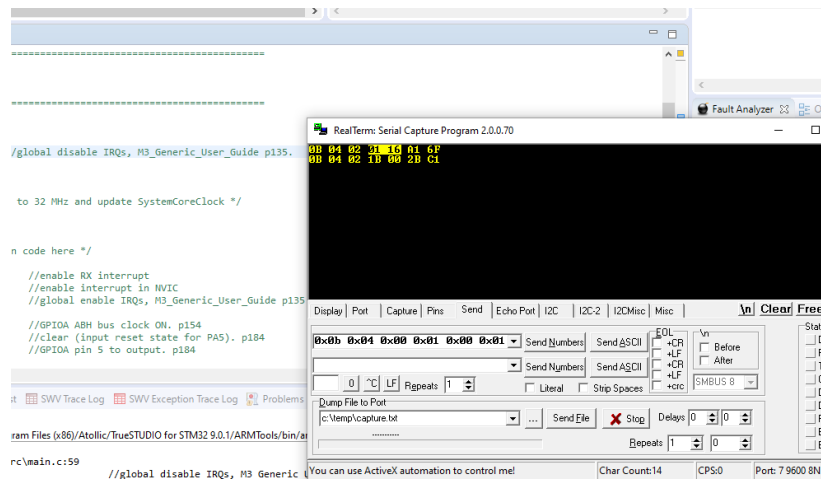We will use a RS 485 cable connected to the Block/3p and then to the MAX chip, and an external power supply (12V) connected to the Block/2P. Thus the Nucleo board will be powered up through the Vin pin and the GND pin.

Then the shield will take the 3.3V back from the Nucleo board to feed the MAX chip.

The LDR in series with the 10k resistor and we take the Voltage of that 10k resistor into pin A0, then with some calculations in the code we may transfer the value in Lux to the IoT-ticket.

Add several capacitors to protect the system.

Next step is designing the PCB in PADS Layout:
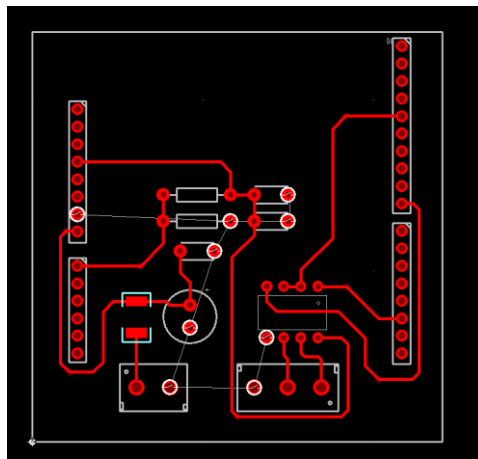
Figure 20. PCB Layout

Since we have not many components, we stack them up in 1 layer only, then we have the CAM file and print the layout onto plastic papers.
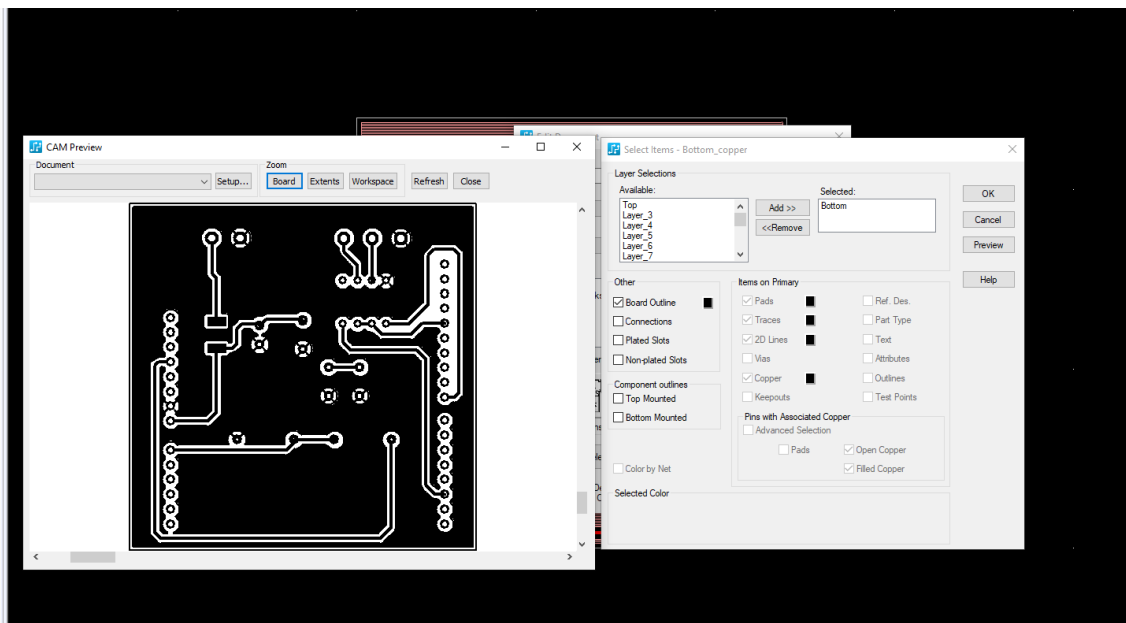
Figure 21. CAM file

There is a problem with the printers that we cannot print the layout with proper size for these row_vertical pins. Then we spend next hours re design the ratio and also the actual distance between those pins in order it to fit the Nucleo board.

Next, we test all the design on breadboard first, before making the actual PCB shield for Nucleo board. Still, we encountered errors and bugs, luckily, we have figured them all out.
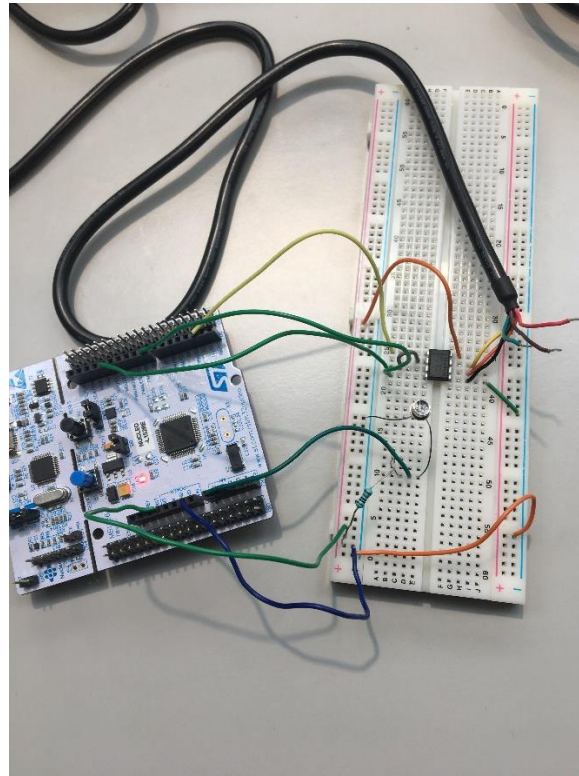
Figure 22. Testing with breadboard

The design works with the Python to IoT-ticket also. The last step is making the PCB.

The process is quite simple: use photon-intensive machine on the boards, then soaking them in the NaOH 0.7% for about 15 minutes, in case of ideal liquid (took us about 4 hours to get to the ideal condition), then drilling holes for connections of the components and also the pins to connect to the Nucleo board.

In the last day of the course, we finally make everything works well!
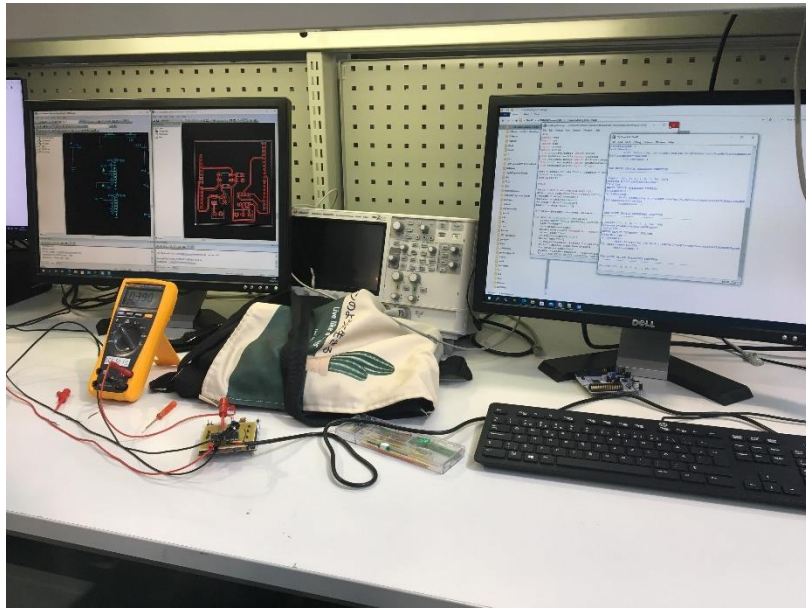
Figure 23. Final "product"

7   CONCLUSION

The course was somehow a little short and we still have things to investigate more but we have learnt a lot throughout the process.

After finishing this project, we learned Nucleo STM32, Light dependent sensor so far. The structure how to combine things to make a project from scratch with use of different platforms (IoT-ticket, from our "Master PC" to the normal screen). Beside that, we were figuring out a majority of problems could be happened with software, circuit, PCB design and the board. However, Which the carefully supported from our Teacher Mr Jani Ahvonen, we complete project with comprehensive study.

REFERENCES

https://portal.vamk.fi/pluginfile.php/707082/mod_resource/content/0/Modbus%20RTU%20slave%20frame%20implementation%20with%20C.pdf

https://www.modbustools.com/modbus.html

https://portal.vamk.fi/pluginfile.php/702327/mod_resource/content/0/STM32L152_reference_manual.pdf

https://portal.vamk.fi/pluginfile.php/702328/mod_resource/content/0/STM32L152RET6_datasheet.pdf

https://portal.vamk.fi/pluginfile.php/702365/mod_resource/content/0/MAX3485.pdf

https://portal.vamk.fi/pluginfile.php/702367/mod_resource/content/0/USB-RS485-WE-1800-BT.pdf

https://www.lammertbies.nl/comm/info/crc-calculation

https://portal.vamk.fi/pluginfile.php/702373/mod_resource/content/0/Modbus%20RTU%20SHT20.pdf

APPENDIX 1

UM1724 User manual 6.3

6.3 Power supply and power selection

The power supply is provided either by the host PC through the USB cable, or by an external source: VIN (From 7 V to 12 V), E5V (5 V) or +3.3V power supply pins on CN6 or CN7. In case VIN, E5V or +3.3V is used to power the STM32 Nucleo board, using an external power supply unit or auxiliary equipment, this power source must comply with the standard EN-60950-1: 2006+A11/2009, and must be Safety Extra Low Voltage (SELV) with limited power capability.

6.3.1 Power supply input from the USB connector

The ST-LINK/V2-1 supports USB power management allowing to request more than 100 mA current to the host PC.

All parts of the STM32 Nucleo board and shield can be powered from the ST-LINK USB connector CN1 (U5V or VBUS). Note that only the ST-LINK part is power supplied before the USB enumeration as the host PC only provides 100 mA to the board at that time. During the USB enumeration, the STM32 Nucleo board requires 300 mA of current to the host PC. If the host is able to provide the required power, the targeted STM32 microcontroller is powered and the red LED LD3 is turned ON, thus the STM32 Nucleo board and its shield can consume a maximum of 300 mA current, not more. If the host is not able to provide the required current, the targeted STM32 microcontroller and the MCU part including the extension board are not power supplied. As a consequence, the red LED LD3 remains turned OFF. In such a case it is mandatory to use an external power supply as explained in the next *Section 6.3.2: External power supply inputs: VIN and E5V*.

When the board is power supplied by USB (U5V) a jumper must be connected between pin 1 and pin 2 of JP5 as shown in *Table 8*.

JP1 is configured according to the maximum current consumption of the board when powered by USB (U5V). JP1 jumper can be set in case the board is powered by USB and maximum current consumption on U5V does not exceed 100 mA (including an eventual extension board or ARDUINO® shield). In such a condition, USB enumeration always succeeds since no more than 100 mA is requested to the PC. Possible configurations of JP1 are summarized in *Table 6*.

Table 6. JP1 configuration table

Jumper state	Power supply	Allowed current
JP1 jumper OFF	USB power through CN1	300 mA max
JP1 jumper ON		100 mA max

Warning: **If the maximum current consumption of the NUCLEO and its extension boards exceeds 300 mA, it is mandatory to power the NUCLEO using an external power supply connected to E5V or VIN.**

Note: *In case the board is powered by a USB charger, there is no USB enumeration, so the led LD3 remains set to OFF permanently and the target STM32 is not powered. In this specific case, the jumper JP1 needs to be set to ON, to allow target STM32 to be powered anyway.*
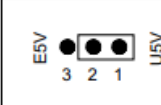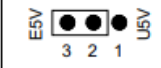
6.3.2   External power supply inputs: VIN and E5V

The external power sources VIN and E5V are summarized in *Table 7*. When the board is power supplied by VIN or E5V, the jumpers configuration must be the following:

- Jumper on JP5 pin 2 and pin 3
- Jumper removed on JP1

Table 7. External power sources

Input power name	Connectors pins	Voltage range	Max current	Limitation
VIN	CN6 pin 8 CN7 pin 24	7 V to 12 V	800 mA	From 7 V to 12 V only and input current capability is linked to input voltage: 800 mA input current when Vin = 7 V 450 mA input current when 7 V < Vin <= 9 V 250 mA input current when 9 V < Vin <= 12 V
E5V	CN7 pin 6	4.75 V to 5.25 V	500 mA	-

Table 8. Power-related jumper

Jumper	Description
JP5	U5V (ST-LINK VBUS) is used as a power source when JP5 is set as shown below (Default setting)
	VIN or E5V is used as a power source when JP5 is set as shown below.

Using VIN or E5V as external power supply

VIN or E5V can be used as an external power supply in case the current consumption of the STM32 Nucleo and extensions boards exceeds the allowed current on USB. In this condition, it is still possible to use the USB for communication, for programming or debugging only, but it is mandatory to power supply the board first using VIN or E5V then connect the USB cable to the PC. Proceeding this way ensures that the enumeration occurs thanks to the external power source.

The following power sequence procedure must be respected:

1. Connect the jumper between pin 2 and pin 3 of JP5
2. Check that JP1 is removed
3. Connect the external power source to VIN or E5V
4. Power on the external power supply 7 V< VIN < 12 V to VIN, or 5 V for E5V
5. Check that LD3 is turned ON
6. Connect the PC to USB connector CN1

If this order is not respected, the board may be supplied by VBUS first then by VIN or E5V, and the following risks may be encountered:

1. If more than 300 mA current is needed by the board, the PC may be damaged or the current supply can be limited by the PC. As a consequence, the board is not powered correctly.
2. 300 mA is requested at enumeration (since JP1 must be OFF) so there is a risk that the request is rejected and the enumeration does not succeed if the PC cannot provide such current. Consequently, the board is not power supplied (LED LD3 remains OFF).

6.3.3 External power supply input: +3.3V

It can be of interest to use the +3.3V (CN6 pin 4 or CN7 pin 12 and pin 16) directly as power input for instance in case the 3.3V is provided by an extension board. When the STM32 Nucleo board is power supplied by +3.3V, the ST-LINK is not powered, thus the programming and debug features are unavailable. The +3.3V external power source is summarized in *Table 9*.

Table 9. +3.3 V external power source

Input power name	Connectors pins	Voltage range	Limitation
+3.3V	CN6 pin 4 CN7 pin 12 and pin 16	3 V to 3.6 V	Used when ST-LINK part of PCB is cut or SB2 and SB12 OFF

Two different configurations are possible when using +3.3V to power the board:

*   ST-LINK is removed (PCB cut) or
*   SB2 (3.3V regulator) and SB12 (NRST) are OFF.

6.3.4 External power supply output

When powered by USB, VIN, or E5V, the +5V (CN6 pin 5 or CN7 pin 18) can be used as an output power supply for an ARDUINO® shield or an extension board. In this case, the maximum current of the power source specified in *Table 7* must be respected.

The +3.3V (CN6 pin 4 or CN7 pin 12 and 16) can be used also as power supply output. The current is limited by the maximum current capability of the regulator U4 (500 mA max).

APPENDIX 2

http://www.cc.puv.fi/~e1800920/ModbusRTU-RS485-LDR.c

http://www.cc.puv.fi/~e1800920/masterpython.py