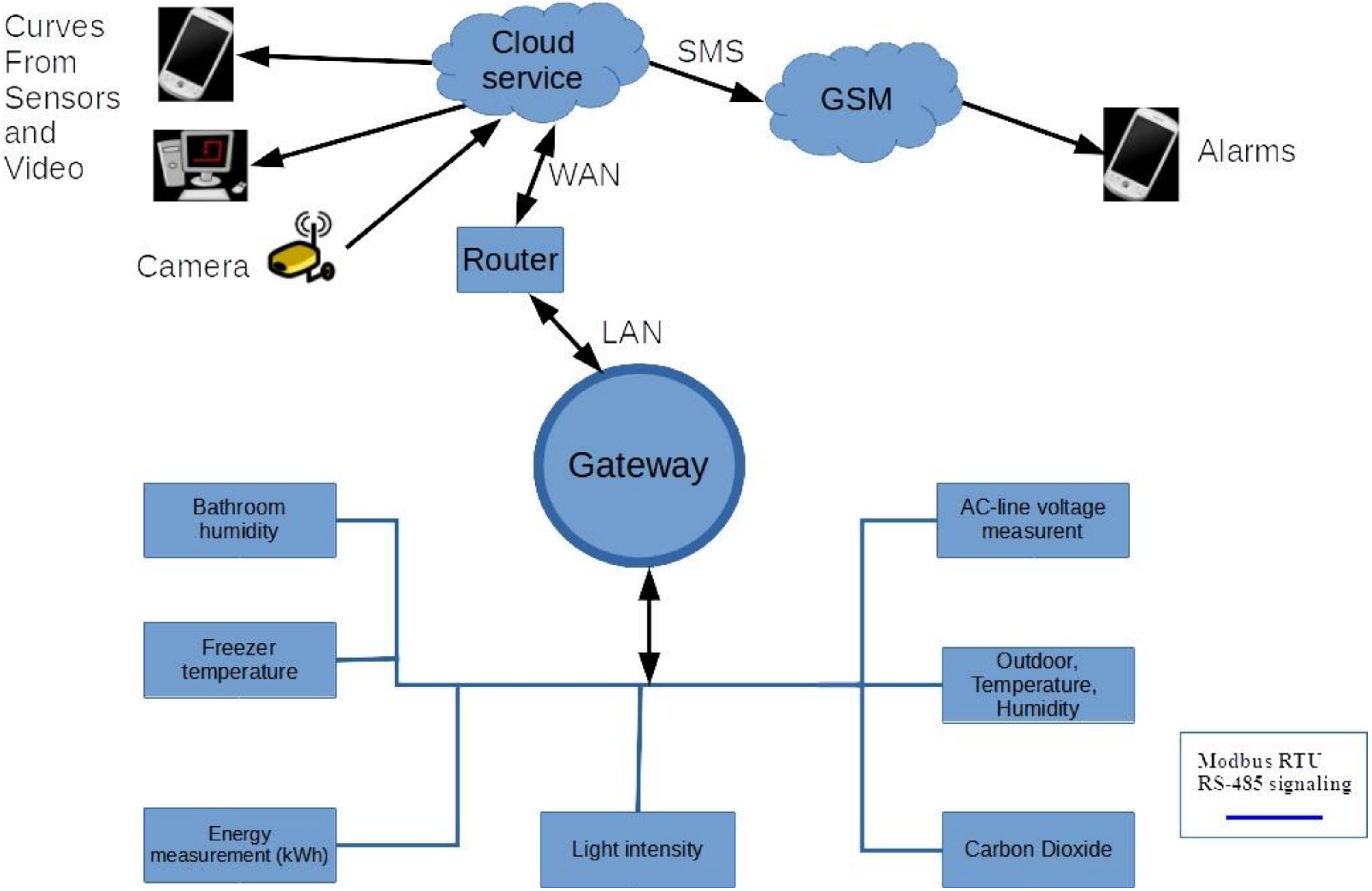
Modbus RTU slave frame implementation with C for ARM microcontroller

Project steps listed

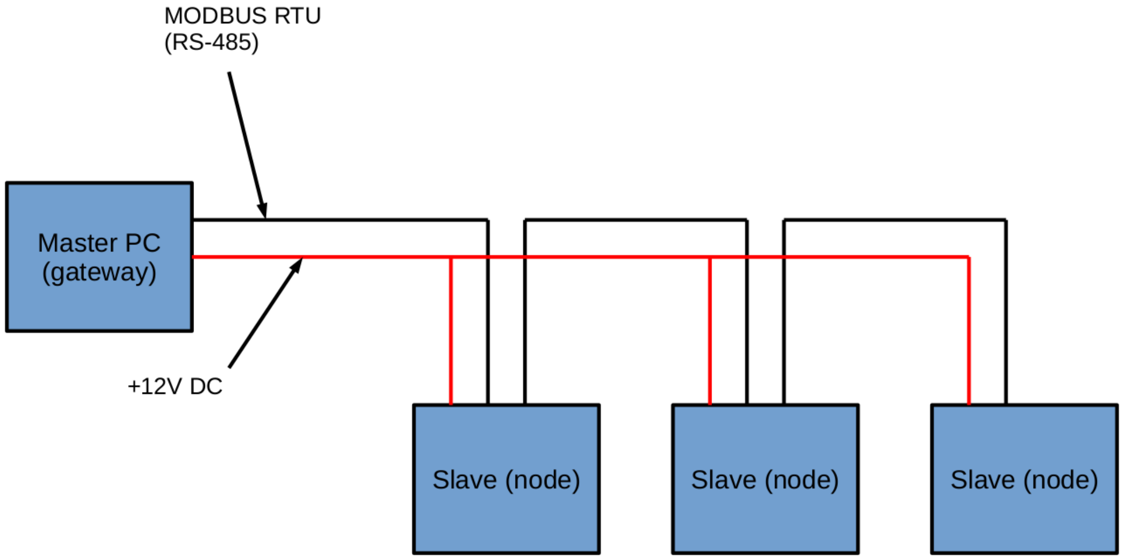
- The sensor and its study
- Signaling RS-485 (HW)
- Frame programming

Each group selects **one rectangular box** and searches a suitable sensor.



Block hardware and sensor code design

Design your own block and test it in the breadboard. The power source for your device is a +12 V DC. Your electronics must be fitted to Nucleo152RE development board (inside Arduino connectors). Choose other connectors so that you can use +12 V DC. Nucleo board can convert +12 V to 3.3V (how?). Serach your sensor datasheet from www.farnell.com and www.elfa.se.



Block hardware and sensor code design

- Design software for your sensor and then create a custom function that can be added later to your MODBUS protocol.

  For example:

  int sensor_value=readsensor(input_register); // read sensor value from your function
- sensor_value=sensor_value*100; //we need to send data without decimal. Master will convert it to decimal value.

To the report: Commented sensor code

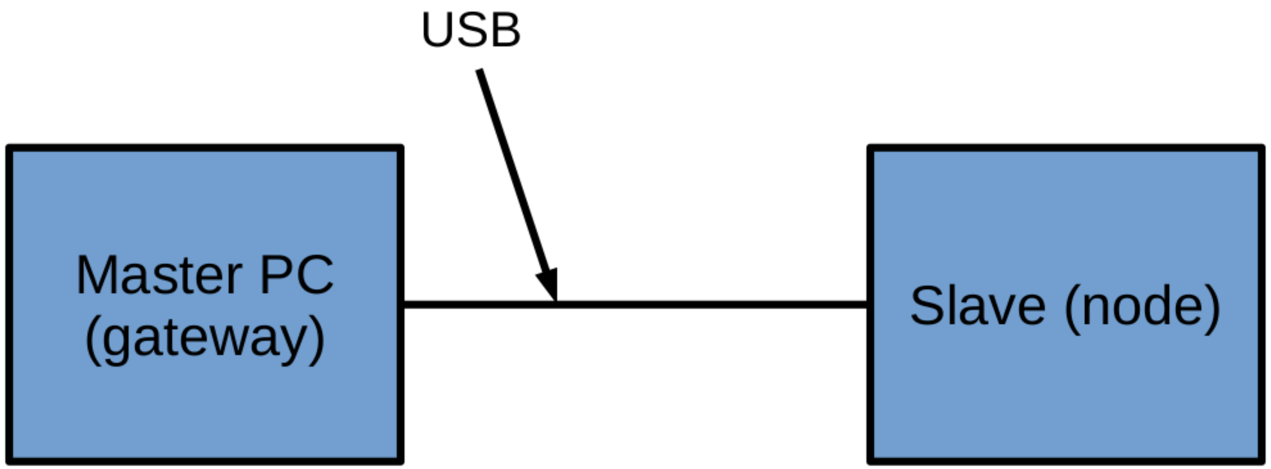
Block hardware and sensor code design

- Draw PADS-Logic circuit diagram for your block.
- Use following components for PADS schematic for RS-485 and +12 V DC:
  - TM_BLOCK/3P, TM_BLOCK/2P, MAX3485CPA+, PTC resistor as a fuse MF-MSMF050-2 - PPTC Resettable Fuse (farnell code 9350314RL), etc...

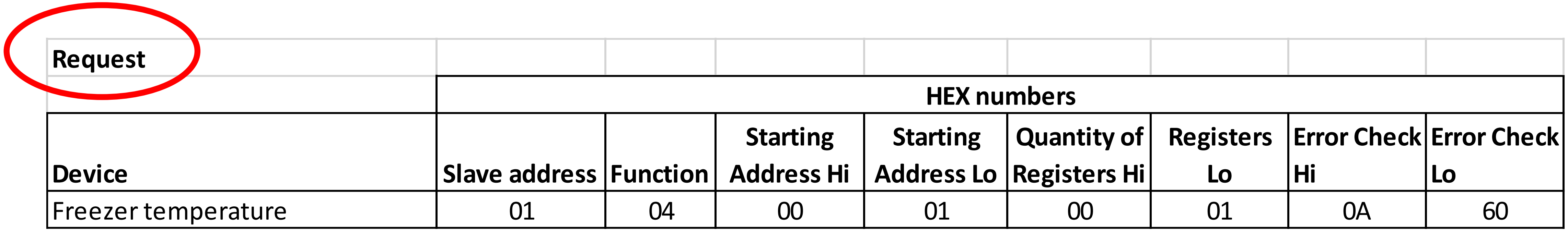
To the report: Commented sensor code

Instructions for MODBUS RTU frame development

- Your hardware connection needs only PC and your slave (nucleo) and your sensor connected to nucleo.



Here are the slave addresses of our project blocks, these can be found from assignment.



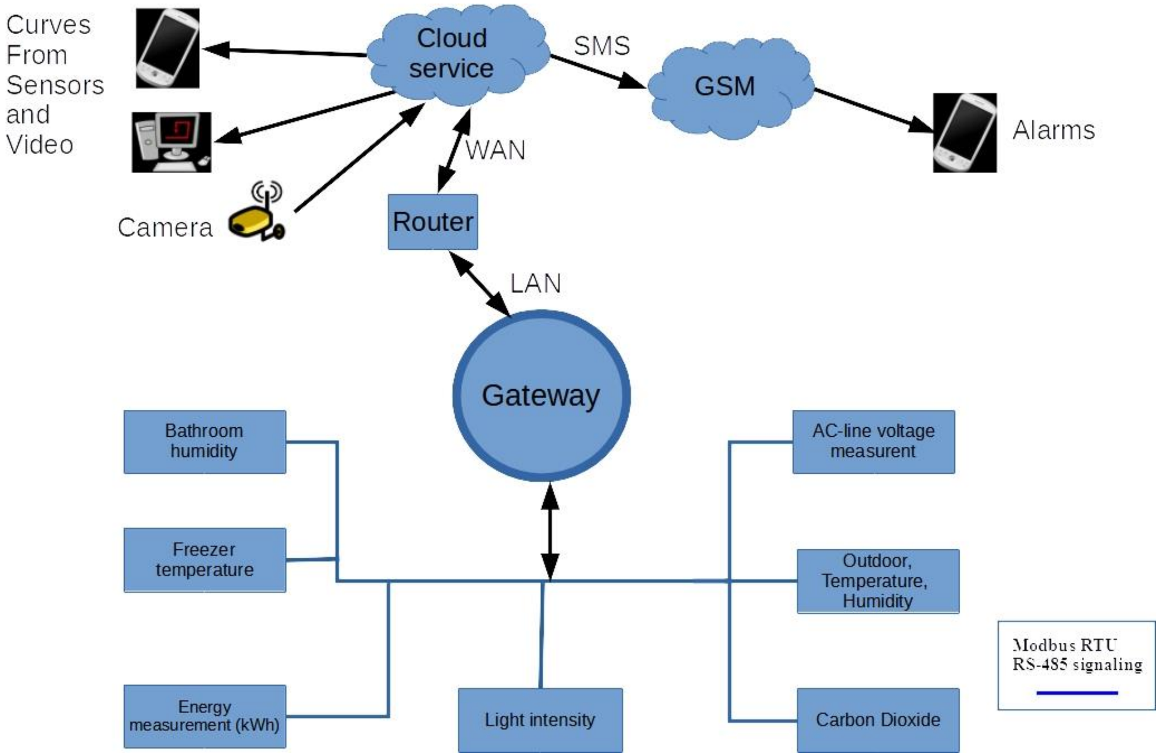
Request								
	HEX numbers							
Device	Slave address	Function	Starting Address Hi	Starting Address Lo	Quantity of Registers Hi	Registers Lo	Error Check Hi	Error Check Lo
Freezer temperature	01	04	00	01	00	01	0A	60

Master sends frame (series of hexadecimal numbers)

Can be written like this to realterm when you want to send numbers. 0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A

0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A

Here are the slave addresses of our project blocks, these can be found in the assignment.



Device	Slave address
Freezer temperature	01
Bathroom humidity	02
Energy measurement (kWh)	03
Light intensity measurement	04
Carbon dioxide measurement	05
Outdood temperature, humidity	06
AC line voltage measurement	07
extra sensor	08

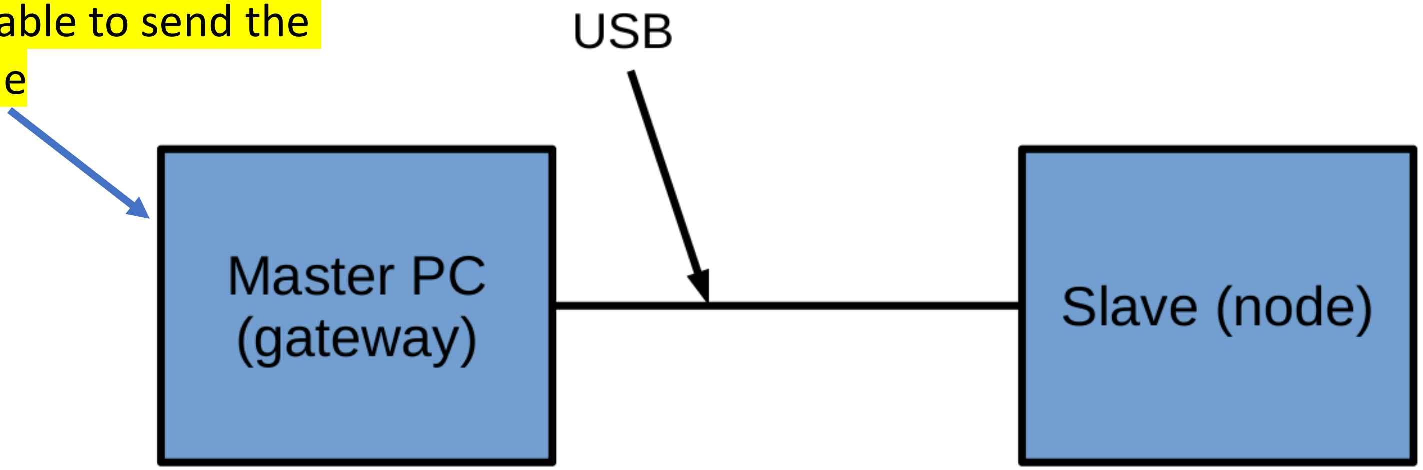
Here are the slave addresses of our project blocks, these can be found in the assignment.

If you want to read both House temperature And humidity so Starting Address Lo can be temperature 01 and humidity 02.

Device	Slave address	Function	Starting Address Hi	Starting Address Lo
Freezer temperature	01	04	00	01
Bathroom humidity	02	04	00	01
Energy measurement (kWh)	03	04	00	01
Light intensity measurement	04	04	00	01
Carbon dioxide measurement	05	04	00	01
Outdood temperature, humidity	06	04	00	01
AC line voltage measurement	07	04	00	01
extra sensor	08	04	00	01

Frame development. Use USART2. You need only PC and Nucleo board

- Create this MODBUS RTU frame with USART2. Later we will convert it to USART1 with MAX3485 chip. But first we develop our software with USART2. Use USART2 RX interrupt. Hints from student pack 18_USART_interrupt.c. Use 9600 baudrate.



Realterm is able to send the master frame

Frame development. Use USART2. You need only PC and Nucleo board

RTU mode (frame)

A frame is sent from the realterm. For example, the following (master sends) request frame in hexadecimal

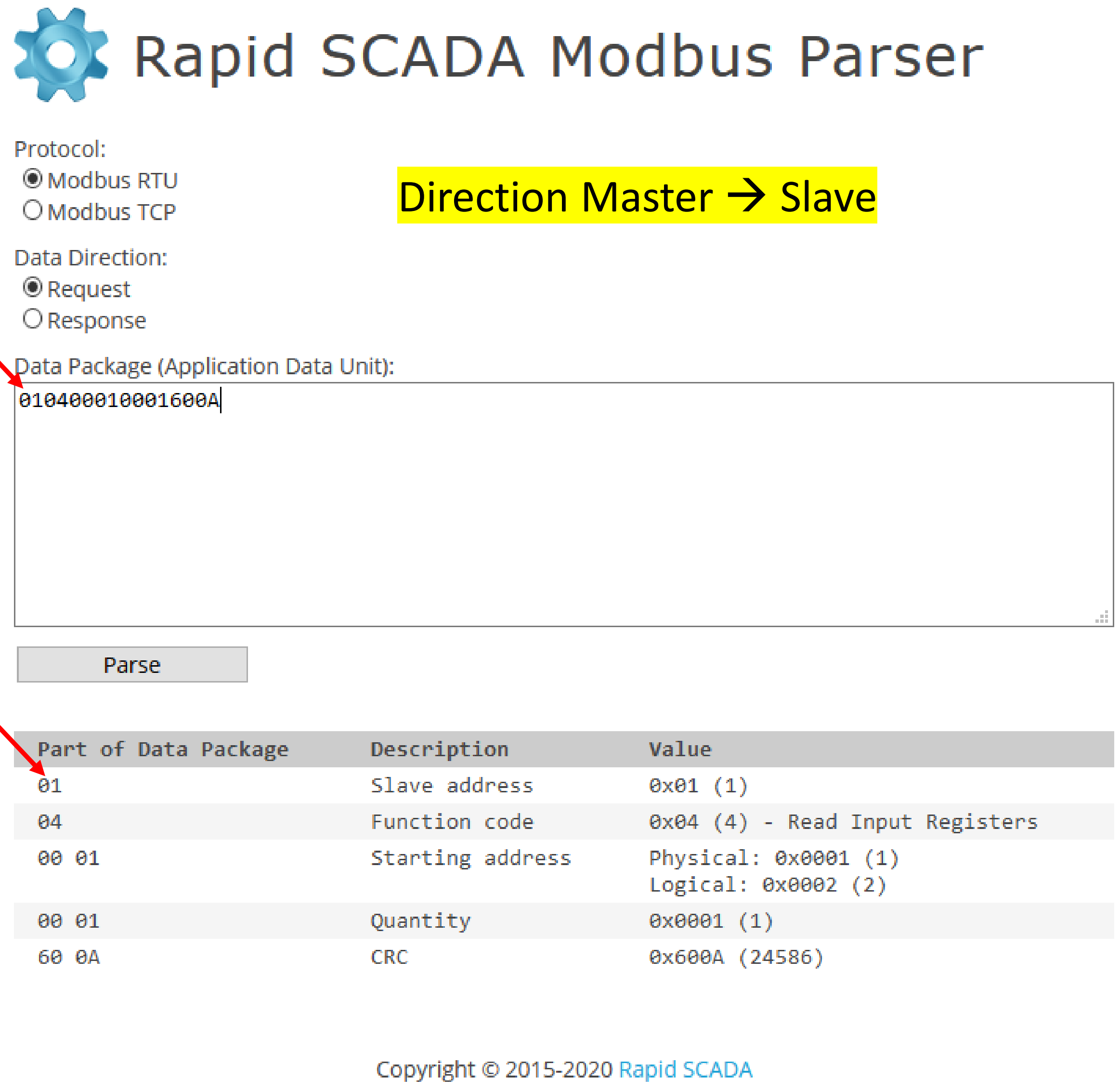
- 0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A Ready-made programs can be used to “unpack” the frame
  - https://rapidscada.net/modbus/ModbusParser.aspx

Modbus RTU “unpacked”

0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A

The first number is slave (block diagram box address eg. freezer temperature 01)

Direction Master → Slave



Rapid SCADA Modbus Parser

Protocol:
● Modbus RTU
○ Modbus TCP

Data Direction:
● Request
○ Response

Data Package (Application Data Unit):

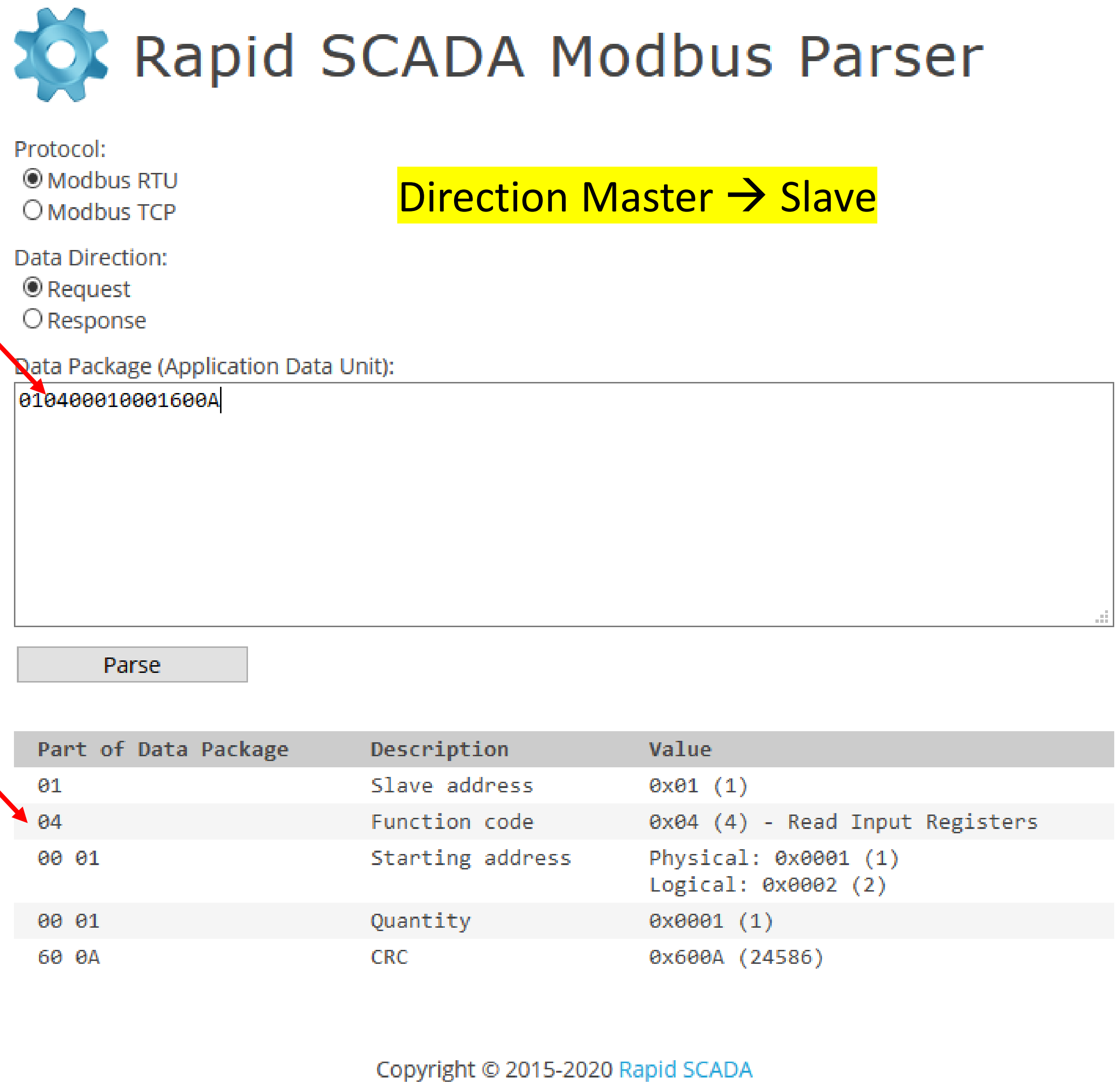
```
010400010001600A
```

Part of Data Package	Description	Value
01	Slave address	0x01 (1)
04	Function code	0x04 (4) - Read Input Registers
00 01	Starting address	Physical: 0x0001 (1) Logical: 0x0002 (2)
00 01	Quantity	0x0001 (1)
60 0A	CRC	0x600A (24586)

Modbus RTU “unpacked”

0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A

The second number “read registers from a slave device”. In this project, this is always the case 0x04.



Direction Master → Slave

Rapid SCADA Modbus Parser

Protocol:
● Modbus RTU
○ Modbus TCP

Data Direction:
● Request
○ Response

Data Package (Application Data Unit):

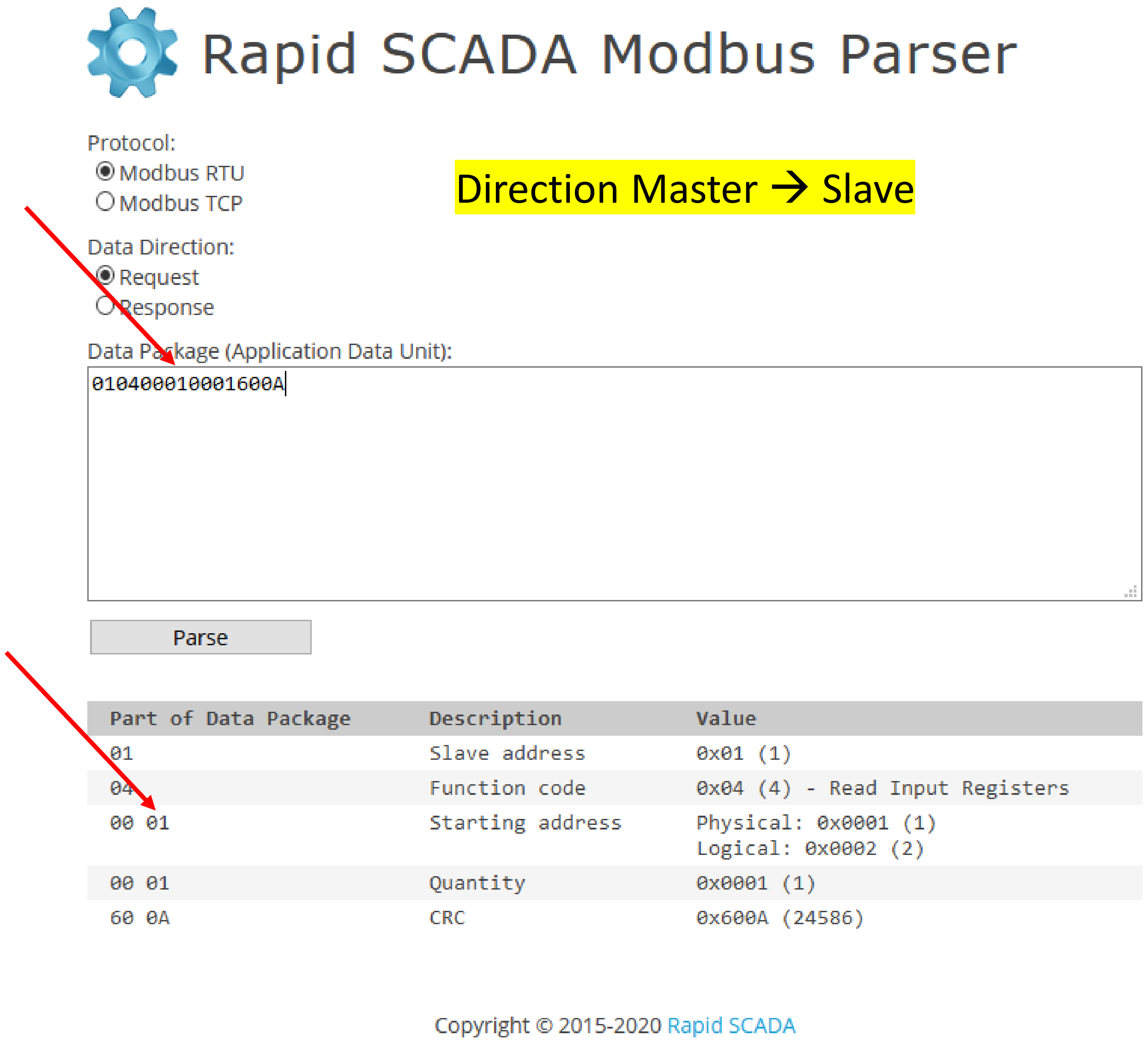
```
010400010001600A
```

Part of Data Package	Description	Value
01	Slave address	0x01 (1)
04	Function code	0x04 (4) - Read Input Registers
00 01	Starting address	Physical: 0x0001 (1) Logical: 0x0002 (2)
00 01	Quantity	0x0001 (1)
60 0A	CRC	0x600A (24586)

Modbus RTU “unpacked”

0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A

If one blue box (one nucleo board) contains many sensors, they could be assigned their own address. Eg 1 temperature and 2 humidity.



Direction Master → Slave

Rapid SCADA Modbus Parser

Protocol:
● Modbus RTU
○ Modbus TCP

Data Direction:
● Request
○ Response

Data Package (Application Data Unit):

```
010400010001600A
```

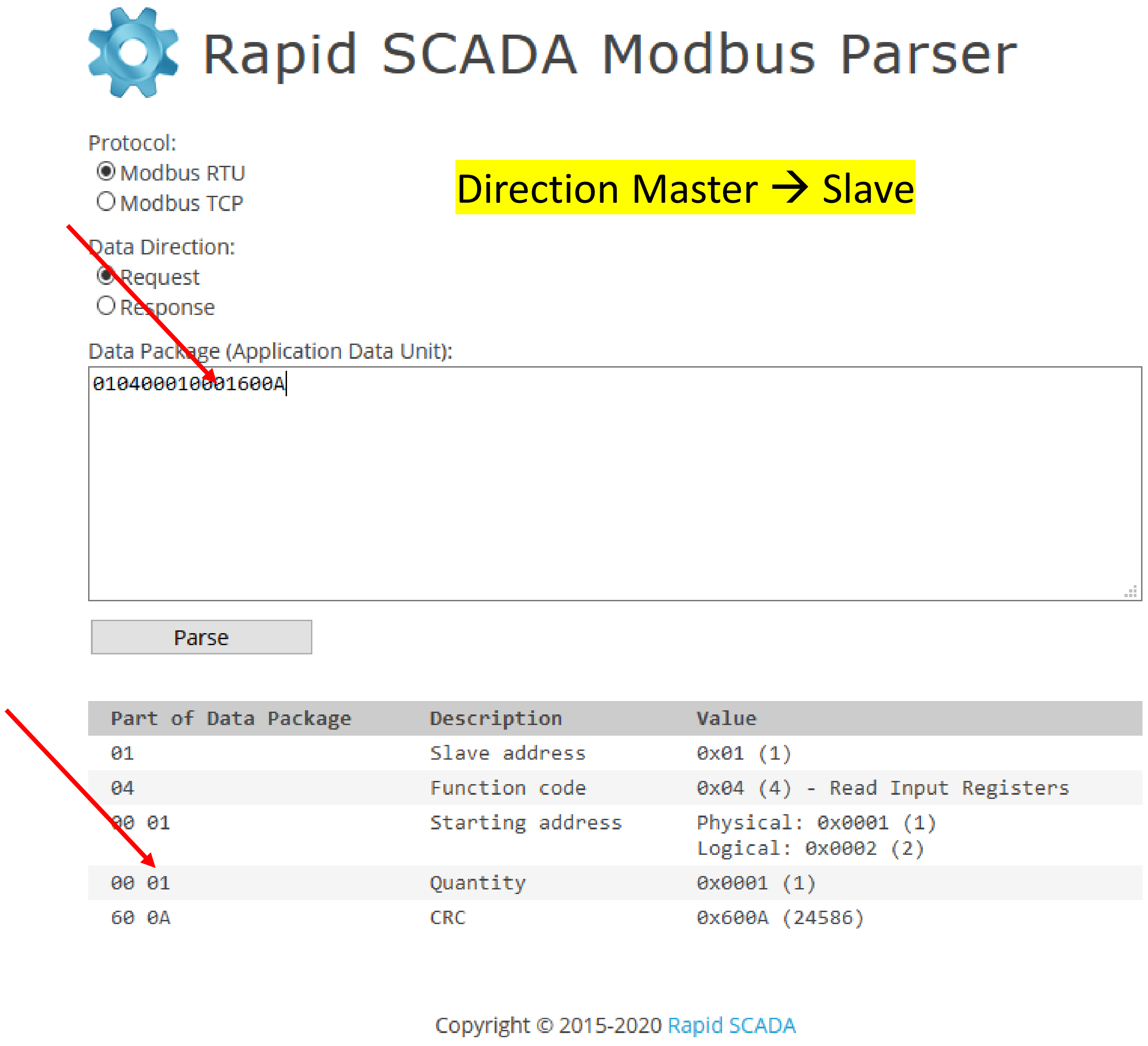
Part of Data Package	Description	Value
01	Slave address	0x01 (1)
04	Function code	0x04 (4) - Read Input Registers
00 01	Starting address	Physical: 0x0001 (1) Logical: 0x0002 (2)
00 01	Quantity	0x0001 (1)
60 0A	CRC	0x600A (24586)

Copyright © 2015-2020 Rapid SCADA

Modbus RTU “unpacked”

0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A

How many 16-bit registers we want to read from a slave device? One register is 16 bit (so two 8-bit characters are read).



Direction Master → Slave

Rapid SCADA Modbus Parser

Protocol:
● Modbus RTU
○ Modbus TCP

Data Direction:
● Request
○ Response

Data Package (Application Data Unit):

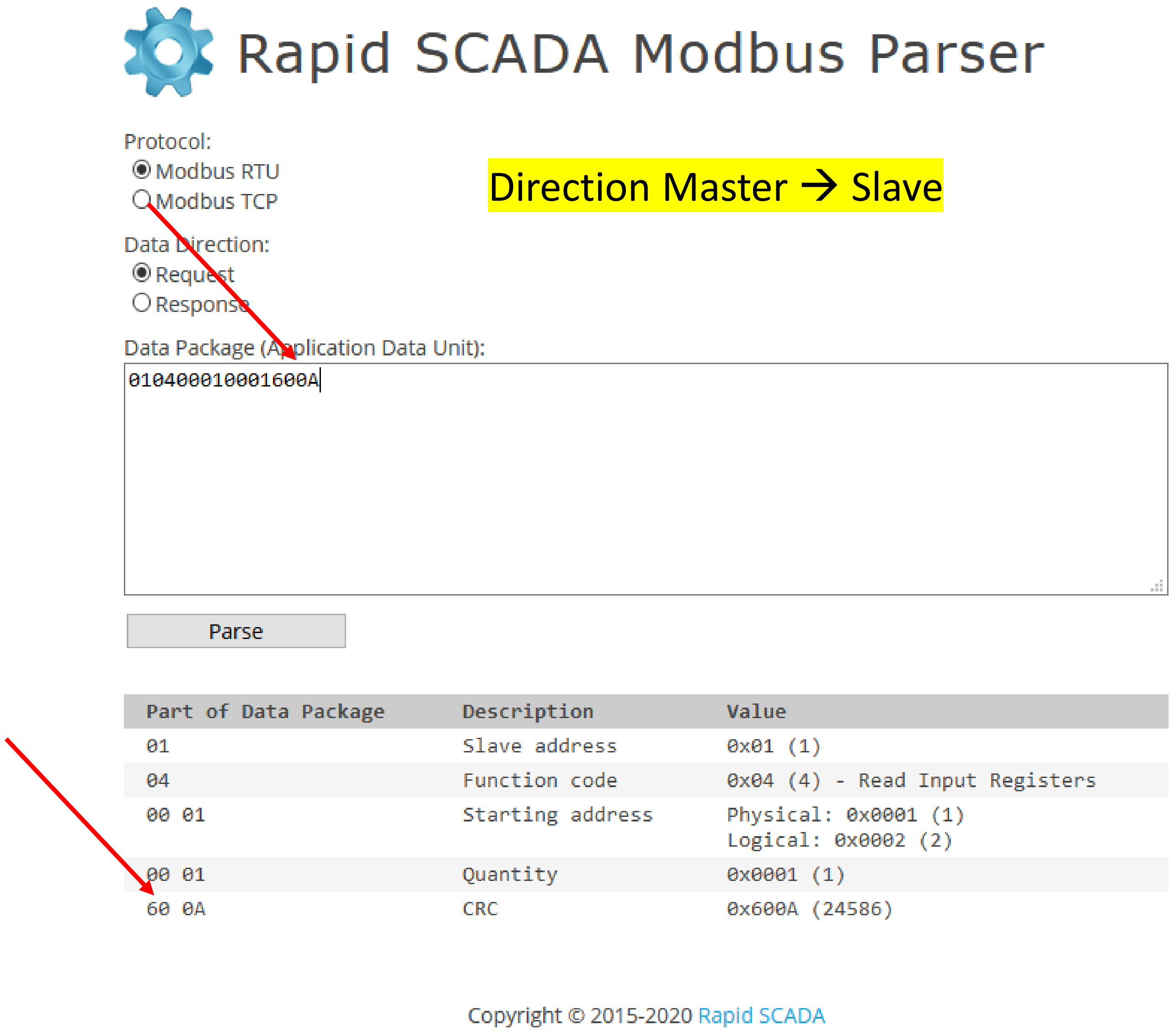
```
010400010001600A
```

Part of Data Package	Description	Value
01	Slave address	0x01 (1)
04	Function code	0x04 (4) - Read Input Registers
00 01	Starting address	Physical: 0x0001 (1) Logical: 0x0002 (2)
00 01	Quantity	0x0001 (1)
60 0A	CRC	0x600A (24586)

Modbus RTU “unpacked”

0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A

Finally, a checksum CRC is calculated from the data. CRC is included.



Rapid SCADA Modbus Parser

Direction Master → Slave

Protocol:
● Modbus RTU
○ Modbus TCP

Data Direction:
● Request
○ Response

Data Package (Application Data Unit):

```
010400010001600A
```

Part of Data Package	Description	Value
01	Slave address	0x01 (1)
04	Function code	0x04 (4) - Read Input Registers
00 01	Starting address	Physical: 0x0001 (1) Logical: 0x0002 (2)
00 01	Quantity	0x0001 (1)
60 0A	CRC	0x600A (24586)

Here we tried another group address "frame"! Now Slave address 0x04

Direction Master → Slave

The Master CRC can be applied to your own block as follows. Eg Light intensity measurement. https://www.lammertbies.nl/comm/info/crc-calculation

CRC-16 (Modbus)

040400010001????

Device	Slave address
Freezer temperature	01
Bathroom humidity	02
Energy measurement (kWh)	03
Light intensity measurement	04
Carbon dioxide measurement	05
Outdood temperature, humidity	06
AC line voltage measurement	07
extra sensor	08

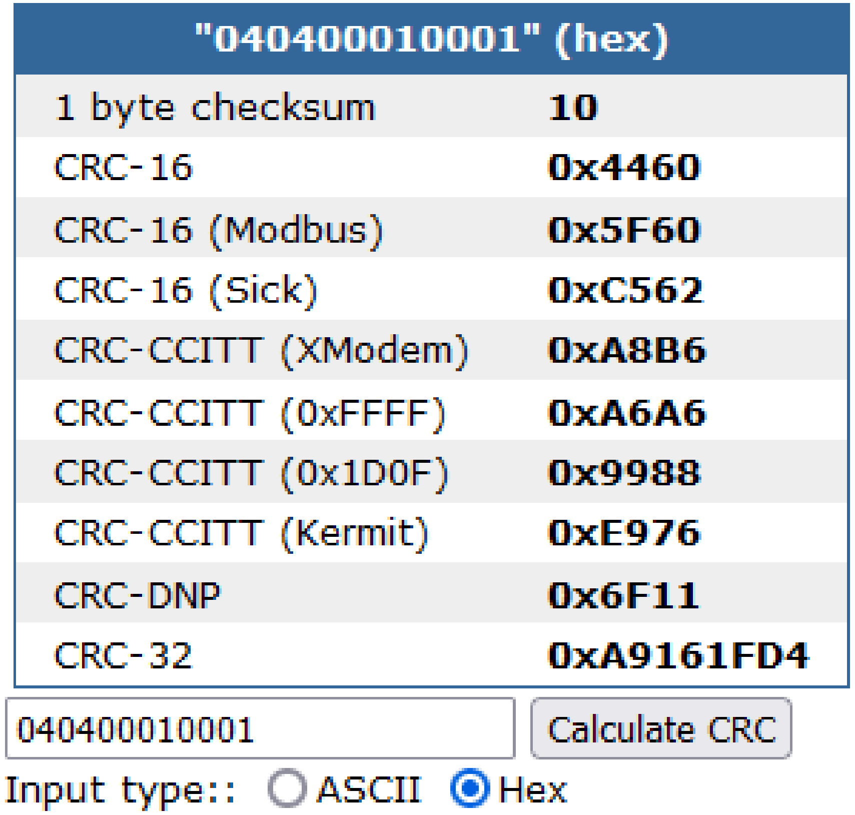
Here we tried another group address “frame”! Now Slave address 0x04

Suunta Master → Slave

The Master CRC can be applied to your own block as follows. Eg Light intensity measurement. https://www.lammertbies.nl/comm/info/crc-calculation

CRC-16 (Modbus)
040400010001????

order 605F in frame



"040400010001" (hex)	
1 byte checksum	10
CRC-16	0x4460
CRC-16 (Modbus)	0x5F60
CRC-16 (Sick)	0xC562
CRC-CCITT (XModem)	0xA8B6
CRC-CCITT (0xFFFF)	0xA6A6
CRC-CCITT (0x1D0F)	0x9988
CRC-CCITT (Kermit)	0xE976
CRC-DNP	0x6F11
CRC-32	0xA9161FD4

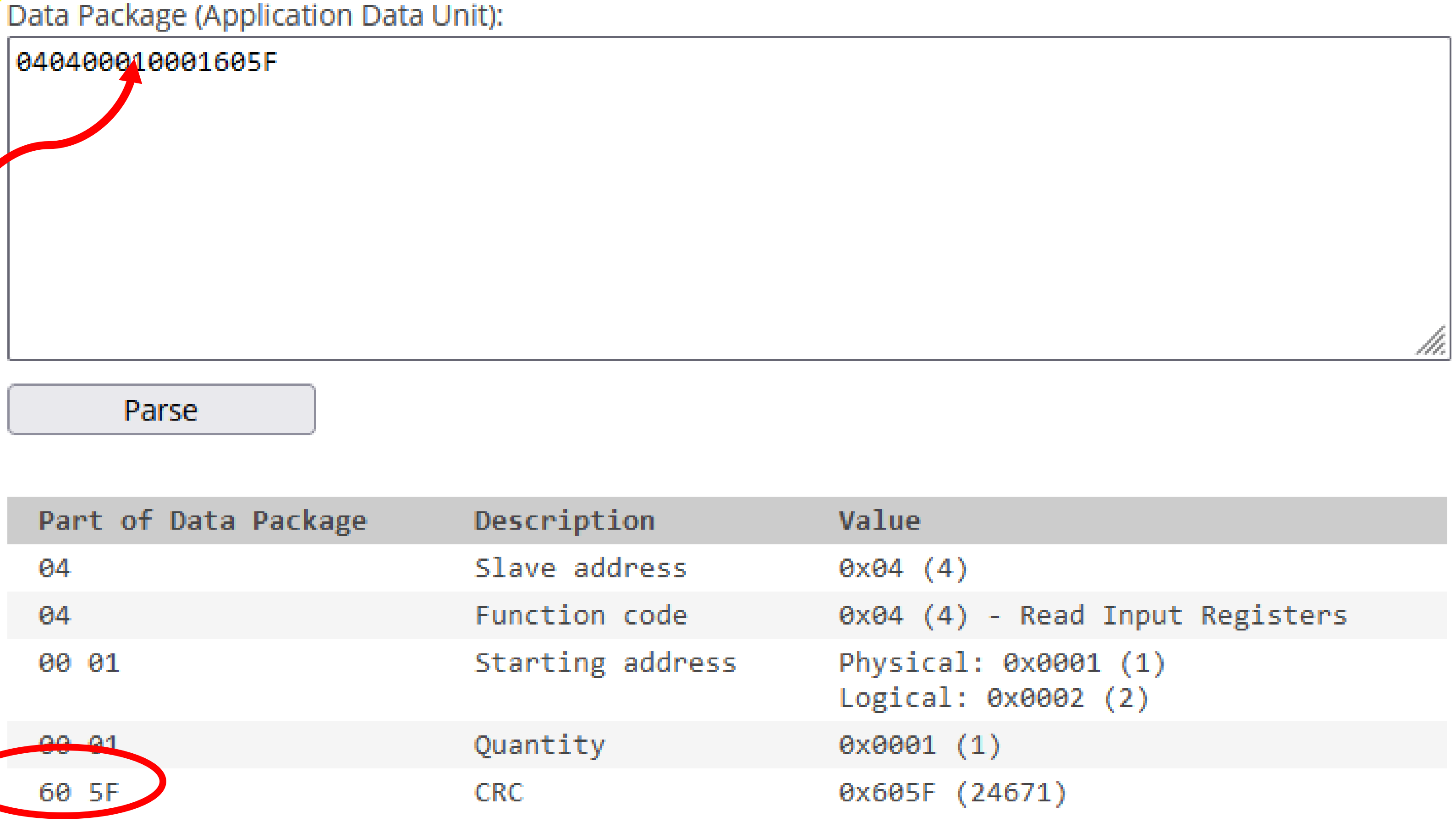
040400010001

Input type:: ○ ASCII ● Hex

Here we tried another group address “frame”! Now Slave address 0x04

Suunta Master → Slave

CRC-16 (Modbus)
040400010001605F



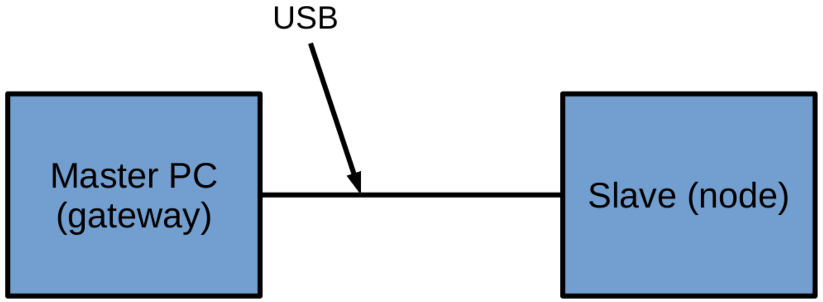
Data Package (Application Data Unit):

```
040400010001605F
```

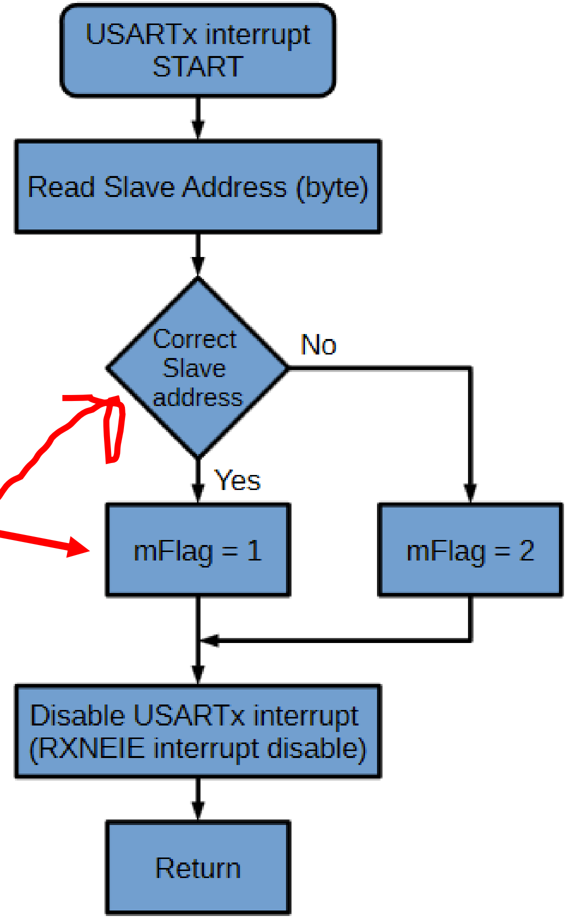
Part of Data Package	Description	Value
04	Slave address	0x04 (4)
04	Function code	0x04 (4) - Read Input Registers
00 01	Starting address	Physical: 0x0001 (1) Logical: 0x0002 (2)
00 01	Quantity	0x0001 (1)
60 5F	CRC	0x605F (24671)

Master request frame now complete. Let's start the development of slave code in C language.

Create this MODBUS RTU frame with USART2. Later we will convert it to USART1 with MAX3485 chip. But first we develop our software with USART2. Use USART2 RX interrupt. Hints from student pack 18_USART_interrupt.c. Use 9600 baudrate.



USARTx interrupt


USARTx interrupt START
Read Slave Address (byte)
Correct Slave address
No
Yes
mFlag = 1
mFlag = 2
Disable USARTx interrupt (RXNEIE interrupt disable)
Return

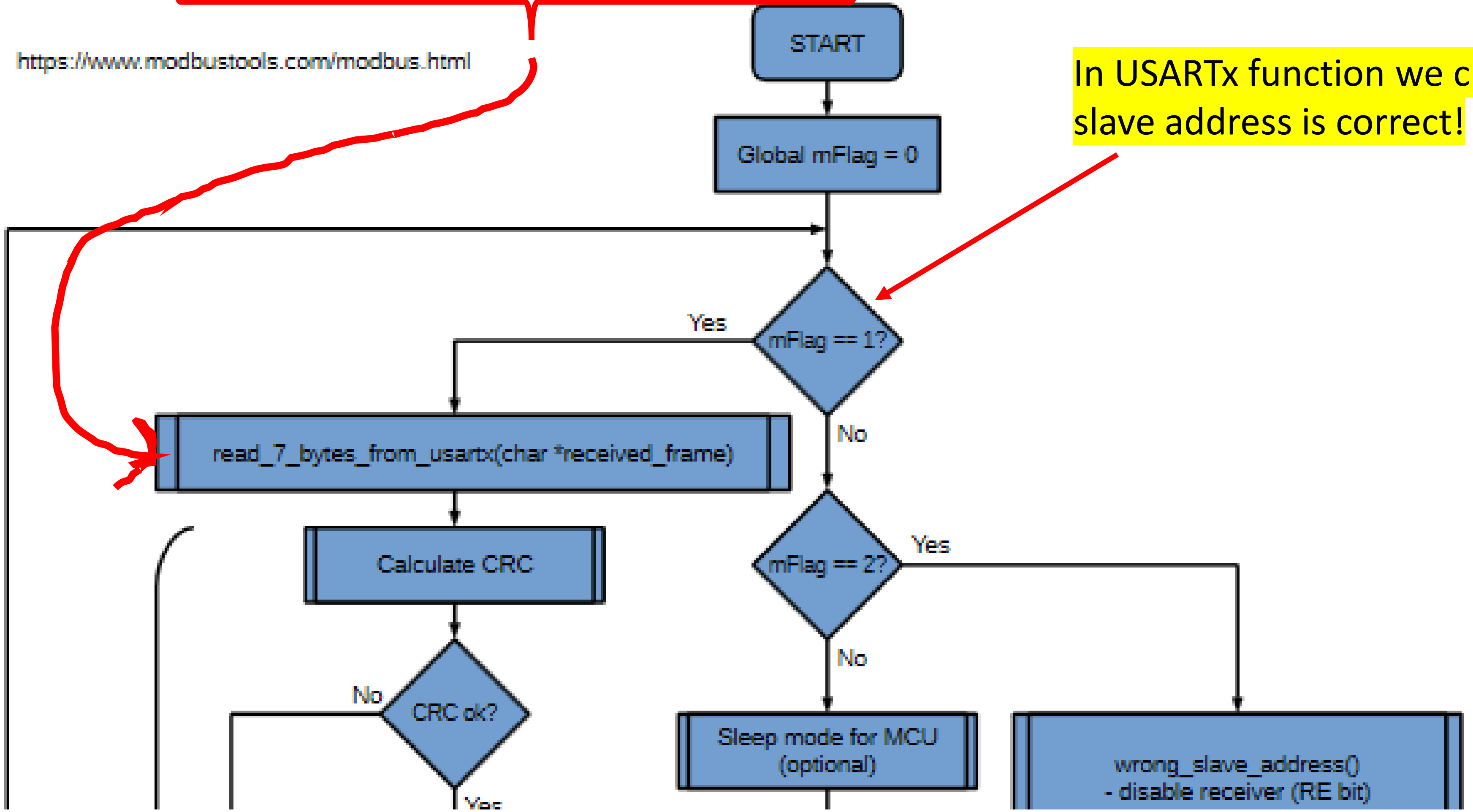
Global Variable

Master request 0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A

Slave program development in C language. Main program:

Master request 0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A

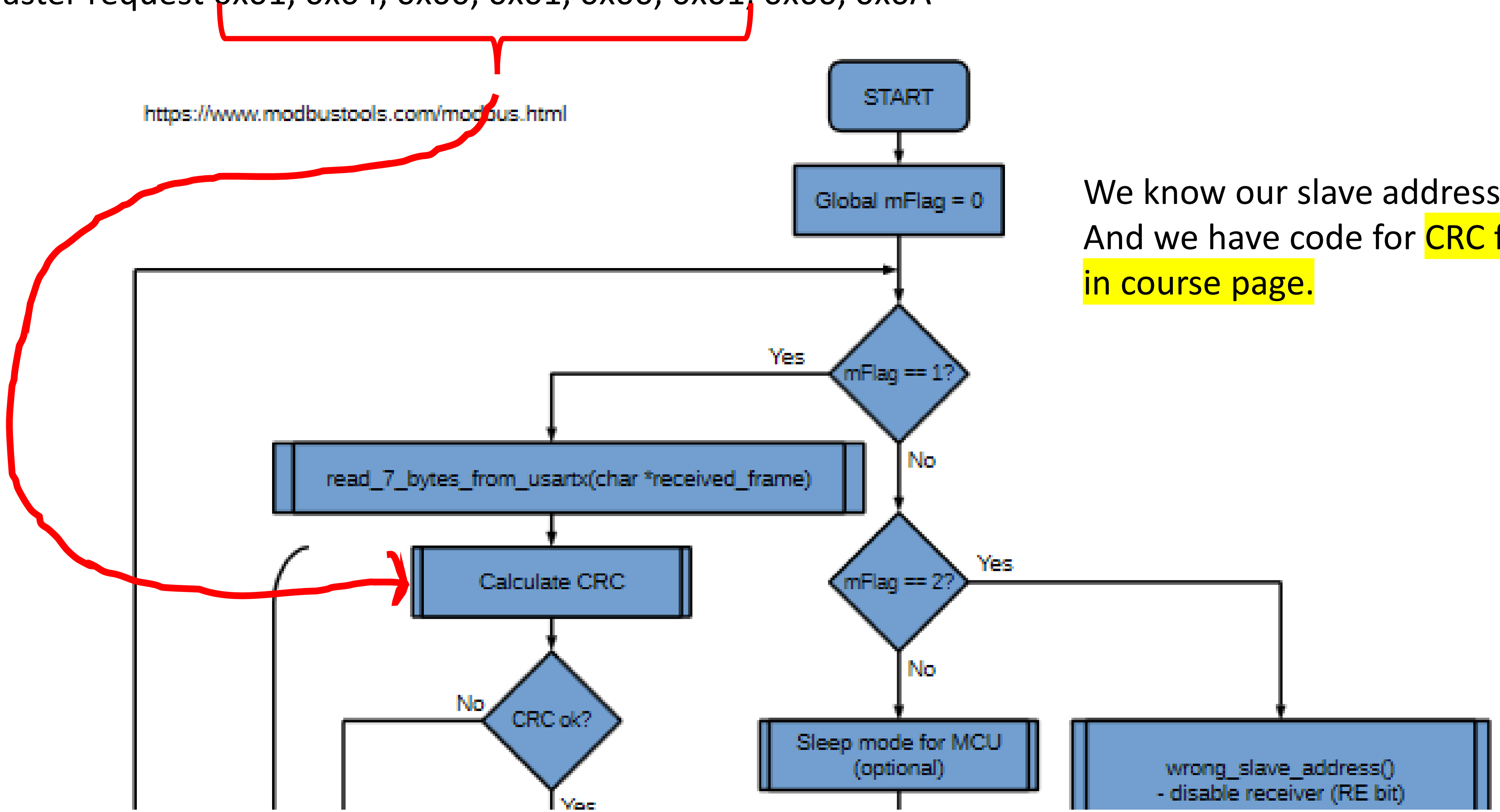
https://www.modbustools.com/modbus.html



In USARTx function we check that slave address is correct!

Slave program development in C language. Main program:

Master request 0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A



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

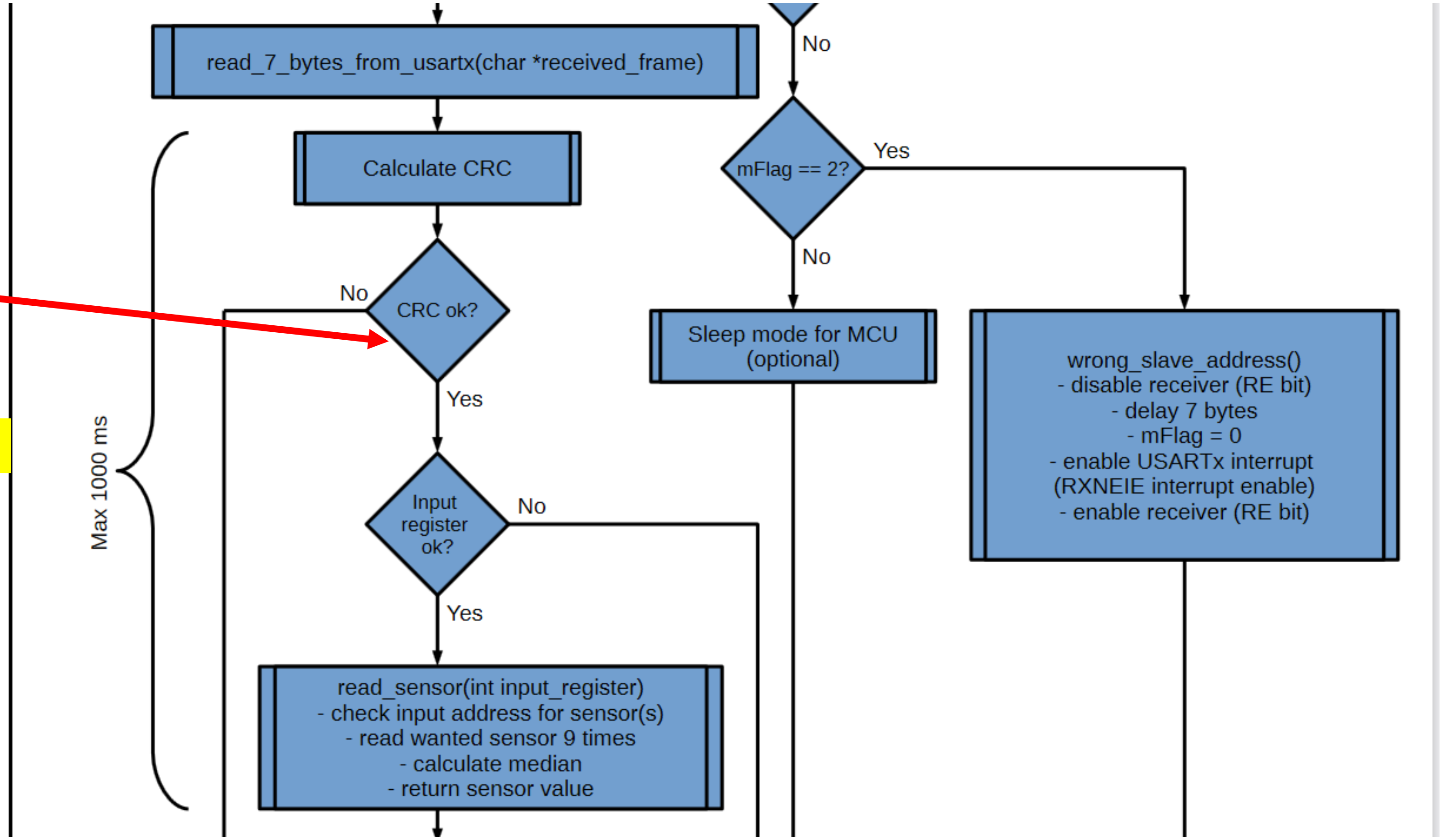
START
Global mFlag = 0
mFlag == 1?
Yes
No
read_7_bytes_from_usartx(char *received_frame)
Calculate CRC
CRC ok?
No
Yes
mFlag == 2?
Yes
No
Sleep mode for MCU (optional)
wrong_slave_address()
- disable receiver (RE bit)

We know our slave address!
And we have code for CRC function in course page.

Slave program development in C language. Main program:

Master request 0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A

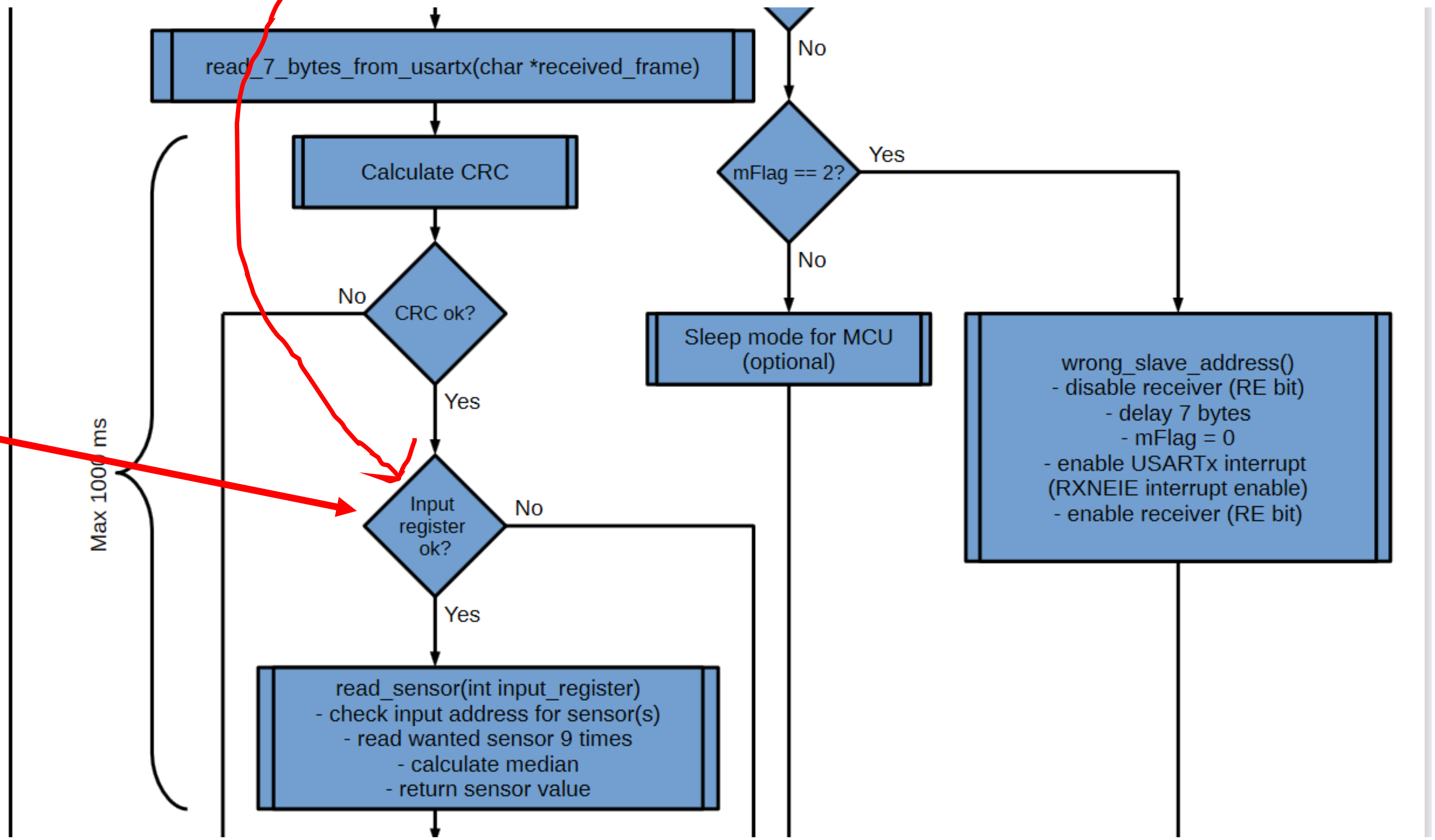
If the CRC calculated by the function is the same as in the received Master request frame then continue…


read_7_bytes_from_usartx(char *received_frame)
Calculate CRC
CRC ok?
No
Yes
Input register ok?
No
Yes
read_sensor(int input_register)
- check input address for sensor(s)
- read wanted sensor 9 times
- calculate median
- return sensor value
Max 1000 ms
No
mFlag == 2?
Yes
No
Sleep mode for MCU (optional)
wrong_slave_address()
- disable receiver (RE bit)
- delay 7 bytes
- mFlag = 0
- enable USARTx interrupt (RXNEIE interrupt enable)
- enable receiver (RE bit)

Slave program development in C language. Main program:

Master request 0x01, 0x04, 0x00, 0x01, 0x00, 0x01, 0x60, 0x0A

If the block contains more than one sensor then the input register can be checked. Sensor 1 0x01 and sensor 2 0x02, etc.... …



read_7_bytes_from_usartx(char *received_frame)

Calculate CRC

CRC ok?

No

Yes

Input register ok?

No

Yes

read_sensor(int input_register)
- check input address for sensor(s)
- read wanted sensor 9 times
- calculate median
- return sensor value

Max 1000 ms

No

mFlag == 2?

Yes

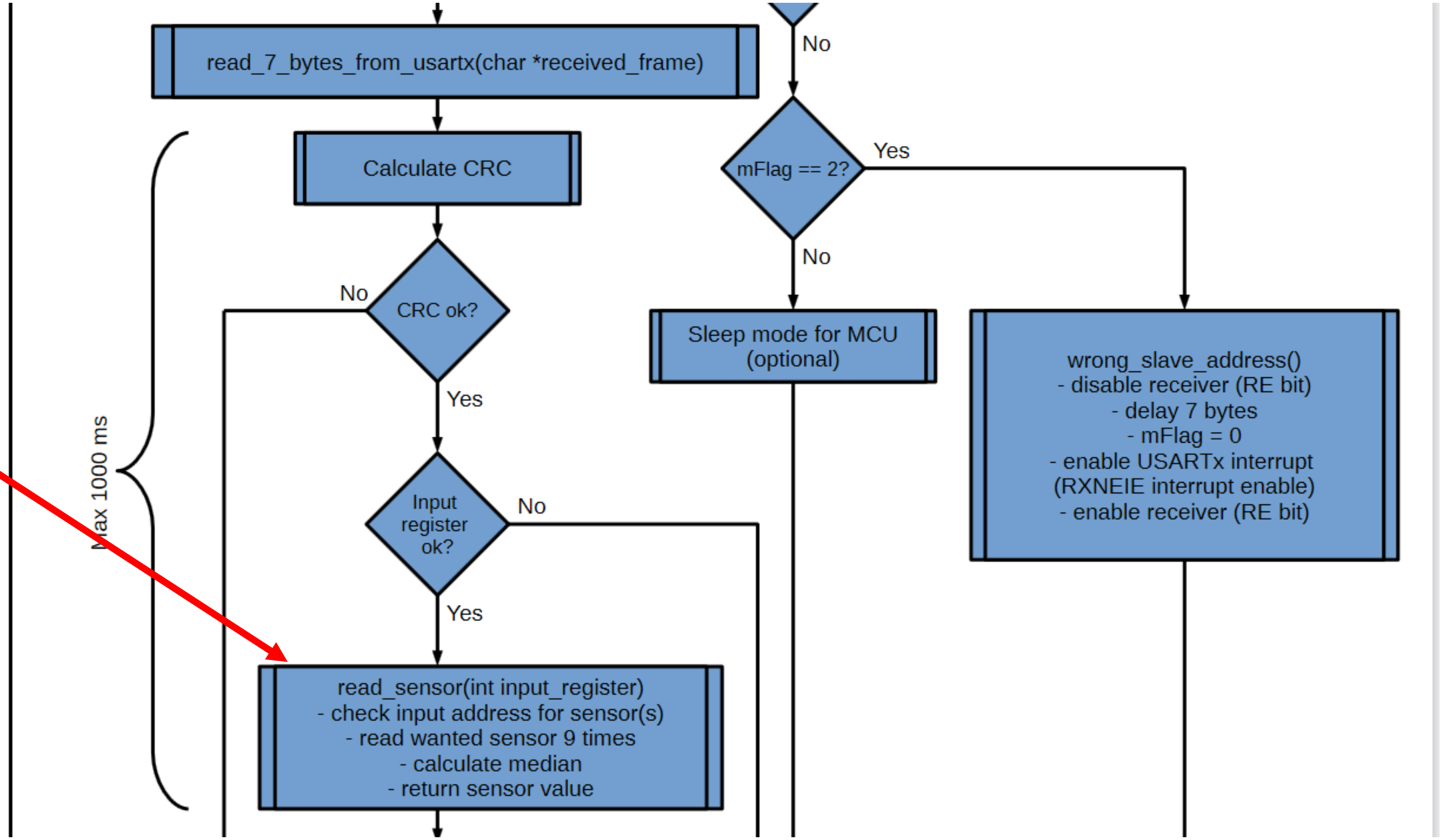
No

Sleep mode for MCU (optional)

wrong_slave_address()
- disable receiver (RE bit)
- delay 7 bytes
- mFlag = 0
- enable USARTx interrupt (RXNEIE interrupt enable)
- enable receiver (RE bit)

Slave program development in C language. Main program

Let's read the sensor 9 times and calculate the median

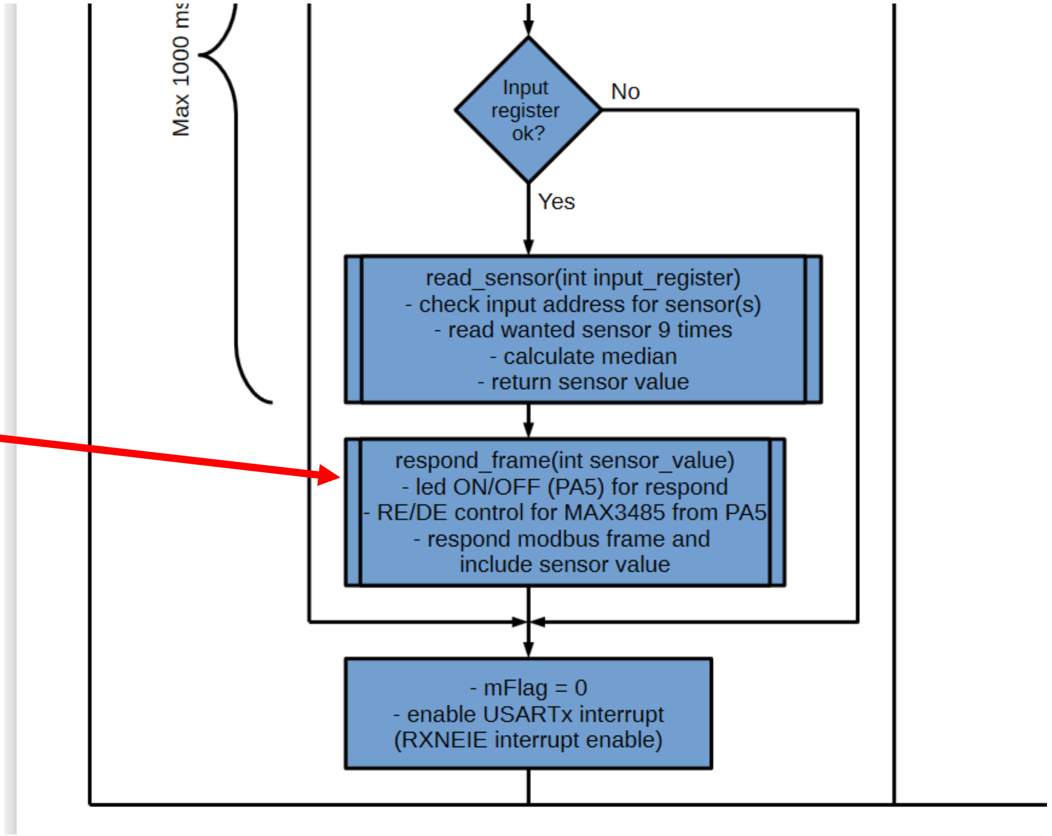


read_7_bytes_from_usartx(char *received_frame)
No
mFlag == 2?
Yes
No
Calculate CRC
CRC ok?
No
Yes
Sleep mode for MCU (optional)
wrong_slave_address()
- disable receiver (RE bit)
- delay 7 bytes
- mFlag = 0
- enable USARTx interrupt (RXNEIE interrupt enable)
- enable receiver (RE bit)
Max 1000 ms
Input register ok?
No
Yes
read_sensor(int input_register)
- check input address for sensor(s)
- read wanted sensor 9 times
- calculate median
- return sensor value

Slave program development in C language. Main program

Respond to the master machine. You need to create a respond frame.

Later when using the real RS-485 signaling! Here you must also control the MAX3485 transmit / receive pin



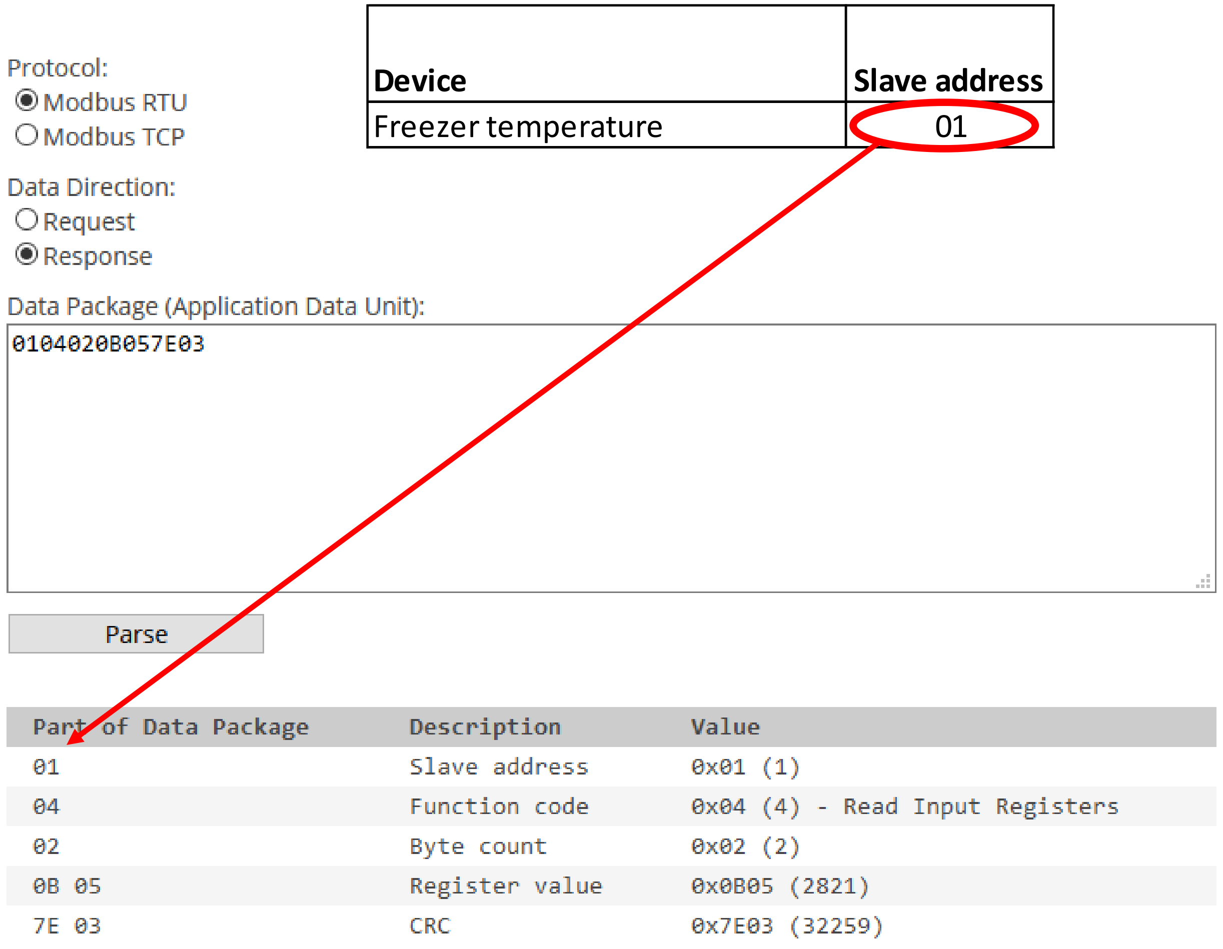
Max 1000 ms
Input register ok?
No
Yes
read_sensor(int input_register)
- check input address for sensor(s)
- read wanted sensor 9 times
- calculate median
- return sensor value
respond_frame(int sensor_value)
- led ON/OFF (PA5) for respond
- RE/DE control for MAX3485 from PA5
- respond modbus frame and include sensor value
- mFlag = 0
- enable USARTx interrupt (RXNEIE interrupt enable)

Direction Slave → Master

Respond frame! The slave sends the master to the device.

0104020B057E03

- Slave address 0x01 in the example
- Function code 0x04 always like this in our case
- Byte count 0x02 we will send in two bytes
- Register value 0x0B05 is 2821 DEC which means the fridge melted (+28.21 Celsius)



Device	Slave address
Freezer temperature	01

Protocol:
● Modbus RTU
○ Modbus TCP

Data Direction:
○ Request
● Response

Data Package (Application Data Unit):

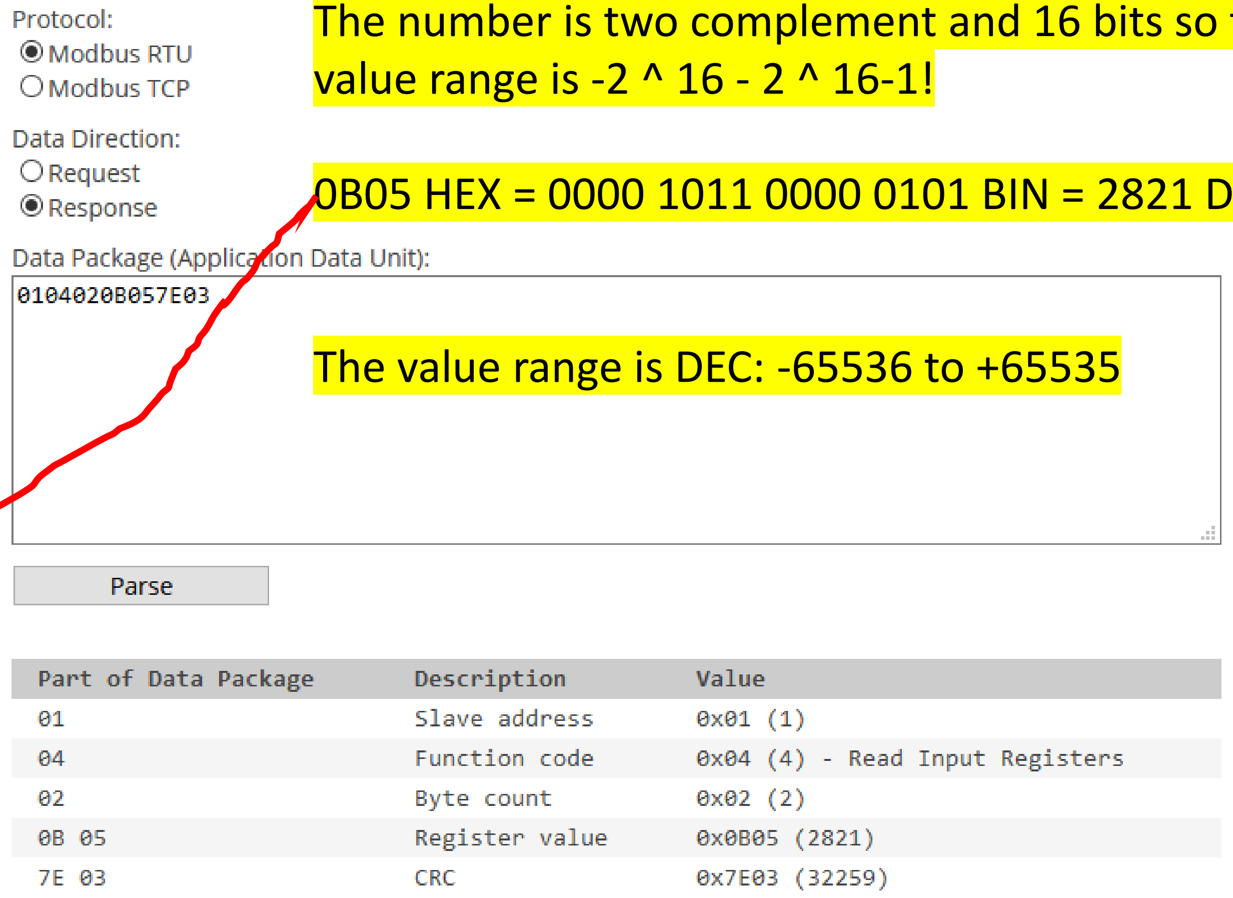
```
0104020B057E03
```

Part of Data Package	Description	Value
01	Slave address	0x01 (1)
04	Function code	0x04 (4) - Read Input Registers
02	Byte count	0x02 (2)
0B 05	Register value	0x0B05 (2821)
7E 03	CRC	0x7E03 (32259)

Respond frame! The slave sends the master to the device.

0104020B057E03

- Slave address 0x01 in the example
- Function code 0x04 always like this in our case
- Byte count 0x02 we will send in two bytes
- Register value 0x0B05 is 2821 DEC which means the fridge melted!! It is 28.21 Celsius! ☹



Protocol:
● Modbus RTU
○ Modbus TCP

Data Direction:
○ Request
● Response

Data Package (Application Data Unit):

```
0104020B057E03
```

Part of Data Package	Description	Value
01	Slave address	0x01 (1)
04	Function code	0x04 (4) - Read Input Registers
02	Byte count	0x02 (2)
0B 05	Register value	0x0B05 (2821)
7E 03	CRC	0x7E03 (32259)

The number is two complement and 16 bits so the value range is -2 ^ 16 - 2 ^ 16-1!

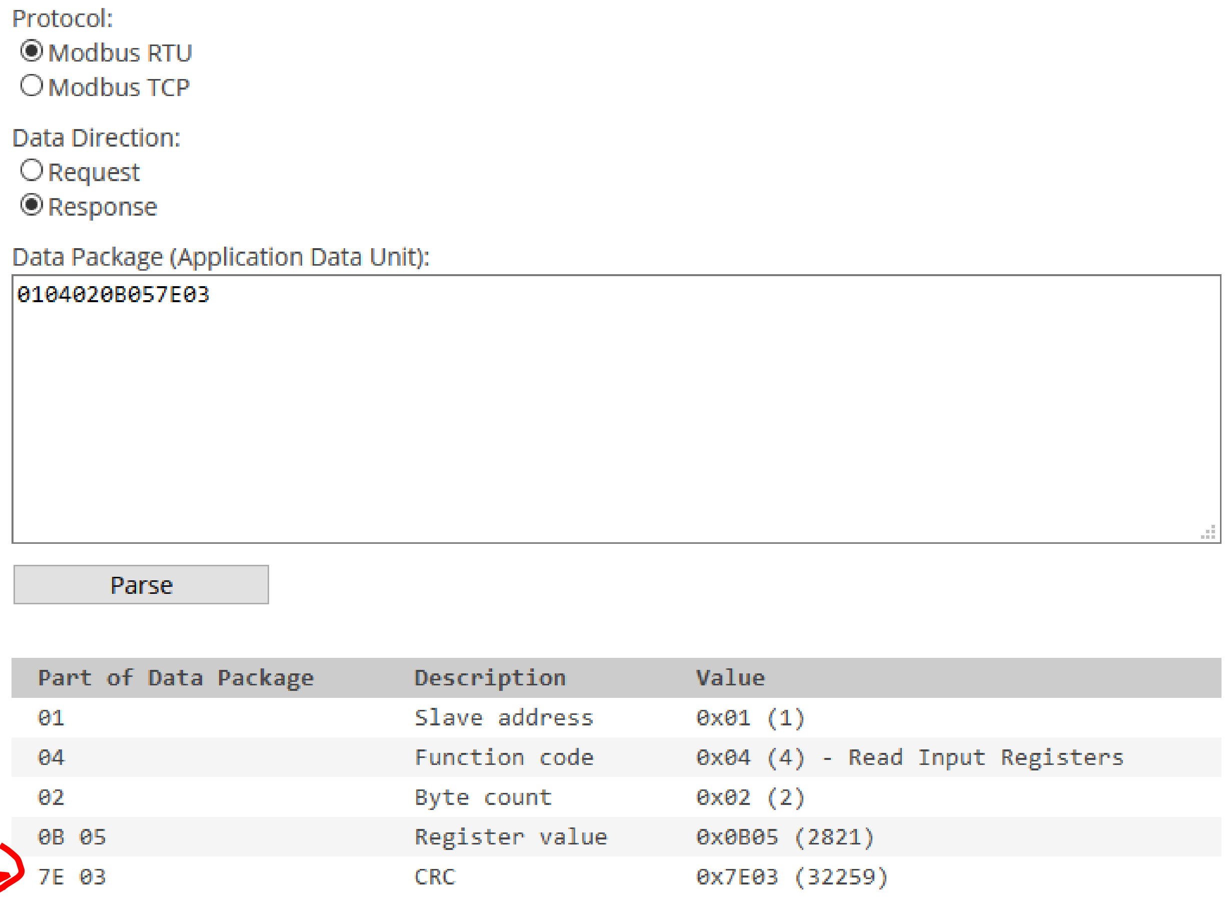
0B05 HEX = 0000 1011 0000 0101 BIN = 2821 DEC

The value range is DEC: -65536 to +65535

Respond frame! The slave sends to master device.

0104020B057E03

- CRC 0x7E03 the CRC check must be calculated in the program for each transmission separately. For example, the temperature changes almost every time. CRC Function given in the exercise.



Protocol:
● Modbus RTU
○ Modbus TCP

Data Direction:
○ Request
● Response

Data Package (Application Data Unit):

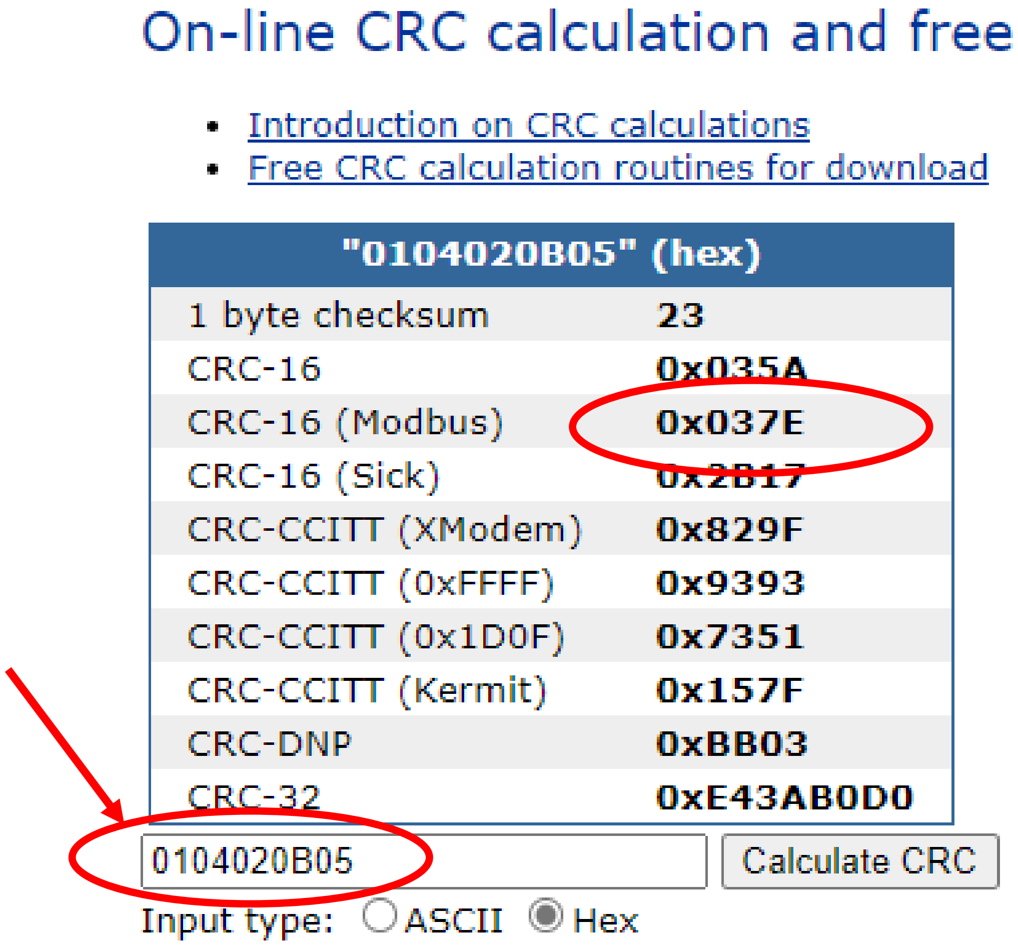
```
0104020B057E03
```

Part of Data Package	Description	Value
01	Slave address	0x01 (1)
04	Function code	0x04 (4) - Read Input Registers
02	Byte count	0x02 (2)
0B 05	Register value	0x0B05 (2821)
7E 03	CRC	0x7E03 (32259)

Respond frame! The slave sends to master device.

0104020B057E03

- CRC 0x7E03 the CRC check must be calculated in the program for each transmission separately. Because, for example, the temperature changes every time. CRC Function given in the exercise.



On-line CRC calculation and free library

- Introduction on CRC calculations
- Free CRC calculation routines for download

"0104020B05" (hex)	
1 byte checksum	23
CRC-16	0x035A
CRC-16 (Modbus)	0x037E
CRC-16 (Sick)	0x2B17
CRC-CCITT (XModem)	0x829F
CRC-CCITT (0xFFFF)	0x9393
CRC-CCITT (0x1D0F)	0x7351
CRC-CCITT (Kermit)	0x157F
CRC-DNP	0xBB03
CRC-32	0xE43AB0D0

0104020B05

Input type: ○ ASCII ● Hex

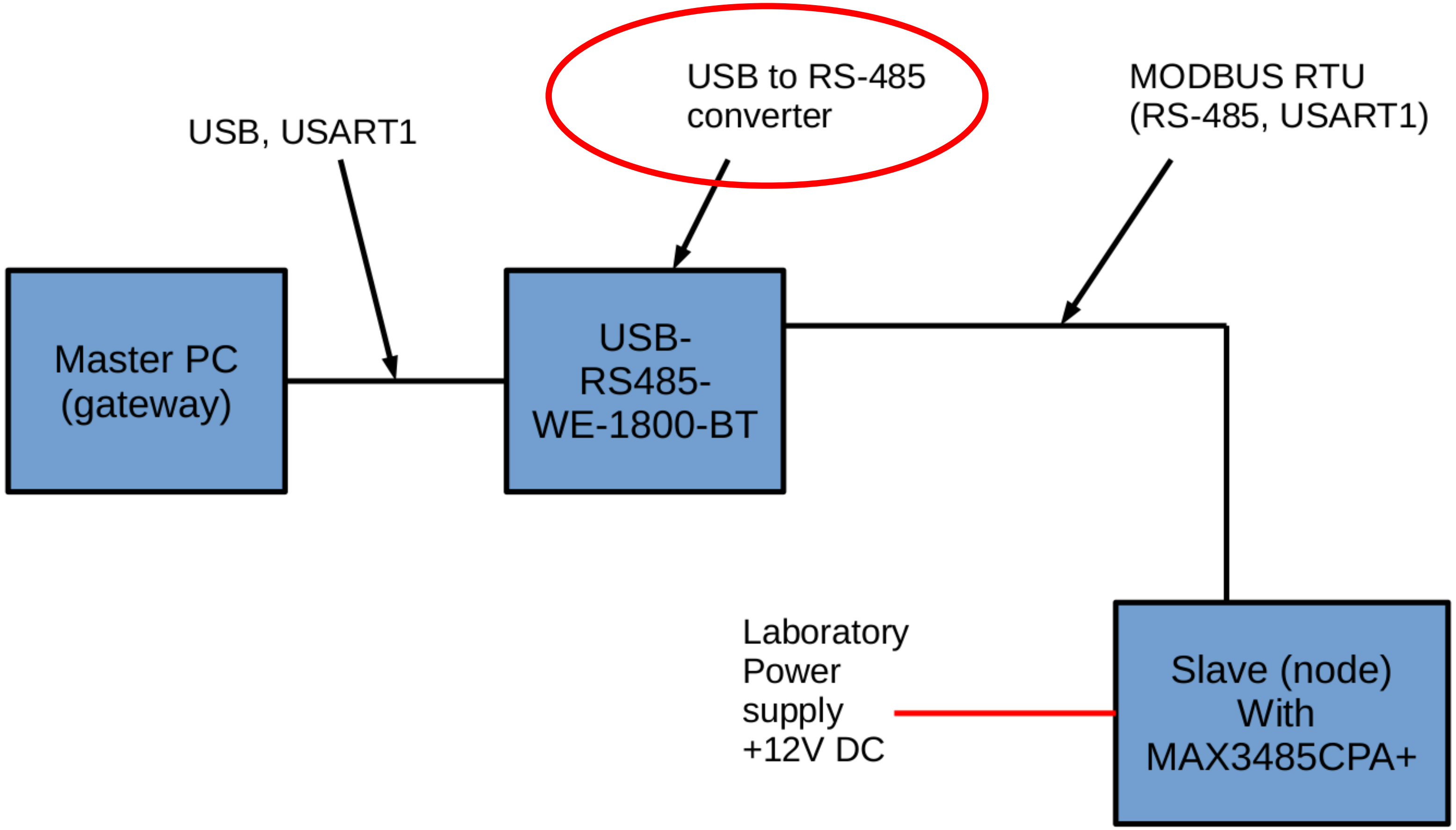
The program is made in pieces !!!!! Initially use USART2 with PC

Take a look at the flowchart and compare to this list!

- Step 1. Only LED ON if right address and off if wrong address. (USART2 interrupt)
- Step 2. USART2 interrupt flowcode implemented (full USARTx flowchart)
- Step 3. USART2 interrupt flowcode implemented and reads 7 bytes in main, if mFlag == 1 and delay 7 bytes (Disable receiver) if mFlag == 2. The delay is not made in the flowchart but is done in the program everything is rejected.
- Step 4. read_7_bytes_from_usartx implemented. If the address is correct then 7 incoming characters are read
- Step 5. crc check ok, crc check master frame
- Step 6. reads sensor, included in the program your own sensor reading
- Step 7. gives resonse to the master and considers that the program flowchart according to anyway.

Master frame ready and Slave frame ready. Now change the USART2 --> USART1 interface!

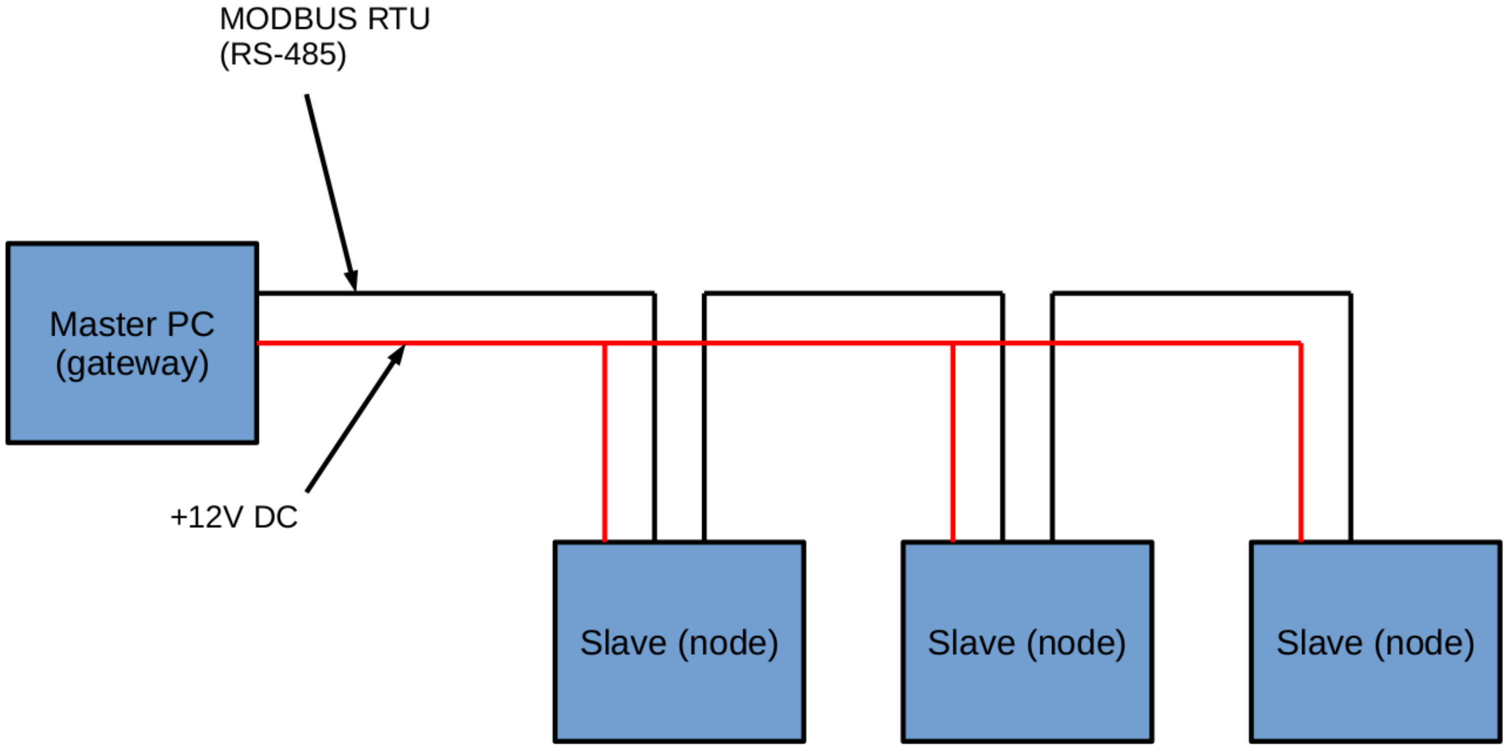
Change your code so that you can use following connection from figure. You need to use breadboard and 120 ohm resistor in the slave end. Remember to check USB_RS485-WE-1800-BT converter signal colors for connection and use laboratory DC-supply for Slave node "nucleo":


USB, USART1
USB to RS-485 converter
MODBUS RTU (RS-485, USART1)
Master PC (gateway)
USB-RS485-WE-1800-BT
Laboratory Power supply +12V DC
Slave (node) With MAX3485CPA+

Data to Wapice IOT Ticket with Python

Finally, if group has time! A python program will be created. Python queries the data from different slave devices and saves the data to the Wapice IoT ticket (cloud).

There is a separate laboratory exercise for this.


MODBUS RTU
(RS-485)
Master PC
(gateway)
+12V DC
Slave (node)
Slave (node)
Slave (node)