RFID READER
ISO 11784 / 11785

PROTOCOL FDX-B

P/N 20001003

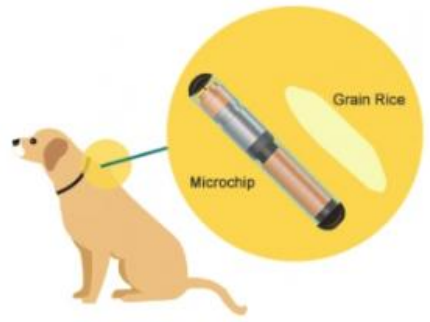
Table of Contents

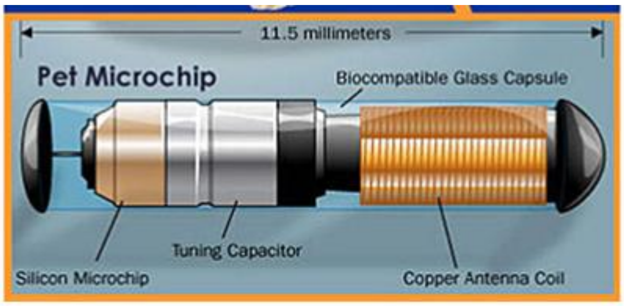
1. Project purpose

Project's goal is to develop a tool for a pet owner or a breeder to be able to check whether a known implanted chip is working or to be able to identify an animal if it lost from its owner. At the same time, basics of radio frequency identification and practical implementation of a communication protocol for hardware and software could be learned.

2. Operating principle

To make a "chip" or a "transponder" communicate with a reader (scanner) contactlessly, technology known as RFID (Radio Frequency Identification) is used. This permits contactless dialogue between the "chip" and the "scanner" using radio waves from the chip which receives power from the scanner. In the case of the identification of domestic animals, low frequencies are used.





The chips which conform to ISO11784 are called FDX-B (Full duplex), FDX-A and HDX (Half duplex). For reasons of size, only FDX-A and FDX-B chips can be injected into domestic animals. HDX chips are used mainly in animal husbandry, generally used in ear tags.

Some countries still have not imposed the FDX-B standard and continue to use the FDX-A type chips (North America, Australia, Taiwan ...). In Europe, all countries use chips which comply with the ISO FDX-B standard.

3. Telegram structure

Transmitted information consists of 128 bits.

msb										first lsb	
1	0	0	0	0	0	0	0	0	0	0	Header 11 bits
		1	1	1	1	1	0	0	0	0	
		1	0	0	0	1	0	0	1	1	
		1	0	0	0	1	0	0	0	0	
		1	0	0	0	1	0	0	0	0	
		1	1	1	0	0	0	0	0	0	38 bit national code
		1	1	1	1	1	1	0	0	1	10 bit country code
		1	-	-	-	-	-	-	-	1	1 bit data block status bit
		1	1	-	-	-	-	-	-	-	1 bit animal application indicator
		1	1	1	0	1	0	1	1	0	
		1	0	1	0	1	1	1	0	1	16 bit checksum
		1	0	1	0	1	0	1	1	0	
		1	0	0	1	1	0	1	0	0	
		1	0	0	0	1	0	0	1	0	24 bit extra data if present

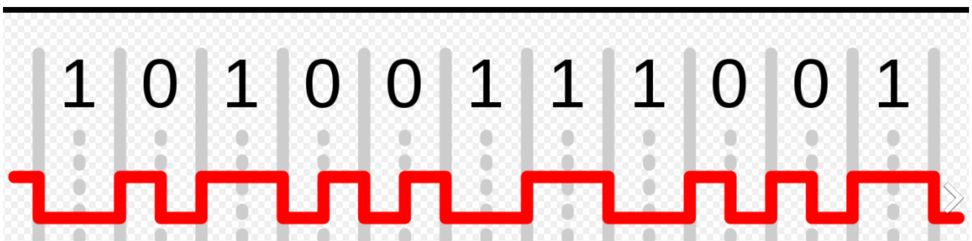
Taulukko 1:Telegram structure

4. Encoding

FDX-B protocol based transponders are defined to operate in the 134.2 kHz band, and employ a differential Manchester encoding, also known as bi-phase encoding scheme to transmit their information. The data bit rate is $f_c/32$, where f_c is the clock frequency.

Characteristics of bi-phase encoding FDX-B:

- Transition in the beginning of each bit
- logic “0”: Transition in the middle of bit period
- logic “1”: No transition during the bit period



When the tag (implanted microchip) enters the electromagnetic field transmitted by the RFID reader it draws power from the field and will commence transmitting its data by varying the electromagnetic field which can be detected by the frontend circuitry. An example is shown in the oscillogram Kuva 1: Oscillogram of ISO 11784/11785 telegram Biphase encoding, FDX-B, Kerry blue terrier “Lily”. The 11 bit header pattern is transmitted to indicate the beginning of the data block. Least significant bit (LSB) is sent first.

This is followed by 38 bits of the identification code. For an animal application this will be the identity code of the animal. This is a unique 12- digit decimal code for each animal.
After every 8 bits a logic 1 bit is sent (“SEP” in the example oscillogram) and it is inserted to differentiate data from the header sequence.

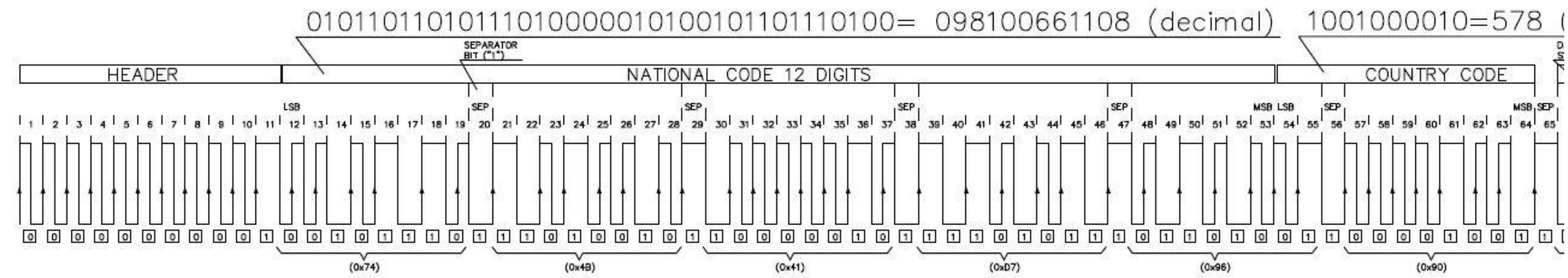
This is followed by the 10-bit country code. A country code is a 3- digit decimal value used to refer to country of origin of the animal or a code of individual manufacturers. Country codes refer to standard ISO 3166 e.g. the example below, 578 (in decimal) corresponds to Norway.

The 1-bit data block (bit 66) status is an indicator bit to indicate whether an additional data block exists. A value of 1 indicates that the transponder contains an additional 24 bit data block. Otherwise it is 0. Following this there are 14 reserved bits allocated for future use.

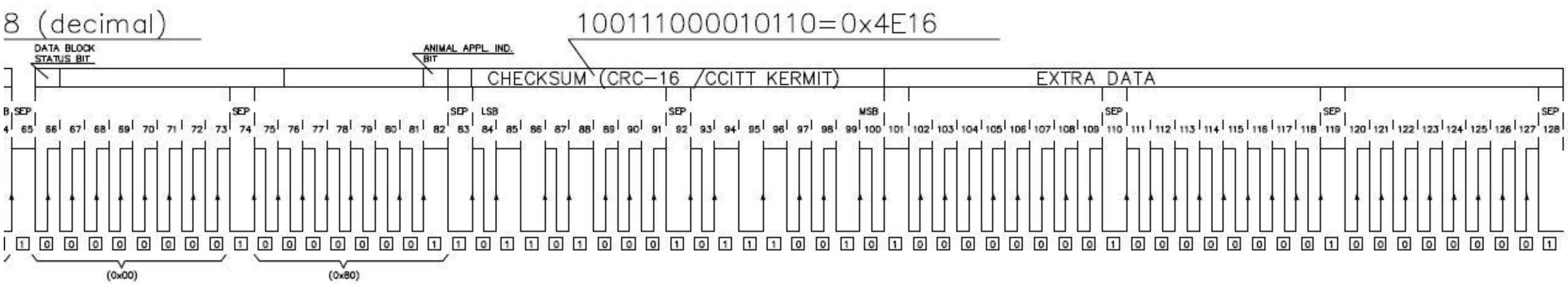
The animal application indicator is a single bit (bit 82) indicating that the transponder is used for animal identification. This value is set to 1 to indicate an animal identification application and 0 otherwise.

A 16 CRC-16/CCITT KERMIT checksum over the preceding 64 bit block (excluding control bits “SEP”) is included after the animal status bit.

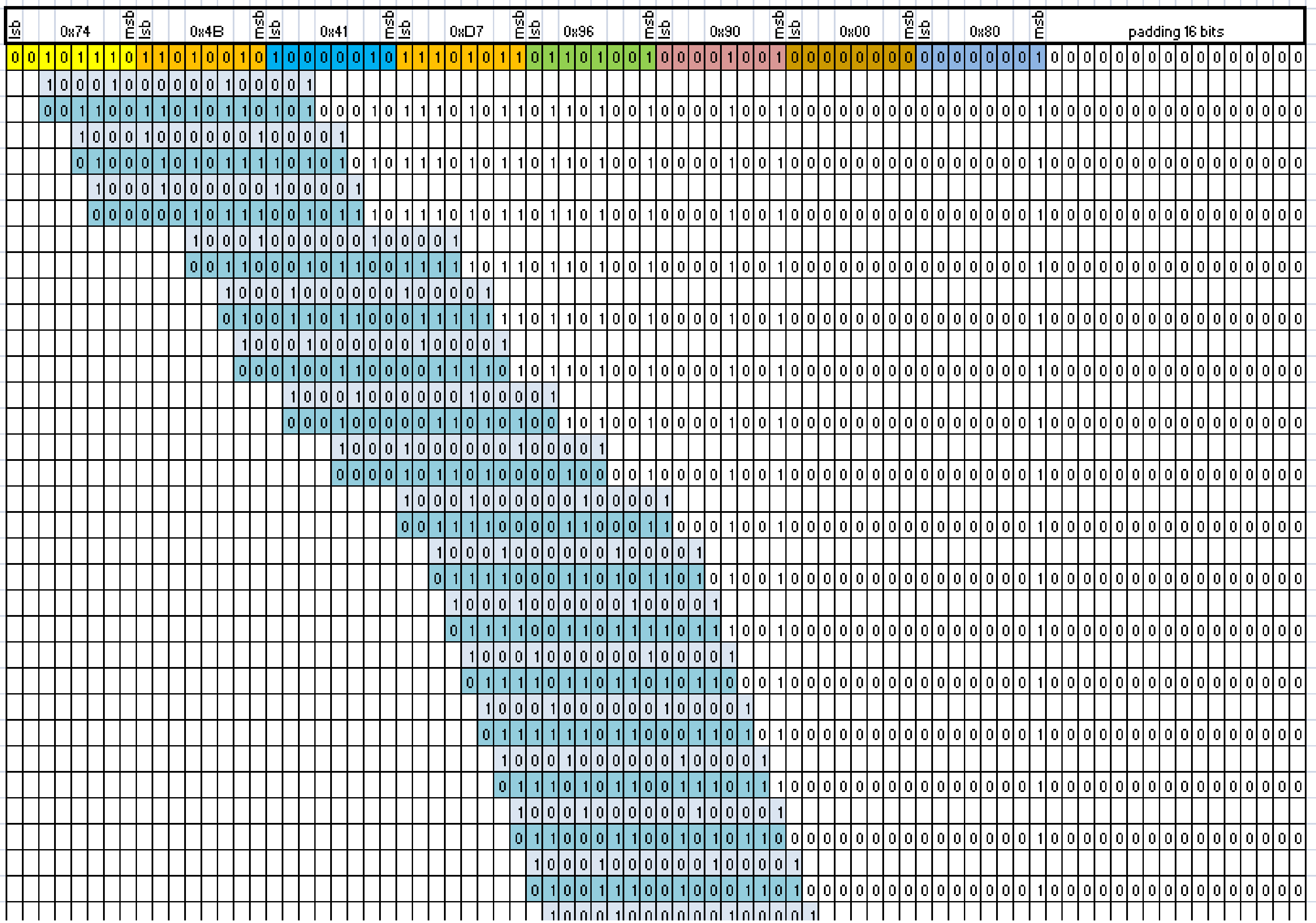
After the CRC checksum bits comes the extra data block. This data block exists if the data block status bit (bit 66) is 1. When the data block status bit is 0 this value will be 000000. The data block may be used to append additional data relevant to the individual application.



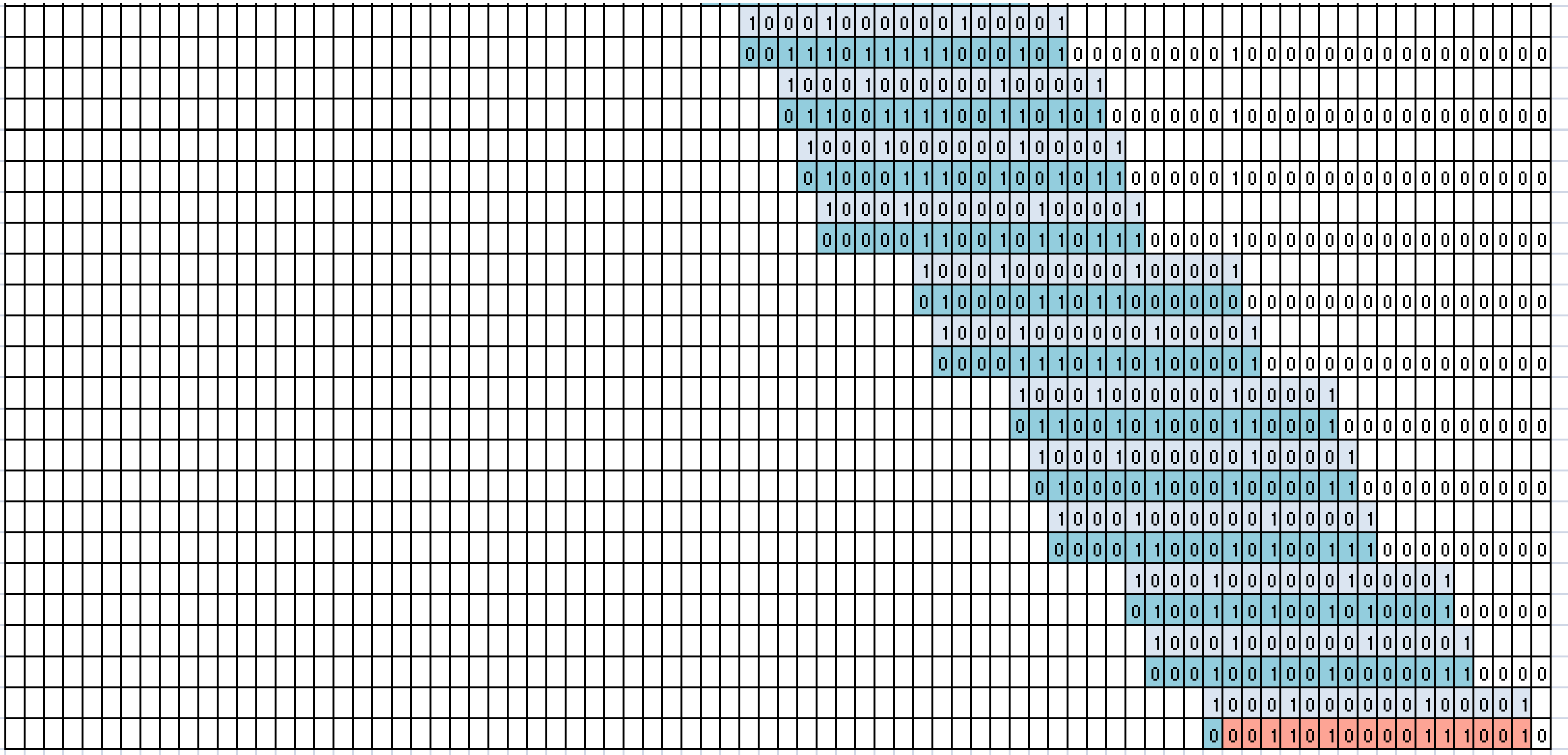
Kuva 1: Oscillogram of ISO 11784/11785 telegram Biphase encoding, FDX-B, Kerry blue terrier “Lily”



Kuva 2: ISO11784/11785 telegram continued



Kuva 3: CRC-16/CCITT KERMIT checksum calculation principle



Kuva 4: CRC-16/CCITT KERMIT checksum calculation, resulting CRC (in red)

5. CRC calculation

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption.

CRC-16 CCITT bit-by-bit calculation principle is presented in Kuva 3: CRC-16/CCITT KERMIT checksum calculation principle. Initial value for calculation are the 8 data bytes followed by 16- bit padding of 0's. Thereafter data bytes, padding included, are exclusive-OR'd (XOR'd) bitwise with polynomial $x^{16}+x^{12}+x^5+1$ (17 bits long, 10001000000100001) as shown in the example (Polynomial = grey, intermediate result = blue). Note polynomial shift to the next "1" and skipping the "0" 's in the intermediate result. When all bits have been handled, that is polynomial reaching the last bit on the right or when the next shift of the polynomial would go past the last bit, the result's 16 MSB's contain the resulting CRC with its MSB on the right.
Microchip reader can then compare the read CRC (bits 84-100, excl. bit 92) with the calculated result of the data bits (bits 12-83, excl. the SEP-bits). If they match, it is likely that all bits have been correctly transmitted and read.
Another method is to use a byte-wise calculation principle. This speeds up data processing by using a pre-calculated CRC lookup table for all 256 possible (00 to FF hexadecimal) byte values. This method requires, however, additional use of data memory.

Process for byte- wise CRC-16 using example's input data and polynomial 0x1021, using reciprocal lookup table

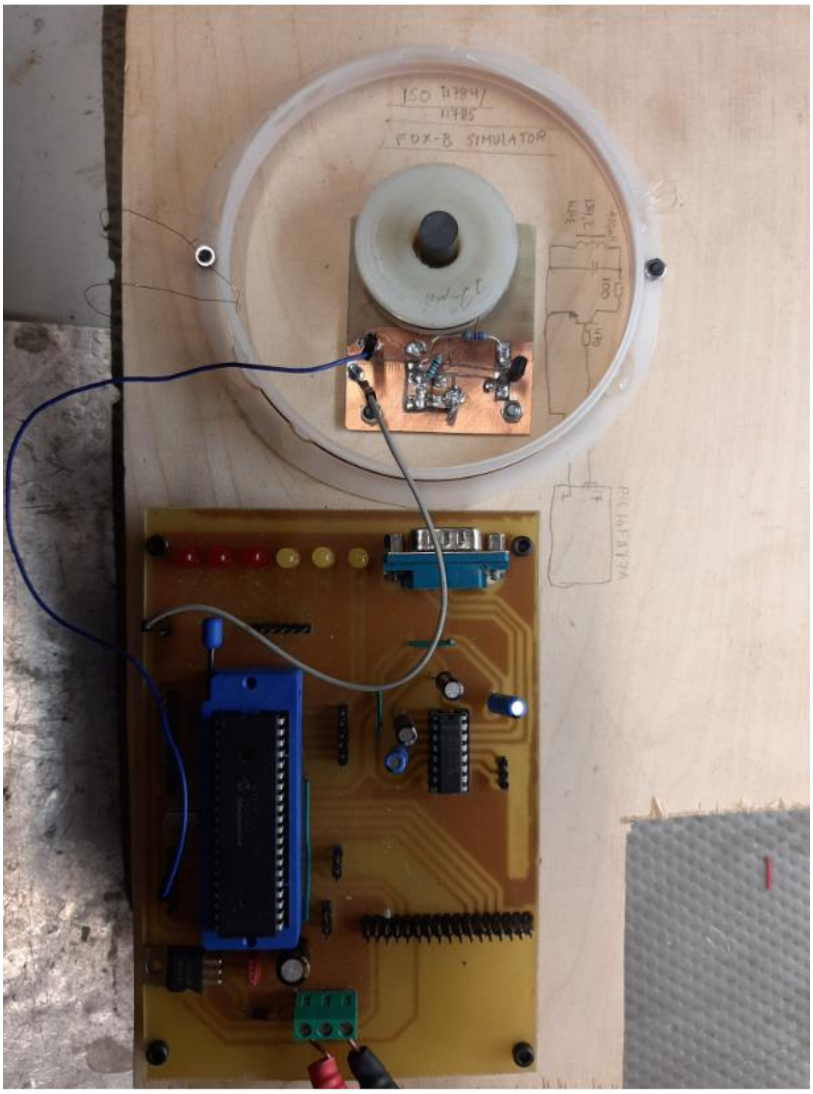
crc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
byte 1 (116)	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	
XOR->div=LSB	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	116
crc16[div]	0	0	1	1	0	1	0	1	1	0	1	0	0	0	1	1	
crc>>8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
intemediate crc	0	0	1	1	0	1	0	1	1	0	1	0	0	0	1	1	13731
byte 2 (75)	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	
XOR->div=LSB	0	0	1	1	0	1	0	1	1	1	1	0	1	0	0	0	232
crc16[div]	0	1	1	0	1	0	1	1	0	1	0	0	0	1	1	0	
crc>>8	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	
intemediate crc	0	1	1	0	1	0	1	1	0	1	1	1	0	0	1	1	27507
byte 3 (65)	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	
XOR->div=LSB	0	1	1	0	1	0	1	1	0	0	1	1	0	0	1	0	50
crc16[div]	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	1	
crc>>8	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	
intemediate crc	0	0	0	1	0	0	1	0	1	1	1	1	1	0	1	0	4858
byte 4(215)	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1	
XOR->div=LSB	0	0	0	1	0	0	1	0	0	0	1	0	1	1	0	1	45
crc16[div]	1	1	1	1	1	0	1	0	1	1	1	0	0	1	1	1	
crc>>8	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	
intemediate crc	1	1	1	1	1	0	1	0	1	1	1	1	0	1	0	1	64245
byte 5(150)	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	
XOR->div=LSB	1	1	1	1	1	0	1	0	0	1	1	0	0	0	1	1	99
crc16[div]	0	1	0	1	0	0	0	1	1	0	0	1	1	1	0	1	
crc>>8	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	
intemediate crc	0	1	0	1	0	0	0	1	0	1	1	0	0	1	1	1	20839
byte 6(144)	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	
XOR->div=LSB	0	1	0	1	0	0	0	1	1	1	1	1	0	1	1	1	247
crc16[div]	1	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	
crc>>8	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	
intemediate crc	1	0	0	0	0	0	1	1	0	1	1	0	0	0	0	1	33633
byte 7(0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
XOR->div=LSB	1	0	0	0	0	0	1	1	0	1	1	0	0	0	0	1	97
crc16[div]	0	1	1	1	0	0	1	0	1	0	0	0	1	1	1	1	
crc>>8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	
intemediate crc	0	1	1	1	0	0	1	0	0	0	0	0	1	1	0	0	29196
byte8(128)	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
XOR->div=LSB	0	1	1	1	0	0	1	0	1	0	0	0	1	1	0	0	140
crc16[div]	0	1	0	0	1	1	1	0	0	1	1	0	0	1	0	0	
crc>>8	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0	
intemediate crc	0	1	0	0	1	1	1	0	0	0	0	1	0	1	1	0	19990

Taulukko 2:Byte-wise CRC calculation with example data (input bytes reflected)

6. Hardware design

6.1. Development tools

A transponder simulator (encoder) was built to produce a reference data stream instead of an actual implanted tag. It is based on a Microchip PIC 16F877 microcontroller. Microcontroller produces a similar output as what was observed in the earlier example and which can then be read by the decoder to be developed. For initial purposes a hard-coded data content was implemented. Microcontroller drives an NPN- transistor to load down the “tag’s” tank circuit, providing the modulation. Encoder flowchart is presented in Kuva 11: Encoder simulator flowchart.

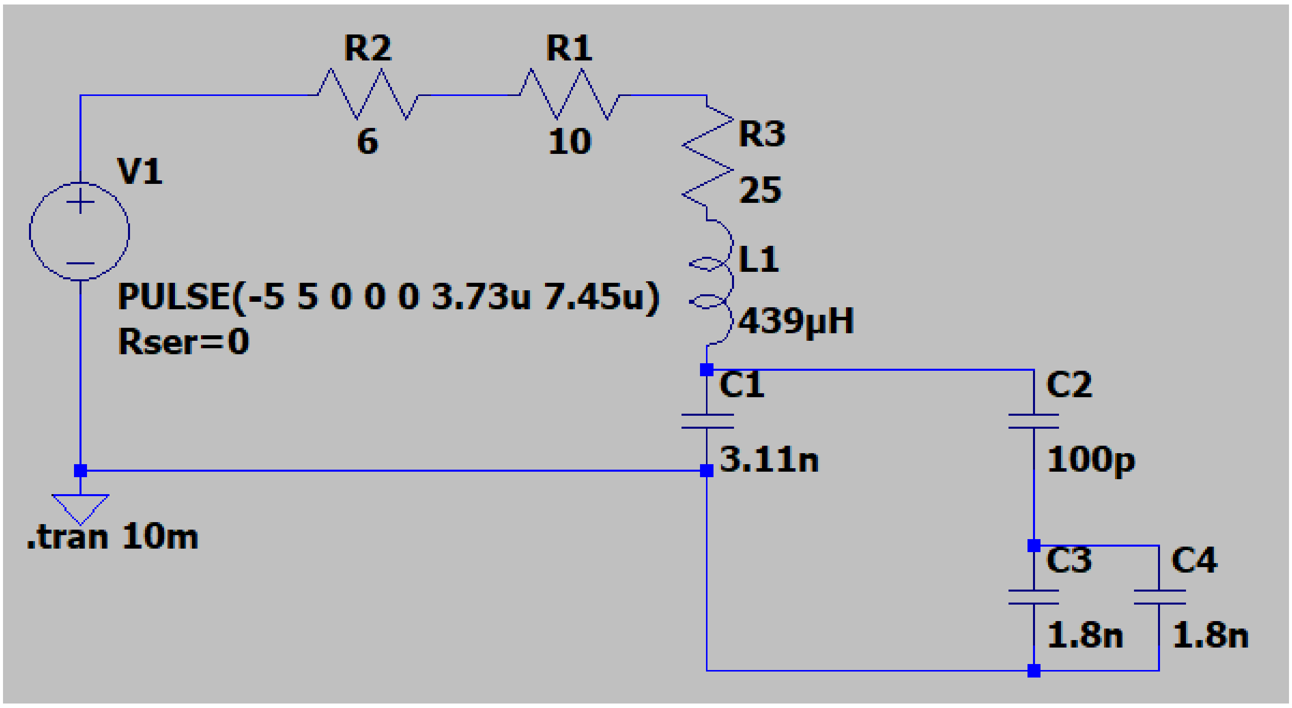


Kuva 5: Tag simulator implementation

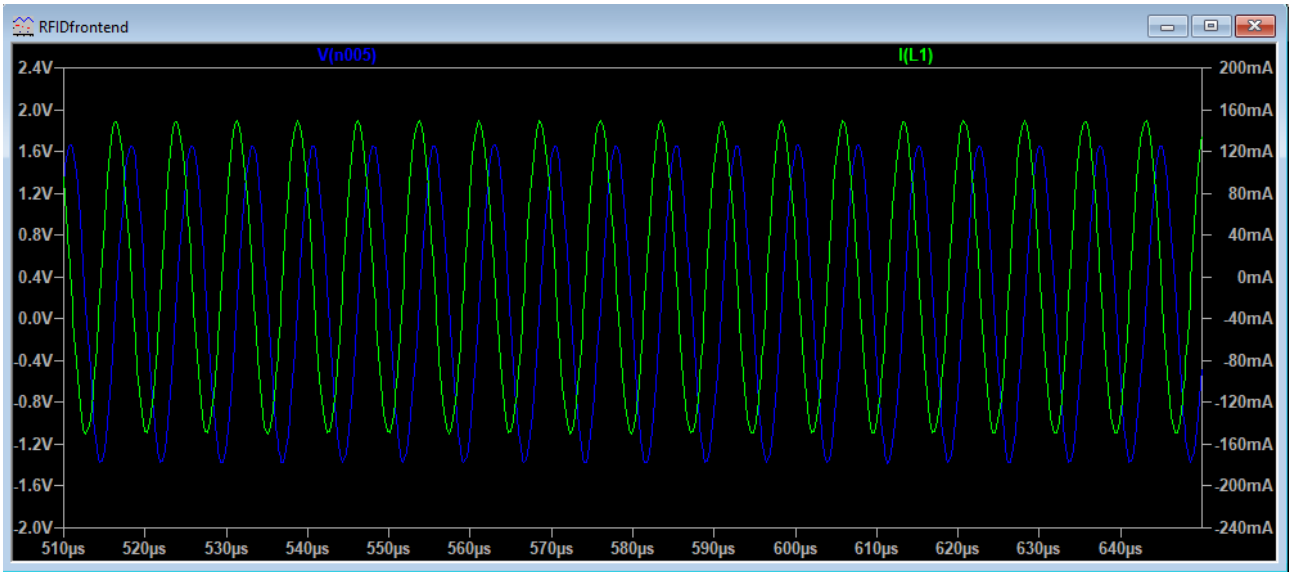
Larger coil around the “tag” circuitry is a “spy coil” enabling measuring the modulation of the RF-field

6.2. Reader frontend design

The electromagnetic field in the scanner is generated by an oscillator and a series-resonant inductor- capacitor (L-C) circuit. At resonance, inductive and capacitive reactance $X(L_A)$, $X(C_0)$ cancel each other out and impedance in the circuit consists only of resistance of the inductor coil and the driver IC's power stage.



Kuva 6: Frontend LTSpice simulation schematic



Kuva 7: Simulation of antenna current (green) and demodulation voltage (blue) at resonance frequency 134.2 kHz

The resonant frequency is

$$f_r = \frac{1}{2\pi\sqrt{L_A C_o}}$$

where

$$C_o = C_{RES} + \frac{C_{DV1} x \, C_{DV2}}{C_{DV1} + C_{DV2}}$$

C_{DV1} and C_{DV2} are the voltage divider capacitors (C2,C3,C4 in simulation providing 37 to 1 divider ratio). At resonance frequency, voltage magnitude over C1 in parallel with C2, C3 and C4, and similarly over L1 (two voltages which are 180 degrees out-of-phase) will be substantially higher than the supply voltage, in class of tens of volts to a few hundred volts, the actual magnitude depending on the resistance of the circuit. With the simulated component values it is about 120 V_{p-p}. Antenna coil peak current is

$$I_{ANTpeak} = \frac{4}{\pi}\left(\frac{V_{DD} - V_{SS}}{R_{ANT} + 2R_{AD} + R_{SER}}\right)$$

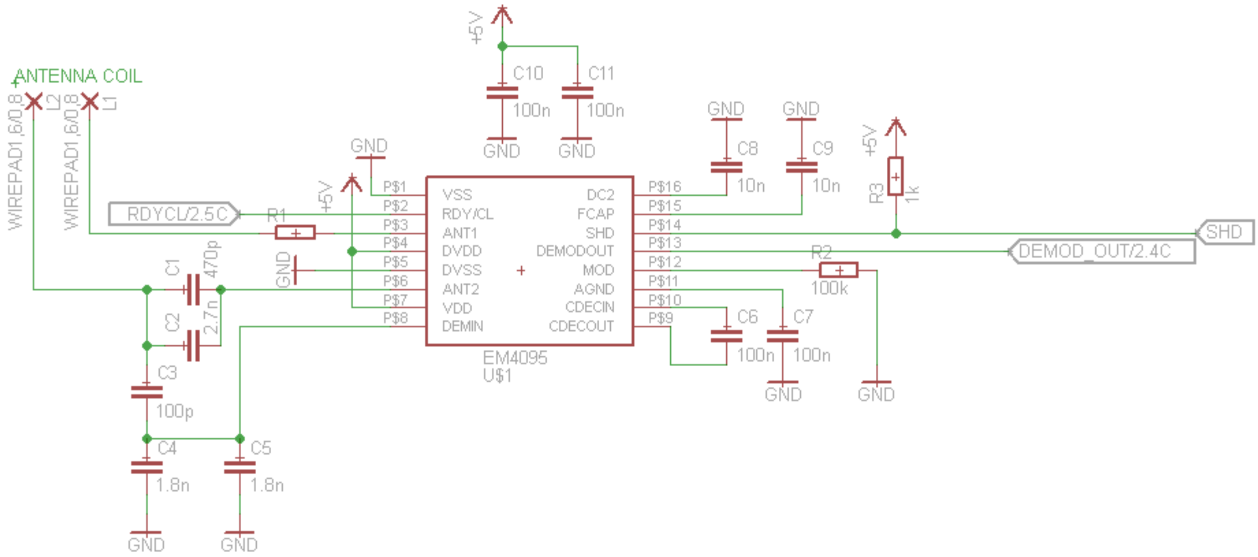
where
V_{DD} = Supply voltage (5V)
V_{SS}= Ground reference voltage (0V)
R_{ANT} = Antenna coil (AC) resistance (R3 in simulation)
R_{AD} = EM4095 driver internal resistance (R2 in simulation)
R_{SER} = Series resistor (R1 in simulation)

Inductance of an air-cored inductor can be calculated using following empiric formula

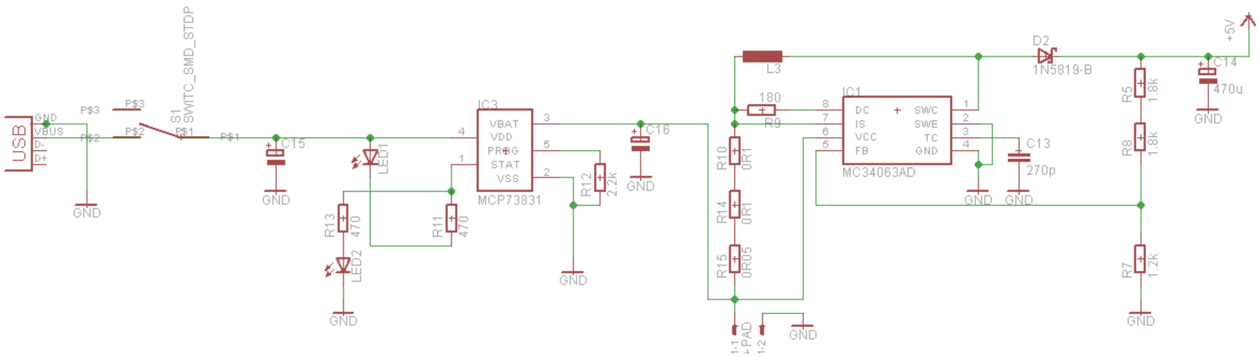
$$\frac{L}{[\mu H]} = \frac{(N\frac{R}{[in]})^2}{9\frac{R}{[in]} + 10\frac{H}{[in]}}$$

where
R= Radius of coil in inches
N= number of turns
H= height of coil in inches

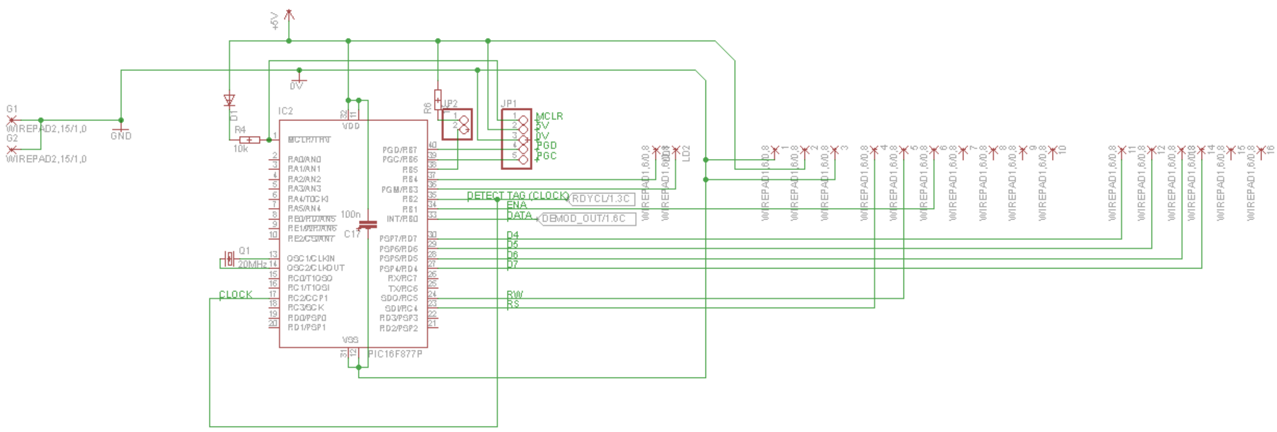
Coil inductance 439µH corresponds to approximately 50 turns of 0.05 mm wire at 0.05 spacing coiled with 45 mm radius. R_{SER} + R_{ANT} shall be 35Ω. In practice coil inductance needs to be slightly less due to stray capacitances, approximately 400 µH.



Kuva 8: Frontend using EM4095 integrated circuit



Kuva 9: Power supply. Li-Ion battery charge & boost converter

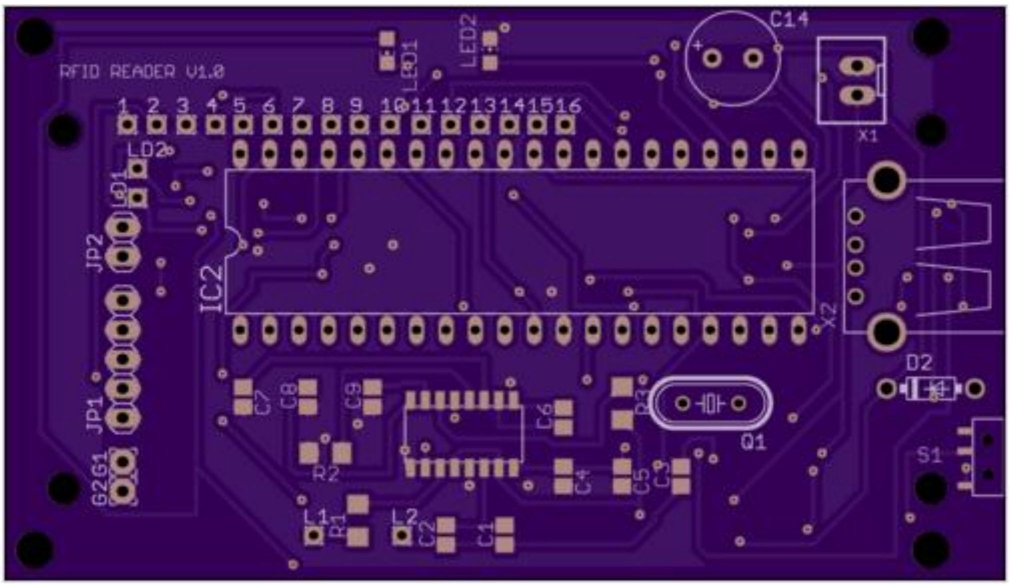


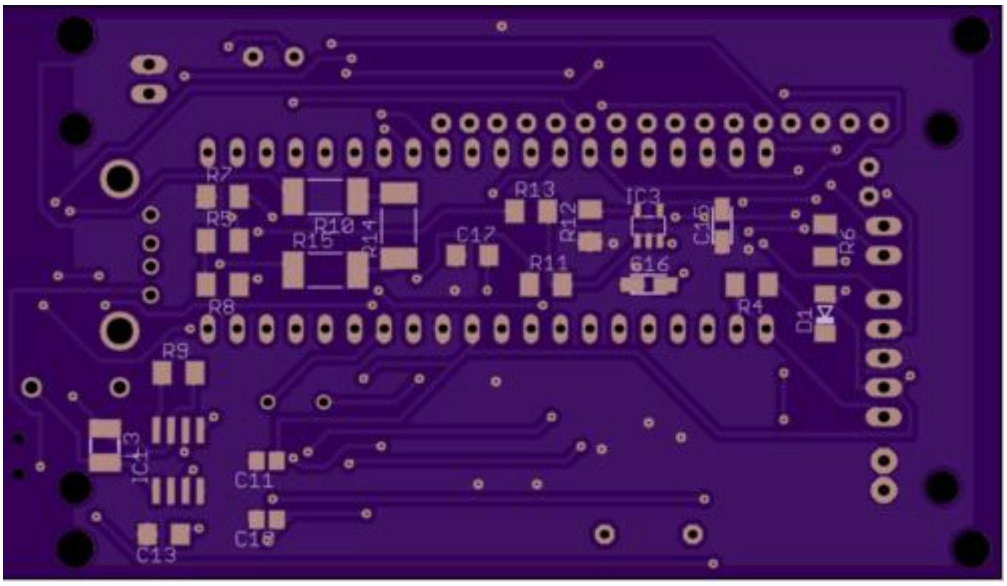
Kuva 10: Data processing, PIC18F452

EM4095 design criteria

- $I_{ANTPeak}$ <250 mA
- $U_{p\ DEMIN}$ max $5V_p$
- $R_{AD} = 3\ \Omega$ (typical)
- MOD pin shall be pulled down to GND for read-only application
- SHD pulled down to GND will enable oscillator
- RDY/CL is output for clock signal
- DEMOD OUT is output for data stream

6.3. PCB design

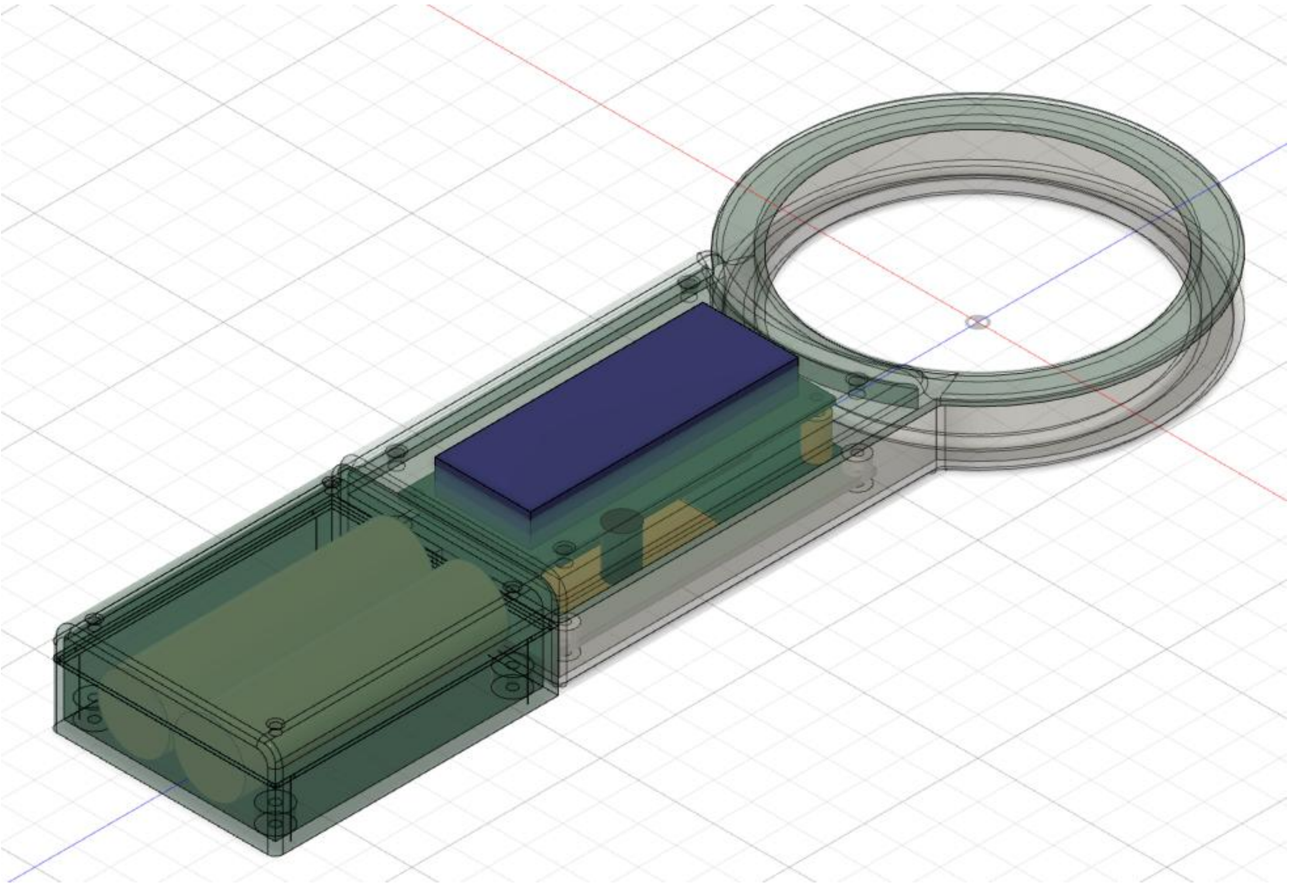


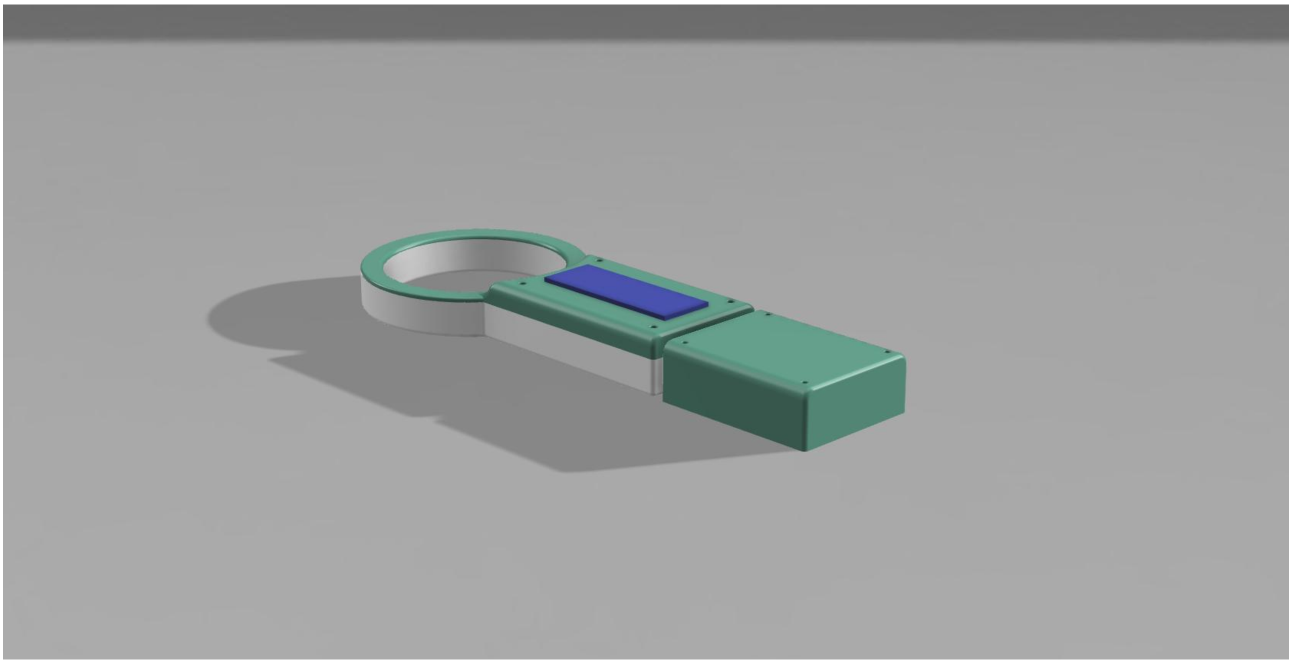


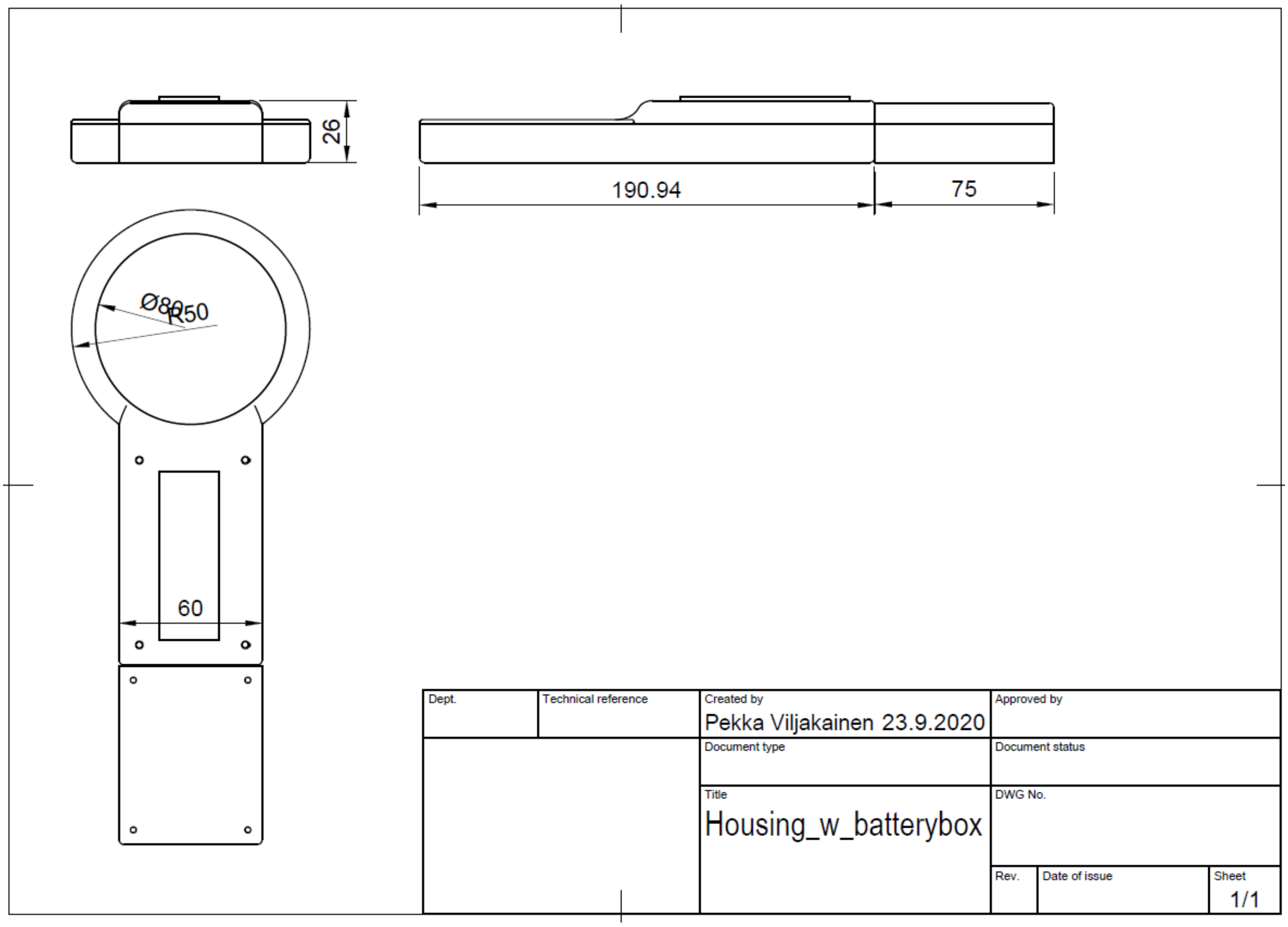
6.4. Display

A 2x16- character back-lit LCD Hitachi 44780- compatible display was used .

6.5. Enclosure





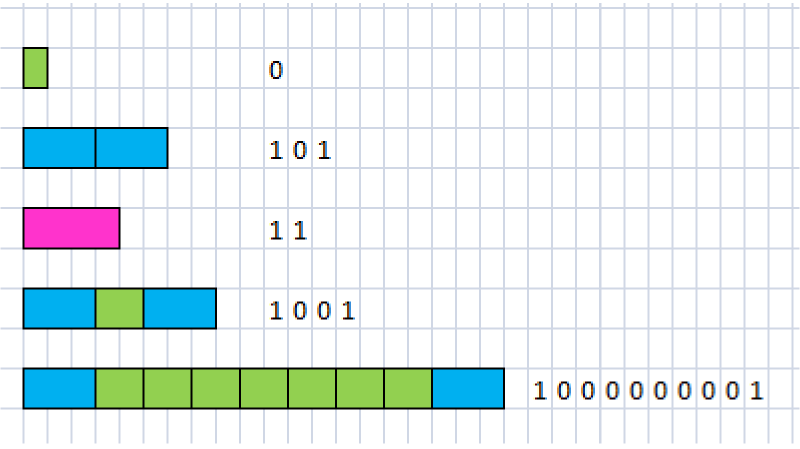


The enclosure was modeled in Autodesk Fusion360 and was made by using a 3D- printer with a 200 x200 mm bed size. Enclosure has four parts: Reader bottom, reader top, battery box bottom and battery box top.

7. Reader firmware

Decoding principle

- Calculate CRC lookup table in memory
- Put state machine in state "Wait for transponder"
- Read clock pulses (using interrupt routine triggered every 16^{th} pulse edge) and determine the average clock period and calculate corresponding data pulse durations t_{short}, $t_{intermediate}$ and t_{long}, check that they are within 134.2 kHz nominal +/- tolerance
- If number of trials is less than 10 put state machine in state "Read data"
- Read sufficient amount of data, 245 intervals (245=2x128+11) in memory using rising edge interrupt and store time intervals between the rising edges in an array
- Disable interrupts
- Put state machine in state "Decode"
- Detect header position in data stream
- Decode bits:
- Until 128 bits decoded
    - Read time from array in memory
    - Select case
        - if time is t_{short}, bit is "0"; save in variable; break
        - if time is t_{long}, the bits are "1 1" ; save in variable; break
        - If time is $t_{intermediate}$, and flag1 is not set, bit is "1", save in variable, set flag1 and read next time; break
        - If flag1 is set and time is $t_{intermediate}$, then bits are " 0 1", save in variable ,reset flag1;break
    - loop
- Separate 8 data bytes
- Calculate CRC-16/CCITT KERMIT using a lookup table over the 8 data bytes
- If CRC's match, put state machine in state "Display", else increase number of trials and put state machine in state ""Read data"
- Display 3-digit country code and 12-digit identification code in decimal format
- Wait for readout time
- Put state machine in state "Wait for transponder"



Taulukko 3: Decoding principle. Green=short; Blue=intermediate; Red=long time

Data display

Double-Dabble Binary-to-BCD Conversion Algorithm

National identification code consists of 12 decimal digits which require 38 bits. As the longest variables the selected microcontroller can handle are 32-bits, the conventional conversion method from binary to decimal using addition of powers of 2 is not easily applicable. Another way, applied here is to convert long binary integers into decimals by using the following method, in which the original binary number is first converted to BCD, a format in which each decimal digit is represented by four bits, which then can then be easily converted into corresponding decimals.

Double-dabble binary-to-BCD conversion a.k.a ”shift and add 3” is as follows (example: consider an 8-bit binary 11111111 or FF_{16}): This principle can be extended to arbitrarily long binary numbers.

1. Shift the binary number left one bit.
2. If 8 shifts have taken place, the result BCD number is in the Hundreds, Tens, and Units column.
3. If the binary value in any of the BCD columns is 5 or greater, add 3 to that value in that BCD column.
4. Go to 1.

Steps to convert an 8-bit binary number to BCD

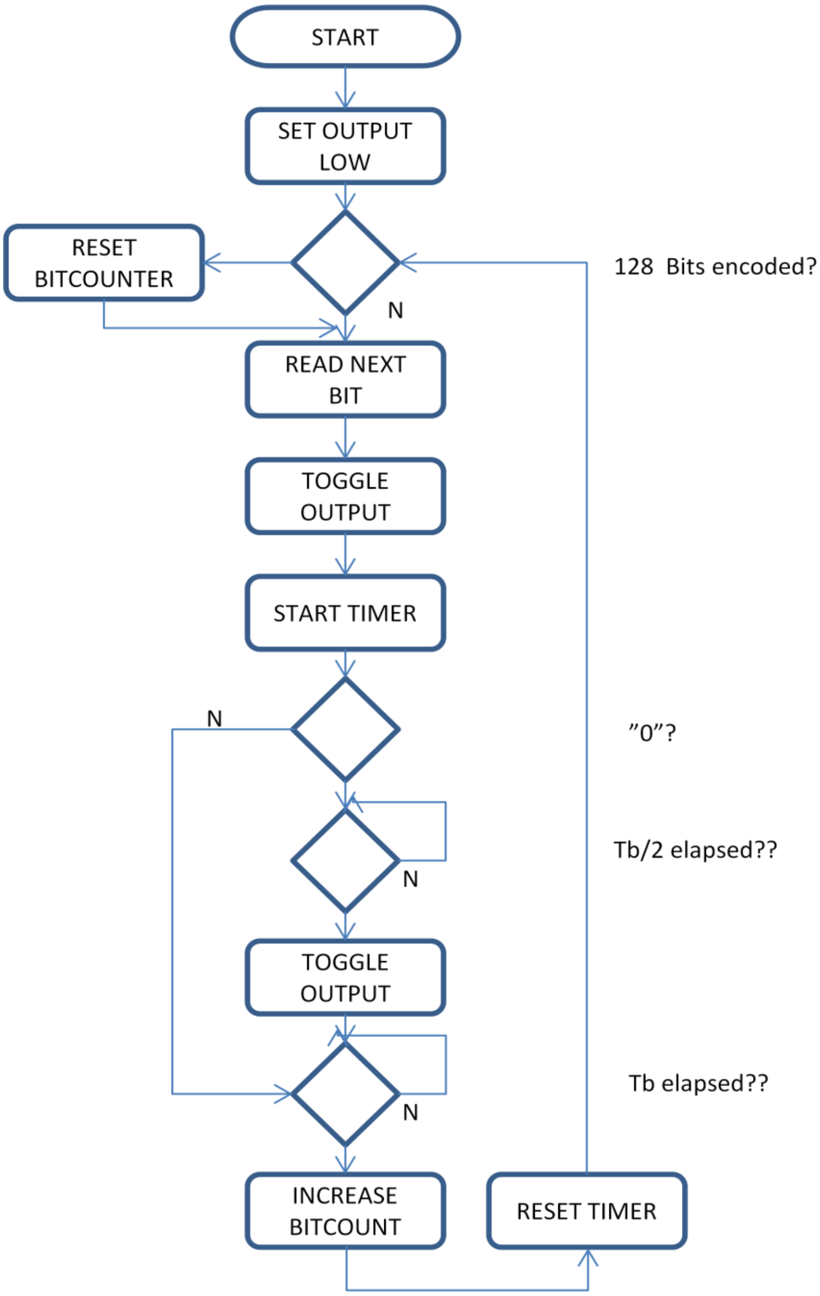
Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1
Shift 1			1	1 1 1 1	1 1 1
Shift 2			1 1	1 1 1 1	1 1
Shift 3			1 1 1	1 1 1 1	1
Add 3			1 0 1 0	1 1 1 1	1
Shift 4		1	0 1 0 1	1 1 1 1	
Add 3		1	1 0 0 0	1 1 1 1	
Shift 5		1 1	0 0 0 1	1 1 1	
Shift 6		1 1 0	0 0 1 1	1 1	
Add 3		1 0 0 1	0 0 1 1	1 1	
Shift 7	1	0 0 1 0	0 1 1 1	1	
Add 3	1	0 0 1 0	1 0 1 0	1	
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		

Taulukko 4: Binary to BCD conversion, double-dabble method

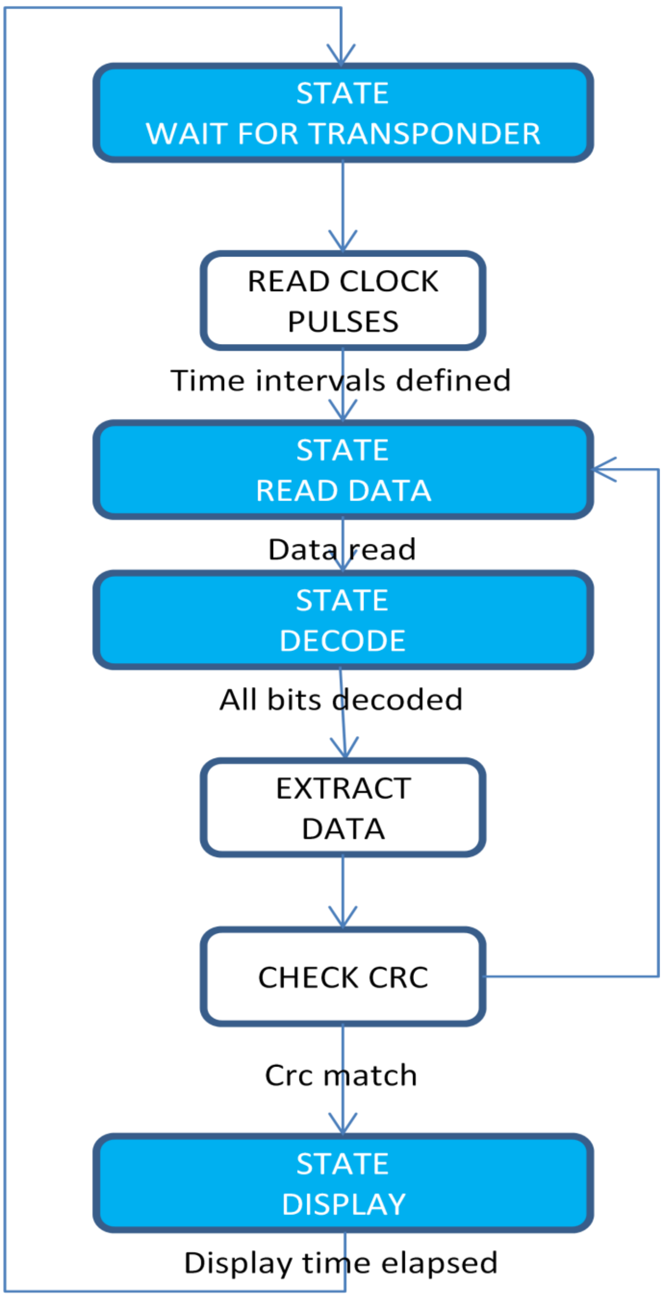
8. Terminology

BCD	Binary Coded Decimal, decimal digits represented by four bits
CRC	Cyclic Redundancy Calculation
ISO 11784:1996	Standard for Radio frequency identification of animals — Code structure
ISO 11785:1996	Standard for Radio frequency identification of animals — Technical concept
ISO 3166	Standard for Country Codes
LSB	Least Significant Bit
MSB	Most significant bit
PCB	Printed Circuit Board
RFID	Radio Frequency Identification

9. Appendix



Kuva 11: Encoder simulator flowchart



Kuva 12: Reader state machine

```
void CRC_table(void)
```

```
{
   Generator=0x1021;
    for (dividend=0;dividend<256;dividend++)
    {
        curByte=(unsigned int)(dividend);
        for (crcbit=0;crcbit<8;crcbit++)
        {
            if((curByte&0x0001)!=0)
            {
                curByte>>=1;
                curByte^=Reflect16(Generator);
            }
            else
                curByte>>=1;
        }
        crctable16[dividend]=(unsigned int) curByte;
    }
}
 unsigned int Reflect16(unsigned int val)
{
    unsigned int resVal = 0;

    for (int i = 0; i < 16; i++)
    {
        if ((val & (1 << i)) != 0)
        {
            resVal |= (unsigned int)(1 << (15 - i));
        }
    }

    return resVal;
}
```

Taulukko 5: Code snippet to generate CRC-16 KERMIT lookup table

Lookup Table:

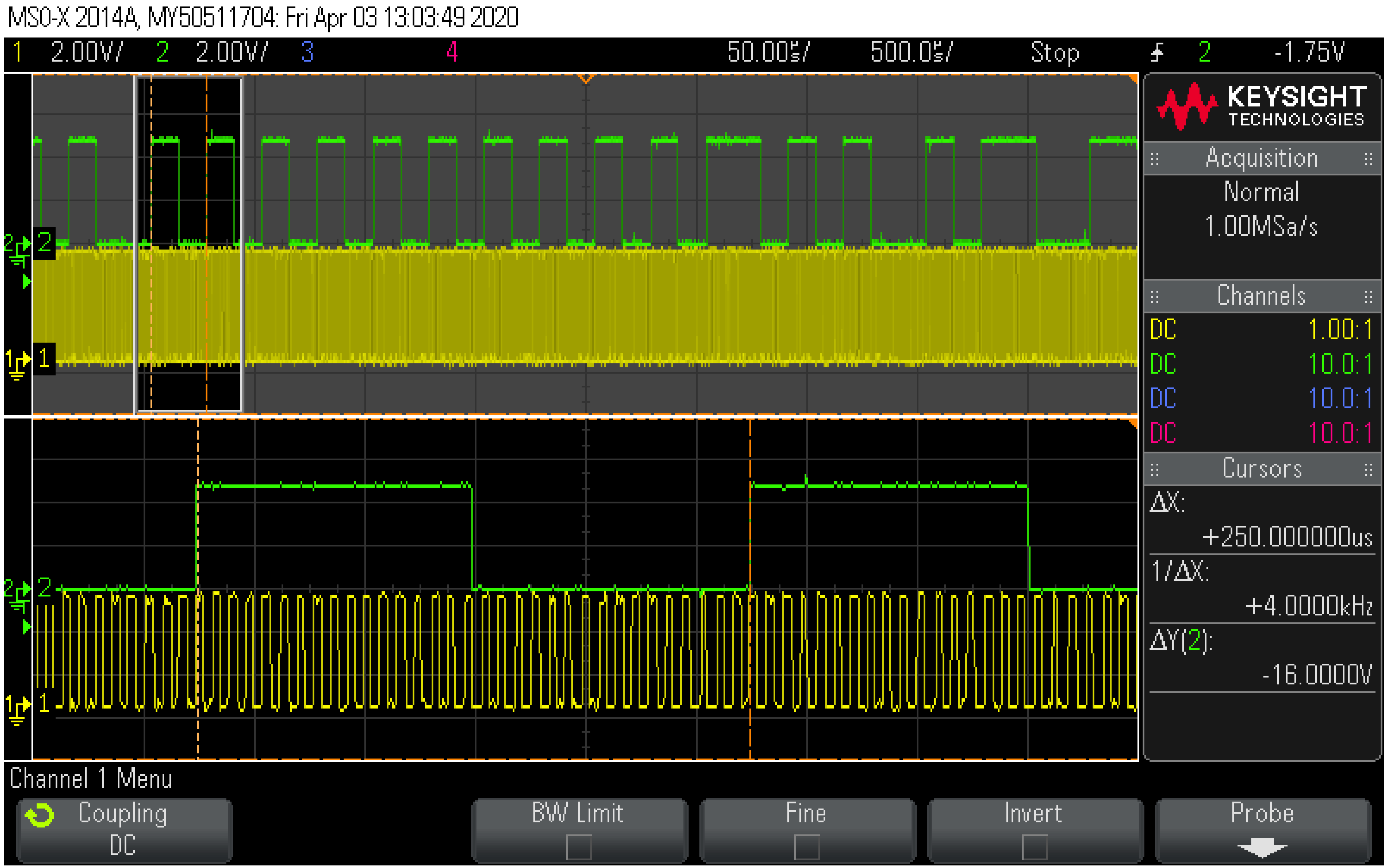
0x0000	0x1189	0x2312	0x329B	0x4624	0x57AD	0x6536	0x74BF	0x8C48	0x9DC1	0xAF5A	0xBED3	0xCA6C	0xDBE5	0xE97E	0xF8F7
0x1081	0x0108	0x3393	0x221A	0x56A5	0x472C	0x75B7	0x643E	0x9CC9	0x8D40	0xBFDB	0xAE52	0xDAED	0xCB64	0xF9FF	0xE876
0x2102	0x308B	0x0210	0x1399	0x6726	0x76AF	0x4434	0x55BD	0xAD4A	0xBCC3	0x8E58	0x9FD1	0xEB6E	0xFAE7	0xC87C	0xD9F5
0x3183	0x200A	0x1291	0x0318	0x77A7	0x662E	0x54B5	0x453C	0xBDCB	0xAC42	0x9ED9	0x8F50	0xFBEF	0xEA66	0xD8FD	0xC974
0x4204	0x538D	0x6116	0x709F	0x0420	0x15A9	0x2732	0x36BB	0xCE4C	0xDFC5	0xED5E	0xFCD7	0x8868	0x99E1	0xAB7A	0xBAF3
0x5285	0x430C	0x7197	0x601E	0x14A1	0x0528	0x37B3	0x263A	0xDECD	0xCF44	0xFDDF	0xEC56	0x98E9	0x8960	0xBBFB	0xAA72
0x6306	0x728F	0x4014	0x519D	0x2522	0x34AB	0x0630	0x17B9	0xEF4E	0xFEC7	0xCC5C	0xDDD5	0xA96A	0xB8E3	0x8A78	0x9BF1
0x7387	0x620E	0x5095	0x411C	0x35A3	0x242A	0x16B1	0x0738	0xFFCF	0xEE46	0xDCDD	0xCD54	0xB9EB	0xA862	0x9AF9	0x8B70
0x8408	0x9581	0xA71A	0xB693	0xC22C	0xD3A5	0xE13E	0xF0B7	0x0840	0x19C9	0x2B52	0x3ADB	0x4E64	0x5FED	0x6D76	0x7CFF
0x9489	0x8500	0xB79B	0xA612	0xD2AD	0xC324	0xF1BF	0xE036	0x18C1	0x0948	0x3BD3	0x2A5A	0x5EE5	0x4F6C	0x7DF7	0x6C7E
0xA50A	0xB483	0x8618	0x9791	0xE32E	0xF2A7	0xC03C	0xD1B5	0x2942	0x38CB	0x0A50	0x1BD9	0x6F66	0x7EEF	0x4C74	0x5DFD
0xB58B	0xA402	0x9699	0x8710	0xF3AF	0xE226	0xD0BD	0xC134	0x39C3	0x284A	0x1AD1	0x0B58	0x7FE7	0x6E6E	0x5CF5	0x4D7C
0xC60C	0xD785	0xE51E	0xF497	0x8028	0x91A1	0xA33A	0xB2B3	0x4A44	0x5BCD	0x6956	0x78DF	0x0C60	0x1DE9	0x2F72	0x3EFB
0xD68D	0xC704	0xF59F	0xE416	0x90A9	0x8120	0xB3BB	0xA232	0x5AC5	0x4B4C	0x79D7	0x685E	0x1CE1	0x0D68	0x3FF3	0x2E7A
0xE70E	0xF687	0xC41C	0xD595	0xA12A	0xB0A3	0x8238	0x93B1	0x6B46	0x7ACF	0x4854	0x59DD	0x2D62	0x3CEB	0x0E70	0x1FF9
0xF78F	0xE606	0xD49D	0xC514	0xB1AB	0xA022	0x92B9	0x8330	0x7BC7	0x6A4E	0x58D5	0x495C	0x3DE3	0x2C6A	0x1EF1	0x0F78

Taulukko 6: CRC-16 KERMIT lookup table in hexadecimal. Index of table is the byte value (0x00 to 0xFF)

```
unsigned char pos;
unsigned int crc;
unsigned char Data[8];


unsigned int Compute_CRC16()
{
    rs232_string("CRC intermediates;");
    crc=0;
    for (char b=0;b<8;b++)
    {
           crc=(unsigned int)(crc^Data[b]);
           pos=(char)(crc & 0xFF);
           crc=(unsigned int)(crc>>8);
           crc=(unsigned int)(crc^(unsigned int)(crctable16[pos]));
    }
    return crc;
}
```

Taulukko 7: Code snippet to calculate byte-wise CRC



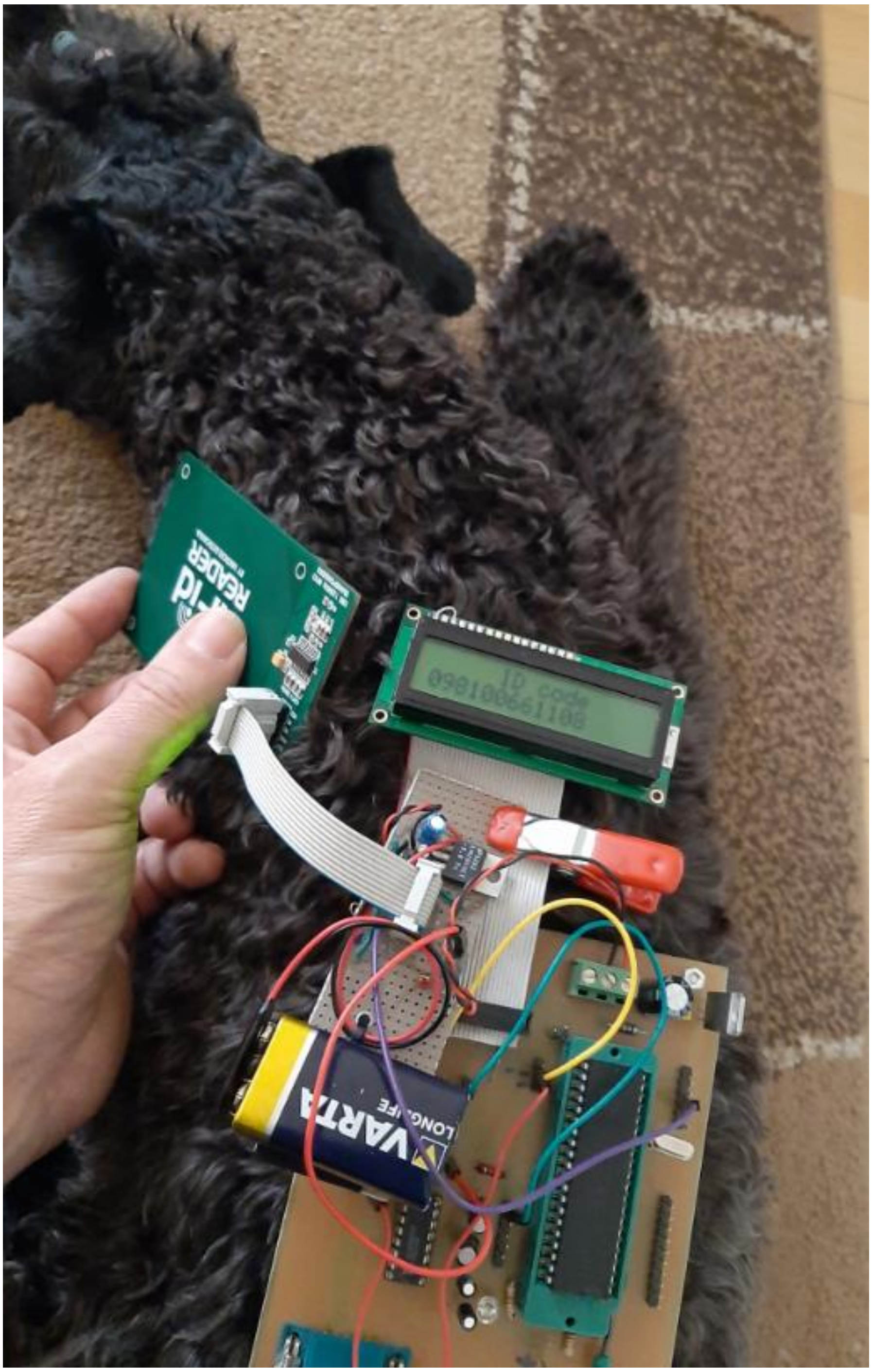
Kuva 13: RFID simulator output, data and clock (green/yellow) at the start of the header sequence

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1	0	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	1	0
24	25	26	27	28	29	30	31	32	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1	0	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1	0	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	1	0

38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73
1	1	1	1	0	1	0	1	1	1	0	1	1	0	1	0	0	1	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0
29	30	31	32	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	0	1	0	1	1	1	0	1	1	0	1	0	0	1	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	1	0	1	1	1	0	1	1	0	1	0	0	1	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0

74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	0	0	0	0	0	0	0	1	1	0	1	1	0	1	0	0	0	1	0	1	1	1	0	0	1	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0	0	0	0	0	0	0	1	1	0	1	1	0	1	0	0	0	1	0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	0	1	1	0	1	1	0	1	0	0	0	1	0	1	1	1	0	0	1	0

101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
28	29	30	31	32	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1



Kuva 14: Prototype software testing

Component	Manuf.	Type	Technical properties	Source	Pcs
PCB				Osh Park	1
USB connector	Cvilux	CU01-SAH1S00	USB-A	Partco	1
Switch SPDT		KYT LIU4 SMD		Partco	1
Inductor			SMD 47µH 1210	Partco	1
Q1			Crystal 20 MHz HC49/S		
Capacitor, tantal		TANSMD-C 47U 16V	47 uF, 16V	Partco	1
Capacitor, electrolytic		470µF 16V 8x12mm	470uF, 16V	Partco	1
Capacitor, ceramic		SMD 1206 270pF		Partco	1
Capacitor, ceramic		CHIPC 0805 2N7	2.7nF, 0805		
Capacitor, ceramic			470pF, 0805		
Capacitor, ceramic			82pF, 0805		
Capacitor, ceramic			1,5nF, 0805		
Capacitor, ceramic			100nF, 0805		
Capacitor, ceramic			10nF, 0805		
Diode		Schottky 1N5819		Partco	1
Diode		0,2A 100V 4ns SOD80		Partco	
LED, smd			PINTALIITOSLED 0805 VIHREÄ 12mcd		1
LED, smd			PINTALIITOSLED 0805 PUNAINEN 12mcd		1
IC		MC34063	1.5A PEAK BOOST/BUCK/INVERTING SWITCHING REG	Partco	
IC	Microchip	MCP73831T-2ATI/OT	Battery Charger for 1 Cell of Li-Ion	Partco/Farnell	
R10, R14			0.1 ohm	Partco	
R15			0.05 ohm	Partco	
R11			470 ohm, 1206		1
R13			470 ohm, 1206		1
R9			180 ohm, 1206		1
R5			1,8 kohm, 1206		1
R8			1,8 kohm, 1206		1
R7			1,2 kohm, 1206		1
R4			10 kohm, 1206		1
R6			1 kohm, 1206		1
R1			?? Ohm, 1206		1
R2			100 kohm, 1206		1
R3			1 kohm, 1206		1
Connector		JST XH	2-POLE	Partco	1
LCD display			16 CHAR 2 ROWS		

Taulukko 8: Bill of materials