Void the warranty!

How to start analyzing blackboxes

Hunz
jabber: hunz@jabber.berlin.ccc.de

Focus of this talk: HW(-SW) blackboxes

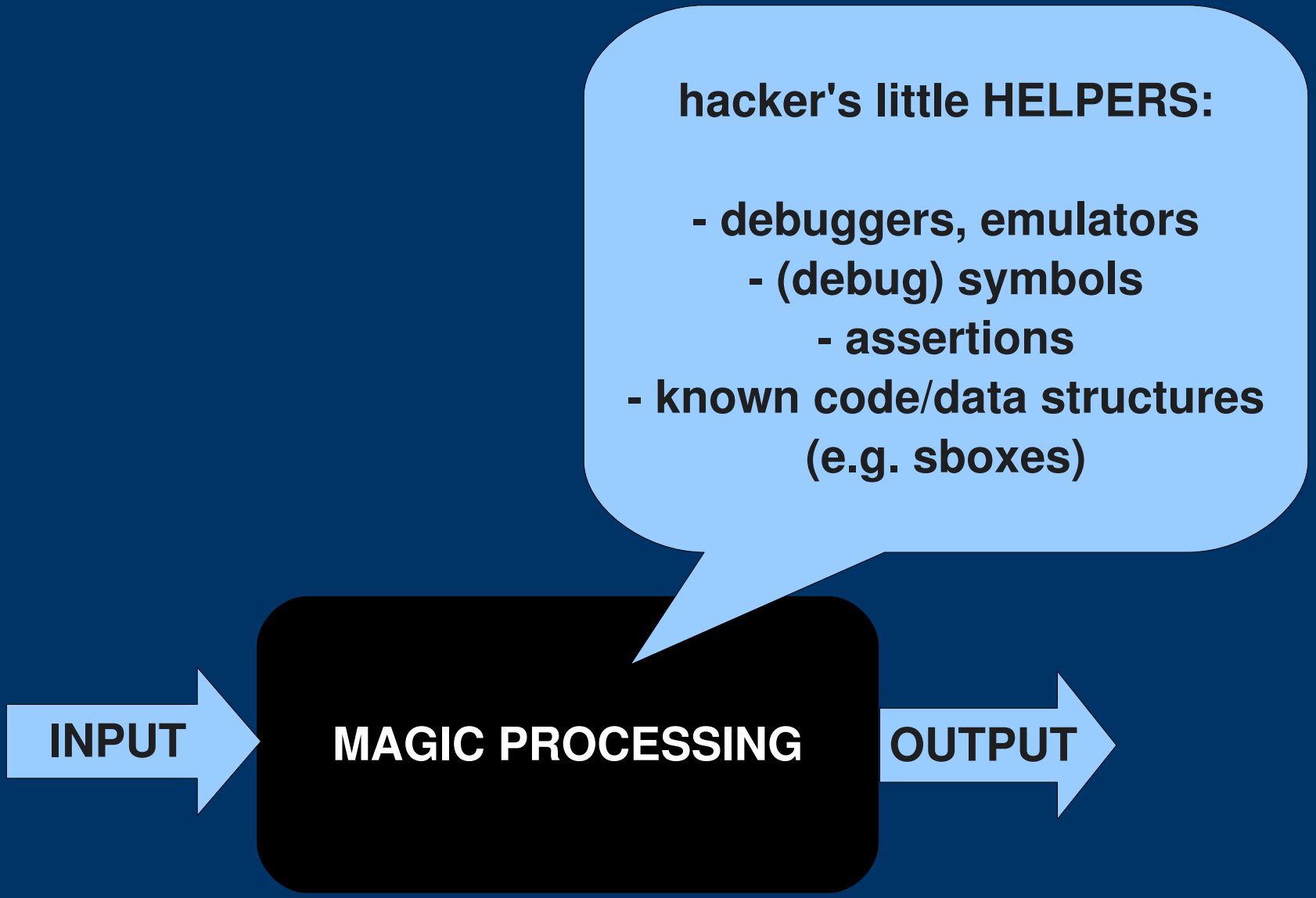
- more precise:
  - non-desktop hardware, with or without software
  - no analog stuff – only digital components
- Why?
  - understanding the functionality
  - extending/abusing it
- Examples:
  - HW/SW: STBs, PDAs, µcontrollers, smartcards, digicams
  - HW: RF(ID), IR

motivation

- HW-only systems are less complex, but wide spreaded yet
- kind of precursor to ubiquitous computing
- people trust those blackboxes
  -> they have an impact on people's everyday life
- getting access to SW requires access to some hardware interface

- thus, hacking those systems can
  <evil> be serious fun </evil> :-)
  <nice> lead to a more critical view of blackbox-systems
  </nice> ;-)

blueprint of a SW-blackbox



SW-blackboxes without those helpers...

- **assumption: one has access to the firmware**
- **question 1: What do you want to do with the system?**
  (understanding everything works only for simple systems)

- 2: what HW interfaces are involved in the process of interest?
- 3: find your way along this path of interest from in- to output

- that's the theory... the practice is often hard and awkward work

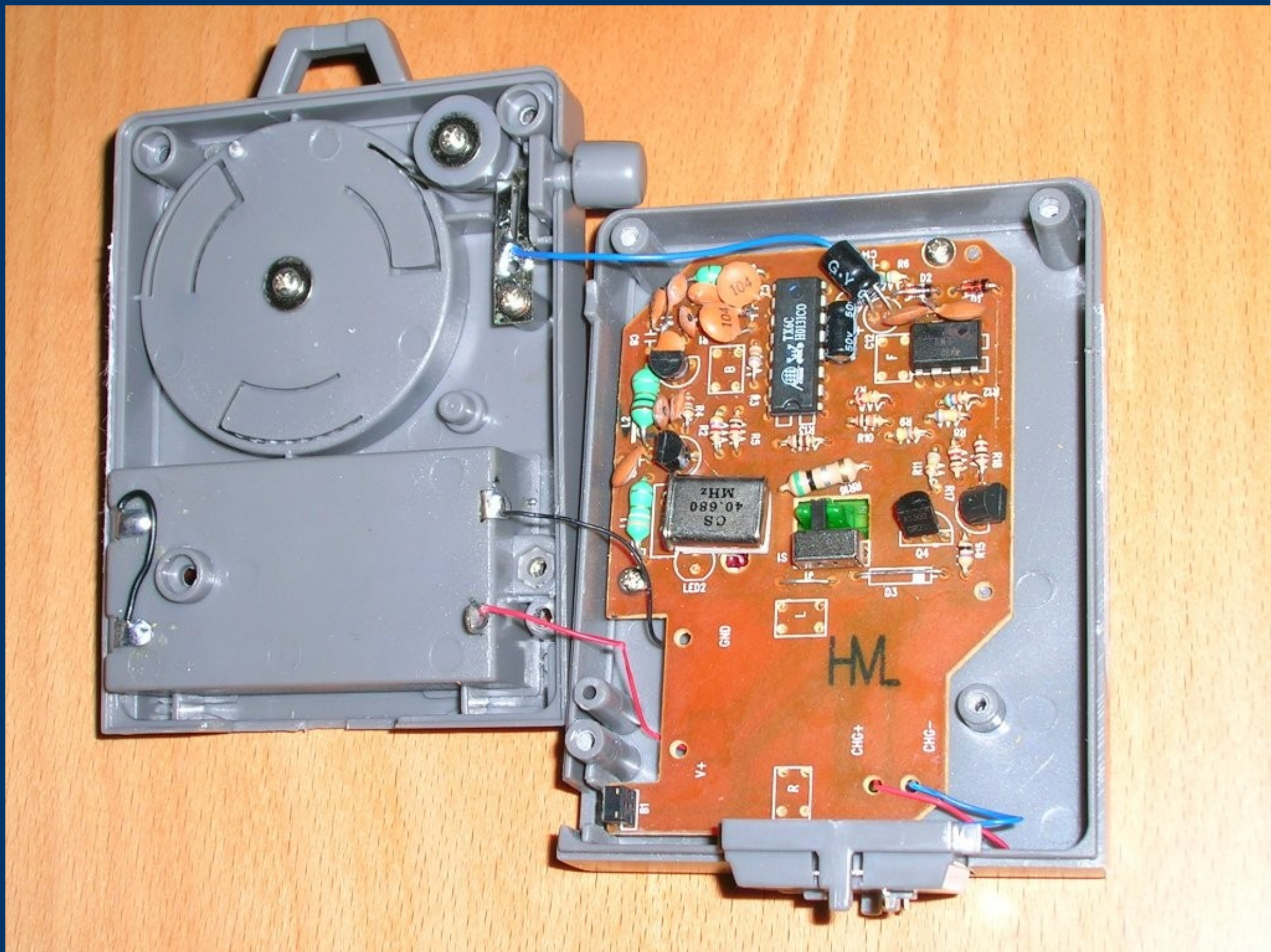
some firmware reversal hints

- identify the architecture
- problems:
  - memory remapping
  - uninitialized jumptables
  - different CPU modes (ARM32/Thumb)
- try to get a memdump of the running firmware!
  - by patching the firmware for example
  - turn off ints before!
    or your dump will be inconsistent!

blueprint of a blackbox as in hardware

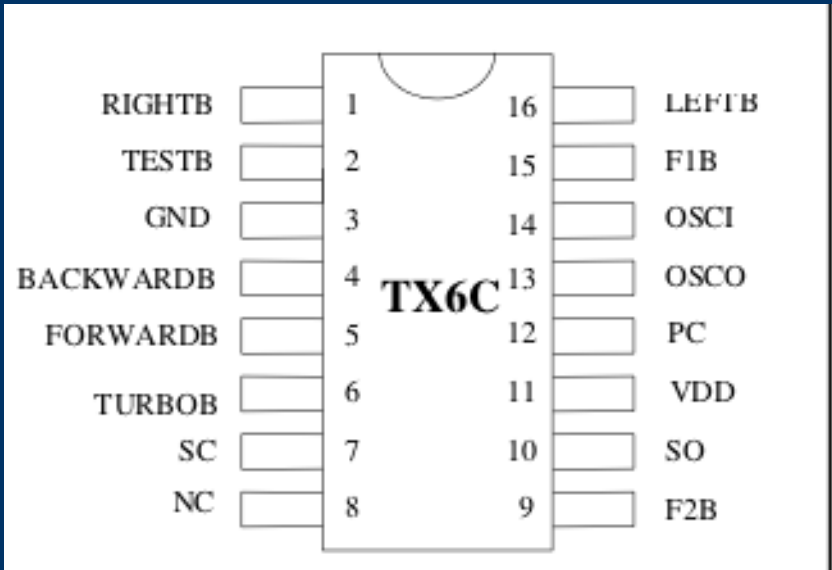
- input can be controlled to a certain degree
- output can be observed
- that's it – guess the rest...

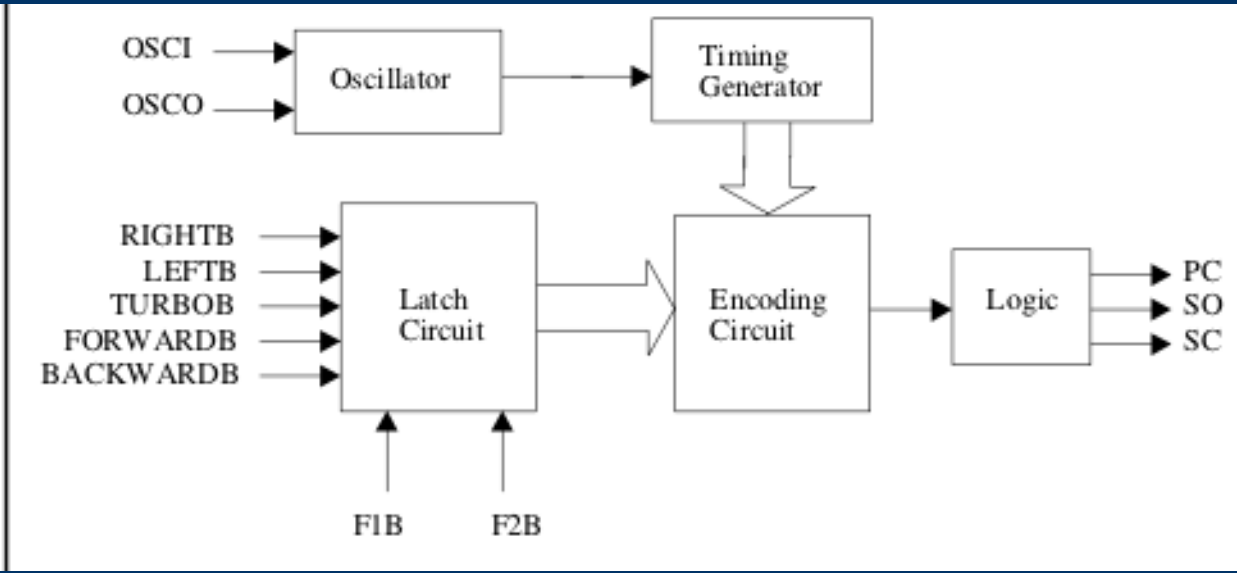

40 MHz
GO
CHARGE
45 SEC
ON
OFF
TURBO BLASTER
CHARGING BASE
R/C MICROSIZERS PRO
HOBBICO

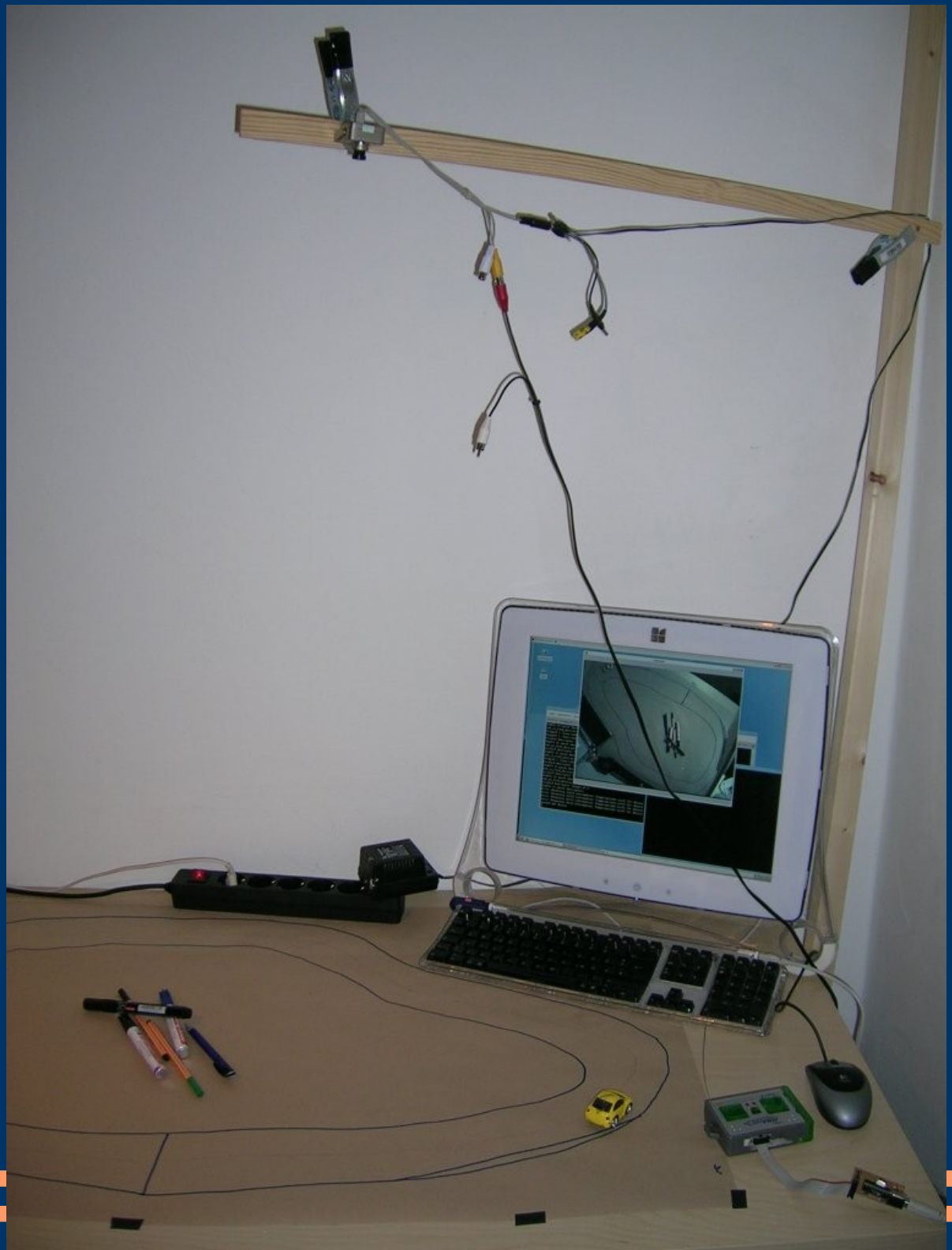

104
G.Y
CS
40.680
MHz
LED2
GND
V+
CHG+
CHG−

simple example: miniature RC-car

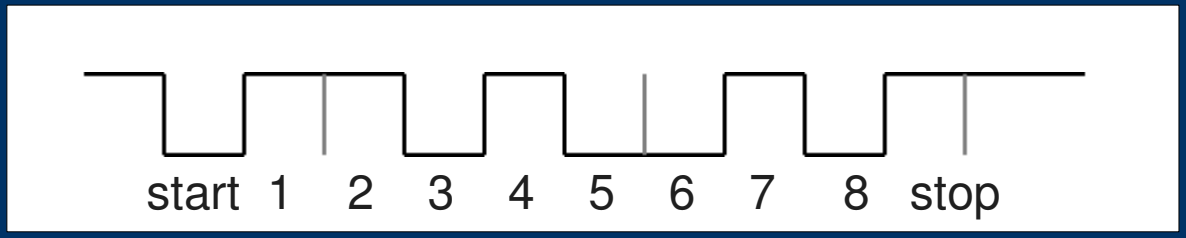
- there's only one interesting IC: TX6C...
- datasheet can be found via google
- there's one pin for every direction (fwd/back/left/right)
- 0 on that pin and it'll drive!
- direct connection to parport possible
- easy to connect to a PC via USB as well







serial transmissions(I): RS232

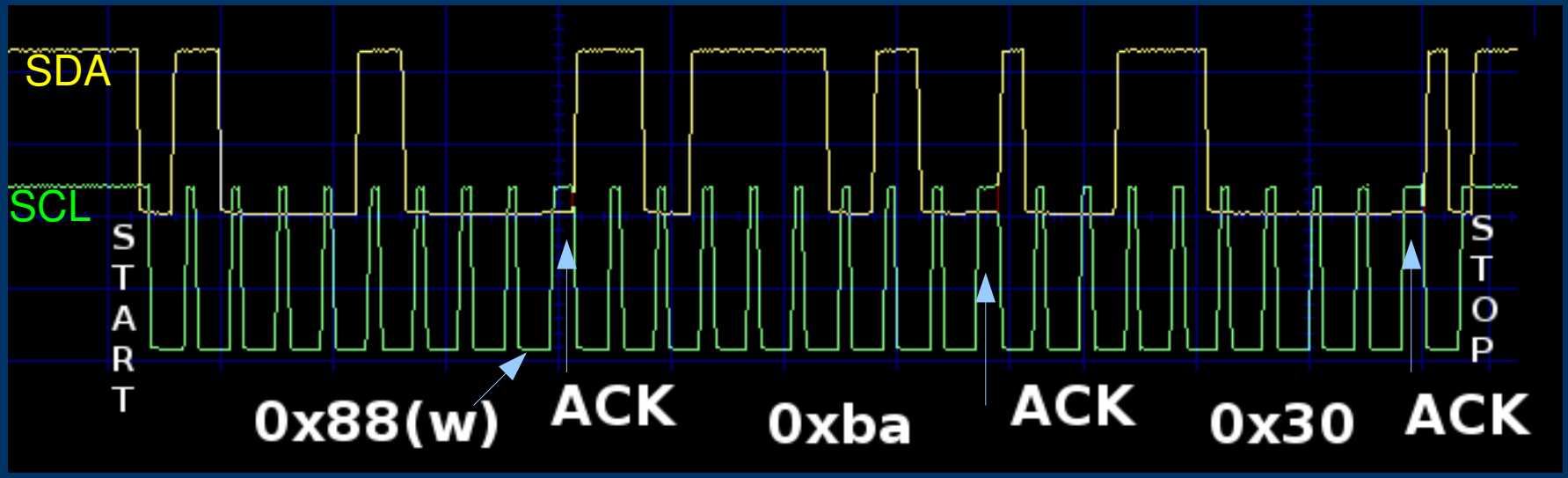


- low speed (<=250kbaud usually), no clock (->async)
- start-, stopbit, (parity) -> bitrate = baudrate – 2 - #(parity bits)!!
- used for communication between more complex controllers that run some kind of software
- **often used for debug stuff!!**
- RX: easy to spot if device sends by itself (2.4/4.8/9.6/etc. kHz)
- TX: bruteforce... - baudrate should be equal to RX
- easy to interface using a PC – don't forget the level shifters!
- usb2serial devices available – quality at high baudrate varies!!
  - can be found in cheap cellphone-usb cables!
  - no levelshifter necessary then!

serial transmissions(II): I²C

- low to medium speed (<=400kHz), clock line -> synchronous
- two lines (SCL: clock, SDA: data) with pullups
- master-slave(s) – master initiates transfer
- used for communication with rather simple ICs (slave)
- slave is addressed by ID (7 or 10 bit)
  (different IDs for different types of ICs)
- master often implemented in software: bitbanging with 2 lines
- SDA sampled on rising edge of SCL
- special bus states: START, STOP, (N)ACK
- START: SCL=1, SDA->0; STOP: SCL=1, SDA->1

serial transmissions(II): I²C

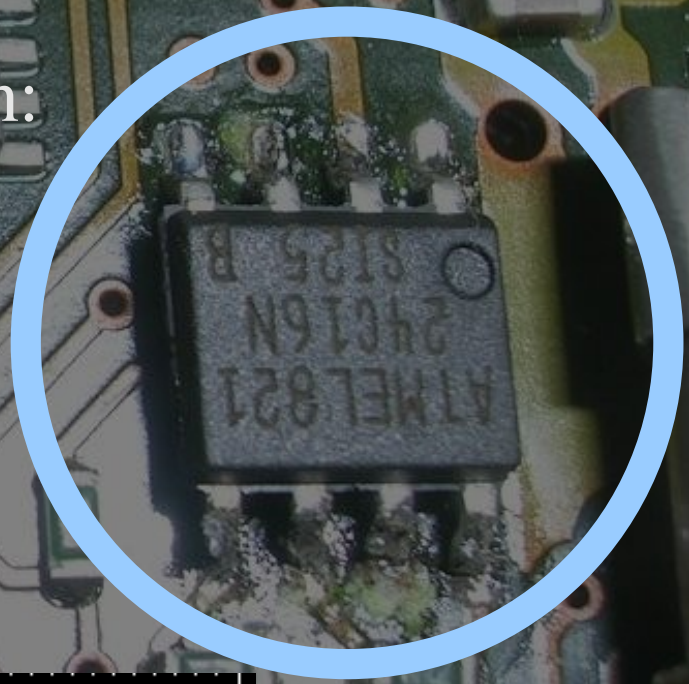


- transmitter sends as long as the receiver acks
- last bit of slaveaddr. is R/nW bit (1: read, 0: write)
- write means: master will send, read: slave has to send
- read from register/addr. X works by doing a dummy-write to register/addr. X, aborting before any data is actually written and doing a read afterwards
- logging can be done with parport and software if low-speed or
- a microcontroller, etc. if hispeed – more details: later!

serial transmissions(II): I²C - Example

serial EEPROMs (usually 24cN where N denotes kbits):

- slaveaddr: 0xcN (default: N=0)
- used to store: config, vendor IDs, serial numbers, code
- easy to interface, easy to sniff
- example: pin number for DECT station:



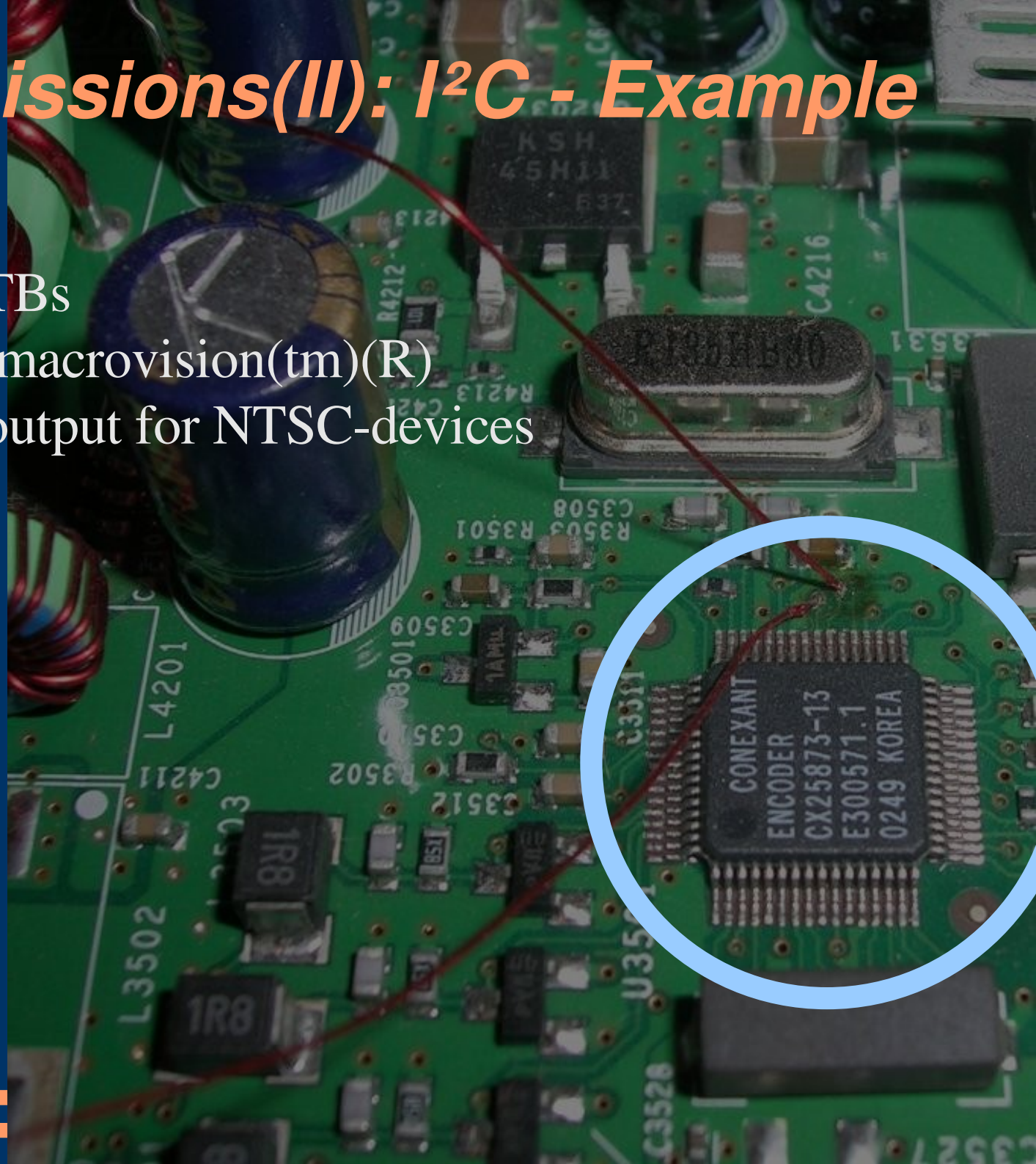
```
0000032A 03 03 03 03 03 03 03 03 01 DF FF FF FF FF FF FF  ................
00000339 FF FF FF FF FF 01 C2 01 3F FF FF FF FF FF FF FF  ........?.......
00000348 FF FF 01 CD 20 13 FF FF FF FF FF FF FF FF FF 03  .... ...........
00000357 32 33 32 33 01 00 FF FF FF FF FF FF FF FF FF FF  2323............
00000366 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ....▮...........
00000375 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00000384 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
```

serial transmissions(II): I²C - Example

video encoder:

- used in many STBs
- one can disable macrovision(tm)(R)
- or enable PAL-output for NTSC-devices
  (or vice-versa)

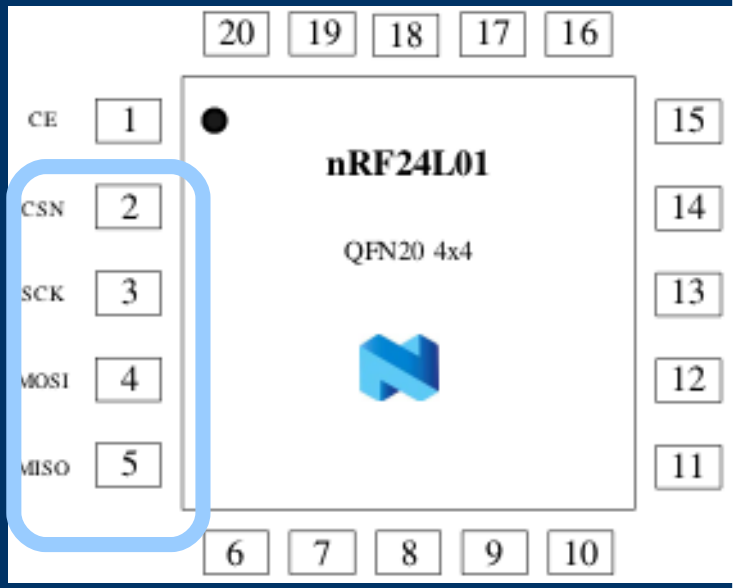
```
88 b2 86
88 b4 25
88 ba 30
88 ba
89 38
88 ba 0a
88 b8
89 00
88 b8 00
88 76 10
88 7a 72
```

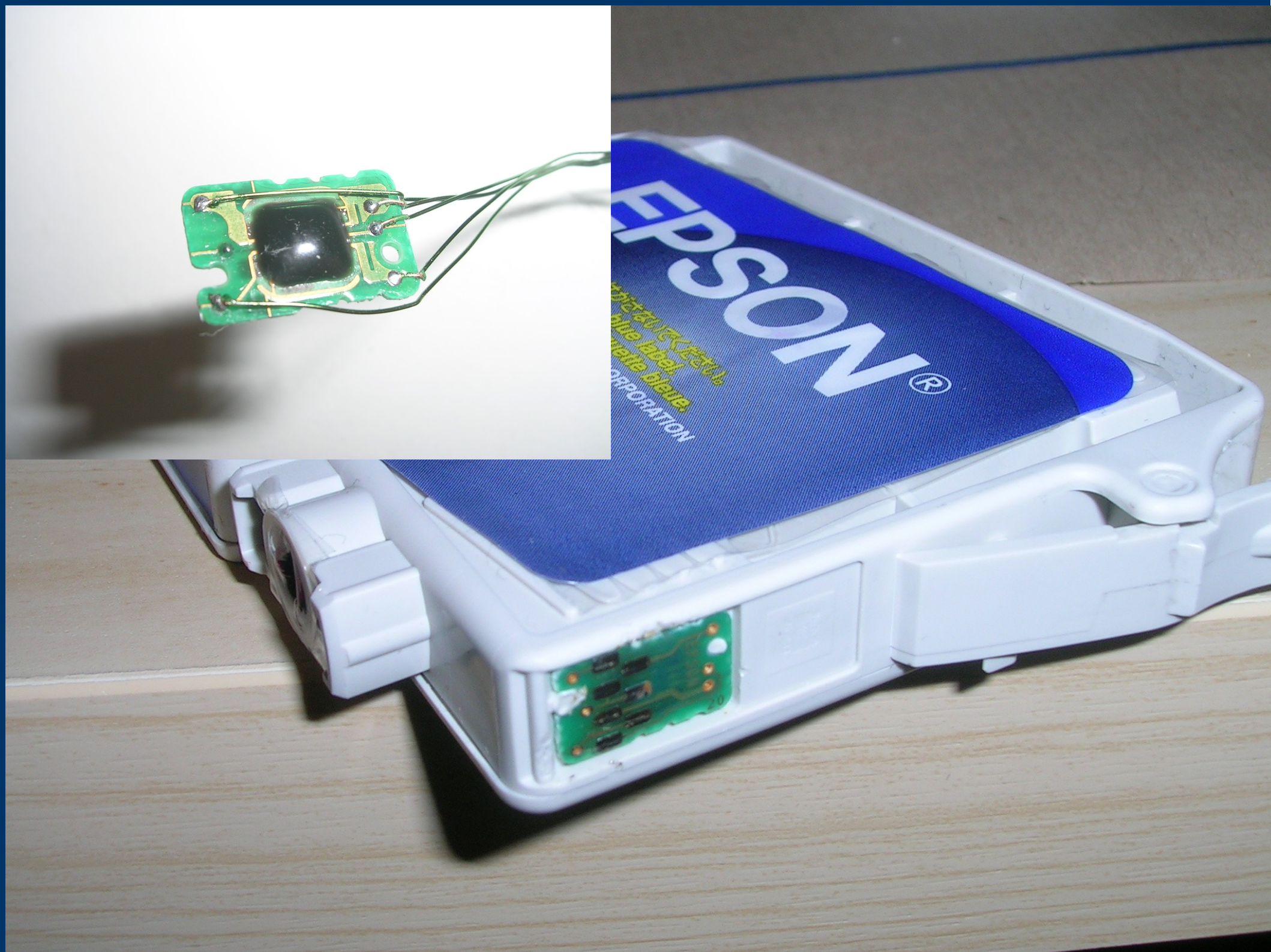


serial transmissions(III): SPI

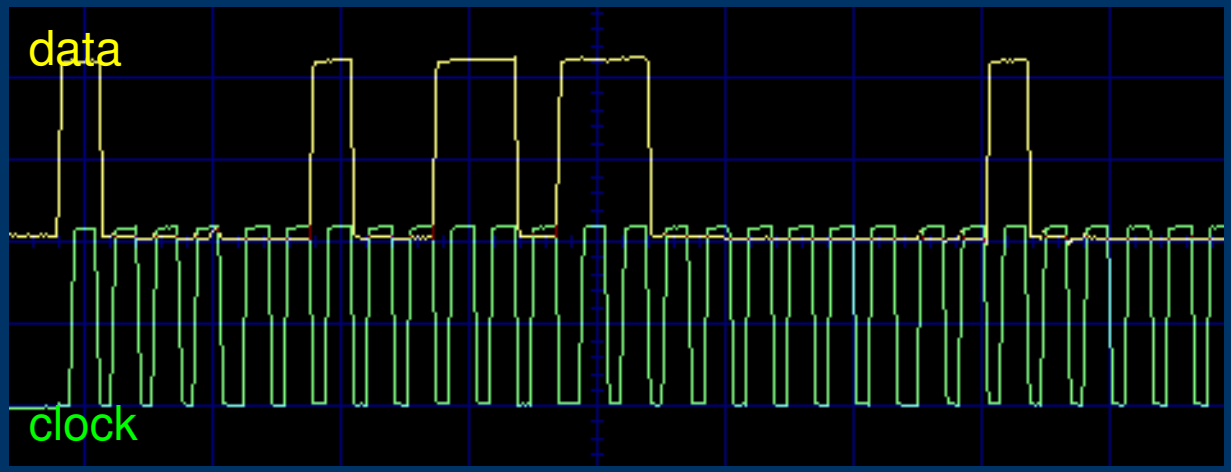
hi speed (several MHz possible), master/slave, synchronous

- SerialClocK, MasterInSlaveOut, MasterOutSlaveIn, SlaveSelect
- if you find a fast clock signal it could be SCK
- usually implemented in hardware
- usage: similar to I²C, but higher speed
- examples: radio modem ICs (sputnik! :-)), SD-cards, ADCs, in-system-programmers
- logging: no simple solution for hi speeds




EPSON®

serial transmissions(IV): ink cartridges



- reset, data and clock lines -> synchronous (read at 62.5kHz)
- printer is master, cartridges are slaves
- after nRST->1: 3? bit cartridge address, 1 bit r/w, databits
- one can write arbitrary data into the chip
- for refill, one reads the data from a new, full cartridge and writes „full“ back once it's empty (that's what chip-resetters do)
- building a fake-chip is also possible with a tiny microcontroller

wireless transmissions: 433 / 868 MHz

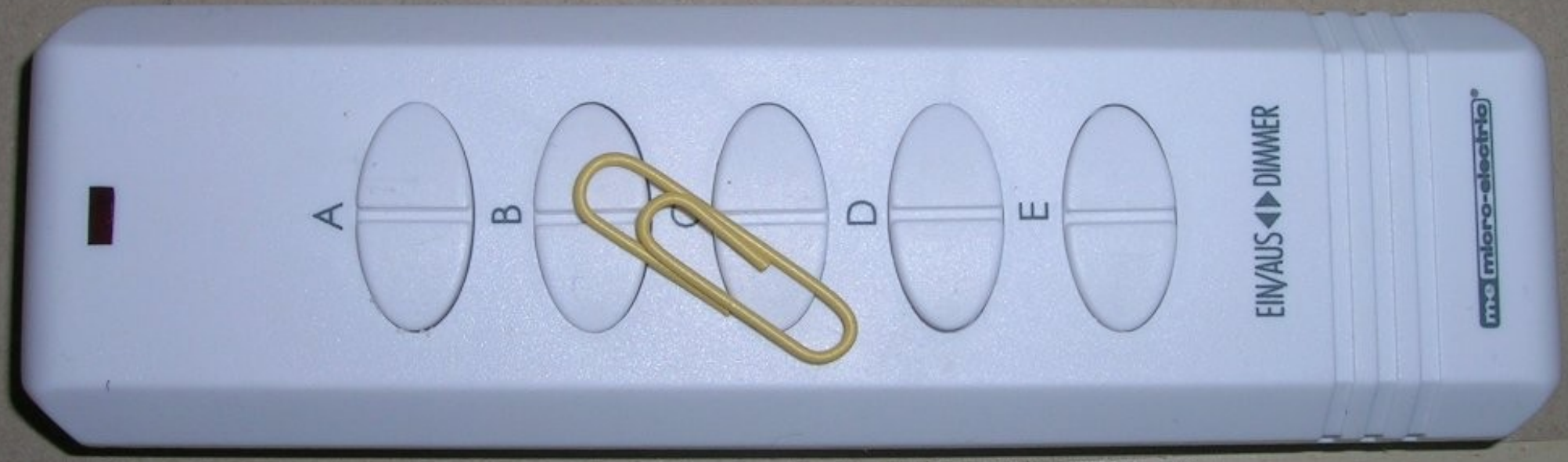
- such devices include: doorbells, mains switches, dimmers, air-pressure sensors, temperature sensors, fire alarms, burglar alarms
- different encodings (Manchester, NRZ, ...)
- different IDs – usually 7-12 bits
- simple en/decoders – example: PT2272
- easy to sniff using a €5 wireless doorbell

- sniffing helpers:
  - your ears
  - soundcard


wireless alarm system (€ 39.??)
1
2
3
4
5
6
7
8
9
*
0
ARMED
BATTERY
FLAMINGO
WIRELESS ALARM SYSTEM HA31
let's hack this one...

McGyver-style! :-)

didn't work :-(



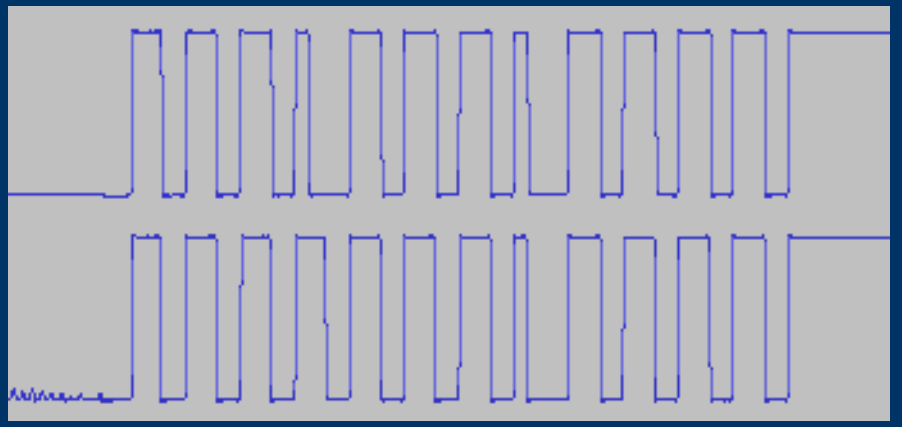
so we need a more sophisticated hackertool...


here it is! ;-)
A
B
C
D
E
EIN/AUS ◀▶ DIMMER
m micro-electric


here it is! ;-)
A
B
C
D
E
EIN/AUS ◀▶ DIMMER
micro-electric
ok, so the wireless sensors are crap, but!
the alarm system is protected agains DoS...

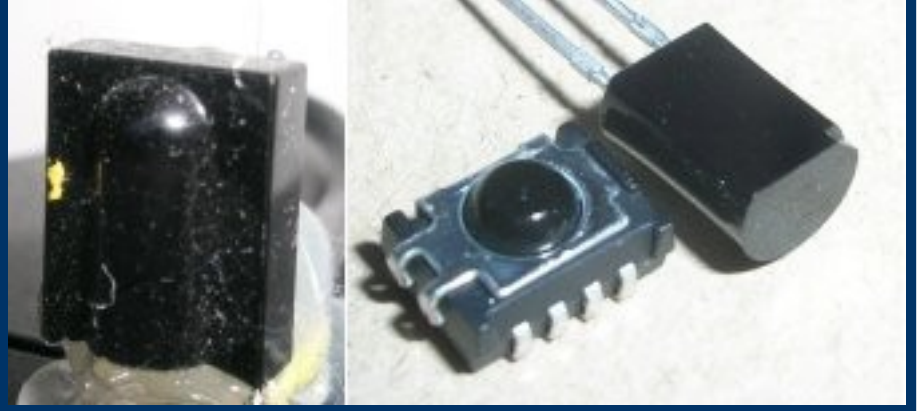
so don't take it off the wall! %-)

looking at the signal



- audacity used here
- sent in case of alarm event
- 1 startbit, 4 addr.bits, 7 foobarbits
  (same for both sensors)
- return to zero signal coding -> Manchester
- boring, the fun wasn't worth the 40 bucks ;-(

similar fun: infrared!



- carrier frequency
  - common: 30-38, 445kHz - sniffing effort depends on the carrier!
- can be used for async. serial transmissions up to 2.4kbaud
- look for daylight filters! (black or dark red)
- where can you find IR transmissions?
  Hifi remotes, RC-vehicles,
  drink/etc.-automats,
  public transfer :-}
- use your cellphone-cam!

IR devices

back (in)to more complex devices: memory interfaces

- parallel, usually 8/16/32 databits, control- (nReaD, nWRite) and
- addresslines (may be multiplexed with datalines)
- used to access Flash-memories
  -> you can also use it to dump the software!
- often high speed (xx MHz) – makes tapping more complicated
- with some fiddling you can dump them without costy hardware!
- read the datasheet, use your parport or a microcontroller

weapons of choice

- for the analog part: oscilloscope!
- expensive, but useful – at least to get a first impression of signals
- low-cost „substitutions“:
  - soundcard (48 / 96kHz (HD) sampling, 2 channels)
  - ADC of AVR µcontrollers: 666.666 kHz sampling rate ((16MHz/2)/12)
    - great: 10bits, up to 200x gain -> few µVolts precision!
    - sufficient to use a IR-diode without analog amplification!

- for the digital part: logic analyzers
- TOO expensive! put something together yourself
- every problem is different -> custom hacks necessary

D'OH we need hard realtime...

- PC parport: (n)nn kHz sampling, **but:** up to 2MByte/s with ECP
- use a µcontroller if possible, a CPLD/FPGA if necessary
  - AVR 8bit: slow but easy, great for hard realtime
  - ARM7 32bit: LPC/AT91 – fast, hard realtime not that easy (pipeline, caches)
- use a ringbuffer! **use a overrun indicator!**
  or you'll waste hours analyzing crap-data!
- data transfer:
  - RS232: 115.2kbaud -> ~9kBytes/s
  - full speed usb: ~1MByte/s, hi speed: <60MByte/s
  - PATA: up to ~130? MByte/s, SATA
  - ethernet doesn't make any sense here

questions you should ask yourself first

- **how much data** do I want to log? how many signals?
  - no fast interface is necessary if it fits completely into a buffer!
- what's the **maximum speed** of the signal?
  - how do I get an appropriate samplerate?
  - what interface do I use to transfer the data to the PC?
- how do I store the data in the buffer? how do I send it to the PC?
  - do you have enough processing power to store it that way?
  - do you have enough throughput to send it that way?
  - example I²C: 1 vs. 4 samples per byte

finding answers

- **how much data?** answer: count the transitions!
  - hardware counter of a microcontroller if fast enough
  - hardware counter using logic else

- **maximum speed?** answer: min. time between 2 transitions!
  - hardware timer of a microcontroller if fast enough
  - hardware timer using logic else

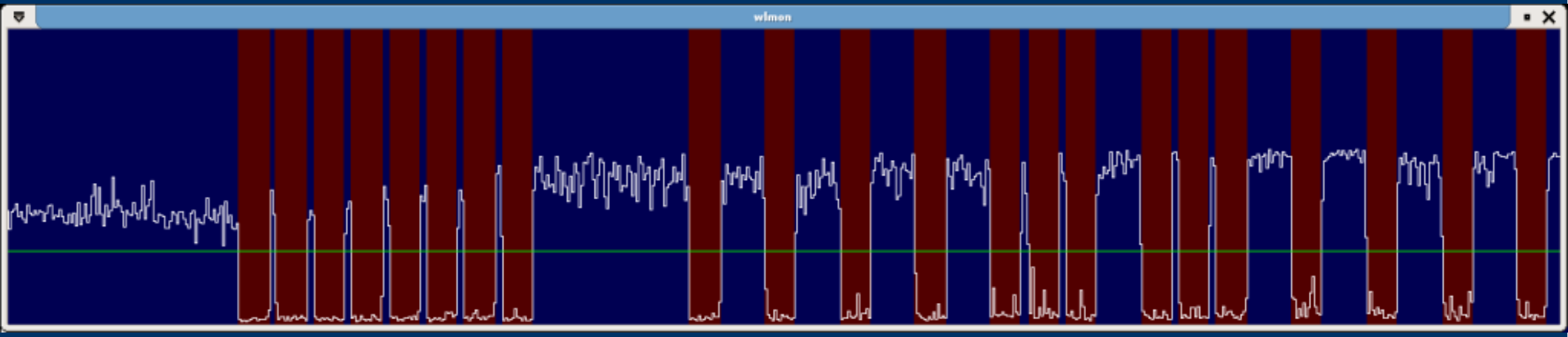
- in general:
  - fast, simple stuff -> job for hardware
  - slow, possibly complex stuff -> job for software

don't panic: the timer in VHDL

```
63  process(CLK, RST)
64  begin
65
66     -- reset
67     if RST='1' then
68
69        TMIN <= (others => '1');
70        LASTSIG <= SIGIN;
71        TIMER <= (others => '0');
72
73     elsif CLK'Event and CLK='1' then
74
75        LASTSIG <= SIGIN; -- still valid in this cycle!
76
77        -- signal changed
78        if LASTSIG /= SIGIN then
79
80           TIMER <= (others => '0'); -- same here
81
82           -- new min
83           if TMIN > TIMER then
84              TMIN <= TIMER;
85           end if;
86
87        elsif TIMER /= "1111111111" then
88           TIMER <= TIMER + 1;
89        end if;
90
91     end if; -- clk
92
93  end process;
94
```

XC9572XL: speed ~66MHz, 50-75% logic usage

analyzing the results



- custom visualizers for signals can be easily put together using SDL_gfx
- if you want to write a picture from your data,
  have a look at the pbm format (ASCII header + raw data)
- print the stuff that drives you nuts to paper!
- use crayons of several colors to work on the signal(s)! :-)

conclusion

- there are a lot of simple, proprietary devices out there
- they have little to zero security features
- they rely on security by obscurity
  hackers are computer-guys - no one will notice this ever!111
- hacking them is simple, though funny
- do it! ;-)


Thanks for listening! :-)
ATMEGA32
16PI
0426B
ATMEGA32
16AU 0612B