(1)

Přenosová rychlost je počet datových bitů za jednotku času.
1 baud je maximální jednotkou přenosu informace za jednotku času, tedy i včetně start/stop bits.

$\frac{176}{8} = 22$

Overhead linky činí $1 - \frac{8}{11}$

Pokud má linka propustnost 16 MB/s datových, potřebuje $16 \cdot \frac{11}{8}$ MB/s
Celkově tedy propustnost 22 MB/s

Pokud budeme mít 32 datových, je overhead pouze $1 - \frac{32}{35}$
na přenosovou rychlost 22 MB/s bude potřebovat $22 \cdot \frac{35}{32}$ MB/s $\doteq$ 24 MB/s

Obecně platí, že čím menší overhead, tím méně musí být nadimenzovaná linka.

$\frac{2}{3} \cdot 35 \approx 24$ opět přibližně nemám kalkulačku

Pokud chápeme MB $\sim$ 1024 kB $\sim$ 1024 · 1024 B, tak rychlost v bajtech u prvního potřebujeme $\doteq$ 22 000 000 bajtů
u druhého potřebujeme $\doteq$ 24 000 000 bajtů.

nemám kalkulačku a nechci ztrácet čas

Což reflektuje skutečnost, že čím menší overhead, tím méně se musí linka předimenzovat.

Handshaking domluvy Master a Slave (clock hold low etc.) je definováno protokolem komunikace.

⑤

Master a Slave je důležitá charakteristika na datové sběrnici.
Aby byl provoz kontrolovatelný a usměrněný, vždy se definuje,
s jakým mistrem!! přenosem, kdo je aktuálně Master a kdo Slave.

při kterém přenosu samozřejmě.

Existují multimaster i singlemaster sběrnice - tedy zda Master
může být jen jedno zařízení či více zařízení.

Master je ten, který pro konkrétní přenos zadává požadavky (řídí komunikaci)
a Slave je ten, který odpovídá vč. vyhovění příkazu.

Typickým příkladem Mastera je procesor, který svým sběrům,
např. registrům definuje, data ze kterých adres mu mají vrátit.

Obecně vždy platí, že pokud jdou data od mastera, je to write, pokud
od slava k masterovi, je to read.

⑥

IEEE 754

$-1555{,}625$ uložit do paměti na 0x0000FA00 jako single (32-bit float)

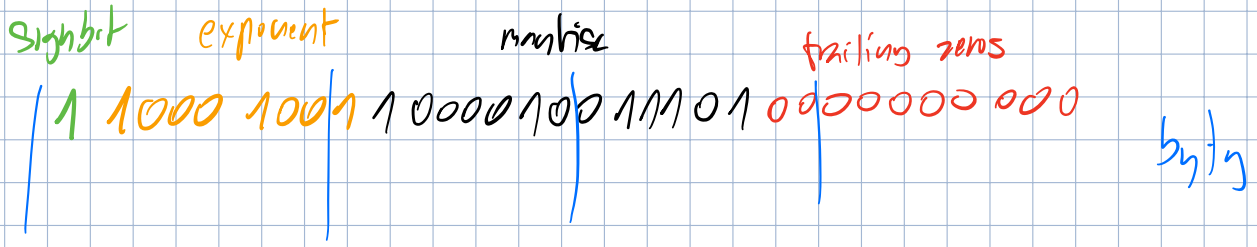
IEEE 754 float: spodních 23 bitů (+ skrytá jednička)
8 bitů exponent (bias [+127])
MSb sign bit

1555 = 11000010011
0,625 = 101

11000010011,101 0000000000

skrytá jednička

exp = 10 → bias +127 → $137_{10} = 1000\ 1001_2$



C 4 C 2 7 4 0 0 hex

0xC4C27400

Takže postupně na adrese 0x0000FA00 a dále: 00 74 C2 C4

- little endian (LSB first)

⑦ ŠEMÍK2 0x0001F02 addr.

a) Windows 1250

b) UTF-16 LE

	UNI	WIN
Š =	0160	8A
E =	0045	45
M =	004D	4D
Í =	00CD	CD
K =	004B	4B
? =	003F	3F

LF!

UNI: 0160 0045 004D 00CD 004B 003F

WIN: 8A 45 4DCD 4B3F

Hex dump ONI:

0x00001F02	3F 00 4B 00 CD 00 4D 00	45 00 60 01 00 00 00 00

Hex dump WIN:

0x00001F02	8A 45 4D CD 4B 3F 00 00	00 00 00 00 00 00 00 00

(3) Zapište posloupnost bez ACK!

- 8 data bits, 1 ACK bit, Negative values in Two's Comp.

Addr: $W = 42_{16}$ $R = 43_{hex}$ (7 bit) – LSb $R/\overline{W}$

– 9th SCL pulse – ACK

Magnetometer X offset = 0x01 (MSB)
0x02 (LSB) (MSB first writing)

Musím poslat: 1 byte (adresa + $R/\overline{W}$)
1 byte (command)
1 byte (adresa)
2 bytes (data)

$2255_{10} = 0000\ 1000 \mid 1100\ 1111_2$

0 8 | C F

08 CF

Data:

[42 77 01 08]
[42 77 02 CF]

↑ start condition
↑ adresa slave
↑ command
↑ bez ACK
↑ adresa in EEPROM
↑ data
↑ stop condition

(b) (tabulka - page 6.)

```
                              uint8      int
                                ↓         ↓
def SetNewOpModeRate(opMode, newRate)      ↙ nejsem si jistý, jaké závorky oddělují data v dictionary
    rates = {(1, uint8(0)), (5, uint8(1)), (10, uint8(2)), (20, uint8(3))}
    check = opMode   # Abych si nezměnil moje data
    if check << 6 != uint(127):   # When not in query mode
        if check << 6 == uint(0) OR check << 6 == uint(256):
            # When in StandBy or Continuous Mode
            opMode = opMode & uint8(159)   # clearing
            OpMode = opMode | rates[newRate]
            # setting up new rate from dict.
            if check << 6 == 0:
                # Set from StandBy to Continuous
                OpMode = opMode | uint8(2)   # setting mode
    return opMode
```

check << 6 ?

?? 000000

0	00	- StandBy
127	01	- Query Mode
256	10	- Continuous
	11	- Not Allowed

clearing rate bits to zero:

```
        ? 01 ? ?? ??
mask:   1 00 1 11 11    ↓ &
 ↑      ? (00) ? ?? ??
159
```

setting Mode

```
        ?????? 00
mask    000000 10    OR
  ↓
        ?????? 10
```

(8) C = C + d + d + 254

16b! c = 0x7000

16b! d = 0x7002

proměnná v $0x0000 - 0x00FF$

254 = 0xFE

1) var0 = d + d

2) C = C + var0

3) C = C + 254

var0 = 0x0000 (2 d)

1)
```
LDAA $7002
CLC
ADCA $7002
STAA $0000
LDAA $7003
ADCA $7003
STAA $0001
```

víceBytová sčítačka

uložení jednotlivých bytů do paměti

d + d

2)
```
LDAA $7000
CLC
ADCA $0000
STAA $0000
LDAA $7001
ADCA $0001
STAA $0001
```

- sčítání míst (hodnotu jsem už přičetl)
- sčítání míst (hodnotu jsem už přičetl)

víceBytová sčítačka

uložení C+2d do var0 paměti

C + 2d

3)
```
LDAA $0000
CLC
ADCA #$FE
STAA $7000
LDAA $0001
ADCA #$00
STAA $7001
```

více bytová carry sčítačka s 00FE (pro případný carry přenos)

uložení zpět do „c"

4) offset = 80 00 00 00 = 00000080 = 128_{10} = 0x0000080

PE: 50 45 00 00 → signatura

pak 20 bytů (velikost COFF Header), pak 4 byty offset a následují 4 byty „SizeOfCode"

SizeOfCode: 00000800 = $8 \cdot 16^2 = 2048$

10)
```
def IsPeExecutable(fileName:str)
    file = open(fileName, "rb")
    mz_sign = file.read(2)
    if mz_sign[0] != 0x4D or mz_sign[1] != 0x5A:
        return False
    else:
        size_of_file = os.path.getsize(fileName)
        if size_of_file > 0x3C:
            file.seek(0x3C)
            pe_sign_offset = file.read(2)
```

→ první dva byty nás zajímají

vychází z hexdump souboru v příloze

```
if pe_sign_offset + 0x3C < size_of_file:
    file.seek(pe_sign_offset)
    pe_sign = file.read(2)
    if pe_sign[0] == 0x50 or pe_sign[1] == 0x45:
        return True
    else:
        return False
else:
    return False
else:
    return False
```