# MRC-100 hasznos terhek

Az együttműködő egyetemi partnerek:

- Debrecen
- Győr
- Óbudai
- Szabadka
- Szeged

## 1. Geometriai és technológiai megkötések

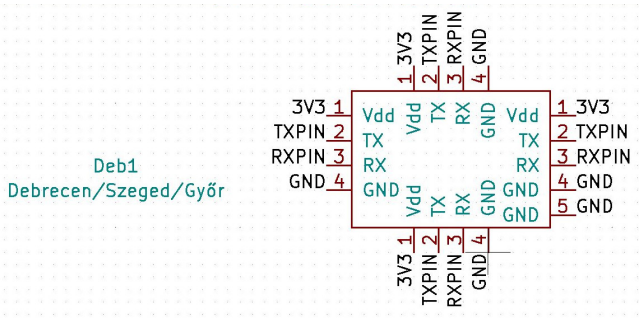
- Méret: 30 x 30 x 3 mm (panelvastagság nélkül).
- Össztömeg: max. 50 g (panel és alkatrészek forrasztva összesen).
- A részegység megvalósítása 0,8 - 1,5 mm vastag FR-4-es áramköri hordozón lehetséges.
- Az alkatrészek elhelyezése kizárólag a felső rétegen lehetséges, az alsó réteg sík kell maradjon, rajzolat és viázás lehet rajta.
- Legalább 2 réz réteg kell (top, bottom), lehet 4, 6 vagy 8 rétegű is.
- Furatfémezés szükséges és alkalmazandó, eltemetett via alkalmazása TILOS.
- Kontúrmarás kell, ennek hatására lesznek a széleken félviák, az MRC-100 belső panelhez való forrasztáshoz.
- Középen nagyméretű fémezett furat a megfelelő mechanikai rögzíthetőség miatt kell, felette alkatrész nem lehet, mert a furaton keresztül lesz forrasztással (ónkitöltés) rögzítve - mechanikai rezonancia. Potenciál: GND (0 V referencia pont).
- Forrasztás gátló lakk alkalmazása TILOS.
- Alkatrész pozíció szita alkalmazása TILOS.
- A réz vezetősávok és lábnyomok aranyozása szükséges (szabványos NYHL technológia, a repülő példányok Eurocircuits-nél való gyártatása szükséges).
- Ón alapú maszk alkalmazása TILOS.
- Az alkatrészek forrasztása kizárólagosan 60/40 %-os ón-ólom eutektikummal lehetséges kézi forrasztási eljárással.
- Ólommentes anyaggal való forrasztás és paszta alapú forrasztás TILOS.

# 2. Elektromos kialakítás

A KiCad (ver. 5.1.12) kapcsolási rajz és nyomtatott huzalozású lemez tervező program használata szükséges. Nyílt forráskódú, szabadon használható: `https://www.kicad.org/`.

## 2.1. Kapcsolási rajz szimbólum, ami tőlünk látszik belőletek

A négyzet alakú modul mind a 4 oldalán rendre ugyanazok a csatlakozási pontok kerülnek kialakításra, az azonosak össze lesznek kötve. Ebből adódóan lehet olyat, hogy a felső oldalon a tápfesz hozzávezetés, alsó oldalon a GND, jobbra a soros adatvolnal TX, balra pl. RX, de ezek lehetnek egy oldalon is (ahogy könnyebb a huzalozást kialakítani). A nem használt viák lebegnek elektromosan a részegységek, mint hasznos terhek paneljein, ekkor csak mechanikai rögzítésre szolgálnak - 1. ábra.



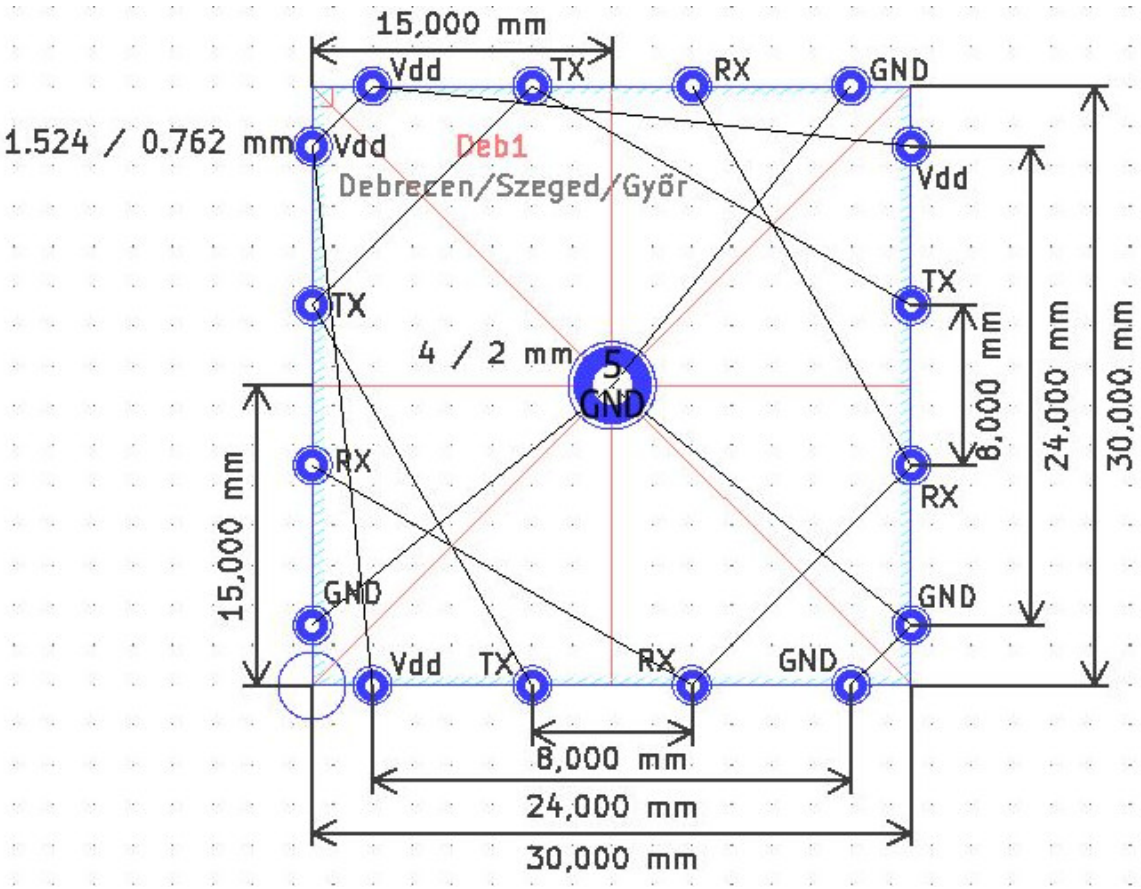

1. ábra. A kapcsolási rajz szimbólum: a hasznos teher, mint modul

1. Vdd - áramhatároló kapcsolóval védett névleges +3,3 V tápfeszültség.
2. TX vagy TXPIN - a hasznos teher soros adatvonalának adó kimenete (UART TX).
3. RX vagy RXPIN - a hasznos teher soros adatvonalának vevő bemenete (UART RX).
4. GND - referencia pont, 0 V-os potenciálú hely, félviák oldalt.
5. GND - referencia pont, 0 V-os potenciálú hely, középen levő nagyméretű furat.

A KiCad szimbólum a csatolt *payload.lib* fájl, lásd 5.1.

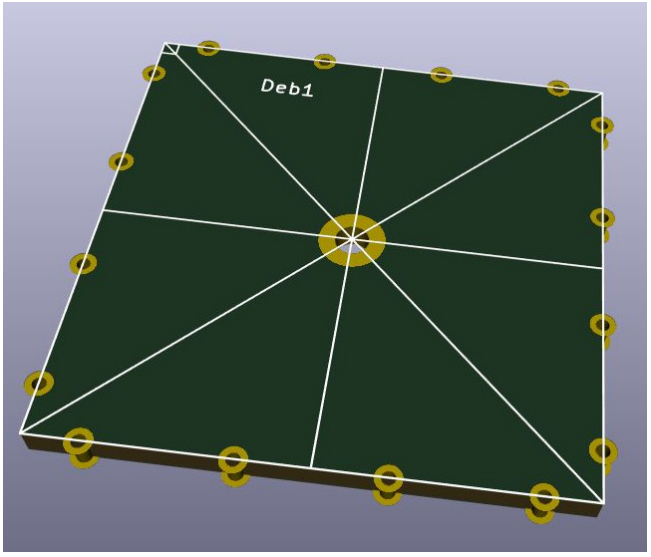
## 2.2. Nyomtatott huzalozású lemez és kontúr

A hasznos terhek paneljének kontúrja, valamint az azon alkalmazandó fél és egész viák geomatriai elrendezése látható a 2. ábrán.




2. ábra. NYHL: kontúr a rögzítést biztosító viákkal és félviákkal

A félviák mérete átmérő 1,524 mm, 0,762 mm-es furattal, ebből 16 darab a 4 oldalon elosztva, a középen levő fémezett via átmérő 4 mm, 2 mm-es furattal.




3. ábra. Alkatrész nélküli hasznos teher panel 3D-ben

Az alkatrész nélküli hasznos teher, mint modul panel 3D-ben látható a 3. ábrán.

A KiCad-ben készített lábnyom fájl: *payload.kicad_mod*, hasonlóan a *lib* fájlhoz, ez is szöveges, és itt a tartalma: 5.2.

# 3. Tápellátás

Minden hasznos teher számára biztosítunk áramhatároló kapcsolóval védett névleges +3,3 V tápfeszültséget, maximális 100 mA fogyasztott áram mellett, amely körátlag (90 perc időtartamra számítva az átlagot) nem haladhatja meg a 34 mA értéket.

A névleges +3,3 V legmagasabb értéke 3,5 V, legalacsonyabb 2,7 V, amikor az energia ellátó rendszer SDC-je egybenyit, és egy lemerült LiIon cella feszültsége jelenik meg a szabályozott energia buszon (ilyen nem valószínű, hogy bekövetkezik, de számoljunk vele).

A modulon belül nem kell alkatrész szinten sem tartalékolni, duplikálni.

# 4. Digitális adatkapcsolat

CMOS +3,3 V-os aszinkron soros porti kommunikáció (aktív LOW, vagyis amikor nincs adatkommunikáció, akkor mind a TX, mind pedig az RX láb logikai magas szinten van), 1 vezetékes UART formában, fél-duplex, egy időben vagy ad vagy vesz. Miközben ad, veszi saját maga jelét (célszerűen adásnál vagy ki kell kapcsolni az UART vevőt, vagy küldött adatbit ellenőrzésre lehet használni, hiszen a TX láb csak lefelé tud majd húzni).

A hasznos terheket tartó paneleken valósítjuk meg a 30x30-as NYHL körül az áramhatároló kapcsolót, és a "soros port - 1 wire UART illesztő" áramkört, amely megakadályozza a 1W-UART adat busz statikus reteszelődését egyes rajta lógó részegységek meghibásodása esetén (ilyen lehet a latch-up jelenség is).

Adatsebesség: 10 000 baud (10 kbaud +/- 1% órajel tolerancia) a teljes -40 ... +80 Celsius fok hőmérséklet tartományban (levegős hőkamrás tesztelni kell).

8 adatbit, nincs paritás, 1 stop bit (nem, nem elírás, 10 000 baud, nem pedig 9600, ui. a szokványos órajelekről csak hibával leoszthatóa 9 600).

## 4.1. Protokoll

- Üzenet váltás: OBC (On-Board Computer, MRC-100 fedélzeti számítógép) üzenet küldés → a periféria (vagyis a hasznos teher) az üzenetet feldolgozza → a periféria válaszol az OBC-nek.
- Csak ismert parancsra szabad válaszolni.
- Hibás checksum (ellenőrző összeg) esetén a vett üzenet figyelmen kívül hagyandó (mintha nem is küldtek volna semmit).
- A hasznos teher tápfeszültség rákapcsolás után csak figyel, bármilyen soros porti adat küldése (jelezve a modul feléledését) szigorúan TILOS.

## 4.2. A digitális üzenetek szerkezete

**<start karakter><üzenet><lezáró szekvencia>**

- start karakter: '$' - piros,
- üzenet: 5 karakter nagybetűs ASCII szám/betű - zöld,
- majd opcionálisan vesszővel elválasztott paramétermezők (csak olvasható karaktereket tartalmazhat) - fekete:
  - üres mező: két vessző között nincs semmi

  - 1 karakteres mező: kisbetű/nagybetű/szám
  - 2 karakteres mező: 8 bites szám BigEndian ASCII HEX stringként (számok és 'A'-'F'; pl. decimális 42 adat → 2A olvasható nagy karakteres BigEndian HEX string)
  - 4 karakteres mező: 16 bites szám (pl. decimális 19906 → 4DC2)
  - 8 karakteres mező: 32 bites szám

- lezáró szekvencia - kék:

  <kettőskereszt><8 bites sorszám BigEndian ASCII HEX><csillag><16 bites checksum BigEndian ASCII HEX><CR><LF>

  Üzenet sorszám a # utáni YY érték: 0...255 érték tartomány, túlcsordul (moduló 256), 8 bites mező BigEndian ASCII HEX stringként, az OBC által küldött parancsból vett sorszámot adjuk meg a válaszüzenetben (ebből tudja az OBC, hogy ki mikor mire válaszol).

  Ellenőrző összeg (checksum) a * utáni XXXX érték: 16 bites mező BigEndian ASCII HEX stringként, Fletcher16 checksum a $ és a * karakterek közötti részre (a $ és * karaktereket nem beleszámítva, a vesszőket is beleszámítva).

Példa üzenet: **$CMDNM ,p1,p2,...,p13#YY*XXXX<CR><LF>**

Az ellenőrző összeg számításához példa kód részlet itt: 5.3.

Budapest, 2022. január 5.

Dr. Dudás Levente

# 5. Függelék

## 5.1. Schematic lib fájl

```
EESchema-LIBRARY Version 2.4
#encoding utf-8
#
# Debrecen
#
DEF Debrecen Deb 0 40 Y Y 1 F N
F0 "Deb" -100 150 50 H V C CNN
F1 "Debrecen" 0 50 50 H V C CNN
F2 "" 0 0 118 H I C CNN
F3 "" 0 0 118 H I C CNN
DRAW
S -800 0 0 -550 0 1 0 N
X Vdd 1 -900 -100 100 R 50 50 1 1 I
X Vdd 1 -550 -650 100 U 50 50 1 1 I
X Vdd 1 -550 100 100 D 50 50 1 1 I
X Vdd 1 100 -100 100 L 50 50 1 1 I
X TX 2 -900 -200 100 R 50 50 1 1 O
X TX 2 -450 -650 100 U 50 50 1 1 O
X TX 2 -450 100 100 D 50 50 1 1 O
X TX 2 100 -200 100 L 50 50 1 1 O
X RX 3 -900 -300 100 R 50 50 1 1 I
X RX 3 -350 -650 100 U 50 50 1 1 I
X RX 3 -350 100 100 D 50 50 1 1 I
X RX 3 100 -300 100 L 50 50 1 1 I
X GND 4 -900 -400 100 R 50 50 1 1 I
X GND 4 -250 -650 100 U 50 50 1 1 I
X GND 4 -250 100 100 D 50 50 1 1 I
X GND 4 100 -400 100 L 50 50 1 1 I
X GND 5 100 -500 100 L 50 50 1 1 I
ENDDRAW
ENDDEF
#
#End Library
```

../payload.lib

## 5.2. PCB kicad_mod fájl

```
(module Debrecen (layer F.Cu) (tedit 61BF50C8)
  (fp_text reference REF** (at -6 3) (layer F.SilkS)
    (effects (font (size 1 1) (thickness 0.15)))
  )
  (fp_text value Debrecen (at -5 5) (layer F.Fab)
    (effects (font (size 1 1) (thickness 0.15)))
  )
  (fp_line (start -15 0) (end 15 0) (layer F.SilkS) (width 0.12))
  (fp_line (start 15 0) (end 15 30) (layer F.SilkS) (width 0.12))
  (fp_line (start 15 30) (end -15 30) (layer F.SilkS) (width 0.12))
  (fp_line (start -15 30) (end -15 0) (layer F.SilkS) (width 0.12))
  (fp_line (start -14 0) (end -14 1) (layer F.SilkS) (width 0.12))
  (fp_line (start -14 1) (end -15 1) (layer F.SilkS) (width 0.12))
  (fp_line (start -15 0) (end 15 30) (layer F.SilkS) (width 0.12))
  (fp_line (start 15 0) (end -15 30) (layer F.SilkS) (width 0.12))
  (fp_line (start -15 15) (end 15 15) (layer F.SilkS) (width 0.12))
  (fp_line (start 0 0) (end 0 30) (layer F.SilkS) (width 0.12))
  (pad 1 thru_hole circle (at -12 0) (size 1.524 1.524) (drill 0.762) (layers *.
    Cu *.Mask))
  (pad 2 thru_hole circle (at -4 0) (size 1.524 1.524) (drill 0.762) (layers *.
    Cu *.Mask))
  (pad 3 thru_hole circle (at 4 0) (size 1.524 1.524) (drill 0.762) (layers *.Cu
     *.Mask))
  (pad 4 thru_hole circle (at 12 0) (size 1.524 1.524) (drill 0.762) (layers *.
    Cu *.Mask))
  (pad 1 thru_hole circle (at -15 3) (size 1.524 1.524) (drill 0.762) (layers *.
    Cu *.Mask))
  (pad 2 thru_hole circle (at -15 11) (size 1.524 1.524) (drill 0.762) (layers
    *.Cu *.Mask))
  (pad 3 thru_hole circle (at -15 19) (size 1.524 1.524) (drill 0.762) (layers
    *.Cu *.Mask))
  (pad 4 thru_hole circle (at -15 27) (size 1.524 1.524) (drill 0.762) (layers
    *.Cu *.Mask))
  (pad 1 thru_hole circle (at 15 3) (size 1.524 1.524) (drill 0.762) (layers *.
    Cu *.Mask))
  (pad 2 thru_hole circle (at 15 11) (size 1.524 1.524) (drill 0.762) (layers *.
    Cu *.Mask))
  (pad 3 thru_hole circle (at 15 19) (size 1.524 1.524) (drill 0.762) (layers *.
    Cu *.Mask))
  (pad 4 thru_hole circle (at 15 27) (size 1.524 1.524) (drill 0.762) (layers *.
    Cu *.Mask))
  (pad 1 thru_hole circle (at -12 30) (size 1.524 1.524) (drill 0.762) (layers
    *.Cu *.Mask))
  (pad 2 thru_hole circle (at -4 30) (size 1.524 1.524) (drill 0.762) (layers *.
    Cu *.Mask))
  (pad 3 thru_hole circle (at 4 30) (size 1.524 1.524) (drill 0.762) (layers *.
    Cu *.Mask))
  (pad 4 thru_hole circle (at 12 30) (size 1.524 1.524) (drill 0.762) (layers *.
    Cu *.Mask))
  (pad 5 thru_hole circle (at 0 15) (size 4 4) (drill 2) (layers *.Cu *.Mask))
)
```

../payload.kicad_mod

## 5.3. Üzenet ellenőrző összeg számítás

```c
/* includes */
#include <stdio.h>
#include <stdint.h>

/* defines */
#define LL 128

/* function prototypes */
uint16_t fastmod255(uint16_t x);
void fletcher16_init(uint16_t * sum);
void fletcher16_update(uint16_t * sum, uint8_t data);
uint16_t fletcher16_get_chksum(uint16_t * sum);

/* main routine */
int main(int argc, char **argv)
{
  // command, different parameters, serial number
  uint8_t cmd[]="CMDHP";
  uint8_t p1=5;
  uint16_t p2=3456;
  uint32_t p3=121212;
  uint8_t serial=123;

  // index variables
  int i,j;

  // msg array
  uint8_t msg[LL];
  // clear msg
  for(i=0;i<LL;i++) msg[i]=0;
  // print command, parameters, serial number: start char $, end char *, before serial #
  sprintf((char*)msg,"$%s,%02X,%04X,%08X#%02X*",cmd,p1,p2,p3,serial);

  // check sum
  uint16_t sum[2];
  // check sum init (clear)
  fletcher16_init(sum);

  // between $ and * calculate check sum, j indexes the next byte cell after *
  for(i=1;i<LL;i++){
    j=i;
    if(msg[i]=='*') break;
    fletcher16_update(sum, msg[i]);
  }
  // get check sum
  uint16_t checksum=fletcher16_get_chksum(sum);
  // append checksum ASCII HEX string ending with CR LF
  sprintf((char*)msg+(j+1),"%04X\r\n",checksum);

  // printing the whole message
  printf("%s",msg);
  // printing the whole message in hexadecimal format
  for(i=0;i<LL;i++){
    if(msg[i]==0) break;
    printf("%02X ",msg[i]);
  }
  printf("\n");
```

```
  return 0;
}

/* subroutines */

uint16_t fastmod255(uint16_t x)
{
  // max(x) == 510
  if (x < 255) return x;
  x -= 255;
  if (x < 255) return x;
  x -= 255;
  return x;
}

void fletcher16_init(uint16_t * sum)
{
  sum[0] = 0;
  sum[1] = 0;
}

void fletcher16_update(uint16_t * sum, uint8_t data)
{
  sum[0] = fastmod255(sum[0] + (uint16_t)data);
  sum[1] = fastmod255(sum[1] + sum[0]);
}

uint16_t fletcher16_get_chksum(uint16_t * sum)
{
  return ((sum[0] & 0x00FF) + ((sum[1] & 0x00FF) << 8));
}
```

./payload.c