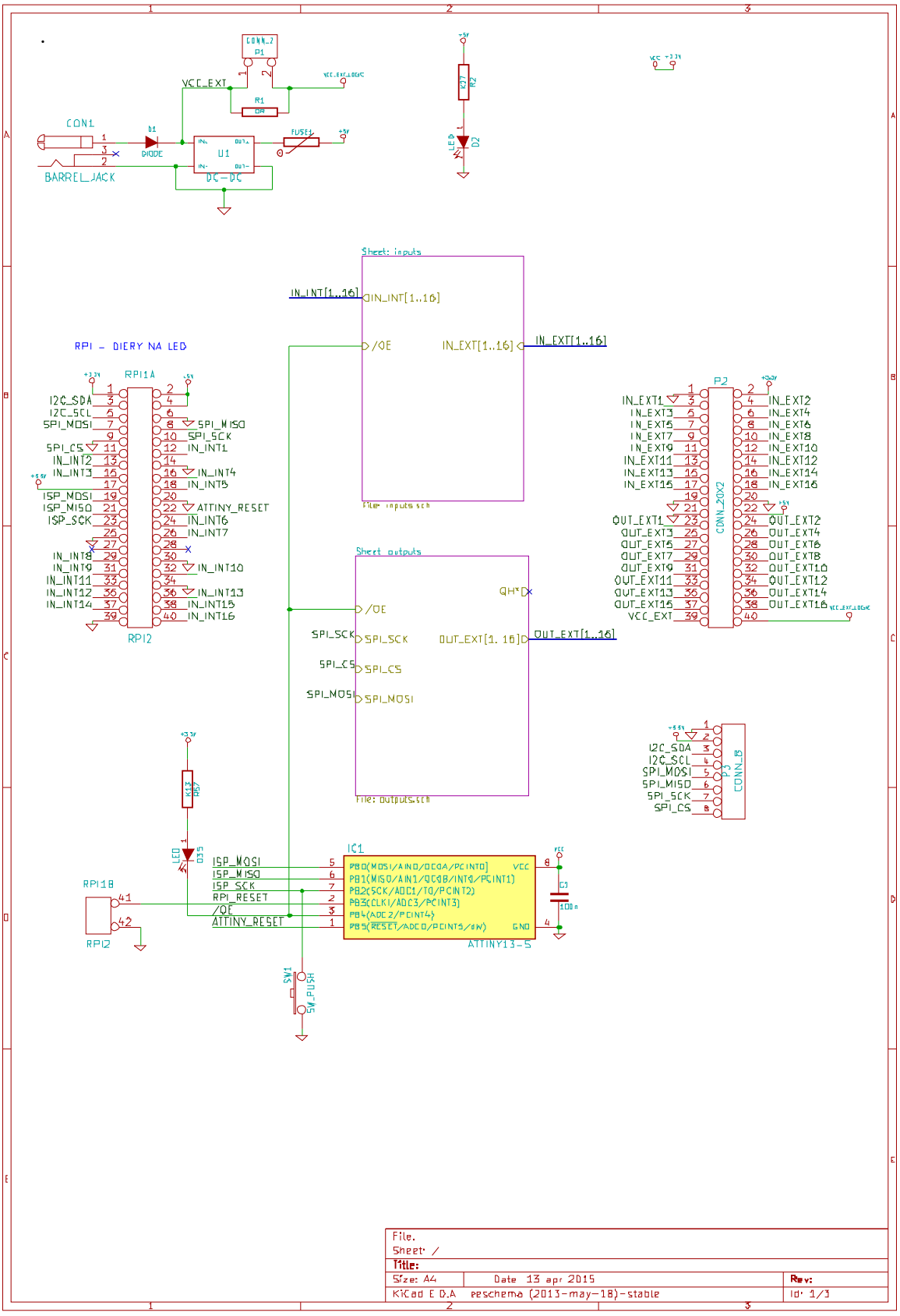
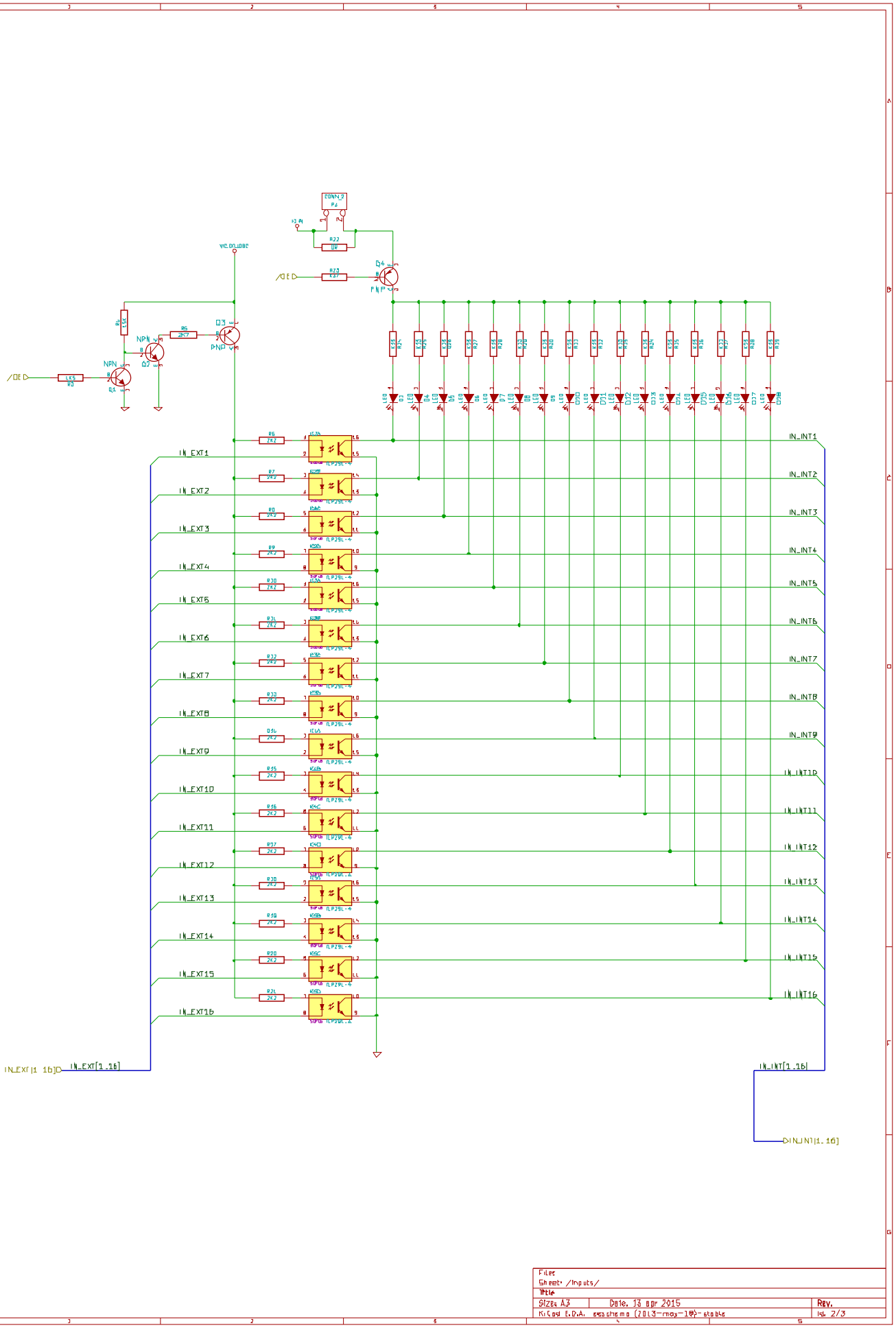
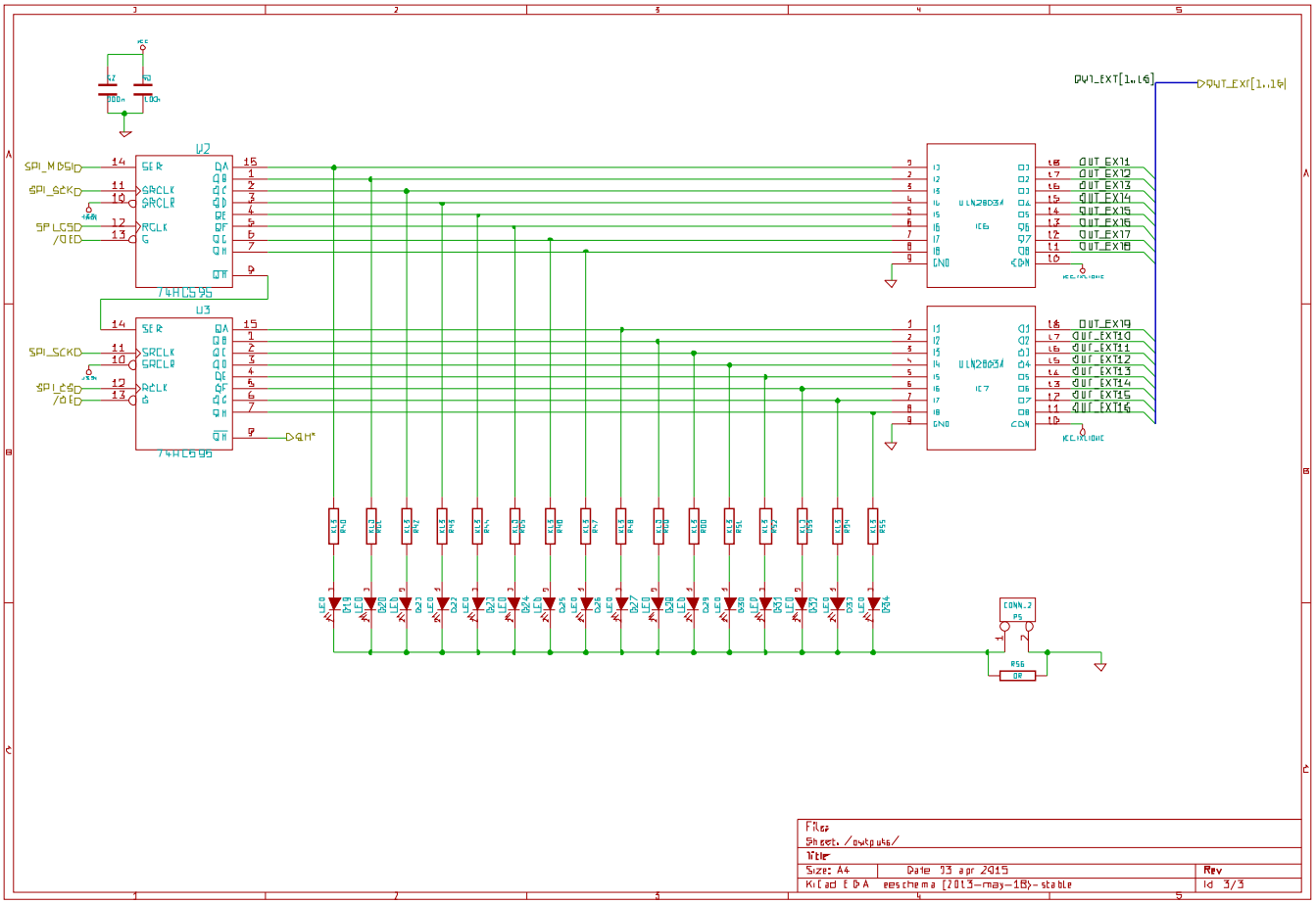
# A SCHÉMA A NÁVRH ROZŠÍRUJÚCEJ DOSKY PLOŠNÝCH SPOJOV

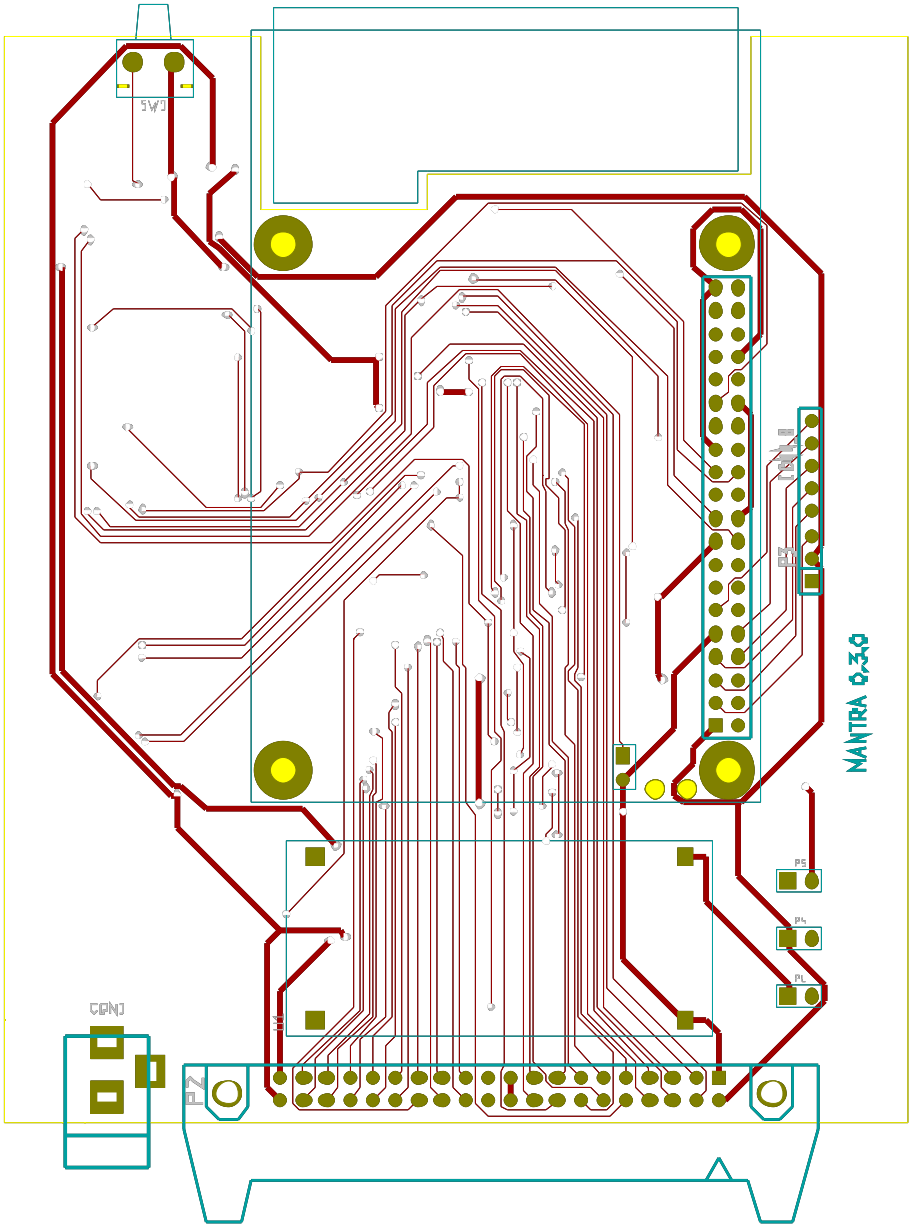



IN_EXT1
IN_EXT2
IN_EXT3
IN_EXT4
IN_EXT5
IN_EXT6
IN_EXT7
IN_EXT8
IN_EXT9
IN_EXT10
IN_EXT11
IN_EXT12
IN_EXT13
IN_EXT14
IN_EXT15
IN_EXT16
IN_INT1
IN_INT2
IN_INT3
IN_INT4
IN_INT5
IN_INT6
IN_INT7
IN_INT8
IN_INT9
IN_INT10
IN_INT11
IN_INT12
IN_INT13
IN_INT14
IN_INT15
IN_INT16
IN_EXT[1..16]
IN_INT[1..16]
/OE
NPN
PNP
ILP29L-4
Filer
Sheet: /Inputs/
Title
Size: A3
Date: 13 apr 2015
Rev.
KiCad E.D.A. eeschema (2013-may-18) - stable
Id: 2/3


OUT_EXT[1..16]
SPI_MOSI
SPI_SCK
SPI_LCS
/OE
SRCLK
SRCLR
RCLK
G
SER
74HC595
U2
U3
ULN2803A
IC6
IC7
GND
COM
OUT_EX11
OUT_EX12
OUT_EX13
OUT_EX14
OUT_EX15
OUT_EX16
OUT_EX17
OUT_EX18
OUT_EX19
OUT_EX10
OUT_EX11
OUT_EX12
OUT_EX13
OUT_EX14
OUT_EX15
OUT_EX16
CONN_2
P5
R56
0R
LED
Date: 23 apr 2015
Size: A4
Rev
Id: 3/3
Sheet: /output6/
File:
Title:
KiCad E.D.A. eeschema (2013-may-18) - stable

# **Návrh** DPS

## Vrchné vrstvy



## Spodné vrstvy

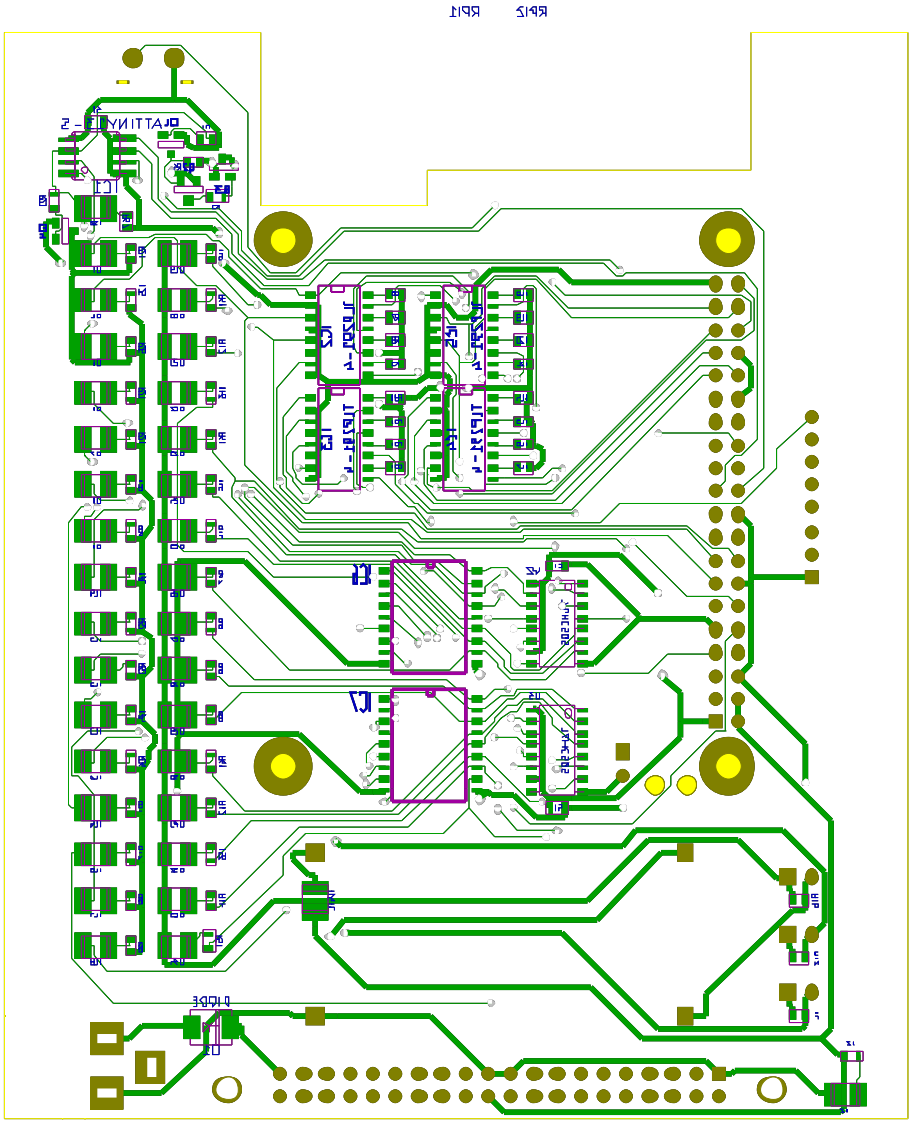
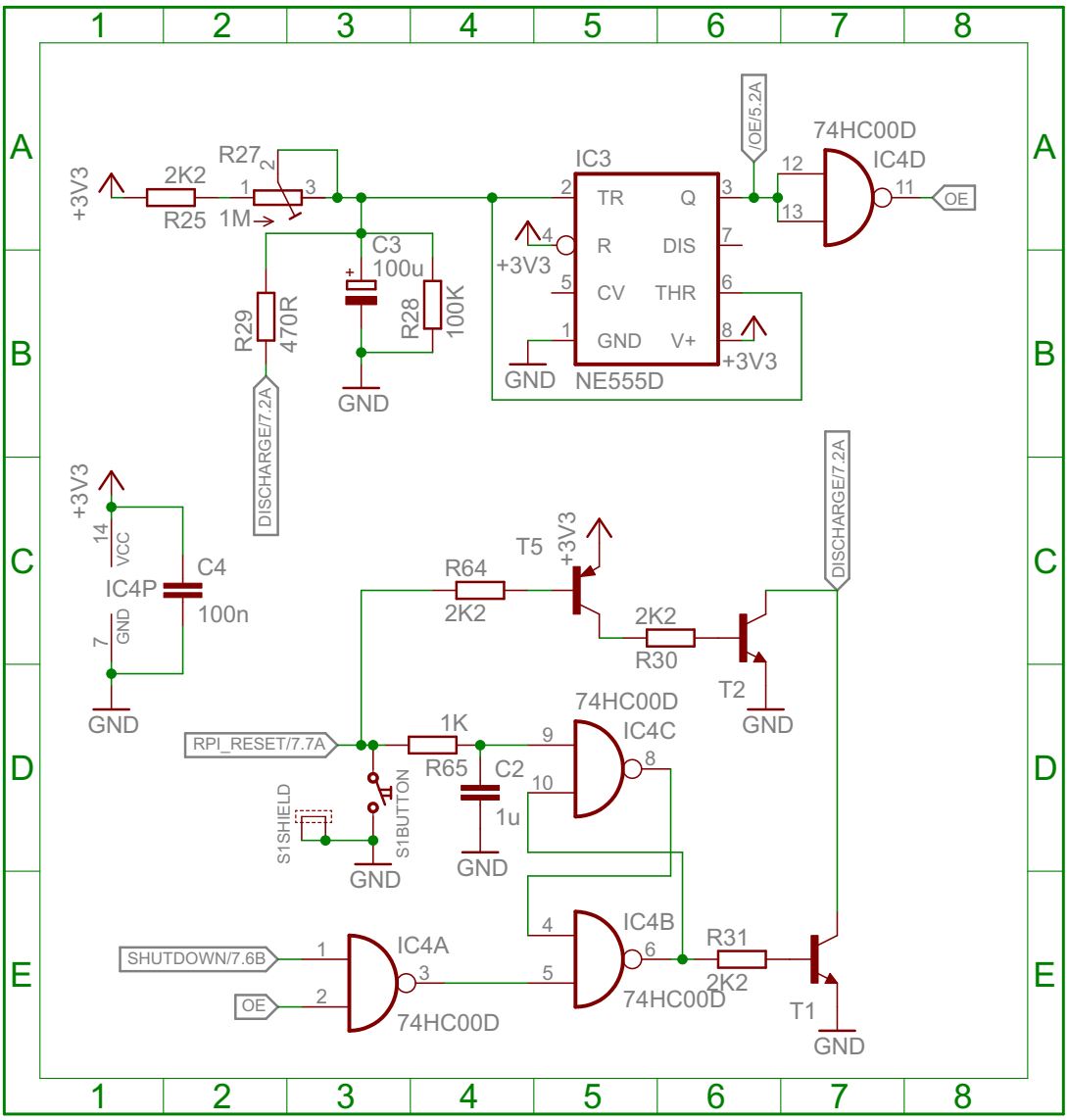


**Schéma prvej verzie resetovacieho obvodu - bez ATtiny**



# B DOKUMENTÁCIA KNIŽNICE PRE RIADIACU JEDNOTKU

Dokumentácia je vygenerovaná programom *Sphinx*, ktorého prioritou je generovať dokumentáciu pre kód v jazyku *Python* vo formáte *HTML*. *Sphinx* dokáže generovať aj vo formáte *PDF*, čo som využil pre túto prácu, výstup však nie je dokonalý. Pre krajšiu dokumentáciu odporúčam vygenerovať si *HTML* dokumentáciu.

Dokumentáciu som písal v angličtine, keďže je určená pre medzinárodnú firmu.

CHAPTER

ONE

# PACKAGES

## 1.1 terminal

Package defines a class *Terminal*, high-level interface to graphical terminal with buttons and leds

### 1.1.1 terminal.terminal

**class** terminal.terminal.**Terminal**(*port*)

Object representing the graphical terminal. Terminal has three general-use windows declared by *WinDefinable.define_win()*:

- **main_win**: whole screen, small font, top-left aligned
- **status**: top part of the screen, big font, middle aligned
- **message**: center of the screen, small font, top-left aligned

**Parameters** **port** (str) – path to the graphical terminal's serial interface character device file

**Variables** ***leds*** – list of *Led* indicators on the graphical terminal

**button_handler**(*button*)

Button event handler registration decorator

**Parameters** **button** (str) – name of the event as described *here*

**clear_button_handler**(*button*, *fn*)

**Parameters**

- **button** (str) – name of the event passed as the *button_handler()*'s button parameter before
- **fn** (function) – function decorated by the *button_handler()* before

**close**()

release resources

**fonts**

Names of all fonts known to the *Terminal*

**reload_fonts**()

Reload fonts from res/fonts

### 1.1.2 terminal.WinDefinable

Meta-mixin for prettier `Window` definition in `Terminal`.

**class** `terminal.WinDefinable.`**`WinDefinable`**

**`define_win`**(*name*, *opts*)
Adds a new attribute to the class from which WinDefinable.define_win was called. Attribute represents a *`Window`* instance, which is loaded lazily, because its constructor needs a terminal as an argument, and the terminal is not yet created when defining windows.

Assigning to the attribute assigns to the *`Window`* instance's text property.

**Parameters**

- **`name`** (`str`) – name of the created attribute
- **`opts`** (`dict`) – kwargs passed to the Window constructor

**class** `terminal.WinDefinable.`**`_WinDescriptor`**(*win*, *opts*)
Descriptor for creating terminal window attributes as described in *`WinDefinable.define_win()`*

### 1.1.3 terminal.OkNg

**class** `terminal.OkNg.`**`OkNg`**
mixes the *`ok_ng()`* method into *`Terminal`*.

**`ok_ng`**(*message*, *stopwatch=None*)
Display the screen with a message and two options OK and NG. Wait until the user presses one of them.

**Parameters**

- **`message`** (`str`) – message to be displayed
- **`stopwatch`** (*`Stopwatch`*) – If given, function uses *`Stopwatch.wait_event()`* method to wait for the OK/NG choice, therefore it can be interrupted by the stopwatch.

### 1.1.4 terminal.ValueSelector

**class** `terminal.ValueSelector.`**`ValueSelector`**
mixes *`selector()`* method into `Terminal`.

**`selector`**(*message*, *values*, *selected*, *stopwatch=None*)
Display the message and let the user select from list of values by pressing arrow keys. Actual selected value is displayed big and in the middle, neighbouring values (if exist) are displayed small, to the left and right of the selected value. User confirms the selection by pressing the OK button.

**Parameters**

- **`message`** (`str`) – message to be displayed
- **`values`** (`list` of `str`) – list of values from which the user selects
- **`selected`** (`int` or `str`) – index of the initially selected value or the value itself
- **`stopwatch`** (*`Stopwatch`*) – If given, function uses *`Stopwatch.wait_event()`* method to wait for the OK button, therefore it can be interrupted by the stopwatch.

**Returns**selected item or None if it was interrupted by the stopwatch

### 1.1.5 terminal.NumberSelector

**class** terminal.NumberSelector.**NumberSelector**
mixes *number_selector()* method into Terminal.

**number_selector**(*message*, *start*, *step*, *stopwatch=None*)
Display the message and let the user select from list of numbers by pressing arrow keys. Actual selected number is displayed big and in the middle. To the left and right of the selected number there is a number smaller and bigger respectively by given step. Neighbouring numbers are displayed small. User confirms the selection by pressing the OK button.

**Parameters**

- **message** (str) – message to be displayed
- **start** (float) – initially selected number
- **step** (float) – step by which the user increments/decrements the selected number
- **stopwatch** (*Stopwatch*) – If given, function uses *Stopwatch.wait_event()* method to wait for the OK button, therefore it can be interrupted by the stopwatch.

### 1.1.6 terminal.Menu

**class** terminal.Menu.**Menu**
mixes *menu()* method into Terminal. This mixin needs to be initialized to generate the list of windows for menu items.

**menu**(*title*, *items*, *selected=0*, *stopwatch=None*)
Display a vertical menu with given title and index, user can move through the menu with up/down arrow keys and confirm the selection by pressing the OK button. Screen moves with cursor only if there is a new item to display that direction.

**Parameters**

- **title** (str) – title displayed above the menu.
- **items** (list of str) – list of menu items
- **selected** (int or str) – index of the initially selected item or the item itself
- **stopwatch** (*Stopwatch*) – If given, function uses *Stopwatch.wait_event()* method to wait for the OK button, therefore it can be interrupted by the stopwatch.

**Returns**selected item or None if it was interrupted by the stopwatch

### 1.1.7 terminal.Pieces

**class** terminal.Pieces.**Pieces**
mixes *pieces* and *pieces_done* property into Terminal.

**pieces**
setter draws the pieces load bar with given number of slots

**pieces_done**
setter fills the given part of the pieces load bar

## 1.2 terminal.matrix_orbital

Low-level interface to the Matrix Orbital graphical terminal

### 1.2.1 terminal.matrix_orbital.matrix_orbital

**class** terminal.matrix_orbital.matrix_orbital.**MatrixOrbital**(*port*)

Abstractiion of the serial commands from Matrix Orbital documentation into more human-friendly methods.

**Parameters** **port** (str) – path to the graphical terminal's serial interface character device file

**close**()

stop *ButtonListener* and close the serial port

**fonts**

list of fonts

**rectangle**(*rect*, *mode='frame'*)

draw a rectangle

**Parameters**

- **rect** (((int, int), (int, int))) – position and size of the rectangle
- **mode** – 'frame' to draw only borders of rectangle, 'filled' to fill with foreground color, 'rubber' to fill with background color

**reload_fonts**()

Reload fonts from res/fonts

**reset**()

send the reset command and wait while the grephical terminal resets

**scroll**

global font scrolling (overriden by scrolling of individual windows)

**set_font_metrics**(***args*)

**Parameters** **args** – See below

**Keyword Arguments**

- line_margin
- top_margin
- char_space
- line_space
- scroll

**window**(*opts*)

define new *Window* in the terminal

**Parameters** **opts** (dict with str keys, see *Window*) – window options passed as **opts to *Window* constructor

**Returns** the created window

**Return type** *Window*

### 1.2.2 terminal.matrix_orbital.window

**class** terminal.matrix_orbital.window.**Window**(*term*, *rect=((0, 0), (0, 0))*, *font=None*, *alignment=('left', 'top')*)

Window is a frame in Matrix Orbital graphical terminal for which a font is defined individually; text can be aligned left, right and middle according to the borders of its window; text wraps in its window. Window doesn't use the window commands built into the Matrix Orbital graphical terminal.

**Parameters**

- **term** (*MatrixOrbital*) – terminal in which the window is created
- **rect** (((int, int), (int, int))) – position and size of the window
- **font** (str) – name of the font used in the window
- **alignment** (tuple (str, str), see *alignment*) – text alignment

**alignment**
text alignment

**Possible values**tuple ('left' or 'right' or 'center', 'top' or 'bottom' or 'center')

**clear**()
clear the window and set its text to ''

**font**
name of the text font currently used in the window

**go_to**(*pos*)
set terminal cursor to the position relative to the position of the window

**Parameters****pos** (tuple (int, int)) – cursor position

**height**
window height

**rewrite**()
rewrite the text in the window

**size**
window size

**Type**tuple (int, int)

**text**
text currently displayed in the window

**Type**str

**width**
window width

### 1.2.3 terminal.matrix_orbital.buttons

Referring to the documentation of the Matrix Orbital graphical terminal, pressing and releasing of the buttons is indicated by terminal's asynchronous write of a single character to the serial line. Every button event is mapped to a unique character in the terminal's memory. BUTTON_MAPPING is a mapping from these characters to human readable event names.

#### Known button event names:

- button_aux_up
- button_up
- button_right
- button_left
- button_ok
- button_aux_down
- button_down
- button_aux_up_released
- button_up_released
- button_right_released
- button_left_released
- button_ok_released
- button_aux_down_released
- button_down_release

**class** terminal.matrix_orbital.buttons.**ButtonListener**(*ser*, *mode='down_up'*)
A StoppableThread reading the serial line and calling handler corresponding to the character read. Handlers are functions with no arguments stored in the handlers dictionary.

**Parameters**

- **ser** (serial.Serial) – pySerial object representing the serial line to read from
- **mode** (str) – initial mode, see *mode*

**Variables** **handlers** – A mapping from *event names* to lists of functions with no arguments that will be all called after the event occurs

**type** dict { str : list [function with no parameters]}

**mode**
Listening mode

**Type** str

**Possible Values**

- ***auto_repeat*: after holding the button longer, event is being repeated,** *_released* handlers are not supported
- *down_up*: *_released* handlers are supported
- *down*: *_released* handlers are not supported

**new_handlers**()
clear all handlers

### 1.2.4 terminal.matrix_orbital.leds

**class** terminal.matrix_orbital.leds.**Led**(*ser*, *number*)
Matrix Orbital graphical terminal is equipped also with LEDs, which are abstracted by instances of this class.

**Parameters**

- **ser** (serial.Serial) – pySerial object representing the graphical terminal's serial line
- **number** (int) – identification number of the LED

**color**
LED color

**Possible values**

- LED_OFF
- LED_GREEN`
- LED_RED
- LED_YELLOW

### 1.2.5 terminal.matrix_orbital.buttons

Module for uploading fonts into Matrix Orbital graphical terminal. Fonts are located in res/fonts/ directory. Font is represented as a directory, the name of which is the name of the font. The directory contains a bitmap for every single character named N.BMP, where N is the decimal position of the character in ASCII table (e.g. 100.BMP for 'd').

**class** terminal.matrix_orbital.fonts.**Font**(*name*, *idn*)
Font name, id and metrics. Main use of this class is in Window class for switching fonts and text alignment.

**Parameters**

- **name** (int) – font name
- **idn** – font ID number

**height**
character height

**width**
character width - only fixed fonts are supported, however Matrix Orbital graphical terminal supports proportional fonts too

terminal.matrix_orbital.fonts.**character_size**(*font*)
open the first character image in the font's directory and get it's dimensions

terminal.matrix_orbital.fonts.**fonts**()

**Returns**list of all fonts, IDs are assigned by enumerating the font names sorted in ascending order

**Return type***Font*

terminal.matrix_orbital.fonts.**load_all_fonts**(*ser*)
upload all fonts to the graphical terminal

**Parameters****ser** (serial.Serial) – pySerial object representing the graphical terminal's serial line

terminal.matrix_orbital.fonts.**font_file**(*font*)
create a font bytestring in format accepted by the Matrix Orbital graphical terminal

**Parameters****font** (str) – font name

**Return type**bytes

terminal.matrix_orbital.fonts.**char_file**(*path*, *size*)
create a single character bytestring for the *font_file()* function

**Parameters**

- **path** (str) – path to the character's bitmap image file
- **size** (int) – size of every character in the font

**Return type** bytes

terminal.matrix_orbital.fonts.**upload_file**(*ser*, *font_id*, *bs*)
upload a file into the graphical terminal

**Parameters**

- **font_id** (int) – ID of the font
- **bs** (bytes) – file bytestring to upload

## 1.3 interpreter

Package defines a function *run()* which interprets a script generated by blockly. In addition it defines also a class *Stopwatch*, which is exported only for bin/terminal_demo, which doesn't use the interpreter (interpreter uses *Stopwatch* internally).

### 1.3.1 interpreter.interpreter

interpreter.interpreter.**run**(*prog*, *term*, *out_spi*)
initialize *Stopwatch* and execute the script with the globals returned by the *lib()* function

**Parameters**

- **prog** (str) – script to be interpreted
- **term** (*Terminal*) – terminal for user interaction

### 1.3.2 interpreter.commands

**class** interpreter.commands.**MantraRuntimeError**
exception raised on NG response

**class** interpreter.commands.**MantraManualInterrupt**
exception raised when the ERROR option is chosen in the interrupt menu

interpreter.commands.**lib**(*term*, *stopwatch*, *out_spi*)
Create closures used as a library in interpreted scripts.

**Parameters**

- **term** (*Terminal*) – graphical terminal used by the closures
- **stopwatch** (*Stopwatch*) – *Stopwatch* used in the closures for user to be able to interrupt the waiting for the events

**Returns** list of generated closures plus the exceptions defined in this module

**Variables** **export** – list of returned objects

**Decorators** **exported**

decorated function will be added to the export list

**Exported closures** procedure_def

define a procedure

**Parameters**

- **name** (str) – procedure name
- **flags (set of str) – two flags are accepted:**
  - displayed – procedure will be displayed in the interrupt menu

**procedure_call**

call a procedure defined by procedure_def

**Parametersname** (str) – procedure name

**shared_procedure_call**

call a shared procedure with given ID

**Parametersname** (str) – shared procedure ID

**select_number**

calls *NumberSelector.number_selector()* with adjusted parameters and stopwatch

**Parameters**

- **start** (float)
- **step** (float)
- **message** (str) – message to be displayed

**select_value**

calls *ValueSelector.selector()* with adjusted parameters and stopwatch

**Parameters**

- **message** (str) – message to be displayed
- **default** (str)
- **values** (str)

**scan_barcode**

read a line of standard input interruptibly by stopwatch

**Parameters**

- **message** (str) – message to be displayed

**manual_inspection**

prompt user for OK/NG response

**Parameters**

- **message** (str) – message to be displayed

**wait_pin**

wait while all given pins are set to their respective values (not implemented yet)

**Parameters**

- **message** (str) – message to be displayed

•**pins** (:class:dict {`int` : `bool`}) – requested values mapped to their pins

**set_pin**

set given pins to their respective values

**Parameters**

•**pins** (`dict` {`int` : `bool`}) – values to be set mapped to their pins

**blink_pin**

start blinking with given frequencies on given pins

**Parameters**

•**pins** (`dict` {`int` : `float`}) – blinking frequencies mapped to the corresponding pins

**assert_pin**

throw *MantraRuntimeError* if pins doesn't match corresponging values

**Parameters**

•**pins** (`dict` {`int` : `bool`}) – values to be matched mapped to their pins

**delay**

display a message and sleep some time; cannot be interrupted by `Stopwatch`

**Parameters**

•**time** (`float`) – time to sleep in seconds

**save_process_result**

send OK/NG result of a process identified by **identifier** to database server

**Parameters**

•**identifier** (`str`) – process identifier

•**value** (`bool`) – OK/NG process result

**get_control_plan**

get data from database for **check_cp_list** and **check_cp_set** commands

**check_cp_list**

check all control plan entries (tool/component barcodes) in fixed order

**check_cp_set**

check all control plan entries (tool/component barcodes) in free order

**verify_as_serial_number**

query database for OK/NG result of supplied arguments; throw *MantraRuntimeError* if the result was NG

**Parameters**

•**supplier**

•**rm_sn**

•**value**

**verify_as_bom**

query database for OK/NG result of supplied arguments; throw *MantraRuntimeError* if the result was NG

**Parameters**

**•value**

**verify_as_operator**

query database for OK/NG result of supplied arguments; throw *MantraRuntimeError* if the result was NG

**Parameters**

**•value**

**verify_as_fg_box_id**

query database for OK/NG result of supplied arguments; throw *MantraRuntimeError* if the result was NG

**Parameters**

**•box_id**

**•sn**

**verify_as_before_process**

query database for OK/NG result of supplied arguments; throw *MantraRuntimeError* if the result was NG

**Parameters**

**•identifiers**

**•value**

**verify_as_control_plan**

query database for OK/NG result of supplied arguments; throw *MantraRuntimeError* if the result was NG

**Parameters**

**•identifier**

**•value**

**verify_as_string**

if the **value** doesn't match the **regex**, throw *MantraRuntimeError*

**Parameters**

**•regex**

**•value**

**save_process_checksheet**

log supplied arguments to the database

**Parameters**

**•item**

**•value**

**save_outgoing_inspection_checksheet**

log supplied arguments to the database

**Parameters**

- **item**
- **value**

**save_processing_log**

log supplied arguments to the database

**Parameters**

- **wip_id**
- **item**
- **value**

**save_product_processing_data**

log supplied arguments to the database

**Parameters**

- **wip_id**
- **item**
- **value**

**save_product_verification_data**

log supplied arguments to the database

**Parameters**

- **wip_id**
- **item**
- **value**

**save_prooduct_rework_data**

log supplied arguments to the database

**Parameters**

- **wip_id**
- **item**
- **value**

**save_lot**

log supplied arguments to the database

**Parameters**

- **part_number**
- **lot_number**

### 1.3.3 interpreter.Stopwatch

**class** interpreter.Stopwatch.**Stopwatch**(*term*, *button='button_ok'*)

*Stopwatch* measures time of holding the OK button. After 4 seconds interrupt menu is displayed, where one can choose from all procedures in procedures list and an EXIT option. There is one additional dummy procedure residing in the procedures list initially called RETURN, which does nothing and also procedure NG which throws MantraRuntimeError.

*Stopwatch* should be used by calling either the *wait_bool()* or the *wait_event()* method.

**Parameters**

- **term** (*Terminal*) – graphical terminal, to which the *Stopwatch* binds
- **button** (str) – button event name, to which the *Stopwatch* listens

**stop**()

clear button handlers unless already cleared

**stopped**

**Returns** was the EXIT option chosen?

**Return type** bool

**wait_bool**(*success_fn*, *redraw*)

wait until the success_fn returns True, but if an interrupt occurs, call confirm_prompt() and if the EXIT option was chosen don't wait anymore

**Parameters**

- **success_fn** (function with no arguments, returning bool) – function that is cyclically called until it returns True
- **redraw** (function with no arguments) – function that restores the screen after returning from the interrupt menu

**Returns** True if terminated by the event, False if terminated by the EXIT choice in the interrupt menu

**wait_event**(*event*, *redraw*)

wait for an event, but if an interrupt occurs, call confirm_prompt() and if the EXIT option was chosen don't wait anymore

**Parameters**

- **event** (threading.Event) – event to wait for
- **redraw** (function with no arguments) – function that restores the screen after returning from the interrupt menu

**Returns** True if terminated by the event, False if terminated by the EXIT choice in the interrupt menu

# C DOKUMENTÁCIA PROGRAMU PRE RESETOVACÍ OBVOD

Dokumentácia je vygnerovaná programom *Doxygen*. Keďže je *Doxygen* oveľa vyzrelejší program, ako *Sphinx*, nemal problém vygenerovať peknú dokumentáciu vo formáte *PDF*.

Dokumentáciu som písal v angličtine, takisto ako dokumentáciu v prílohe B.

# Chapter 1

# Main Page

Welcome to the documentation for program stored in MANTRA Reset circuit's ATtiny microcontroller. Reset circuit handles disabling of inputs and outputs when RPi is in shutdown state. After startup, RPi sends a message to the ATtiny, that it has just been initialized. ATtiny should then enable outputs by setting the OE_NON pin low. ATtiny holds OE_NON low until it receives a message from RPi that it's going to shut down.

Besides this behaviour ATtiny handles a power button. The exact mapping of actions to times of hold of the button is not yet determined, but it should be able to send RPi a message to shut down or reboot and it should be also able to reboot the RPi forcibly by toggling the RPI_RESET pin.

# Chapter 4

# Class Documentation

## 4.1 stopwatch Struct Reference

stopwatch for time measurement

`#include <stopwatch.h>`

### Public Attributes

- uint8_t start

  *Here stopwatch stores the starting value of the timer register TCNT0.*
- uint8_t counter

  *Here stopwatch counts how many timer register overflows passed.*
- uint8_t running

  *flag indicating whether the stopwatch runs or is stopped*

### 4.1.1 Detailed Description

stopwatch for time measurement

The documentation for this struct was generated from the following file:

- stopwatch.h

# Chapter 5

# File Documentation

## 5.1 communication.c File Reference

see communication.h

```
#include <util/delay.h>
#include <avr/io.h>
#include "communication.h"
#include "stopwatch.h"
#include "pins.h"
```

### Macros

- #define **START** 2
- #define **STOP** 3
- #define **ERROR** 4
- #define **BIT_POS** (1<<((edge - 3)/2))

### Functions

- uint8_t mark (uint8_t level)

  *translate a one logical level into mark according to the time passed from the last call of this function*
- void comm_in_change (uint8_t value)

  *This function should always be called after the change of logical level on the receiving pin.*
- void communication_tick (void)

  *Calls stopwatch_tick() for the receiver's internal stopwatch.*
- void rectangle (unsigned char type)

  *send a given mark*
- void comm_write (uint8_t byte)

  *Send data through the communication link (transmitting pin), function blocks.*

### Variables

- struct stopwatch **comm_in_stopwatch**
- volatile uint8_t rpi_state = RPI_OFF

  *in this variable there's always the last byte received from RPi*

### 5.1.1 Detailed Description

see communication.h

### 5.1.2 Function Documentation

#### 5.1.2.1 void comm_in_change ( uint8_t *value* )

This function should always be called after the change of logical level on the receiving pin.

This is the base function for receiving. It translates the incoming signal's edge times (with use of stopwatch) to bytes and stores them into rpi_state.

**Parameters**

| | |
|---|---|
| *value* | logical level on the receiving pin after the change |

#### 5.1.2.2 void comm_write ( uint8_t *byte* )

Send data through the communication link (transmitting pin), function blocks.

**Parameters**

| | |
|---|---|
| *byte* | of data to be sent |

#### 5.1.2.3 uint8_t mark ( uint8_t *level* )

translate a one logical level into mark according to the time passed from the last call of this function

This function is called twice for every mark. The first time for the high level, the second time for the low level. If the mark is OK, it should return the same results for the two levels.

**Parameters**

| | |
|---|---|
| *level* | needed for the translation (e. g. signal for mark BIT_1 should remain cca 85ms in high and cca 25ms in low level) |

## 5.2 communication.h File Reference

communication with RPi with custom bit-banging protocol

```
#include <stdint.h>
```

### Macros

- #define **RPI_OFF** 0
- #define **RPI_ON** 1
- #define **START_ACK** 0
- #define **SHUTDOWN** 1

### Functions

- void comm_in_change (uint8_t value)

*This function should always be called after the change of logical level on the receiving pin.*

- void communication_tick (void)

  *Calls stopwatch_tick() for the receiver's internal stopwatch.*

- void comm_write (uint8_t byte)

  *Send data through the communication link (transmitting pin), function blocks.*

### Variables

- volatile uint8_t rpi_state

  *in this variable there's always the last byte received from RPi*

### 5.2.1 Detailed Description

communication with RPi with custom bit-banging protocol

### 5.2.2 Function Documentation

#### 5.2.2.1 void comm_in_change ( uint8_t *value* )

This function should always be called after the change of logical level on the receiving pin.

This is the base function for receiving. It translates the incoming signal's edge times (with use of stopwatch) to bytes and stores them into rpi_state.

**Parameters**

| | |
|---|---|
| *value* | logical level on the receiving pin after the change |

#### 5.2.2.2 void comm_write ( uint8_t *byte* )

Send data through the communication link (transmitting pin), function blocks.

**Parameters**

| | |
|---|---|
| *byte* | of data to be sent |

## 5.3 main.c File Reference

Initialization, global interrupt handlers and main.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <util/atomic.h>
#include "pins.h"
#include "power_button.h"
#include "communication.h"
#include "stopwatch.h"
```

### Functions

- void init (void)

*Global initialization.*

- ISR (TIM0_OVF_vect)

  *Send ticks to stopwatches for powerboutton and communication receiver after every timer overflow.*

- ISR (PCINT0_vect)

  *Handle external interrupts.*

- int main (void)

### 5.3.1 Detailed Description

Initialization, global interrupt handlers and main.

### 5.3.2 Function Documentation

#### 5.3.2.1 void init ( void )

Global initialization.

- setup GPIO
- setup timer for stopwatch ticks
- setup external interrupts for power button and communication receiver

#### 5.3.2.2 ISR ( PCINT0_vect )

Handle external interrupts.

Find out which external interrupt pin changed its value, run corresponding change function with the new state of the pin. Power button's pin has inverse polarity to the state of button.

#### 5.3.2.3 int main ( void )

Put it all together. When the Raspberry Pi is off, wait for start message or power button click (toggle reset of Raspberry Pi after power button click to start it up). When the Raspberry Pi is on, wait for stop message or power button click (send shutdown message after power button click), disable I/O after that.

## 5.4 pins.h File Reference

Wiring definitions on PB.

### Macros

- #define **COMM_IN** 0
- #define **COMM_OUT** 1
- #define **PWR_BUTTON** 2
- #define **RPI_RESET** 3
- #define **OE_NON** 4

### 5.4.1 Detailed Description

Wiring definitions on PB.

## 5.5 power_button.h File Reference

handling power button events according to the time of holding

`#include <stdint.h>`

### Macros

- #define **PWR_BUTTON_NOTHING** 0
- #define **PWR_BUTTON_CLICKED** 1

### Functions

- void power_button_change (uint8_t pressed)

  *Handle the change of the power button's state.*
- void power_button_tick (void)

  *stopwatch_tick() for the power button handler's internal stopwatch*

### Variables

- volatile unsigned int power_button

#### 5.5.1 Detailed Description

handling power button events according to the time of holding

#### 5.5.2 Variable Documentation

##### 5.5.2.1 volatile unsigned int power_button

this variable is initialized to PWR_BUTTON_NOTHING and after every press and after every click of power button (at least 10ms long) it is set to PWR_BUTTON_CLICKED

## 5.6 stopwatch.h File Reference

stopwatch for time measurement

`#include <stdint.h>`

### Classes

- struct stopwatch

  *stopwatch for time measurement*