Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

DIPLOMOVÁ PRÁCE

DESKY

PLZEŇ, 2013 Stanislav Šimek

Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

Diplomová práce

Implementace a testování ovladače systému REX pro platformu Raspberry Pi a Gert Board

PLZEŇ, 2013 Stanislav Šimek

P R O H L Á Š E N Í

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni, dne 30. 8. 2013 _____

Stanislav Šimek

Poděkování

V první řadě bych chtěl poděkovat Ing. Ondřeji Ježkovi za vedení práce a cenné rady. Dále bych chtěl poděkovat Ing. Janu Kovandovi za stylistické připomínky a Ing. Jiřímu Žahourovi za obstarání a pomoc s měřicími přístroji.

Abstrakt

Na začátku této diplomové práce je popsán mikropočítač Raspberry Pi a možnosti vytváření programů komunikující s jeho periferiemi. Dále je popsaná rozšiřující karta pro Raspberry Pi - Gert Board. Jsou popsány její části a způsoby připojení k Raspberry Pi.

Následuje popis řídicího systému REX a ovladače RPiDrv, který umožňuje REXu bežícímu na Raspberry Pi komunikaci s jeho periferiemi. Poté je popsán program RPiCfg, který slouží pro vytvoření konfiguračního souboru pro REX. V konfiguračním souboru jsou uloženy upřesňující nastavení jednotlivých periférií. Následně je popsáno použití obou programů na jednoduchých modelech.

Na konci práce jsou testovány Raspberry Pi a ovladač RPiDrv. Testy jsou zaměřeny na rychlost komunikace mezi REXem a periferiémi Raspoberry Pi. Následně je Raspberry Pi připojeno k reálným modelům. Je otestována vazba mezi REXem a ovladačem RPiDrv a správné nastavení periférií pomocí RPiCfg.

Klíčová slova: Raspberry Pi, Gert Board, REX, testování, RPiCfg, RPiDrv, periférie

Abstract

At the beginning of this master thesis a microcomputer Raspberry Pi and options for creating programs with its peripherals are described. After that is described an extending card Gert Board for Raspberry Pi. Parts of Gert Board and methods of a connection with Raspberry Pi are described.

This is followed by a description of the control system REX and driver RPiDrv. The driver allows REX that runs on Raspberry Pi to communicate with Raspberry's peripherals. Next is described program RPiCfg which is used for creating a configuration file for REX. In the configuration file are saved detailed settings of individual peripherals. This is followed by a description of using both programs on simple models.

At the end of thesis Raspberry Pi and driver RPiDrv are tested. Tests are focused on speed of communication between REX and peripherals of Raspberry Pi. Further is Raspberry Pi connected to real models. The binding between REX and driver RPiDrv with correct settings of peripherals using the RPiCfg is tested.

Key words: Raspberry Pi, Gert Board, REX, testing, RPiCfg, RPiDrv, peripherals

Obsah

1 Úvod

Cílem předložené diplomové práce je představit platformu Raspberry Pi a vytvořit pro tuto platformu odpovídající ovladač. Tato platforma může být využita v praxi pro řízení průmyslových zařízení. Velkou výhodou platformy je nízká pořizovací cena (cca 400 - 900 Kč), což je v porovnání k existujícím ekvivalentům výrazné cenové snížení. Navíc tato platforma podporuje operační systém GNU/Linux. Implementace ovladače umožní propojit pokročilé řídicí algoritmy systému Rex přímo s řízenými procesy a laboratorními modely.

V diplomové práci budeme nejprve seznámeni s platformou Raspberry Pi. Dozvíme se něco o výhodách této platformy a také získáme několik informací o periferiích, které Raspberry Pi nabízí. Poté se naučíme vytvářet programy komunikující s periferiemi a budeme seznámeni s úskalími, které se vyskytují u této platformy. Mimo jiné budeme seznámeni s rozmístěním jednotlivých pinů, abychom byli schopni správně propojit Raspberry Pi s okolním světem.

Bude nám také představena rozšiřující karta Gert Board, jež přináší spousty nových prvků, se kterými může Raspberry Pi komunikovat a ovládat je. Dále nám bude ukázáno, jak správně propojit Raspberry Pi s deskou Gert Board a na co si dát případně pozor. Na jednotlivých příkladech nám bude vysvětleno, jak používat jednotlivé prvky karty Gert Board, jak je správně propojit a jakým stylem s nimi komunikovat z Raspberry Pi.

V další části diplomové práce budeme seznámeni s knihovnami RPiDrv, které umožňují řídicímu systému REX komunikovat s periferiemi na Raspberry Pi. Zároveň nám bude předveden program RPiCfg, který slouží k vytvoření konfiguračního souboru upřesňujícího vlastnosti knihovny RPiDrv. Budou popsány prvky, jež můžeme nastavit. Také bude vysvětlen jejich význam. S těmito znalostmi budeme schopni vytvořit funkční model v programu RexDraw využívající periférie Raspberry Pi.

Sérií testů bude odměřeno, jakých dosahuje Raspberry Pi latencí při komunikaci se svými periferiemi. Poznatky získané z těchto testů nám pomohou určit, kde v praxi by se následně dalo Raspberry Pi použít.

Poté bude Raspberry Pi připojeno k laboratorním modelům, díky čemuž bude možné ověřit funkčnost knihovny RPiDrv v praxi.

Na závěr práce budou prodiskutovány další možnosti připojení senzorů, jež by rozšířily použitelnost zmíněného Raspberry Pi.

2 Raspberry Pi

Co je vlastně Raspberry Pi? Raspberry Pi je miniaturní počítač o rozměrech 86 x 52 x 21 mm. Poháněn je mikroprocesorem BCM2835 (Broadcom 2013), což je takzvaný "system one a chip". Čip obsahuje výpočetní procesor, jádro pro zpracování obrazu, atd. (OpenGL, IPS, ...). Procesor je taktován na 700 MHz s možností přetaktování až k 1 GHz. Operační paměť dosahuje velikosti 512 MB. Raspberry Pi (dále jen RPi) je vybaveno HDMI a RCA portem pro připojení externích obrazovek, USB konektorem, síťovým konektorem (záleží na modelu RPi, *Model A* síťový konektor neobsahuje, *Model B* ano), audio konektorem Jack 3,5 mm a slotem pro připojení SD/MMC karty. Vše je naletováno na jednom plošném spoji. Hardwarovou konfiguraci tedy není možné měnit.

Raspberry Pi má několik výhod. Jednou z největších je bezesporu cena. Model A se prodává za méně než pět set korun (Model A, 256 MB paměti, 1xUSB) a cena modelu B se pohybuje okolo jednoho tisíce korun (Model B, 512 MB paměti, 2xUSB, 1xLAN). Velmi malé rozměry byly již uvedeny. RPi neobsahuje žádné pohyblivé části, což zvyšuje jeho odolnost vůči vibracím. Požadované napájení pro RPi je 5 V, není tedy problém použít jako zdroj elektrické energie bateriové články. U RPi není potřeba chladit elektrické součástky - nevyžaduje žádný chladič, ať už aktivní nebo pasivní.

RPi používá jako operační systém GNU/Linux (dále jen Linux). Je možné si vybrat z několika podporovaných distribucí (Debian, Arch Linux, …). Systém Windows toto zařízení zatím nepodporuje. Bude-li k RPi připojena klávesnice, myš a monitor, může sloužit jako plnohodnotný počítač. Výrobce uvádí, že výkon je dostatečný pro sledování videa ve Full HD kvalitě. Je možné využít i dalších funkcí operačního systému Linux. Mimo vlastnosti klasického počítače RPi obsahuje i různé periférie: vstupně výstupní piny, SPI, UART, I^2C, PWM a PCM

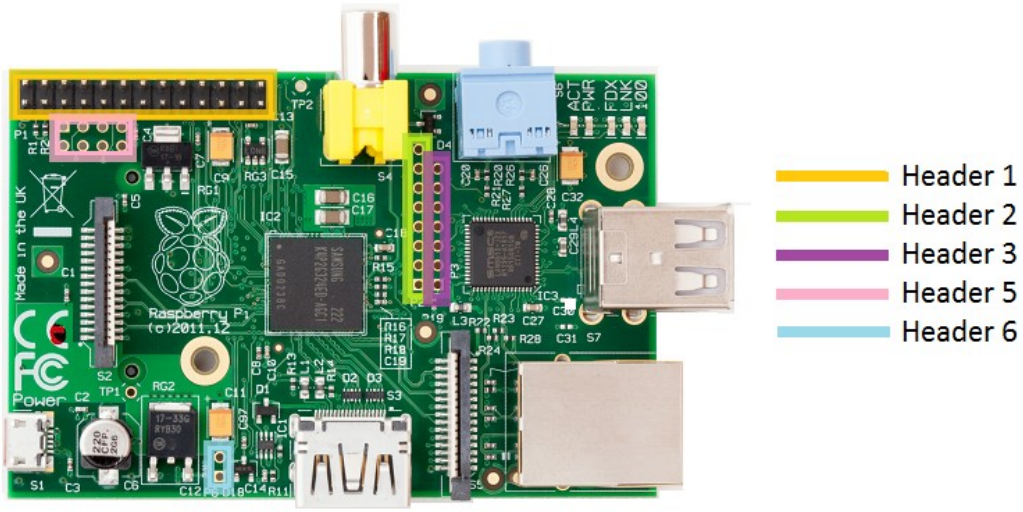
2.1 Vstupně výstupní piny

RPi je rozšířeno o několik vstupně výstupních pinů (General Purpose Input Output - GPIO). Každý GPIO může být nastaven jako:

- Vstupní pin, který je schopen detekovat, zda je k němu připojená logická 0, tedy napětí v rozsahu 0 - 0,8 V nebo logická 1 tj. napětí v rozsahu 2 - 3,3 V.
- Výstupní pin, který má schopnost nastavit svou hodnotu na logickou 0 o napětí 0 - 0,4 V nebo logickou 1 o napětí 2,4 - 3,3V.

- Alternativní funkce, mezi které patří UART, I^2C, SPI, PWM a PCM. Tyto funkce lze nastavit pomocí registrů. Více v kapitole 2.2. Kompletní seznam funkcí přiřaditelných pinům je v (Broadcom Corporation 2012: 102–103).

Rozložení GPIO na RPi je zobrazeno na Obr. 2.1.



Obr. 2.1 Rozložení pinů na Raspberry Pi

HEADER 1. Hlavní header Raspberry Pi, obsahuje 17 x GPIO, napájení a signálové země. Význam jednotlivých pinů je zobrazen v Tab. 2.1.

Power 3V3	Pin 01	Pin 02	Power 5V
GPIO 2	Pin 03	Pin 04	Power 5V
GPIO 3	Pin 05	Pin 06	Ground
GPIO 4	Pin 07	Pin 08	GPIO 14
Ground	Pin 09	Pin 10	GPIO 15
GPIO 17	Pin 11	Pin 12	GPIO 18
GPIO 27	Pin 13	Pin 14	Ground
GPIO 22	Pin 15	Pin 16	GPIO 23
Power 3V3	Pin 17	Pin 18	GPIO 24
GPI0 10	Pin 19	Pin 20	Ground
GPI0 9	Pin 21	Pin 22	GPIO 25
GPIO 11	Pin 23	Pin 24	GPIO 8
Ground	Pin 25	Pin 26	GPIO 7

Tab. 2.1 Význam pinů - header 1

HEADER 2. Tento header patří do skupiny JTAG (Join Test Action Group) a je využíván k testování RPi ve fázi výroby pro testování plošného spoje a pamětí. Rozložení pinů je zobrazeno v Tab. 2.2. Piny headeru nejsou na RPi standardně připájeny.

HEADER 3. Servisní header JTAG LAN9512. Rozložení pinů je zobrazeno v Tab. 2.3. Piny headeru nejsou na RPi standardně připájeny.

Ground	Pin 08
Ground	Pin 07
TCLK	Pin 06
TMS	Pin 05
TDO	Pin 04
TDI	Pin 03
nTRST	Pin 02
Power 3V3	Pin 01

Tab. 2.2 Význam pinů - header 2

Ground	Pin 07
Nepřipojen	Pin 06
TCLK	Pin 05
TMS	Pin 04
TDO	Pin 03
TDI	Pin 02
nTRST	Pin 01

Tab. 2.3 Význam pinů - header 3

HEADER 5. Rozšiřující header RPi. Přidává UART, I^2C nebo čtyři GPIO piny. Piny headeru nejsou na RPi standardně připájeny. Rozložení pinů je zobrazeno v Tab. 2.4.

HEADER 6. Header sloužící k restartu BCM2835. K restartu dojde propojením pinu P1 (RUN) se zemí (GND). Rozložení pinů je zobrazeno v Tab. 2.5.

Power 5V	Pin 01	Pin 02	Power 3V3
GPIO 28	Pin 03	Pin 04	GPIO 29
GPIO 30	Pin 05	Pin 06	GPIO 31
Ground	Pin 07	Pin 08	Ground

Tab. 2.4 Význam pinů - header 5

Ground	Pin 02
Run	Pin 01

Tab. 2.5 Význam pinů - header 6

2.2 Programování pro RPI

Programovat na RPi je možné v libovolných programovacích jazycích podporovaných operačním systémem Linux. Jelikož je potřeba při vytváření ovladače zajistit co nejvyšší rychlost a přistupovat na hardwarovou úroveň RPi, byl vybrán jazyk C/C++. RPi disponuje kompilátorem GCC/G++ se všemi standardními knihovnami. Periférie RPi jsou ovládány pomocí registrů popsaných v (Broadcom Corporation 2012: 4–202.) Registry jsou zpřístupněny na adresovém rozsahu 0x2000 0000 až 0x20FF FFFF (15 MB) ve fyzické paměti. Pro komunikaci s periferiemi BCM2835 je tedy nutné získat tento adresový prostor pod kontrolu.

2.2.1 Přístup k periferiím

Jak již bylo uvedeno, je potřeba získat přístup do adresového prostoru 0x2000 0000 až 0x20FF FFFF. Ideálním řešením by bylo vytvořit pointer s požadovanou adresou a přistupovat do paměti pomocí pointeru. Toto řešení však není možné, protože dnešní operační systémy včetně Linuxu nedovolují přistupovat tímto způsobem do adresového

prostoru, který nepatří danému programu. Linux však umožňuje přístup do paměti pomocí souboru "/dev/mem". Je tedy potřeba vytvořit spojení mezi naším programem a souborem *mem*. K tomu slouží funkce *mmap*. Tato funkce je schopna vzít soubor (nebo jeho část) a namapovat ho do paměti. Návratová hodnota této funkce je pointer na začátek namapovaného souboru. Chceme-li následně číst nebo zapisovat data ze souboru, je to možné udělat pomocí pointerů. Namapujeme-li soubor mem, konkrétně jeho část 0x2000 0000 až 0x20FF FFFF, získáme možnost číst a zapisovat data
do registrů BCM2835. Adresy jednotlivých registrů jsou uvedeny v (Broadcom Corporation 2012: 8–202).

2.2.2 Čtení a zápis z libovolného registru

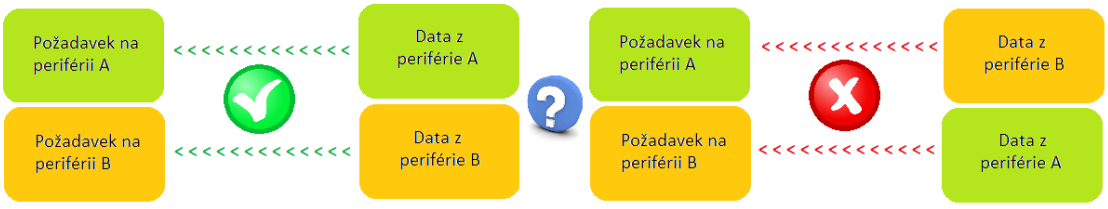
Proces čtení z jedné periférie:

V případě, že několikrát čteme z jedné periférie, nedochází v průběhu čtení k žádným problémům. Čtení probíhá následovně:

1. Je vytvořen požadavek na čtení dat z periférie.
2. Požadavek je zpracován obslužným systémem BCM2835.
3. Je navrácena požadovaná hodnota.

Proces čtení z dvou a více periférií:

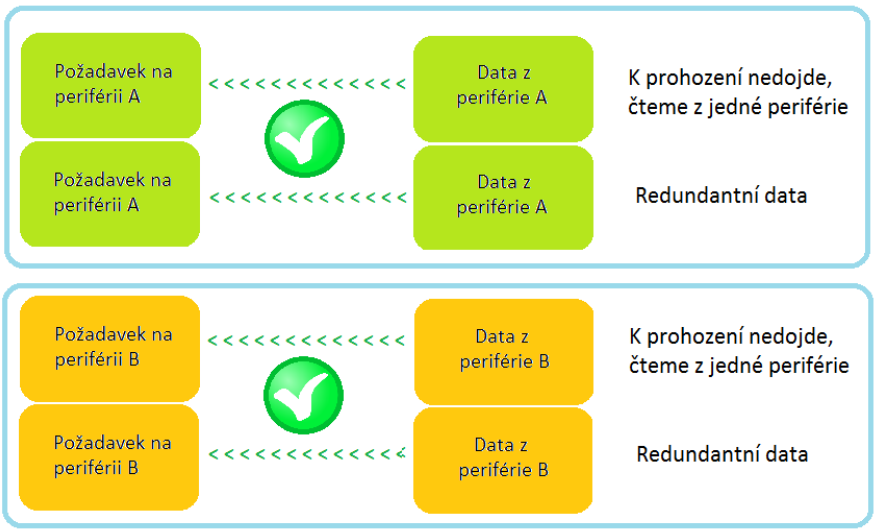
Může nastat následující problém. Přijde požadavek na čtení dat z *periferie A* a zároveň požadavek na čtení dat z *periférie B*. Dle očekávání by měla být navrácena data z *periférie A* následována daty z *periferie B*. To však není zaručeno (Broadcom Corporation 2012: 7). Není jisté, v jakém pořadí data přijdou. Zda přijdou ve správném, jak je znázorněno na Obr. 2.3Obr. vlevo, nebo zda přijdou prohozena, jak je znázorněno
na Obr. 2.3 vpravo. Je na uživateli, aby udělal příslušná opatření v případě čtení z více periférii.



Obr. 2.3 Nejistota pořadí navrácení dat při čtení z více periférii

V případě zápisu je situace obdobná. Jestliže je zapisováno pouze na jednu periférii, nedojde k žádnému problému. Za podmínky, že bude zapisováno na více periférií současně, mohou být zapisovaná data prohozena.

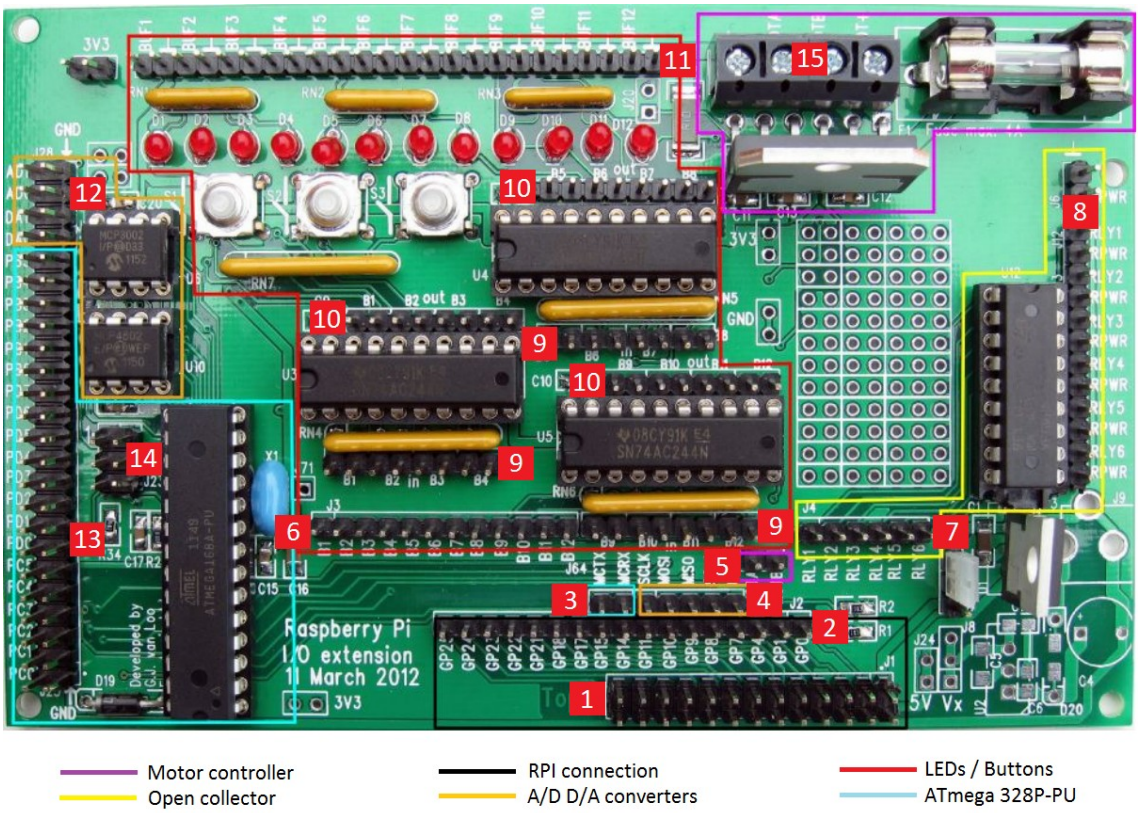
Řešením tohoto problému je zdvojení zapisovaných, případně čtených dat, jak je zobrazeno na Obr 2.4, jelikož při čtení z jedné periférie k prohození nedochází. Je zde však zvýšena režie zápisu a čtení, protože jedna hodnota je vždy redundantní.



Obr. 2.4 Ochrana před prohozením dat

3 Gert Board

Gert Board je karta rozšiřující RPi. Karta obsahuje LED diody, tlačítka, A/D a D/A převodníky, motor controller, ATmega328P-PU a několik otevřených kolektorů pro spínání externích obvodů. Rozmístění důležitých součástek a význam pinů je na Obr. 3.1. V následujících kapitolách budou popsány jednotlivé prvky Gert Board (dále jen GB). Nejprve je však potřeba připojit GB k RPi. To je realizováno spojením *header 1* na RPi a *header 1* na GB. Je nutné dávat pozor na správnou orientaci propojovacího kabelu, aby pin 01 na GB odpovídal pinu 01 na RPi.

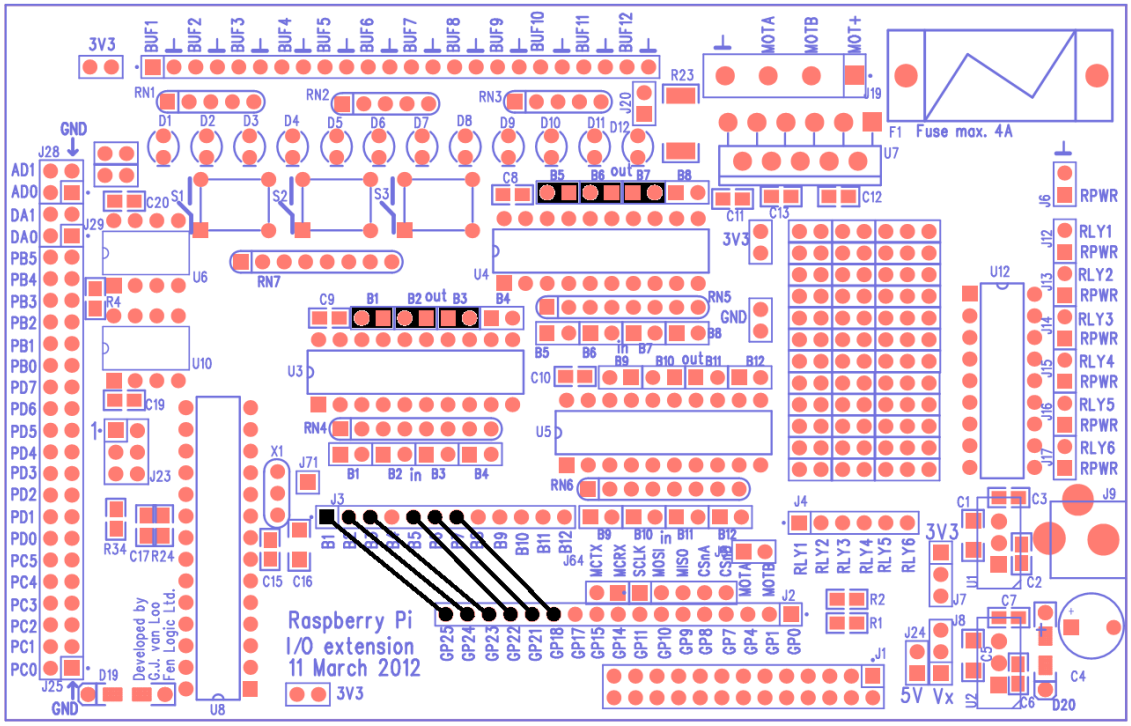


Obr. 3.1 Rozložení součástek a význam headeru na desce Gert Board

1. Propojení s RPi
2. Vyvedené GPIO piny
3. Piny UART k ATmega
4. Piny SPI k A/D D/A převodníkům
5. Piny k řízení PWM
6. Piny k diodám / tlačítkům
7. Piny k ovládání otevřeného kolektoru
8. Výstup otevřeného kolektoru
9. Vstupní svorky k tlačítkům / diodám
10. Výstupní svorky k tlačítkům / diodám
11. Ovládací piny k tlačítkům / diodám
12. Vstupy a výstupy A/D D/A převodníků
13. Vyvedené piny ATmega328P-PU
14. Piny k programování ATmega328P-PU
15. Výstup PWM

3.1 Tlačítka a LED diody

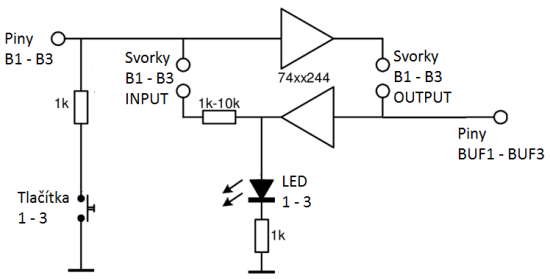
Na desce Gert Board se nachází dvanáct pinů označených BUF1-BUF12 (Obr. 3.2 nahoře). Tyto piny mohou být jak vstupní, tak výstupní. Záleží na tom, zda svorka spojuje B1-12 IN nebo B1-12 OUT (viz vnitřní zapojení Obr. 3.3, Obr. 3.4). LED diody zobrazují, zda je k pinům připojeným k B1-B12 (Obr. 3.2 header J3) přiváděno napětí, respektive zda piny dodávají napětí do obvodu. GB rovněž obsahuje tři tlačítka. Vnitřní zapojení obvodu s tlačítky je zobrazeno na Obr. 3.3.



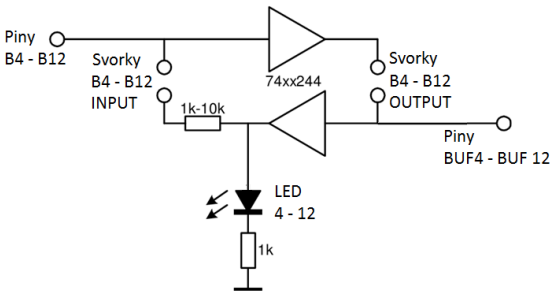
Obr. 3.2 Schéma zapojení tlačítek a LED diod

Testování tlačítek a diod na desce Gert Board bylo provedeno dle schématu na Obr. 3.2. GPIO 23, 24 a 25 byly nastaveny jako vstupy a GPIO 18, 22 a 27 jako výstupy. Testovací program měl následující funkci:

- Hodnota přečtena na vstupu GPIO 25 (obvod s tlačítkem 1) je zapsána na výstup GPIO 22 (LED dioda 5).
- Hodnota přečtena na vstupu GPIO 24 (obvod s tlačítkem 2) je zapsána na výstup GPIO 22 (LED dioda 6).
- Hodnota přečtena na vstupu GPIO 23 (obvod s tlačítkem 3) je zapsána na výstup GPIO 22 (LED dioda 7).



Obr. 3.3 Vnitřní zapojení BUF1 - BUF3

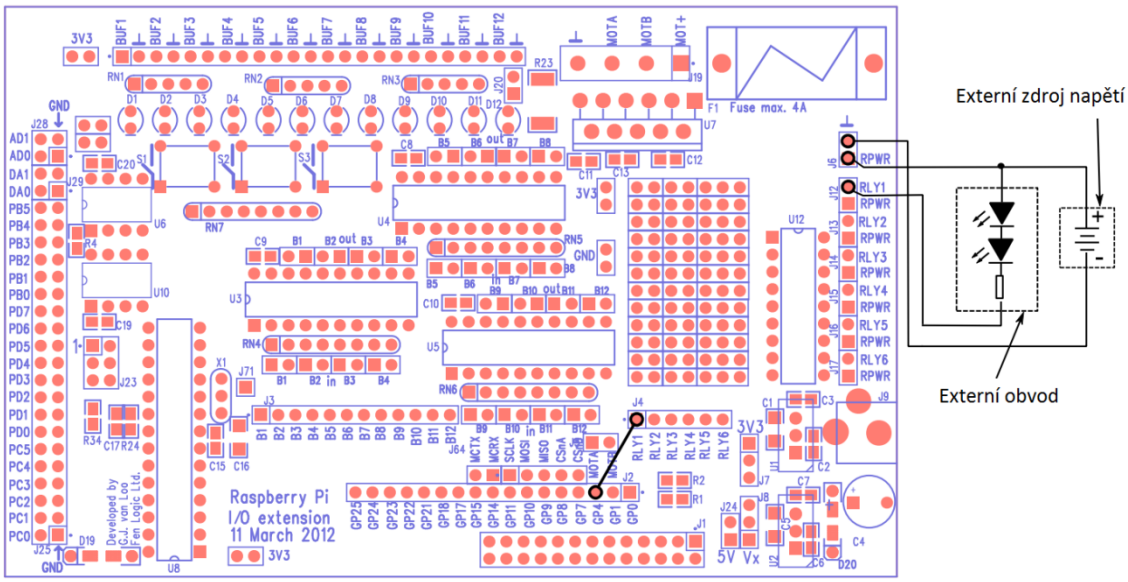


Obr. 3.4 Vnitřní zapojení BUF4 - BUF12

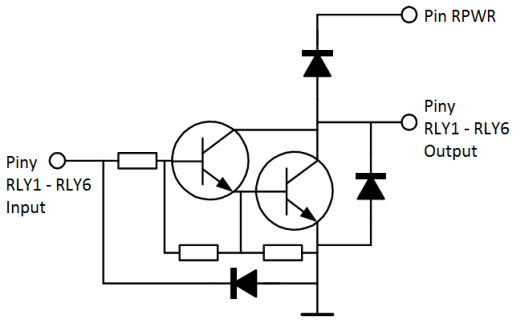
Z důvodu snížení množství propojovacích vodičů, měly vstupní piny připojen *pull up* rezistor a byly spojeny svorky B1-B3 OUT. Nebude-li tlačítko stisknuto, objeví se na pinu vlivem *pull up* rezistoru logická 1. Stiskneme-li tlačítko, napětí proteče do země a na pinu bude logická 0. Pokud by byly spojeny svorky B1-B3 IN, bylo by potřeba na piny BUF1-BUF3 přivést napětí logické 1.

3.2 Otevřený kolektor

GB obsahuje šest otevřených kolektorů, které se používají pro spínání externích elektrických obvodů. Konkrétně se jedná o součástku ULN2803a (ULN2803a 1997), která umožňuje sepnout obvod, ve kterém je napětí až 50 voltů a proud 500 mA. K ovládání otevřeného kolektoru stačí nastavit jeden z pinů jako výstupní a propojit ho dle schématu Obr. 3.5. V našem případě byl zvolen GPIO 4. Logická 1 na pinu RLY1 header J1 obvod sepne, logická 0 rozepne. Vnitřní zapojení je zobrazeno na Obr. 3.6.



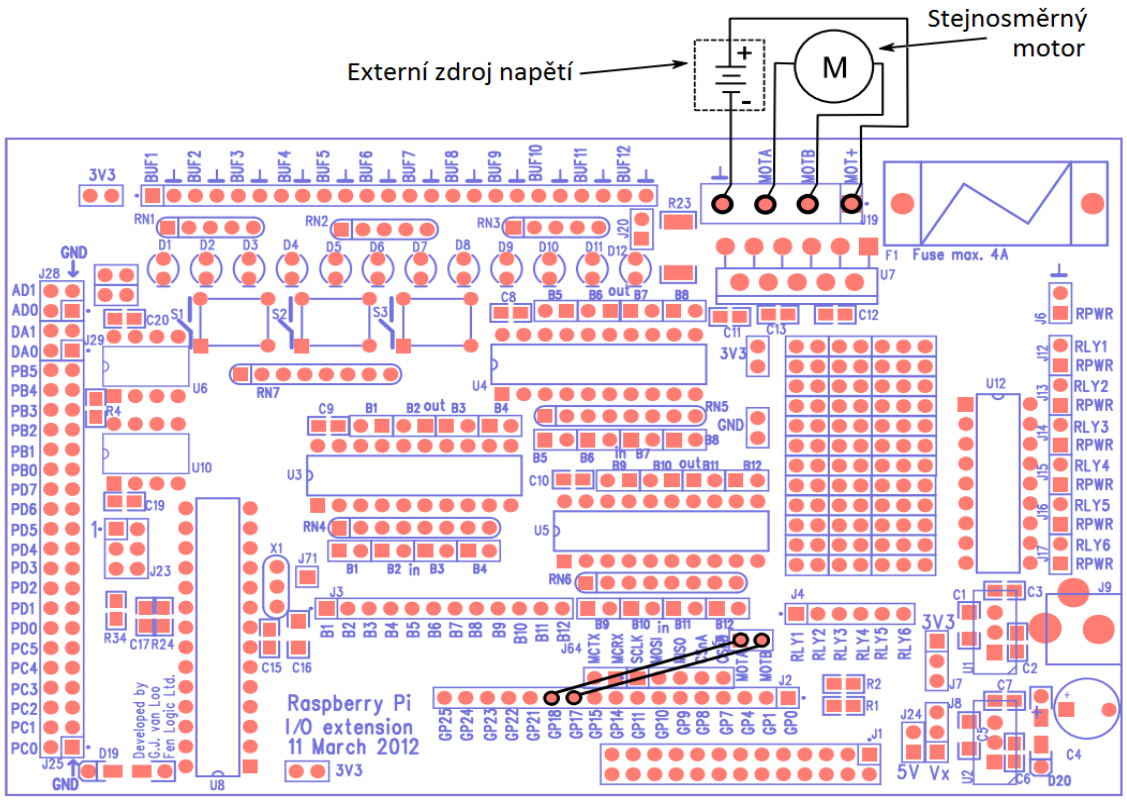
Obr. 3.5 Schéma zapojení otevřeného kolektoru



Obr. 3.6 Vnitřní zapojení otevřeného kolektoru

3.3 PWM

Další součástí GB je součástka L6203 (L6203 1997). Součástka přivádí napětí z externího zdroje připojeného ke svorkám MOT+ a ⊥ (Obr. 3.7 nahoře) na svorky MOTA a MOTB (Obr. 3.7 nahoře). Ovládací piny jsou označeny rovněž MOTA, MOTB na Obr. 3.7 uprostřed. Maximální hodnota napětí externího zdroje je 48 V a maximální dodávaný proud součástkou je 4 A (chvilkově až 5 A). Součástka slouží převážně k řízení motorů pomocí PWM. Zdrojem PWM je na RPi pin GPIO 18, alternativní funkce 5. Druhý pin může být libovolně zvolen. Musí být však nastaven jako výstupní.



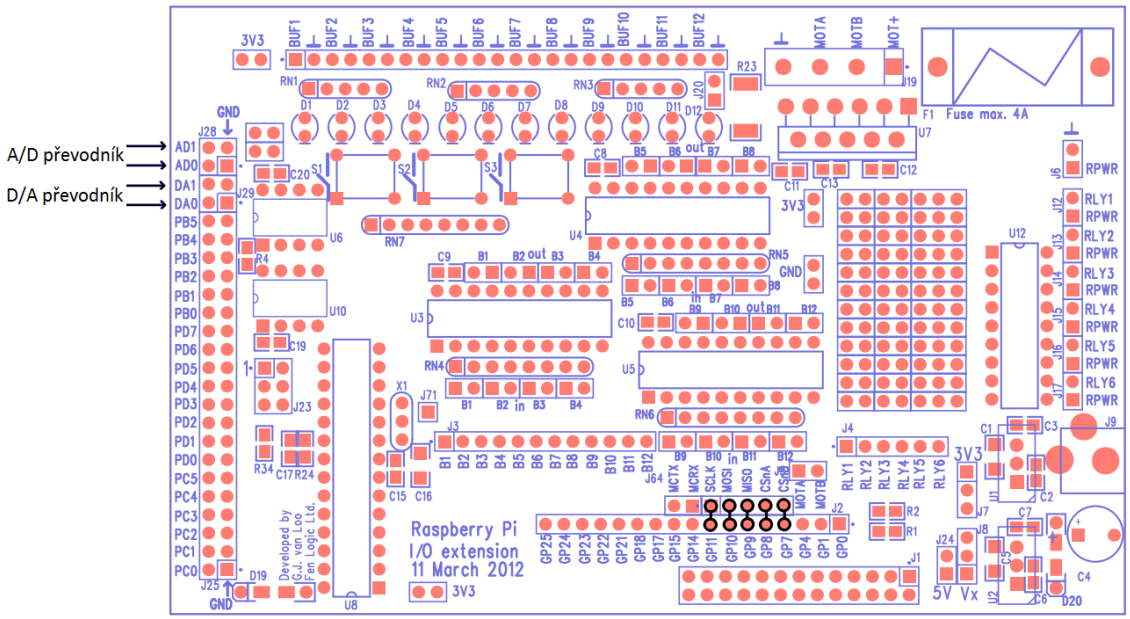
Obr. 3.7 Schéma zapojení PWM

3.4 A/D D/A převodníky

Mezi další rozšíření karty Gert Board patří A/D a D/A převodníky. Převodníky komunikují s RPi pomocí SPI sběrnice. Je nutné použít následující piny (alternativní funkce 0):

- GPIO 7 Chip select 1 Připojen D/A převodník (Na GB CSnB)
- GPIO 8 Chip select 0 Připojen A/D převodník (Na GB CSnA)
- GPIO 9 Master In Slave Out
- GPIO 10 Master Out Slave In
- GPIO 11 Serial clock

Schéma zapojení je zobrazeno na Obr. 3.8.



Obr. 3.8 Schéma zapojení A/D, D/A převodníky

3.4.1 A/D převodník

A/D převodník je reprezentován čipem MCP3002 (MCP3002 2007). Jedná se o dvoukanálový desetibitový převodník. Abychom získali z A/D převodníku hodnotu napětí, je na něj potřeba přenést následujících 16 bitů. Jejich význam je zobrazen v Tab. 3.1.

	Byte 1								Byte 2							
bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Označení	Start bit	SGL/DIFF	ODD/SIGN	MSBF	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx

Start bit	Udává začátek sekvence, musí být nastaven vždy na 1
SGL/DIFF	Určuje mód přenosu 1 = Single ended mode, 0 pseudo-differencial mode
ODD/SIGN	Pokud je bit SGL/DIFF roven 1, tak se tímto bitem volí kanál
	Pokud je bit SGL/DIFF roven 0, tak tak se určuje polarita kanálů
MSBF	Určuje bitové pořadí 1 = MSB , 0 = LSB
xxx	Nepodstatné bity

Tab. 3.1 Rozmístění 16 bitů odesílaných na A/D převodník

Během odesílání bitů na A/D převodník jsou z něj zároveň bity přijímány. Jejich rozložení a význam je zobrazen v Tab. 3.2.

	Byte 1								Byte 2							
bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Označení	xxx	xxx	xxx	xxx	xxx	NULL	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0

B0 -B9	Jednotlivé bity z A/D převodníku
xxx	Nepodstatné bity

Tab. 3.2 Rozmístění 16bitu přijímaných z A/D převodníků

Hodnota A/D převodníku je z přijatých bajtů získána:

```
hodnota = ((byte1 & 0x03) << 8) | byte2
```

Operací `(byte1 & 0x03)` jsou odstraněny bezvýznamné bity v prvním bajtu, které mohou nabývat libovolné hodnoty. Dále je potřeba přesunout nejvyšší dva bity na správnou pozici (bylo použito MSB) bitovým posuvem a tento výsledek spojit s nižšími osmi bity. Hodnota navrácená A/D převodníkem se pohybuje v rozmezí 0 až 1023, kde hodnota 0 odpovídá 0V a hodnota 1023 odpovídá 3,3V. Napětí na A/D převodníku z přijaté hodnoty získáme dle vzorce:

$$\text{Výsledné U} = \frac{\text{Naměřená hodnota}}{1023} * 3{,}3\text{V}$$

3.4.2 D/A převodník

D/A převodník je reprezentován čipem MCP4802 (Microchip Technology Inc. 2010: 1–5, 8–11, 13–17). Jedná se o dvoukanálový převodník s rozlišením 8 bitů. Aby byl převodník nastaven na požadovanou hodnotu napětí, je na něj potřeba odeslat 2 x 8 bitů. Význam bitů je zobrazen Tab. 3.3.

	Byte 1								Byte 2							
bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Označení	A/B	xxx	GA	SHDN	B7	B6	B5	B4	B3	B2	B1	B0	xxx	xxx	xxx	xxx

A/B	Výběr kanálu
GA	Zesílení signálu, 1 = 1x tj max výstup 2,048V 0 = 2x tj 4,094V
SHDN	Odstavení převodníku z činosti
Data 0-7	Hodnota D/A převodníku
xxx	Nepodstatné bity

Tab. 3.3 Význam 16 bitů odesílaných na D/A převodník

Na převodník odesíláme hodnotu 0 - 255, kde hodnota 0 odpovídá 0V a hodnota 255 odpovídá 2,048V. Výsledná hodnota napětí je dána vztahem:

$$\text{Výstupní napětí [V]} = \frac{\text{Odeslaná hodnota}}{255} * 2{,}048\text{V}$$

Zároveň s odesíláním bitů na D/A převodník jsou z D/A převodníku bity přijímány. Význam bitů je zobrazen v Tab. 3.4.

	Byte 1								Byte 2							
bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Označení	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx

xxx	Nepodstatné bity

Tab. 3.4 Rozmístění 16 bitu přijímaných z D/A převodníků

Přijaté bity nemají žádný význam, avšak je nutné je přečíst, aby nezůstávaly v SPI frontě a nebyly mylně přečteny jinou funkcí.

3.5 ATmega 328P-PU

Gerdboard obsahuje mikrokontrolér typu ATmega328P-PU. Jedná se o 8bitový mikrokontrolér o taktu 20 MHz s 32KB pamětí. ATmega dále obsahuje:

- 2x osmibitový čítač/časovač
- 1x čítač reálného času
- 6x PWM
- 14x deseti bitový A/D převodník
- UATR
- SPI master/slave
- I^2C
- analogový komparátor
- a mnoho dalších funkcí (Atmel Corporation 2012: 1–650).

3.5.1 Instalace softwaru

K programování ATmega pomocí RPi je nejprve nutné nainstalovat nějaký programovací nástroj. V našem případě bylo použito Arduino IDE, které lze nalézt v repozitáři Linuxu. Stačí tedy použít příkaz:

```
sudo apt-get install arduino
```

Nyní je třeba nainstalovat soubory upravující Arduino IDE pro RPi. Toho bude dosaženo použitím níže zobrazených příkazů. První tři příkazy nainstalují upravenou verzi Arduino IDE a druhé tři příkazy nastaví potřebné parametry.

```
wget http://project-downloads.drogon.net/Gert Board/avrdude_5.10-4_armhf.deb

sudo dpkg -i avrdude_5.10-4_armhf.deb

sudo chmod 4755 /usr/bin/avrdude

wget http://project-downloads.drogon.net/Gert Board/setup.sh

chmod +x setup.sh

sudo ./setup.sh

zdroj: (Gordons Projects 2013).
```

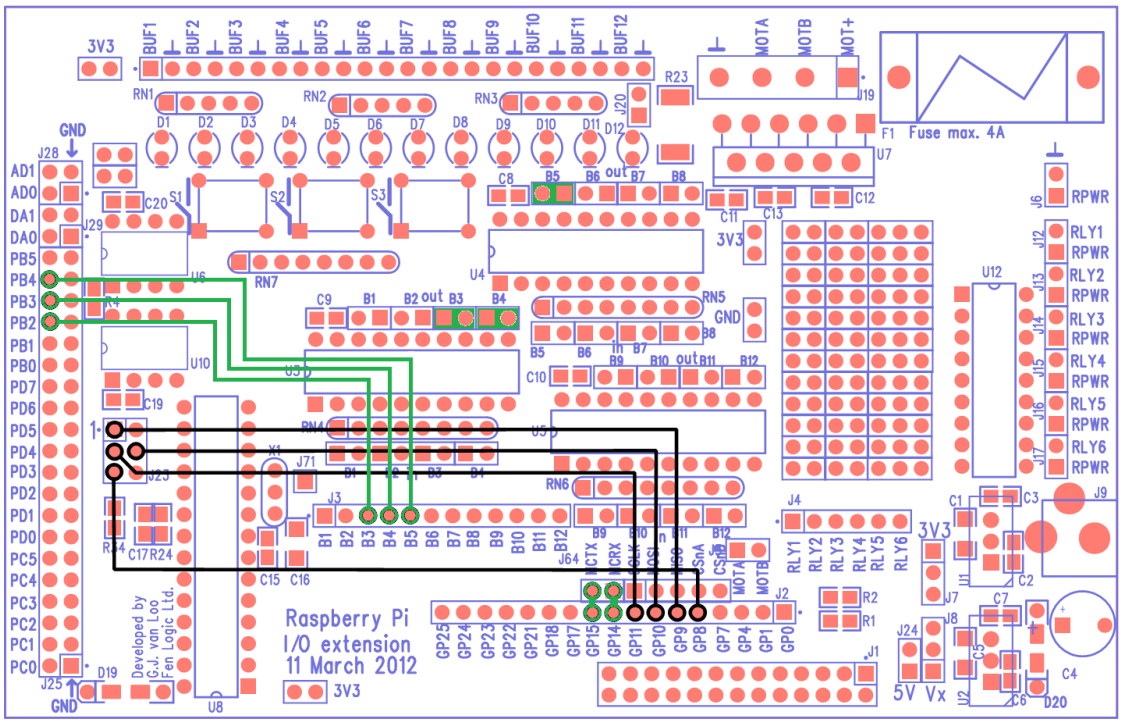
Po instalaci těchto souborů a restartu RPi je téměř vše připraveno k programovaní ATmega. Ještě je třeba zvolit typ čipu. To je provedeno příkazem:

```
avrsetup
```

A nyní již můžeme spustit Arduino IDE na RPi. V *Tools* položka *Board* je potřeba nastavit, že používáme *Gert Board with ATmega328* a v *Tools* položka *Programmer* nastavit *Raspberry PI GPIO*. Poté nám již nic nestojí v cestě a můžeme programovat ATmega. Všechny kroky k instalaci jsou k nahlédnutí na stránkách projektu Drogon[1].

3.5.2 Programování ATmega

ATmega se programuje SPI sběrnicí, propojení je zobrazeno na Obr. 3.9 - černé propojení.



Obr. 3.9 ATmega, černě: propojení pro programování, zeleně: příklad programu ATmega

Jakmile je napsán příslušný program, je třeba ho přesunout na mikrokontroler použitím Arduino IDE. Použijeme-li program Arduino IDE, jsou základem programu funkce *setup()* a *loop()*. Funkce *setup()* je volána pouze jednou, a to na začátku běhu programu. Obvykle slouží k inicializaci pinů. Po dokončení funkce *setup()* následuje funkce *loop()*. Obsah funkce je opakován cyklicky, dokud není ATmega odpojeno od elektrického zdroje. Program je psán v jazyku C. Syntaxe a možné použité funkce jsou vypsány na

[1] https://projects.drogon.net/raspberry-pi/Gert Board/arduino-ide-installation-isp/
https://projects.drogon.net/raspberry-pi/Gert Board/using-the-arduino-ide/
https://projects.drogon.net/raspberry-pi/Gert Board/Gert Board-atmega-io-vs-arduino-pins/.

webových stránkách[2]. Ukázkový příklad je zobrazen v Zdroj. kód 3.1 a konkrétní zapojení k níže napsanému kódu je na Obr. 3.9 - zelené propojení.

3.5.3 Ukázka programu ATmega

```
/* Inicializace globálních proměnných */
int tlacitko = 10;                          //PB2
int pwm = 11;                               //PB3
int led = 12;                               //PB4
unsigned char tmp;

/* Funkce volaná na začátku běhu programu */
void setup(){
      pinMode(tlacitko, INPUT_PULLUP);      //Nastaví pin jako vstup s PU
      pinMode(led,OUTPUT);                  //Nastaví pin jako výstup
      pinMode(pwm,OUTPUT);                  //Nastaví pin jako výstup
      Serial.begin(9600);                   //Nastavení UART (rychlost)
}

/* Periodicky se opakující funkce */
void loop() {
      tmp = Serial.read();          // Přečte byte z UART fronty
      analogWrite(pwm,tmp);         // Zapíše na pin pomocí PWM
      tmp = digitalRead(tlacitko);  // Přečte hodnotu z pinu
      digitalWrite(led,tmp);        // Zapíše hodnotu na pin
      delay(1);                     // Čeká 1 ms
}
```

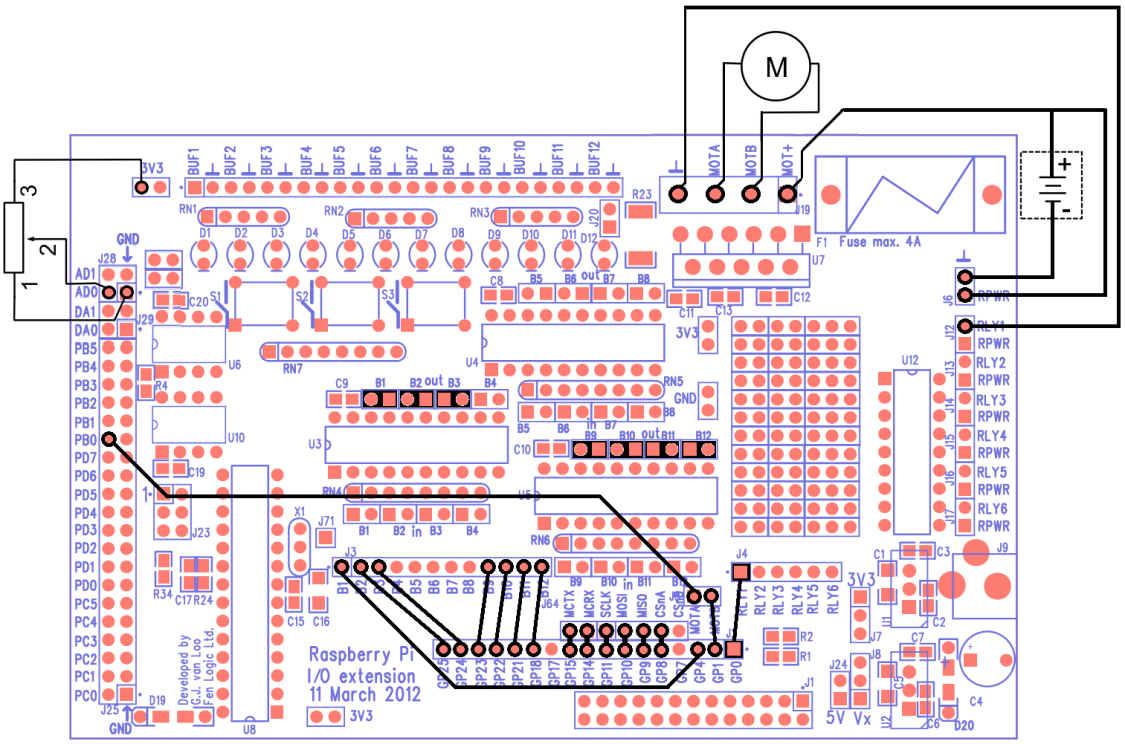
Zdroj. kód 3.1 ATmega příklad

3.6 Testovácí program Gert Board

Pro desku Gert Board byl napsán program v jazyce C/C++ demonstrující její vlastnosti. Zapojení je zobrazeno na Obr. 3.10. Funkce programu byly následující:

- Z A/D převodníku byla čtena hodnota určující otáčky motoru
- Hodnota otáček motoru byla odeslána na ATmega pomocí UART
- ATmega generovala PWM dle otáček motoru
- LED diody D8-D12 zobrazují směr a rychlost otáčení
- Tlačítko S1 slouží k zapnutí / vypnutí napájení motoru pomocí otevřeného kolektoru
- Tlačítko S2 slouží k nastavení směru 0
- Tlačítko S3 slouží k nastavení směru 1

[2] http://arduino.cc/en/Reference/HomePage



Obr. 3.10 Schéma zapojení pro testování karty Gert Board

Zdrojový demonstrující vlastnosti karty Gert Board je k nahlédnutí viz Zdroj. kód 3.2.Nastavení pinů, piny GPIO 2,3,18,22,23 a 27 jsou nastaveny jako výstupní piny. Piny 25, 24, 4 jsou nastaveny jako vstupní piny. Detailnější funkčnost je možné vyčíst z komentářů ve zdrojovém kódu.

```
void nastav_piny(){                                          /* Nastaví všechny piny */
        gpio_fsel(23,GPIO_FSEL_OUTP); gpio_fsel(22,GPIO_FSEL_OUTP);
        gpio_fsel(27,GPIO_FSEL_OUTP); gpio_fsel(18,GPIO_FSEL_OUTP);
        gpio_fsel(3,GPIO_FSEL_OUTP);  gpio_fsel(2,GPIO_FSEL_OUTP);
        gpio_fsel(25,GPIO_FSEL_INPT); gpio_fsel(24,GPIO_FSEL_INPT);
        gpio_fsel(4,GPIO_FSEL_INPT);  gpio_set_pud(25,GPIO_PUD_UP);
        gpio_set_pud(24,GPIO_PUD_UP); gpio_set_pud(4,GPIO_PUD_UP);
        uart_begin(); spi_begin();    gpio_len(0|(1<<24)|(1<<25)|(1<<4));
        spi_setClockDivider(250);     uart_setup(9600,8,0,0,0);
}
void odnastav_piny() {                                       /* Zruší nastaveni pinů */
        gpio_set_pud(25,GPIO_PUD_OFF); gpio_set_pud(24,GPIO_PUD_OFF);
        gpio_set_pud(4,GPIO_PUD_OFF); spi_end(); uart_end();
}
void ledky(int val,int smer) {                               /* Ovládání led diod */
        int rychlost;
        if(val < 50)                          rychlost = 600;
        if(50  <= val && val < 100)           rychlost = 500;
        if(100 <= val && val < 150)           rychlost = 400;
        if(150 <= val && val < 200)           rychlost = 300;
        if(200 <= val && val < 250)     rychlost = 200;
        if(255 <= val )                 rychlost = 100;
        if(smer >0) {                                        /* Směr 1 */
                gpio_set(23); delay(rychlost); gpio_set(22); gpio_clr(23); delay(rychlost);
                gpio_set(27); gpio_clr(22); delay(rychlost); gpio_set(18); gpio_clr(27);
                delay(rychlost); gpio_clr(18); gpio_clr(3);
        }
        if(smer <0) {                                        /* Směr 0 */
                gpio_set(18); delay(rychlost); gpio_set(27); gpio_clr(18); delay(rychlost);
                gpio_set(22); gpio_clr(27); delay(rychlost); gpio_set(23); gpio_clr(22);
                delay(rychlost); gpio_clr(23); gpio_set(3);
        }
        if(val == 0) delay(500);                             /* Motor stojí */
}
int main(int argc, char **argv) {
        int16_t tl1,tl2,tl3, ad, smer = -1, on = 0, on_pred=0;
        double pomer = 1023/255;
        if (!init()) return EXIT_FAILURE;                    /* Inicializace BCM */
        nastav_piny();

        while(1) {
                if(gpio_eds(4)) on = (on)?0:1;               /* Kontroluje spuštění */
                else on_pred = 0;

                if(!on) {                                    /* Když není zapnuto -> zpět*/
                   gpio_clr(2);
                   delay(500);
                   continue;
                } else gpio_set(2);                          /* Připoj zdroj pomoci OC*/

                tl2 = gpio_eds(25); tl3 = gpio_eds(24);      /* Načtení dat z tlač. */
                ad = read_from_AD(0, 0);                     /* Přečtení hodnoty z A/D*/
                if(tl2) smer = 1;                            /* Směr 1 */
                if(tl3) smer = -1;                           /* Směr 0 */
                if(tl2*tl3) break;                           /* Zastav program */
                ad /=pomer;                                  /* Přepočti z 1024 na 255*/
                ledky(ad,smer);                              /* Aktivace Led diod */
                uart_write_to_atmega(ad*smer,NULL);          /* Odešli na ATmega */
        }

        odnastav_piny();
        if (!close()) return EXIT_FAILURE;                   /* Ukončení BCM */
        return EXIT_SUCCESS;
}
```

Zdroj. kód 3.2 Ukázka programu k desce Gert Board

4 Řidicí systém REX

"Řídicí systém REX je otevřený a škálovatelný systém vhodný pro vnořené řízení (embedded control), přenositelný na různé platformy s překladači jazyka C a C++ od jednoúčelových řídicích desek s jednoduchou exekutivou reálného času až po procesní stanice se standardními operačními systémy (Windows XP/Vista/7, Windows CE, Linux, PharLap ETS, apod.). Kompatibilita řídicího systému REX s programovým balíkem Matlab/Simulink je jednou ze základních myšlenek návrhu systému REX. Je možné využít veškeré možnosti Simulinku pro simulaci a odladění algoritmů. Po simulačním ověření lze řídicí algoritmy přeložit do binárních konfiguračních souborů, které je možné pomocí diagnostického protokolu založeného na standardu TCP/IP poslat přímo do cílových zařízení a podle nich zahájit řízení bez nutnosti odstavení zařízení. Ekvivalentní chování simulace a řízení v reálném čase zaručuje rozsáhlá knihovna funkčních bloků ve verzích jak pro Simulink tak i pro každou cílovou platformu." (Zdroj: REX_Getting_Started_CZ.pdf)

Bylo by tedy dobré, aby REX rozšířil své pole působnosti i na periférie platformy RPi. K tomu, aby REX mohl ovládat periférie RPi, slouží ovladač RPiDrv.

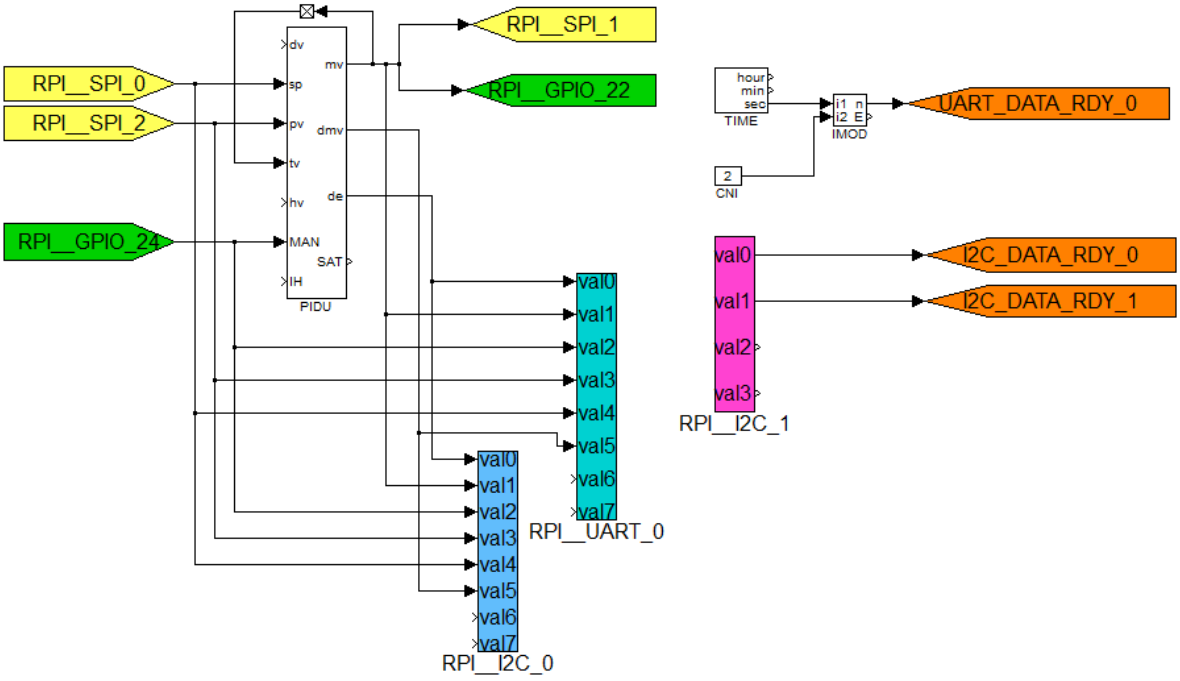
4.1 Ovladač RPiDrv

Ovladač RPiDrv přidává řídicímu systému REX možnost komunikovat s periferiemi RPi. Také je rozšířen o funkce, které jsou schopny komunikovat s A/D, D/A převodníky, ATmega a Motor Controlerem na kartě Gert Board. V následující části práce budou popsány základní rysy ovladače. Jelikož se ovladač skládá z dvou částí, bude popsána každá zvlášť.

4.1.1 Host část

Hostovská knihovna je potřebná pro počítač, na kterém je nainstalován program RexDraw a RexView. Je zde vytvářen příslušný model, respektive výsledný *.rex* soubor.

Hostovská knihovna vytváří spojení mezi příslušnou periférií na RPi / Desce Gert Board a bloky REXu. Pro čtení z periférie je nutné použít blok *From* a blok *Goto* pro zápis na periférii. Příklad spojení je uveden na Obr. 4.1, popis jednotlivých bloků je zobrazen pod Obr. 4.1.



Obr. 4.1 Příklad modelu REXu používajícího bloky RPI

Vstupy

RPI__SPI_0	(AD převodník, kanál 0)
RPI__SPI_2	(AD převodník, kanál 1)
RPI__GPIO_24	(Vstupní pin)
RPI__I2C_1	(Přijímá data pomocí I^2C)

Výstupy

RPI__SPI_1	(DA převodník kanál 0)
RPI__GPIO_22	(Výstupní pin)
RPI__UART_0	(Odesílá data pomoci UART)
RPI__I2C_0	(Odesílá data pomocí I^2C)

Ovládací prvky

UART_DATA_RDY_0	(Aktivuje komunikaci UART 0)
I2C_DATA_RDY_0	(Aktivuje komunikaci I^2C 0)
I2C_DATA_RDY_1	(Aktivuje komunikaci I^2C 1)

Spojení je vytvářeno tak, že ve fázi kompilace je každému prvku přiřazen takzvaný *handle*. *Handle* je unikátní číslo, které smí používat právě jeden blok, ať už *From,* nebo *Goto*. Bloky musí mít název v pevně daném formátu, aby byly přiřazeny ke správné periférii. Název se skládá ze tří částí, jak je naznačeno v tabulce Tab. 4.1.

- První část udává příslušnost k ovladači RPi. Jiný název nebude brán ovladačem v potaz.

- Druhá část definuje typ periférie, ze které bude chtít číst nebo na ni zapisovat. Bude-li první část souhlasit a druhá nikoliv, nastane při kompilaci chyba. A to důvodu nenalezení odpovídající periferie.
- Třetí část udává identifikátor jednotlivé periférie. Jedná-li se o GPIO, odpovídá číslo použitému pinu. V ostatních případech slouží jako identifikátor k rozlišení jednotlivých bloků mezi sebou.

RPI__	GPIO_	#
	I2C_	#
	SPI_	#
	UART_	#
	PWM_	#
	PCM_	#
	ISS_	#
	FCE_	#

Tab. 4.1 Vzory RPi bloků v REXu

Možnosti použitelných bloků jsou zobrazeny v Tab. 4.2.

GPIO_2	I2C_0	FCE_0	SPI_0	UART_IN	PCM_IN	ISS_IN	PWM_OUT
GPIO_3	I2C_1	FCE_1	SPI_1	UART_OUT	PCM_OUT	ISS_OUT	
GPIO_4	I2C_2	FCE_2	SPI_2				
GPIO_7	...	FCE_3	SPI_3				
GPIO_8	...	FCE_4	SPI_4				
GPIO_9	...	FCE_5	SPI_5				
GPIO_10	...	FCE_6					
GPIO_11	...	FCE_7					
GPIO_14	...	FCE_8					
GPIO_15	...	FCE_9					
GPIO_17	...	FCE_10					
GPIO_18	...	FCE_11					
GPIO_22	...	FCE_12					
GPIO_23	...	FCE_13					
GPIO_24	...	FCE_14					
GPIO_25	...	FCE_15					
GPIO_27	...	FCE_16					
GPIO_28	...	FCE_17					
GPIO_29	I2C_47	FCE_18					
GPIO_30	I2C_48	FCE_19					
GPIO_31	I2C_49						

Tab. 4.2 Výčet možných periférií v REXu

Nyní již víme, jak vytvořit spojení mezi blokem Rexu a periférií RPI. Zatím však není periférie nijak nastavena. Je tedy potřeba vytvořit konfigurační soubor, ve kterém bude uvedeno nastavení jednotlivých periférií. K vytvoření konfiguračního souboru slouží program RPiCfg.

4.2 RPiCfg

RPiCfg je grafický program napsaný v jazyce C/C++ za použití Qt framework, konkrétně se jedná o verzi 4.8.1. Program je určen k vytvoření konfiguračního souboru. Jak program spustit a ovládat bude popsáno v následujících kapitolách.

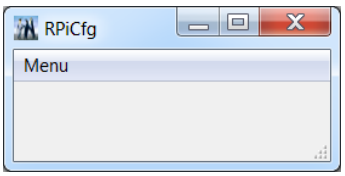
Program je vytvořen ve dvou variantách:

RPICfg.exe Pro spuštění programu je nutné mít následující soubory:

- RPiCfg.exe
- GPIOs.png
- QtCore4.dll
- QtGui.dll

Součást knihovny RPiDrv_T.dll Pro spuštění programu z REXu je nezbytné kliknout na tlačítko *Special edit* v nastavení bloku IODRV.

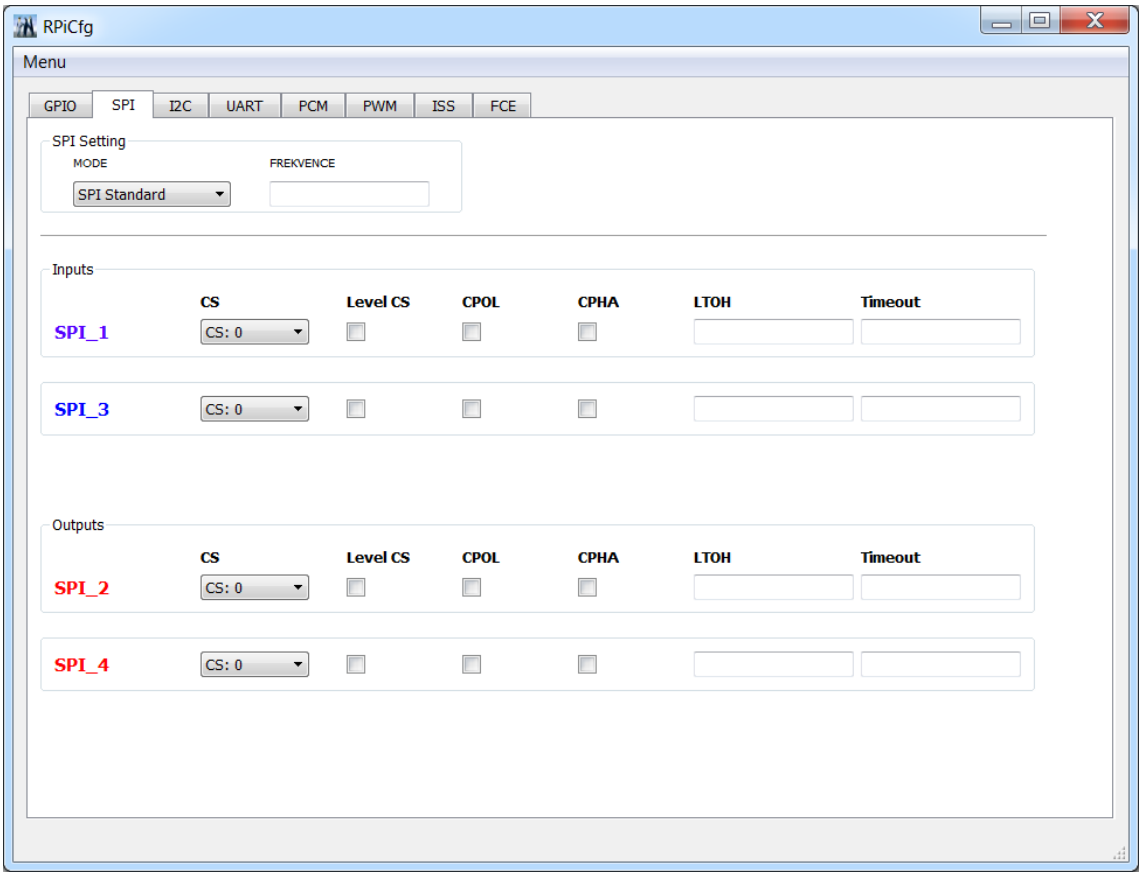
Po spuštění programu, ať už první, nebo druhou cestou, se nám zobrazí okno viz Obr. 4.2.



Obr. 4.2 Úvodní okno RPiCfg

Nyní je třeba vybrat příslušný *.mdl* soubor, který obsahuje bloky pro komunikaci s RPi perifériemi. Je důležité, aby soubor *.mdl* obsahoval všechny bloky RPi, které se budou používat. Pouze k nalezeným blokům bude možné vytvořit nastavení. Soubor vybereme pomocí *Menu - open file*. Jakmile je soubor vybrán, jsou v něm vyhledány bloky RPi a je vygenerováno okno vyobrazené na Obr. 4.3. (vzhled okna bude záviset na použitých perifériích). U jednotlivých bloků je zjištěno, zda se jedná

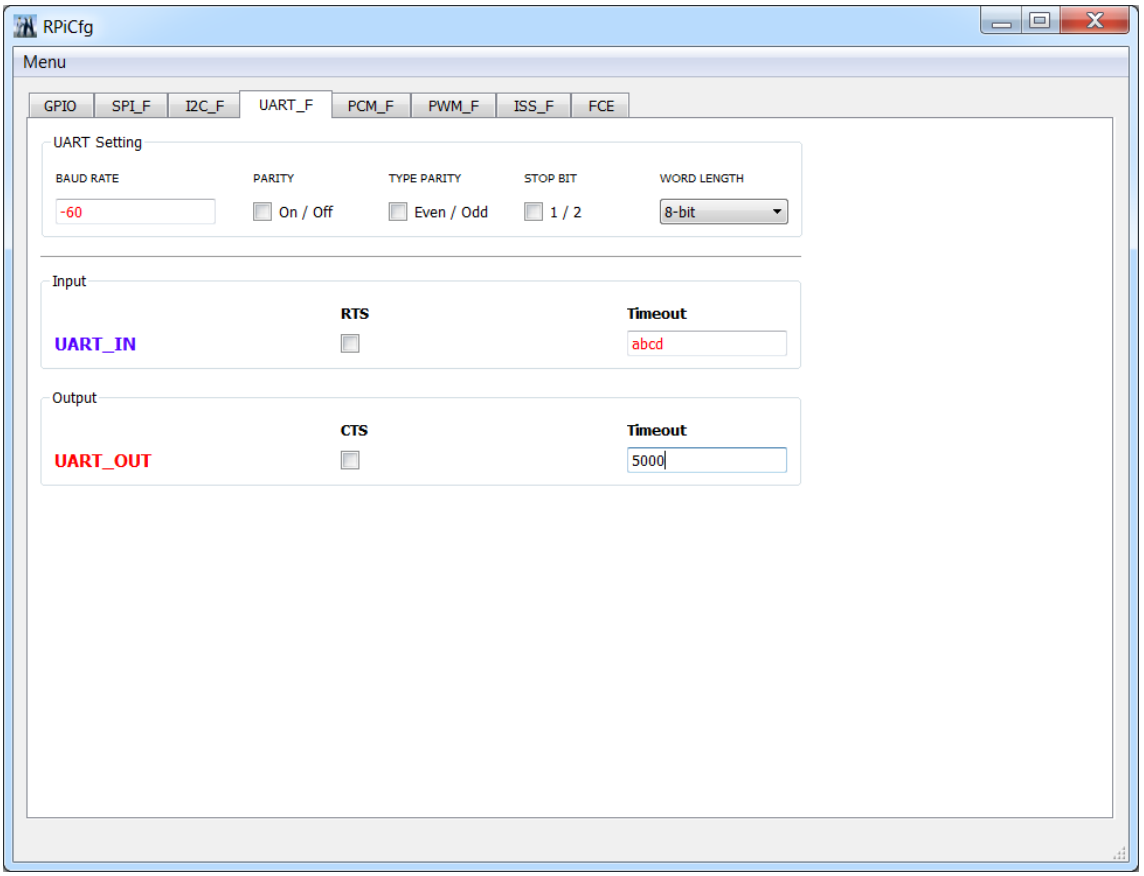
o blok *From* nebo *Goto*, tedy jestli bude z periférie čteno nebo na ní budou data zapisována.



Obr. 4.3 Ukázka okna RPiCfg s perifériemi

Jsou-li v modelu použity různé typy periférií (UART, SPI, ...), je nutné pro jejich nastavení aktivovat příslušnou záložku se jménem periférie. Některé periférie obsahují položky, které platí pro všechny bloky. Tyto položky se nacházejí v horní části okna. Viz Obr. 4.3 - *Frekvence* a *použitý mód*. Program RPiDrv obsahuje záložku FCE. Tato záložka slouží k přiřazení předdefinované funkce danému bloku. To slouží k přiřazení k již definovaným funkcím, které nevyžadují žádné další nastavení.

Po vyplnění všech kolonek je možné provést kontrolu textových polí. Zkontroluje se, zda textová pole obsahují hodnoty v určených mezích. Mechanizmus je aktivován dvojklikem levým či pravým tlačítkem myši (myš musí být nad oknem aplikace). Kontrola je provedena nad všemi záložkami aplikace. Vyskytují-li se v záložce chyby, je za název záložky přidán příznak "**_F**" a text v textových polích je obarven na červeno. Viz Obr. 4.4.



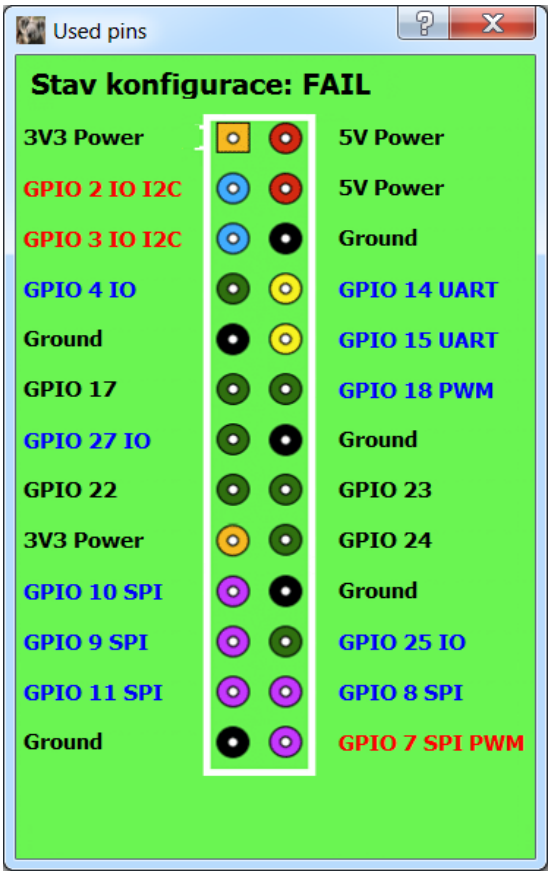
Obr. 4.4 Detekce chyb programu RPiCfg

Zadáme-li do textového pole novou, již validní hodnotu, je potřeba aktivovat kontrolu znovu, aby se změny projevily.

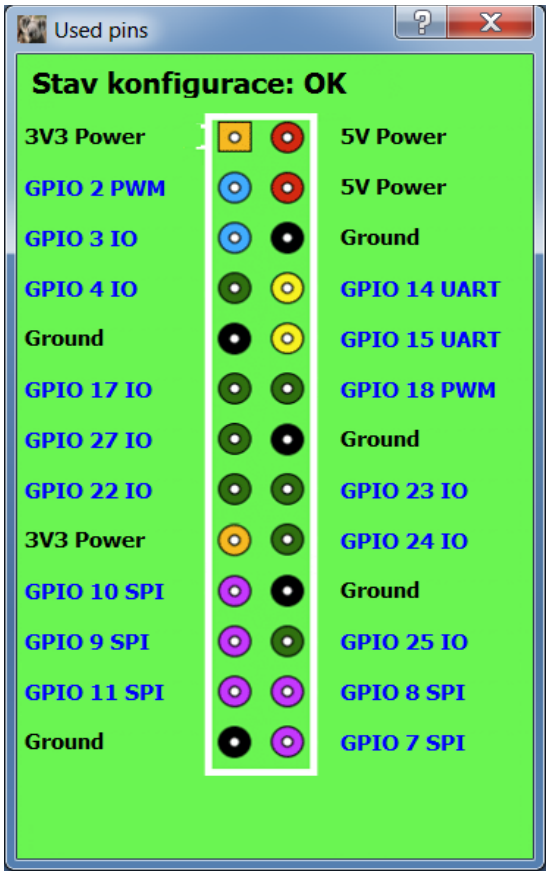
Jsme-li spokojeni s veškerým zadaným obsahem, můžeme vygenerovat soubor *.rio*, který obsahuje námi požadované nastavení periférií. Před uložením nastavení se zobrazí okno zobrazující využití a případnou kolizi pinů. Ke kolizi může dojít, když bude v modelu použito zároveň:

- SPI & GPIO 7, GPIO 8, GPIO 9, GPIO 10, GPIO 11
- UART & GPIO 14, GPIO 15, GPIO 30(CTS), GPIO 31(RTS)
- PWM & GPIO 18
- I^2C & GPIO 2, GPIO 3
- PCM & GPIO 29, GPIO 29, GPIO 30, GPIO 31

V případě, že je vše v pořádku, zobrazí se nám dialog na Obr. 4.6. V případě, že došlo k nějaké kolizi, jsou popisy pinů zobrazeny červeně, viz Obr. 4.5. Dialogové okno je pouze informační. Pokud si je uživatel jist svými kroky, v uložení mu nebude zabráněno. Nyní je vytvořen konfigurační soubor.



Obr. 4.5 Konfigurace narazila na chyby



Obr. 4.6 Konfigurace OK

4.3 Zavedení a použití ovladače v Rexu

Verze REXu musí být stejná nebo vyšší než je 2_08. U nižších verzí není zaručena funkčnost ovladače. Před spuštěním REXu je nezbytné nakopírovat do složky *bin* v kořenovém adresáři REXu následující soubory:

- RPiDrv_H.dll
- rpivs.dll
- GPIOs.png
- QtCore4.dll
- QtGui.dll.

Nyní je možné Rex spustit a nastavit exekutivu.

4.3.1 Nastavení Rex exekutivy

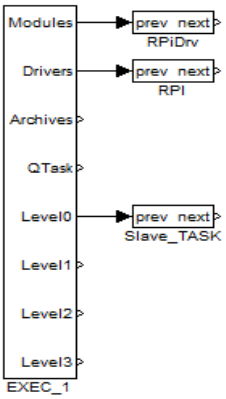
Základem REXu exekutivy je blok EXEC z knihovny EXEC. Pro zavedení RPI ovladače je nutné splnit následující kroky:

- Ve vlastnostech bloku nastavit parametr "target" jako "PC - linux"
- K portu Modules připojit blok MODULE z knihovny EXECLIB
- Blok MODULE pojmenovat RPiDrv
- K portu Drivers připojit blok IODRV z knihovny EXECLIB

U bloku IODRV je nutné nastavit:

- Jméno bloku **RPI**
- Module: **RPiDrv**
- classname: **RPiDrv**
- cfgname: <jméno konfiguračního souboru vytvořeného v RPiCfg>

Příklad exekutivy je zobrazen na Obr. 4.7.



Obr. 4.7 Ukázka exekutivy

V tuto chvíli je vše připraveno ke kompilaci modelu. V průběhu kompilace načte hostovská část konfigurační soubor vytvořený v RPiCfg, ve kterém je kompletní nastavení periférií, a přibalí ho k souboru *.rex*. Soubor *.rex* je následně odeslán na RPi jedním z programů RexDraw nebo RexView. Soubor je přijat REXem na RPi a nastavení je předáno druhé části ovladače - Target část.

4.3.2 Target část

Před odesláním souboru je nutné zkontrolovat, zda je verze RexCore na RPi buď stejná nebo vyšší než je 2_08. U nižších verzí není zaručena funkčnost knihovny. Zároveň je

nezbytné nakopírovat knihovnou RPiDrv_T.so do adresáře /usr/lib a přiřadit jí příslušná práva příkazem:

```
sudo chmod 755 RPiDrv_T.so
```

Jakmile je vše hotovo, je možné na Raspberry Pi odeslat soubor *.rex*.

4.4 Funkce ovladače

Jakmile je knihovna zinicializována, je zavolána funkce *open(),* ve které dochází k namapování příslušných adres, viz kapitola 2.2. Po dokončení mapování se volá funkce *turn_off_pull(),* která zruší všem pinům předchozí nastavené *pull up* a *pull down* rezistory, viz kapitola 5.1.1. To je vykonáváno z důvodu, že nastavení *pull up* a *pull down* rezistorů je permanentního rázu a zůstává nastavené i při vypnutí RPi. Nebylo by příliš vhodné, aby měl jeden z pinů nastaven zároveň *pull up* a *pull down* rezistor. Následuje funkce *set_param(),* která přednastavuje vlastnosti jednotlivých pinů, jako je například:

- přiřadí pinu funkci vstupu / výstupu
- pin bude detekovat náběžné hrany
- k pinu bude připojen pull up rezistor
- nebo celých skupin, inicializace SPI, UART atd.

Nastavení jednotlivých periférií je součástí souboru *.rex*. Respektive se jedná o nastavení vytvořené v RPiCfg. V případě, že soubor *.rex* nebude obsahovat nastavení, nedojde k inicializaci periférii.

Např. očekáváme od pinu funkci vstupu, která z důvodu absence nastavení nebyla pinu přiřazena. To však nezabraňuje číst data z registru, v němž je uložen stav pinu. Přečtená hodnota může být náhodná.

Dále knihovna obsahuje funkce *XRead* a *XWrite*, pomocí kterých komunikuje program REX s knihovnou RPiDrv_T.so. Funkce *XRead* obsahuje několik parametrů, mezi kterými je *handle*. Ten určuje, z jaké periférie se bude číst. Tato funkce navrací přečtenou hodnotu. Funkce *XWrite* je REXem volána s několika parametry, mezi které patří opět *handle* a hodnota, jež má být zapsána na odpovídající periférii. V tomto okamžiku jsme již schopni vytvořit model, který komunikuje s periférierniemi na RPi. Nyní se seznámíme detailněji s vlastnostmi ovladače.

5 Vlastnosti ovladače

V původním návrhu diplomové práce bylo mým úkolem zjistit, jaké má ovladač nedostatky. Jelikož jsem ovladač neobdržel, ale napsal jsem ho sám, popsal bych spíše jeho vlastnosti, shrnul jeho výhody a nevýhody. Ovladač zpřístupňuje následující periférie:

- GPIO
- UART
- PWM
- SPI
- I^2C
- PCM

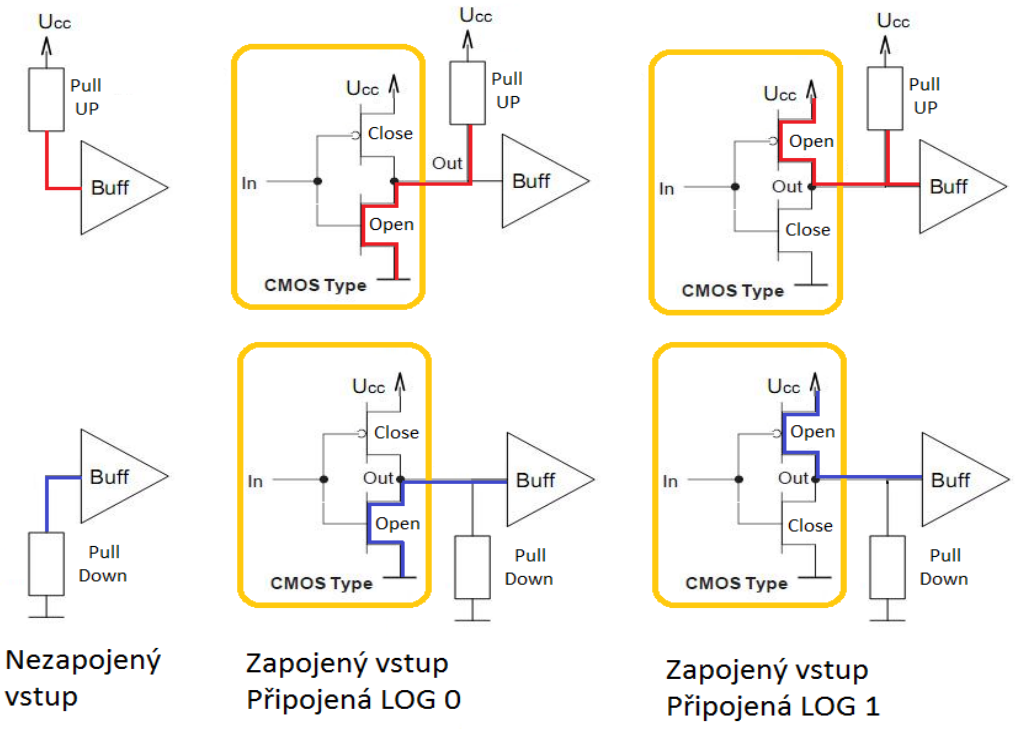
Vlastnosti, které lze nastavit pomocí ovladače RPi, jsou popsány v následujících kapitolách.

5.1 GPIO

U GPIO je možnost nastavit jej jako vstupní pin a výstupní pin. V případě, že bude pin nastaven jako vstupní, je u něj možné volit následující vlastnosti:

5.1.1 Pull up a pull down rezistor.

Možnost připojit *pull up* a *pull down* rezistor. Význam rezistoru je zobrazen na Obr. 5.1.



Obr. 5.1 Významy Pull UP a Pull DOWN rezistorů

5.1.2 Detekční vlastnosti GPIO

Základní vlastnosti GPIO je čtení aktuální hodnoty, která je k pinu připojena. Hodnota se však může libovolně měnit mezi čteními, a tím pádem nám může uniknout důležitá událost. K zaznamenání těchto jevů je RPi vybaveno různými detekčními vlastnostmi. Aktivace detekce probíhá tak, že je zapsána 1, na bit odpovídající GPIO pinu, do příslušného registru.

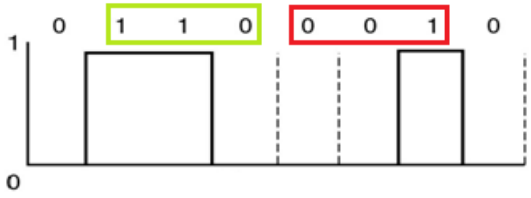
Př. Budeme-li chtít aktivovat na GPIO 7 detekci náběžné hrany, je potřeba do registru (Broadcom Corporation 2012: 98) zapsat na sedmý bit 1.

Bude-li náběžná hrana na GPIO 7 detekována, mohou nastat 2 události (dle nastavení) :

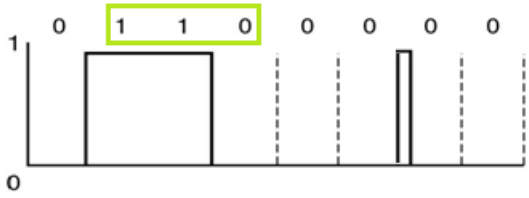
- Nastane přerušení. Je možné tomuto jevu přiřadit kód, který se v případě přerušení vykoná.
- V registru EDS se objeví na příslušném bitu hodnota 1.

Ovladač využívá pouze druhého jevu. Detekovatelné události jsou následující.

Detekce vzestupné a sestupné hrany. Detekce probíhá tím způsobem, že jsou ze signálu odebírány vzorky s určitou periodou viz Obr. 5.2. (o odebírání vzorků se stará BCM2835). V případě, že odebíraná data odpovídají vzoru 011 nebo 100, tak je nastavena hodnota 1 příslušného bitu v registru EDS. Nevýhodou této detekce je perioda, s jakou jsou vzorky odebírány. Je možné, že velmi krátké hrany nebudou zaznamenány, jako na Obr. 5.3.

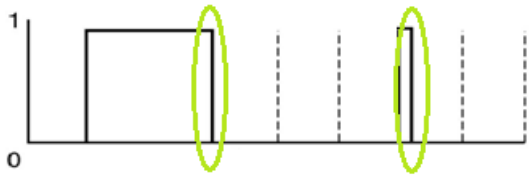


Obr. 5.2 Detekce hrany synchronní metodou - OK

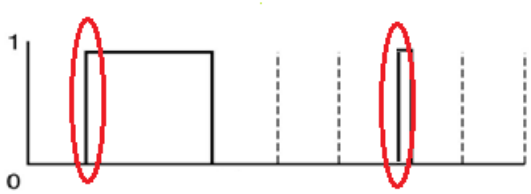


Obr. 5.3 Detekce hrany synchronní metodou - Chyba

Asynchronní detekce vzestupné / sestupné hrany jsou zobrazeny na Obr. 5.4 a Obr. 5.5. Je používán hranový detektor, který není závislý na vzorkovací periodě. Proto mohou být detekovány i velmi krátké hrany. Nastane-li tato událost, je nastaven příslušný bit v registru EDS.

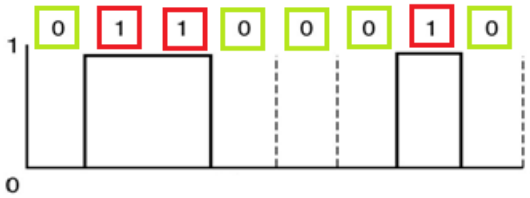


Obr. 5.4 Detekce sestupné hrany asynchronní metodou



Obr. 5.5 Detekce vzestupné hrany asynchronní metodou

Detekce hladiny 0 - 1 je zobrazena na Obr. 5.6. V případě, že je detekována zvolená hladina, je nastaven příslušný bit registru EDS.



Obr. 5.6 Detekce hladiny

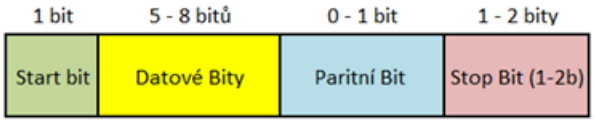
Detekční vlastnosti je možné kombinovat. Například je možné detekovat vzestupnou i sestupnou hranu zároveň. Výhodou je, že jsme schopni zjistit, že k jedné z těchto událostí došlo. Nevýhodou je, že nevíme ke které, protože obě události nastavují stejný bit registru EDS.

V případě výstupu má pin pouze jedno nastavení a tím je maximální odebíraný proud, po jaký je pin schopen udržet hodnotu logické 0 (popřípadě 1). Nastavení maximálního proudu se nevztahuje na jeden pin, ale na celou skupinu. GPIO 0 - 31 patří do jedné skupiny.

Všechny uvedené vlastnosti je možné nastavit z RPiCfg.

5.2 UART

Knihovna RPiDrv je schopna komunikovat s jiným zařízením pomocí protokolu UART (Universal Asynchronous Transmiter Reciever). RPi disponuje dvěma oddělenými frontami pro zápis a čtení, které jsou dimenzovány na šestnáct bajtů, konkrétně fronta pro odesílání dat má velikost 16x8 bitů, a do fronty pro přijímání dat je možné uložit 16x12 bitů. Fronta pro přijímání obsahuje dvanáct bitů z důvodu, že uchovává 1x start bit, 5-8x datové bit, 1x paritní bit a 1-2 stop bity viz Obr. 5.7 Rozložení bitů v UART ramci.



Obr. 5.7 Rozložení bitů v UART ramci

Nyní se zaměříme na možnosti nastavení protokolu UART.

5.2.1 Časový mód

Přijímací a odesílací funkce mohou být někdy velmi časově náročné. Bude-li například požadováno přijmout N zpráv a odesílatel tento počet bajtu neodešle, bude se čekat až nekonečně dlouho, dokud nebude přijat příslušný počet. Pokud bude komunikováno pouze pomocí periférie UART, může být tento jev žádoucí. Pokud však bude potřeba obsluhovat i jiné zařízení, již to žádoucí není. Z uvedeného důvodu je tedy možné nastavit maximální dobu čekání v us a po jejím uplynutí bude funkce ukončena. Dále je však uživatele nutné informovat, že nastala tato událost a také, kolik zpráv bylo přijato. K těmto informacím je možné se dostat pomocí bloků *FROM* v REXu s názvy:

- RPI__fUART_TIMEOUT_TX Indikuje vypršení času na UART_TX
- RPI__fUART_TIMEOUT_RX Indikuje vypršení času na UART_RX
- RPI__fUART_NO_OF_TX Počet odeslaných zpráv na UART_TX
- RPI__fUART_NO_OF_RX Počet přijatých zpráv na UART_RX

5.2.2 Chybová hlášení

Knihovna rovněž obsahuje několik bloků *FROM*, které umožňují monitorovat stav UART periférie. Mezi vlajky patří:

- RPI__fUART_OE: Vlajka indikuje přetečení, tj. na UART jsou posílány další zprávy, ačkoliv je přijímací fronta plná. Tyto bajty jsou nenávratně ztraceny. Je na uživateli, aby doba mezi čteními z přijímací fronty byla úměrná počtu přijímaných bajtů.
- RPI__fUART_BE: Vlajka indikuje chybu během přijímání bajtu.
- RPI__fUART_PE: Vlajka indikuje chybu parity, parita nesouhlasí s paritou, jež byla nastavena v RPiCfg
- RPI_fUART_FE: Vlajka indikuje chybu stop bitu.

5.2.3 Nastavení komunikace

Pomocí knihovny lze nastavit, kolik vodičů bude použito při komunikaci. Přesněji řečeno, zda budou používány CTS a RTS. (Je možné využít i pouze CTS nebo pouze RTS, avšak pozor, pro možnost využití těchto signálů je nutné mít napájený Header 5 viz kapitola 2.1.) Další nastavitelné položky:

- Přenosová rychlost. Ovladač umožňuje nastavit přenosovou rychlost od 476 b/s do 31 250 000 b/s.
- Parita. Je možné nastavit, zda bude používán paritní bit, či nikoliv. Bude-li parita použita, je možné nastavit, zda bude sudá nebo lichá.
- Stop bit. Je možné zvolit počet stop bitů. Mezi standardní možnosti patří 1; 1,5 a 2 stop bity. RPi však možnost 1,5 bitu nepodporuje.
- Počet datových bitů. Knihovna umožňuje volit počet datových bitů ve zprávě. Umožňuje nastavit standardní délky, tedy od 5 do 8 bitů.

REX periodicky čte / zapisuje data s délkou periody nastavenou v bloku EXEC. Pokud bude délka periody nastavena na 100 ms, budou data každých 100 ms odesílána na cílové zařízení. Naskýtá se otázka, zda je potřeba posílat / číst data ze zařízení každou periodu. Z tohoto důvodu knihovna obsahuje blok *Goto* s názvem RPI__fUART_DATA_RDY_# (kde # je nahrazuje „IN“ nebo „OUT“ odpovídajícího UART bloku). Tedy:

- RPI__fUART_DATA_RDY_IN patří k RPI__UART_IN
- RPI__fUART_DATA_RDY_OUT patří k RPI__UART_OUT

Data jsou odesílána pouze v případě, je-li hodnota bloku UART_DATA_RDY_# různá od nuly. V případě, že blok nebude využit, budou data odesílána v každém cyklu.

Pro mikrokontroler ATmega obsahuje knihovna funkci *uart_write_to_atmega*. Vstupním parametrem funkce je znaménkové 16bitové celé číslo. Znaménko určuje směr otáčení a absolutní hodnota rychlost otáčení. Funkce se stará o vygenerování příslušných dat a následné odeslání pomocí UART na mikrokontroler. Pro správnou funkci knihovny musí být na ATmega nahrán program *Atmega_pwm*.

5.3 PWM

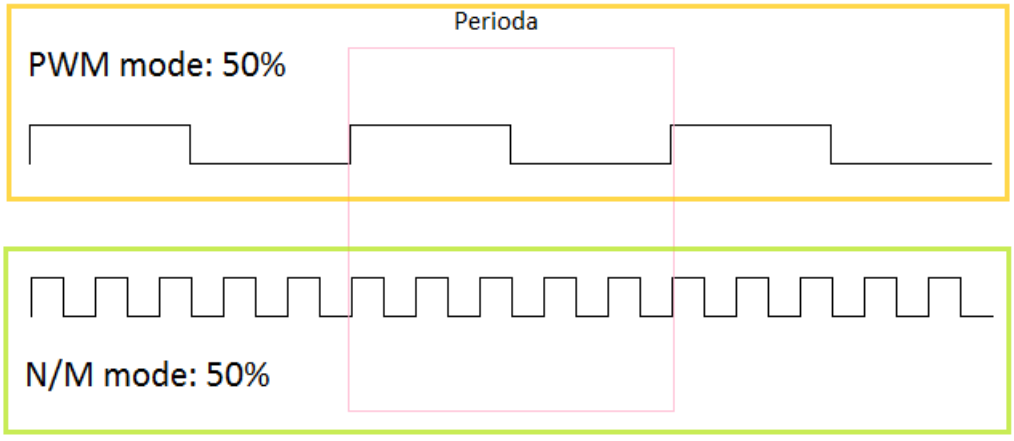
Knihovna je schopna ovládat periférii pro generování pulzní šířkové modulace, neboli PWM. Je možné vybrat si z několika módů a vlastností, které budou popsány v dalších odstavcích.

5.3.1 Módy PWM

První mód je označen jako PWM mód. Hodnotu, již chceme mít na výstupu PWM, stačí jednou zapsat do registru. Po zbytek běhu programu je hodnota generována PWM výstupem. Druhý mód je označen jako sériový. Nyní je používána odesílací fronta jako v případě UART. Fronta má velikost 16x32 bitů. Data jsou z fronty zapisována na výstupní piny PWM. V případě použití sériového módu je možné zvolit další nastavení, která definují, jak se bude chovat výstup v případě, že je fronta prázdná. Jednou z možností je aktivace prvku RPTL, který bude opakovat poslední zprávu ve frontě, dokud nebude fronta obsahovat nové zprávy. Druhou možností je definovat, jaká hodnota bude hodnota výstupu. Tedy, zda bude setrvávat v hodnotě LOW nebo HIGH.

5.3.2 Nastavení reprezentace dat

Mezi další nastavení PWM patří i algoritmus, který je používán k reprezentaci dat. První algoritmus je označován jako PWM. Princip tohoto algoritmu spočívá v tom, že: odesílaná hodnota je reprezentována jako poměr N/M, kde N označuje počet cyklů, po které bude hodnota výstupu v hodnotě HIGH, a hodnota M označuje celkový počet cyklů. Cílem toho algoritmu je, aby libovolný počet cyklů, co možná nejlépe, aproximoval reprezentovanou hodnotu. Druhým algoritmem je M/S. Nyní jsou data reprezentována, jak je standardem u PWM. Názorné srovnání je zobrazeno na Obr. 5.8.



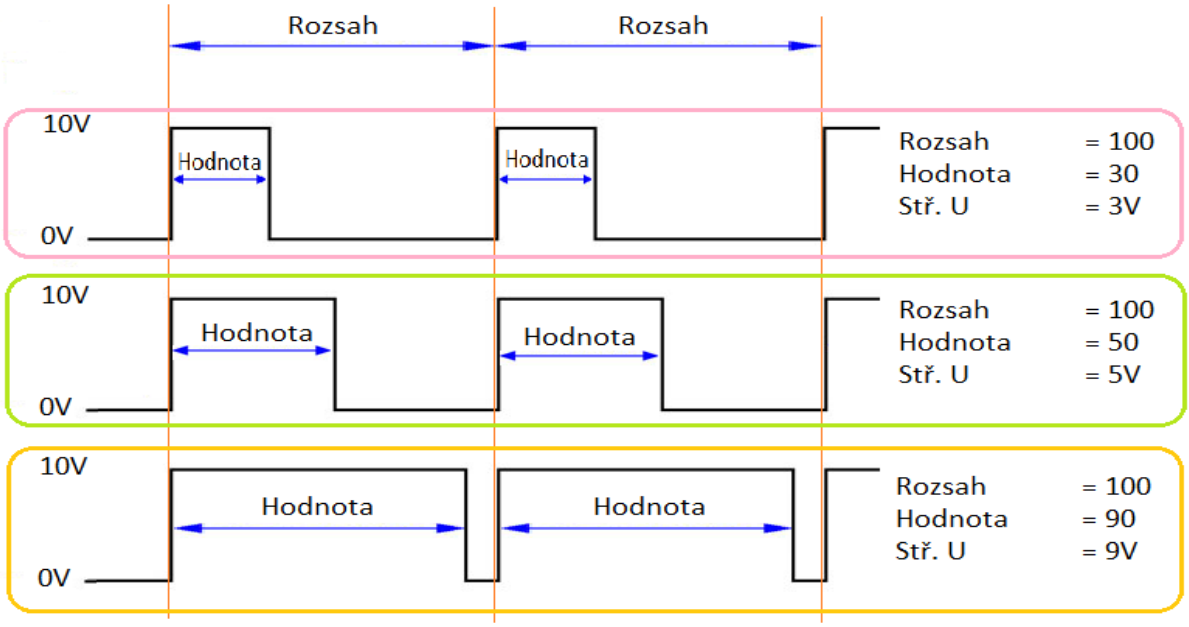
Obr. 5.8 Srovnání PWM a N/M módu

5.3.3 Nastavení PWM

Mezi nastavení PWM patří:

- Frekvence, kterou je PWM přenášeno.
- Rozsah. Určuje, na kolik dílů bude rozdělena jedna perioda.

Na Obr. 5.9 je zobrazen příklad významu rozsahu a požadované hodnoty. Perioda je rozdělena na 100 dílů. Je-li *Hodnota* rovna 30, bude po dobu 30 dílů výstup PWM sepnutý. Dále jsou uvedeny *Hodnoty* 50 a 90. Na Obr. 5.9 je uvedeno, jaká by byla střední hodnota napětí v případě přiváděného napětí 10 V. Není dovoleno, aby byla *Hodnota* větší, než je počet dílů. Bude-li hodnota větší, ovladač ji přednastaví na maximální hodnotu rozsahu.



Obr. 5.9 Příklad PWM

5.3.4 Funkce pro ovládání motoru

Knihovna obsahuje funkci *pwm_motor_set(uint32_t value)*, která je koncipována tak, aby mohla být ovládána jediným parametrem. Znaménko u řídicí hodnoty určuje směr otáčení motoru a absolutní hodnota udává počet dílů, po které bude PWM sepnuta. Funkce nezabraňuje prudkým změnám směru. Pokud by byla hodnota v jednom taktu v maximální záporné hodnotě a v druhém v kladné, tak tomu není programově zabráněno. Je na uživateli, aby tyto stavy ošetřil zastavením motoru. Funkce není ošetřena záměrně. Záleží na konkrétním typu motoru, za jakou dobu je schopen změny směru.

5.4 SPI

Mezi další komunikační protokoly podporované knihovnou patří i protokol SPI. SPI disponuje, stejně jako předchozí komunikační protokoly, dvěma frontami. Jedna pro příjem dat a druhá pro odesílání. Velikost obou front je shodná, a to 16×32 bitů. Mezi základní nastavení patří, v jakém módu bude komunikace probíhat.

5.4.1 Módy SPI

Knihovna podporuje několik SPI režimů, mezi které patří:

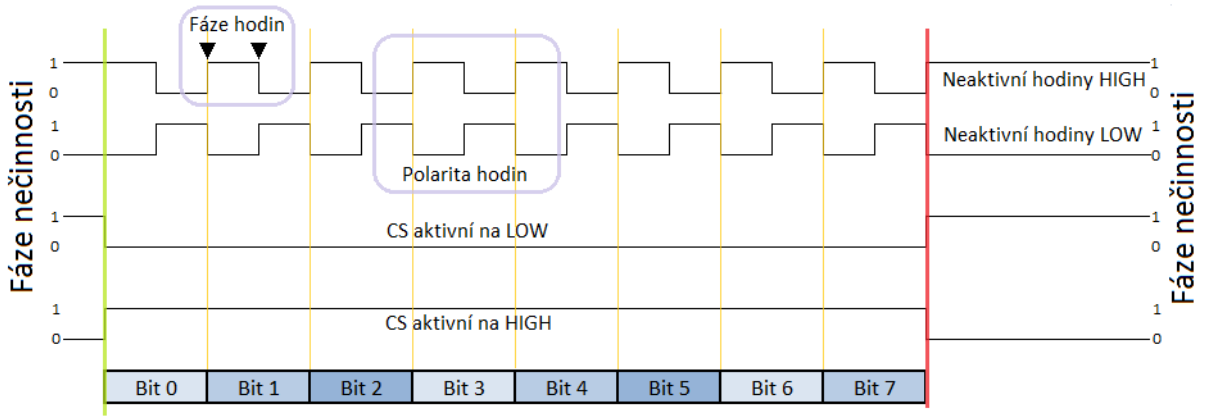
- SPI mode, klasické třívodičové zapojení.
- Bidirectional mode, dvouvodičové zapojení - obousměrný provoz po jednom datovém vodiči.
- LoSSi mode.

U LoSSi módu lze nastavit zpoždění výstupního kanálu (LTOH), maximální hodnota je 15 taktů.

5.4.2 Nastavení SPI

U SPI komunikace je možné nastavit následující parametry (parametry jsou zobrazeny na Obr. 5.10.):

- Frekvence přenosu.
- Volba Chip selectu.
- Hladina aktivace Chip Selectu. Na začátku komunikace je nutné upozornit slave jednotku, že s ní bude komunikováno. To se děje vodičem CS. V případě, že neprobíhá žádná komunikace, je standardně hladina napětí nastavena na logickou jedna. Nová zpráva začíná nastavení hodnoty na 0. Tím je slave jednotka upozorněna na začínající komunikaci. Hladiny je možné pomocí knihovny prohodit.
- Polarita hodin. Knihovnou je možné nastavit, v jaké hladině budou setrvávat hodiny v neaktivním stavu.
- Fáze hodin. V jaké části datového bitu bude odečítána hodnota.



Obr. 5.10 Příklad SPI

5.4.3 Funkce pro SPI

Knihovna obsahuje jak funkce univerzální, jejichž vlastnosti lze libovolně nastavit, tak i funkce jednoúčelové, které slouží pro komunikaci s D/A A/D převodníky na desce Gert Board. V programu RPiCfg stačí zvolit, s jakým z převodníků bude komunikováno a jaký kanál bude použit. Funkce se postará o nastavení potřebných parametrů a vygenerování příslušných bajtů (viz kap 3.4). Funkce *write_to_da()* se stará o zapsání hodnoty na příslušný kanál na D/A převodník. Naopak funkce *read_from_ad()* navrací hodnotu přečtenou z A/D převodníku z příslušného kanálu.

Obdobně jako u přenosu UART nemusí být vyžadováno zasílání dat každou periodu. Proto je zde blok *Goto* s názvem RPI__SPI_DATA_RDY_#, který aktivuje a deaktivuje zasílání dat. Samozřejmostí je možnost nastavení časového limitu, viz kapitola 5.2.1. S tím jsou i spojené bloky:

- RPI__fSPI_TIMEOUT_TX Indikuje vypršení času na SPI_TX
- RPI__fSPI_TIMEOUT_RX Indikuje vypršení času na SPI_RX
- RPI__fSPI_NO_OF_TX Počet odeslaných zpráv na SPI_TX
- RPI__fSPI_NO_OF_RX Počet přijatých zpráv na SPI_RX

5.5 I^2C

Součástí knihovny jsou funkce podporující přenos dat protokolem I^2C. Jedná se o funkce *I2C_read()* a *I2C_write()*. Pomocí těchto funkcí je možné přenést *n* bajtů na zvolenou slave jednotku.

5.5.1 Nastavení I2C

Mezi parametry I^2C bloků patří:

- Adresa slave jednotky. Podpora 7 i 10 bitové adresy.
- Frekvence přenosu.
- REDL nastavuje počet taktů, které proběhnou po vzestupné hraně, než se začnou vzorkovat příchozí data (Hodnota musí být menší než frekvence / 2).
- FEDL nastavuje počet taktů, které proběhnou po sestupné hraně, než se začnou vysílat další data. (Hodnota musí být menší než frekvence / 2).
- TOUT počet taktů, které proběhnou po vzestupné hraně, po kterých se určí, že slave jednotka neodpovídá.

5.5.2 Podpůrné bloky I2C

Rovněž existují bloky I2C_DATA_RDY_# sloužící k aktivaci a deaktivaci přenosu a bloky vypršení časového limitu:

- RPI__fI2C_TIMEOUT_TX Indikuje vypršení času na I2C_TX
- RPI__fI2C_TIMEOUT_RX Indikuje vypršení času na I2C_RX
- RPI__fI2C_NO_OF_TX Počet odeslaných zpráv na I2C_TX
- RPI__fI2C_NO_OF_RX Počet přijatých zpráv na SPI_RX

Navíc I2C obsahuje své specifické bloky, jež jsou:

- RPI__fI2C_CLKT Slave jednotka neodpovídá
- RPI__fI2C_ERR Došlo k chybě potvrzení zprávy

5.6 SPI a I^2C slave

Pomocí knihovny je možné nastavit, aby se SPI nebo I2C periférie chovalo jako slave jednotka. Lze nastavit pouze jednu periférii toho typu do slave režimu, protože v tomto režimu tyto dvě periférie sdílí společně fronty pro odesílání a přijímání dat.

5.6.1 Stavy I2C a SPI

Pro zjištění stavu jednotky slouží několik vlajek, které jsou popsány níže:

- RPI_ISS_UE vlajka je nastavena v případě, že I2C nebo SPi master požaduje data, avšak čtecí fronta žádná data neobsahuje.
- RPI_fISS_OE vlajka je nastavena v případě, že jsou posílána nová data a přijímací fronta je plná.

- RPI_fISS_BRK pokud je nastavena vlajka na hodnotu 1, jsou zastaveny probíhající operace a fronta je vyprázdněna.

5.6.2 Nastavení I2C

Následující parametry je možné nastavit pomocí knihovny, abychom docílili požadovaného chování SLAVE jednotky I2C.

- Adresa slave jednotky.
- Nastavení Host Control. V případě, že je aktivován, master jednotka obdrží obsah registru STAT nebo CTRL jako první zprávu.
- GPUSTAT obsah STAT registru.
- HCTRL obsah CTRL registru.

5.6.3 Nastavení SPI

Následující parametry je možné nastavit pomocí knihovny, abychom docílili požadovaného chování SLAVE jednotky SPI viz kapitola 5.4.2

- CPOL nastavení polarity
- CPHA nastavení fáze

5.7 PCM

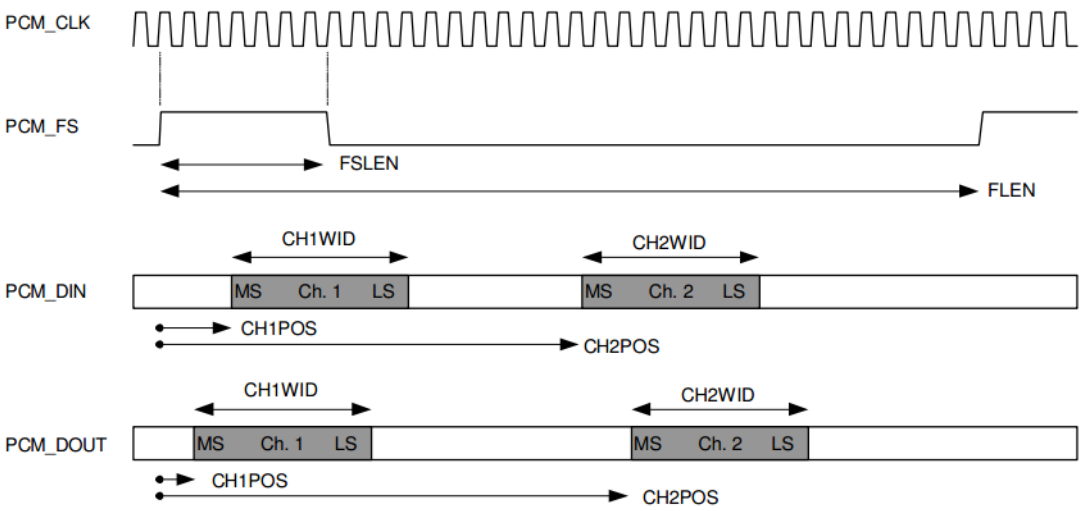
Díky knihovně je možné používat pulzní kódovou modulaci, která je součástí RPi. Pro plné využití pulzní kódové modulace je nutné mít napájený Header 5 (kapitola 2.1). Pokud není header 5 přítomen, je možné použít PCM pouze jako výstup. (Knihovna nijak nekontroluje přítomnosti pinů.) RPi obsahuje dvě fronty. Jednu pro přijímání a druhou pro odesílání. Obě fronty mají totožnou velikost, a to konkrétně 64x32bitů.

5.7.1 Nastavení PCM

Knihovna nabízí možnosti nastavit následující funkce. Ilustrace některých z nich jsou zobrazeny na Obr. 5.11:

- PDM faktor 16/32.
- PDM mode
- FRXP Data z každého kanálu jsou zapsaná do přijímací fronty. Data jsou zapisována tak, že první zpráva bude z kanálu jedna, druhá z kanálu dva atd. Při zapnutí této funkce budou zprávy zredukovány na 16 bitů a budou poskládány tak, že kanál jedna bude mít dolních 16 bitů a kanál dva horních 16 bitů.

- FTXP. Stejné jako FRXP, jen pro odesílací frontu.
- PCM_CLOCK_MODE. Vypnuto = master, PCM hodiny jsou výstupem a řídí běh. Zapnuto = slave, PCM hodiny jsou vstupem.
- CLKI Inverze hodin. Vypnuto = výstup se mění na vzestupnou hranu, vstupy jsou vzorkovány na sestupnou. Zapnuto = opak.
- FSM Vypnuto = master mode, PCM_FS je výstup a je generován. Zapnuto = Slave mode PCM_FS je vstup a je čten.
- FSI Vypnuto = v master mode. FS je 0 a jde do 1, když indikuje zprávu. Ve slave modu začíná na 1 a pak jde do 0. Zapnuto = opak.
- FLEN délka rámce v bitech.
- CHxWEX Povoluje, aby mohla mít zpráva více než 24bitů u kanálu x(1 nebo 2).
- CHxEN Zapne kanál x.
- CHxPOS Pozice prvního bitu v rámci.
- CHxWID Počet bitů v kanálu.
- RX Sign extend doplní chybějící MSB bity na hodnotu 1. Spjato s CHxWID, ve které je uložena délka zprávy. Bude-li mít zpráva například 22 bitů, zbylých 10 do 32 bude doplněno na 1.
- FSLEN nastaví délku synchronizačního rámce v hodinových cyklech, pouze když je FSM on PCM_FS zůstane aktivní když FSLEN≤FLEN.



Obr. 5.11 Ukázka jednotlivých nastavení na PCM rámci

5.7.2 Podpůrné bloky PCM

Knihovna podporuje následující bloky pro zjištění stavu periférie PCM:

- RPI__fPCM_RXR přijímací fronta je plná a vyžaduje čtení
- RPI__fPCM_TXW odesílací fronta je prázdná a vyžaduje nová data
- RPI__fPCM_RXERR u přijímací fronty došlo k chybě
- RPI__fPCM_TXERR u odesílací fronty došlo k chybě
- RPI__fPCM_RXSYNC přijímací fronta není synchronizována
- RPI__fPCM_TXSYNC odesílací fronta není synchronizována

K odstranění některých chybových stavů stačí, aby chyba ustala, a hodnota stavu se vrátí zpět do normálu. Někdy je však potřeba zapsat na příslušný bit hodnotu 1, aby příznak chyby zmizel. K potvrzení slouží blok RPI__fPCM_CS. Zapsáním následujících hodnot odstraníme příslušné chybové stavy:

- Zapsáním 0x100000 synchronizujeme fronty
- Zapsáním 0x10000 odstraníme příznak chyby v přijímací frontě
- Zapsáním 0x8000 odstraníme příznak chyby v odesílací frontě

Rovněž existují bloky RPI__PCM_DATA_RDY_# sloužící k aktivaci a deaktivaci přenosu a bloky vypršení časového limitu:

- RPI__fPCM_TIMEOUT_TX Indikuje vypršení času na PCM_TX
- RPI__fPCM_TIMEOUT_RX Indikuje vypršení času na PCM_RX
- RPI__fPCM_NO_OF_TX Počet odeslaných zpráv na PCM_TX
- RPI__fPCM_NO_OF_RX Počet přijatých zpráv na PCM_RX

5.8 Vlastnosti knihovny

V následujících řádcích budou shrnuty vlastnosti napsané knihovny.

Výhody

- Kompletní možnost ovládání I/O pinů
- Podpora protokolu I^2C (7 i 10 bitové adresy)
- Podpora PWM
- Podpora protokolu SPI
- Podpora A/D D/A převodníku na desce Gert Board
- Podpora protokolu UART
- Vysoká podpora PCM

Nevýhody

- Knihovna nepodporuje přerušení.
- V případě, že nejsou odbavována data z příchozích front, knihovna tuto událost nijak neošetřuje, pouze o této události oznámí nadřazenou vrstvu, tedy REX.
- Nepodporuje DMA.

Právě jsme byli seznámeni se všemi vlastnostmi ovladače a můžeme přejít k testování.

6 Testování ovladače

Aby bylo možné určit případné použití Raspberry Pi, je důležité zjistit, jak rychle je schopno komunikovat se svými periferiemi, jaké jsou limitní časy zápisu a čtení z jednotlivých periférií.

6.1 Test zápisu a čtení z pinu

Účelem testu je zjistit za jak dlouho se objeví logická hodnota na pinu od jejího zapsání do registru. Analogicky je rovněž důležitá hodnota čtení. Nebo-li, za jakou dobu je možné zjistit případnou změnu hodnoty na určitém pinu. Měření těchto časů probíhalo následovně:

- PIN_1 byl nastaven jako výstup
- PIN_2 byl nastaven jako vstup
- Piny byly propojeny svorkou
- Hodiny byly spuštěny
- Na PIN_1 byla zapsána hodnota
- PIN_2 byl čten tak dlouho, dokud se na něm neobjevila zapisovaná hodnota
- Hodiny byly zastaveny

Hodiny byly spuštěny v době, kdy byla na výstupní pin odeslána hodnota a zastaveny, jakmile byla daná hodnota přečtena na pinu vstupním viz Zdroj. kód 6.1.

```
START                                          // Spuštění hodin
gpio_write(PIN_1 ,value);                      // Zápis hodnoty na pin
while(value != gpio_lev(PIN_2));               // Přečtení hodnoty z pinu
STOP                                           // Zastavení hodin
```

Zdroj. kód 6.1 Měření zápis / čtení modifikace 1

Jelikož byl tento časový úsek velmi krátký a hraničil s minimální možnou měřitelnou dobou použitých funkcí (byla použita funkce *clock_gettime*), bylo nutné změřit více zápisů / čtení, aby byla režie měřící funkce zanedbána. Aby bylo zaručeno, že se v aktuálním kroku čte právě zapsaná hodnota, byla tato hodnota v každé iteraci změněna viz Zdroj. kód 6.2.

```
START                                          // Spuštění hodin
for(j = 0;j<1000000;j++) {                     // 1000000x
       gpio_write(PIN_1,value);                // Zápis hodnoty
       while(value != gpio_lev(PIN_2));        // čtení
       value = (value ? 0:1);                  // Změna hodnoty na opačnou
}
STOP                                           // Zastavení hodin
```

Zdroj. kód 6.2 Měření zápis / čtení modifikace 2

Rozebereme-li blíže část kódu, Zdroj. kód 6.2, naskýtá se další otázka. Kolikrát proběhne cyklus *while,* než se na pinu zobrazí dříve zapsaná hodnota? Jednoduchou modifikací získáme Zdroj. kód 6.3.

```
for(j = 0;j<1000000;j++) {              // 1000000x
      gpio_write(PIN_1,value);          // Zápis hodnoty
      tmp = gpio_lev(PIN_2);            // čtení
      if(value != tmp) return;          // Kontrola shody
      value = (value ? 0:1);            // Změna hodnoty na opačnou
}
```

Zdroj. kód 6.3 Měření zápis / čtení modifikace 3

Je zjištěno, že cyklus neproběhne ani jednou (otestováno 1 000 000 000 vždy se stejným výsledkem). Z tohoto poznatku tedy vyplývá, že cyklus while a změna hodnoty viz Zdroj. kód 6.3, jsou nadbytečné. Po jejich odstranění v kódu zůstane pouze zápis hodnoty a čtení, viz Zdroj. kód 6.4.

```
START                                   // Spuštění hodin
for(j = 0;j<1000000;j++) {              // 1000000x
      gpio_write(PIN_1,value);          // Zápis hodnoty
      tmp = gpio_lev(PIN_2);            // čtení
}
STOP                                    // Zastavení hodin
```

Zdroj. kód 6.4 Měření zápis / čtení modifikace 4

Další modifikací již můžeme jednoduše zjistit délku trvání zápisu a čtení, viz Zdroj. kód 6.5.

```
START                                   // Spuštění hodin
for(j = 0;j<1000000;j++) {              // 1000000x
      gpio_write(PIN_1,value);          // Zápis hodnoty
}
STOP                                    // Zastavení hodin
```

Zdroj. kód 6.5 Měření zápis / čtení modifikace 5

Doba čtení a zápisu je zobrazena na grafech Graf 6.1,
Graf 6.2. Testování čtení a zápisu je opakováno 1 000 000.
Test byl proveden pouze na jedné dvojici pinů. Pokud nahlédneme na implementaci čtení viz Zdroj. kód 6.6.

```
uint32_t hodnota_registru = *gpio_lev;
uint8_t hodnota_pinu = (hodnota_registru & (1 << PIN)) ? HIGH : LOW;
```

Zdroj. kód 6.6 Získání hodnoty na vstupním pinu

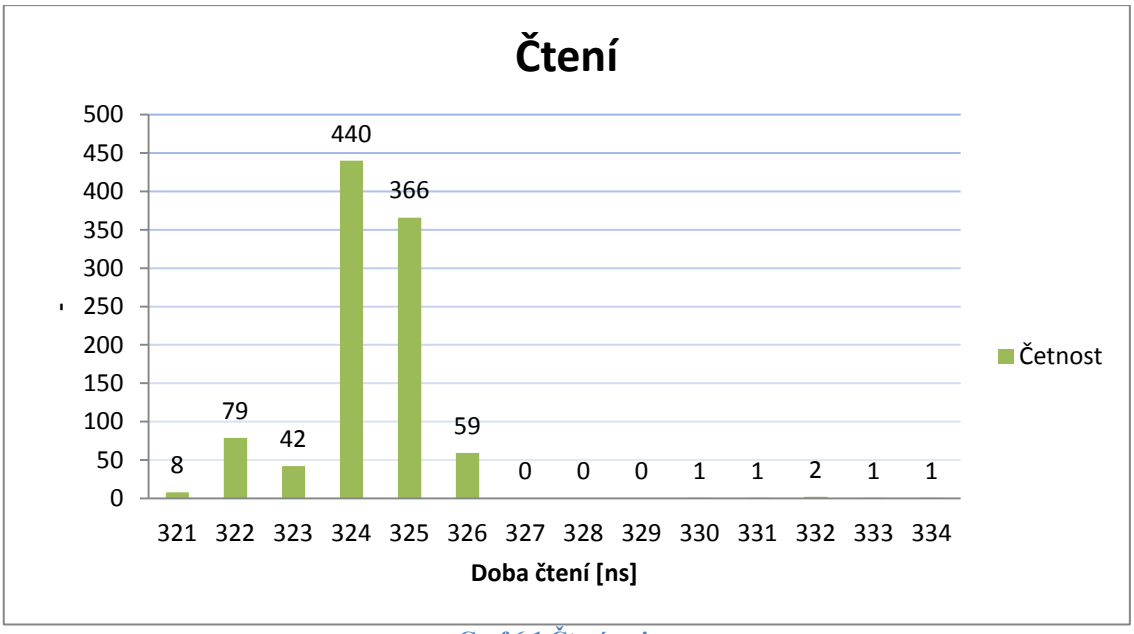
Vidíme, že je přečten celý registr. Ten však obsahuje informaci o všech pinech (Broadcom Corporation 2012: 96). Výsledná hodnota pinu je získána viz druhá řádka Zdroj. kód 6.6. Ekvivalentní je i zápis. S tím rozdílem, že je zde potřeba zapisovat na dvě adresy registru viz Zdroj. kód 6.7. V případě, že chceme na pin zapsat logickou

1, musí být 1 na příslušném bitu v registru SET0. Chceme-li zapsat logickou nulu, je postup stejný, jen je hodnota zapsána do registru CLR0. (Broadcom Corporation 2012: 97)

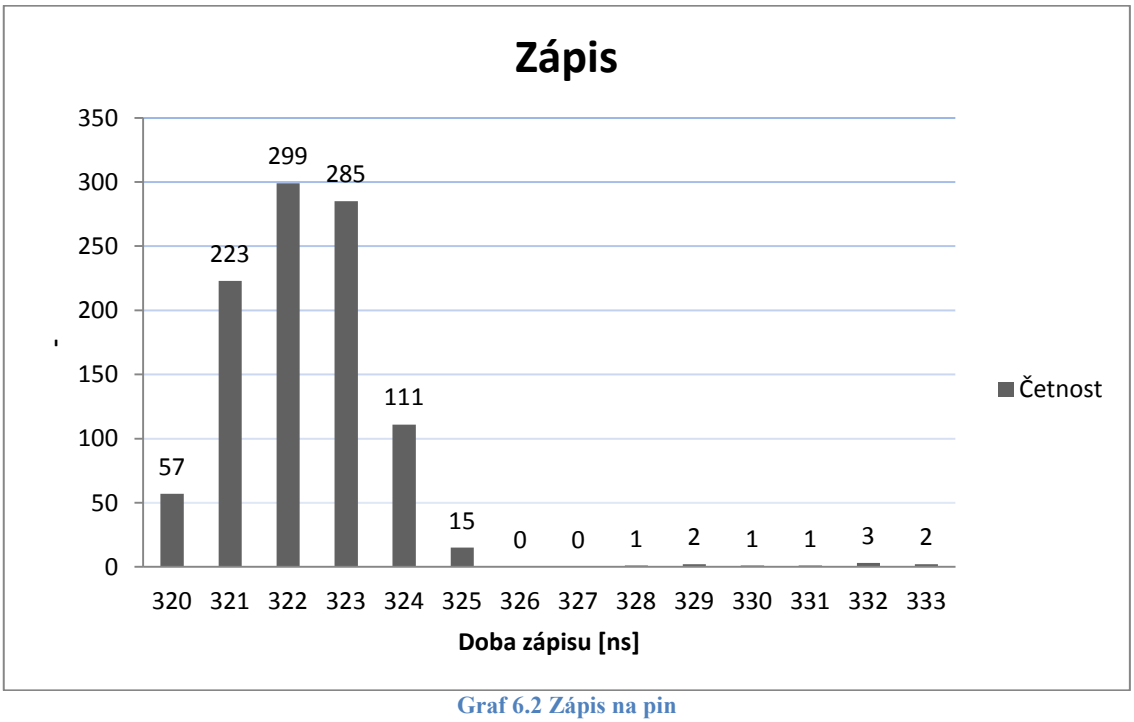
```
if (value)                              // Kontrola zapisované hodnoty
        *gpio_set = (1<<PIN);           // Zápis logické 1
else
        *gpio_clr = (1<<PIN);           // Zápis logické 0
```

Zdroj. kód 6.7 Zápis hodnoty na pin

Naměřené hodnoty jsou zobrazeny v grafech Graf 6.1, Graf 6.2.



Graf 6.1 Čtení z pinu



Graf 6.2 Zápis na pin

V Graf 6.1 a Graf 6.2 jsou uvedeny průměrné hodnoty.

$$\text{Průměrna hodnota cyklu} = \frac{\text{Celkový naměřený čas}}{\text{Počet iterací for cyklu}}$$

Softwarově naměřené hodnoty můžeme porovnat s hodnotami naměřenými osciloskopem.

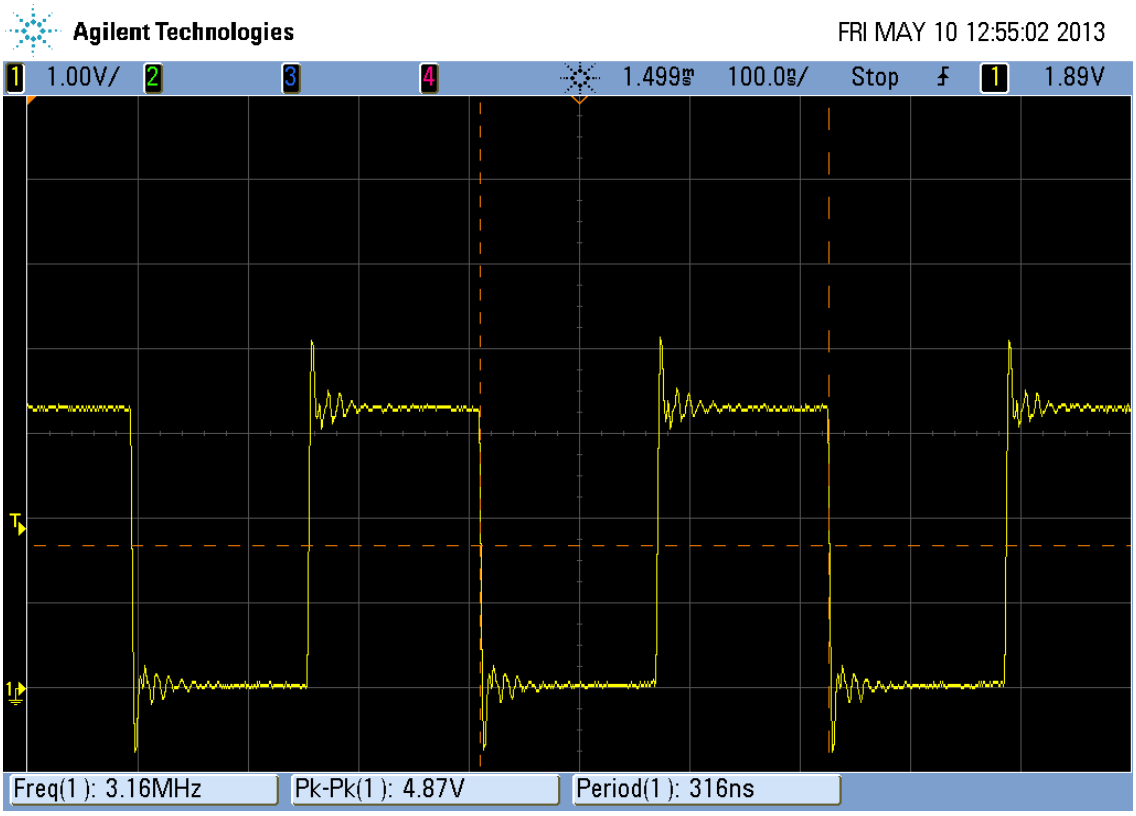
Měření zápisu pomocí osciloskopu. Postup měření byl následující:

- PIN_1 byl nastaven jako výstup.
- Na PIN_1 byla připojena osciloskopická sonda.
- Země osciloskopu a RPi byly spojeny.
- Na pin byla neustále zapisována hodnota logické 0 a logické 1 viz Zdroj. kód 6.8.

```
while(1) {
	gpio_write(PIN_1, LOW);		// Zápis log 0
	gpio_write(PIN_1, HIGH);	// Zápis log 1
}
```

Zdroj. kód 6.8 Použitý kód při osciloskopickém měření

Výsledek měření je vidět na Obr. 6.1. Naměřená průměrná hodnota byla 316 ns



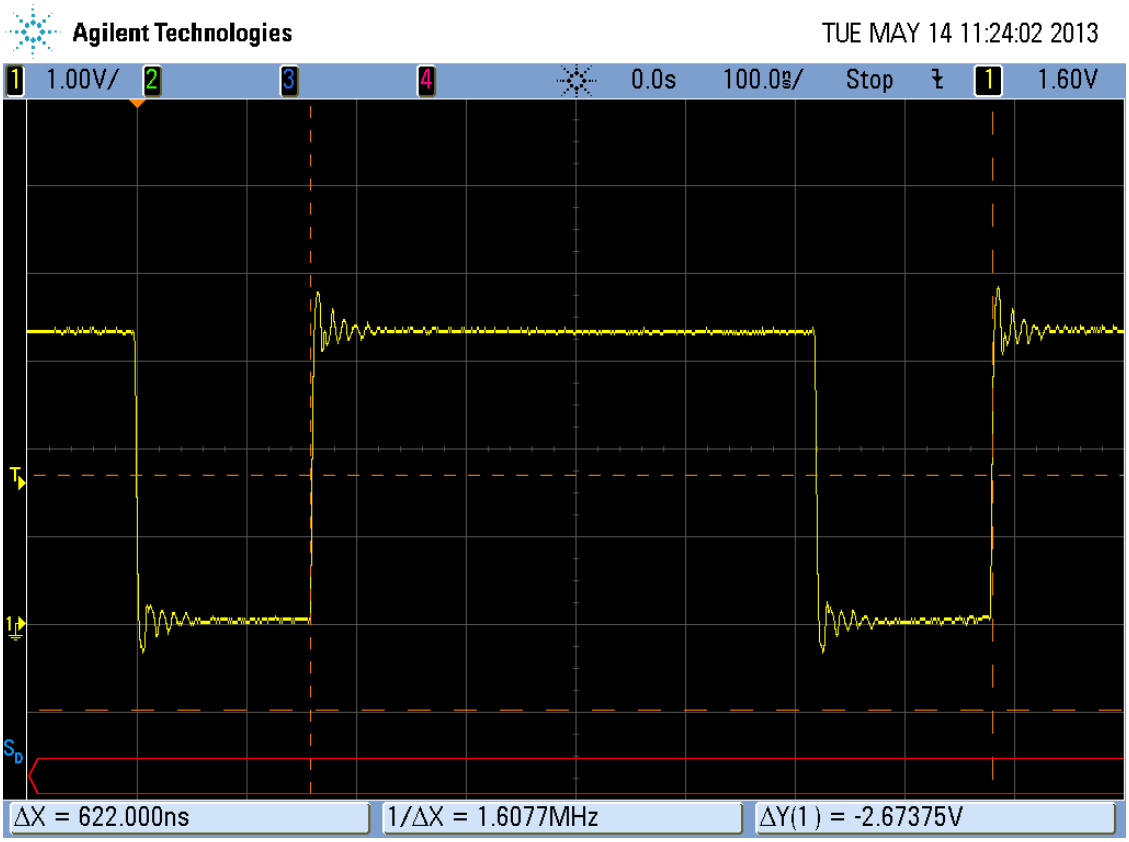
Obr. 6.1 Osciloskopické měření - zápisu na pin

Nyní se podíváme na dobu čtení z pinu. Naměřené hodnoty lze vidět na Obr. 6.2. Postup měření byl následující:

- PIN_1 byl nastaven jako výstup
- PIN_2 byl nastaven jako vstup
- Piny propojeny svorkou
- Na svorku byla připojena osciloskopická sonda
- Země osciloskopu a RPi byly spojeny
- Poté byl neustále opakován následující Zdroj. kód 6.9.

```
while(1) {
          gpio_write(PIN_1, HIGH);          // Zápis log 1
          gpio_lev(PIN_2)                   // Čtení z pinu
          gpio_write(PIN_1, LOW);           // Zápis log 0
}
```

Zdroj. kód 6.9 Použitý kód při osciloskopickém měření doby čtení



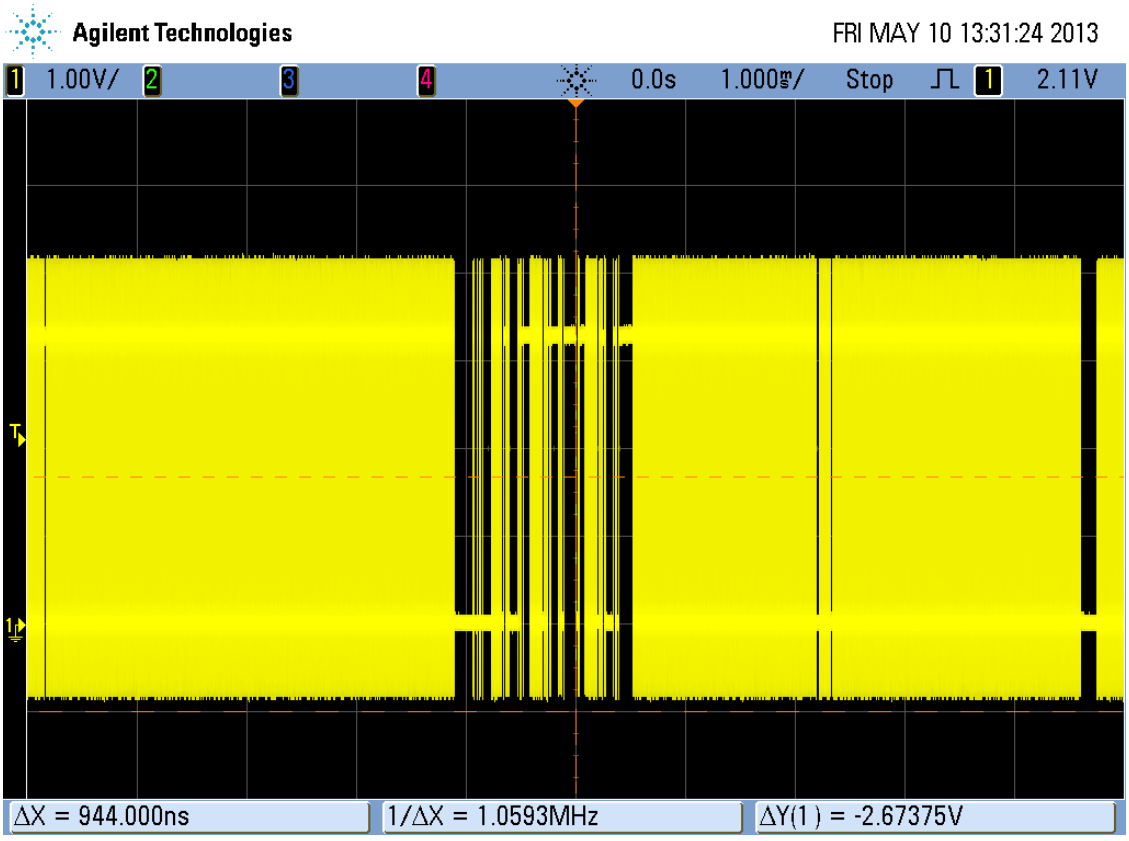
Obr. 6.2 Osciloskopické měření - čtení z pinu

Naměřený čas obsahuje zápis logické jedna a zápis logické nuly. Tento čas je znám z předchozího měření, doba čtení je tedy rovna:

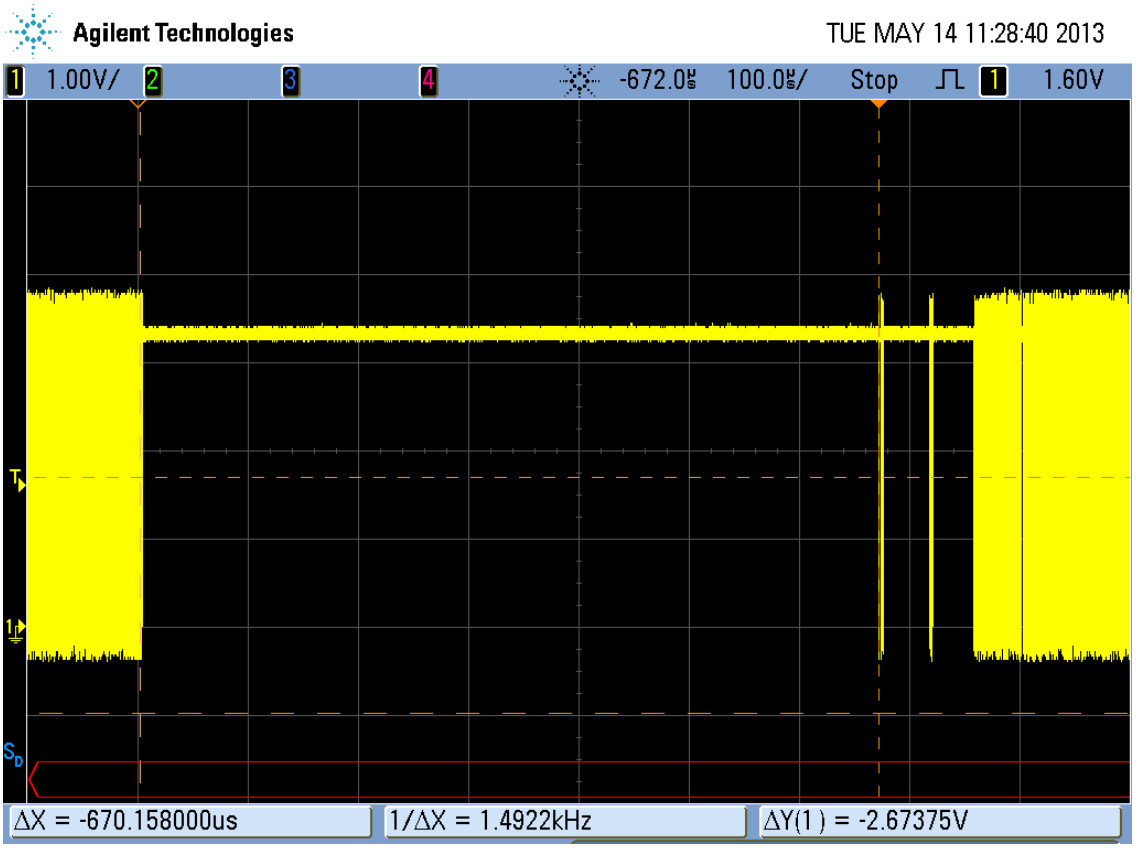
$$\text{Doba čtení} = \text{Celkový naměřený čas} - \text{doba zápisu}$$

$$306ns = 622ns - 316ns$$

Nyní se podíváme na průběh signálu po dobu delšího časového úseku. Na Obr. 6.3 je zobrazen průběh signálu po dobu 10 milisekund. Jsou zde vidět časové úseky dosahující desítky až stovky µs, viz detail na Obr. 6.4, během kterých nedochází ke změně hodnoty (Obr. 6.4 černá část). Nejdelší doba nečinnosti během měření byla 670 µs. Je to způsobeno tím, že při běhu programu dochází k přerušení a čas procesoru je přiřazen jinému procesu a následně je přiřazen zpět.



Obr. 6.3 Osciloskopické měření - ukázka přerušení



Obr. 6.4 Osciloskopické měření - ukázka přerušení detail

Průměrná hodnota naměřená osciloskopem byla brána pouze z hodnot menších než byla 1µs. Z toho důvodu jsou časy naměřené osciloskopem menší než časy naměřené softwarově.

Pozn. Program měl nastavenou nejvyšší možnou prioritu. Kernel Linuxu byl preemptivní.

6.2 Testování SPI sběrnice

Mezi hojně využívané sběrnice patří bezesporu SPI. Velká řada senzorů a čidel používá ke komunikaci s řídicí jednotkou právě sběrnici SPI. Můžeme k nim zařadit například inkrementální rotační čítače, teplotní senzory, měřiče vzdálenosti, snímače otáček, AD / DA převodníky atd.

Důležitým faktorem, co se týče časových limitů, je zjistit, jak dlouho trvá čtení a zápis ze sběrnice na RPi. K testu posloužily A/D převodník MCP3002 a D/A převodník MCP4802, jež obsahuje karta Gert Board.

6.2.1 Testování čtení

Test čtení probíhal následovně:

- PIN_1 byl nastaven jako výstup.
- Byla provedena počáteční inicializace SPI sběrnice. Pinům GPIO 7 - GPIO 11 byly přiřazeny příslušné funkce.
- Byla nastavena frekvence, fáze hodin, polarita a jiné potřebné parametry.
- PIN_1 byl svorkou propojen s vstupem A/D převodníku.
- Hodiny byly spuštěny.
- Na PIN_1 byla zapsaná hodnota log 1.
- Z D/A převodníku bylo čteno, dokud se na výstupu neobjevila příslušná hodnota.
- Hodiny byly zastaveny.

Jelikož se jedná o desetibitový převodník, tak by log 1 na výstupu pinu měla odpovídat hodnotě 1023. Hodnota z A/D převodníku by měla být rovna nule, při nastavení log 0 na výstupním pinu. V předchozích větách bylo záměrně uvedeno "měla být", ale jelikož logické hodnoty mají své tolerance, tak není nikdy zaručeno, že log 0 na výstupu bude právě 0 V. Na to je potřeba brát v průběhu testování zřetel. Zdroj. kód 6.10 realizující test:

```
START                                   // Spuštění hodin
gpio_set(1<<PIN_1);                     // Zápis logické 1
do {
      *spi_fifo = byte1;                // Zápis prvních 8 bitů viz kap
      *spi_fifo = byte2;                // Zápis druhých 8 bitů viz kap

      while (!(*bcm_spi0 & SPI0_DONE));
                                        // Vyčkání do přijetí všech bitů
      byte1 = *spi_fifo;                // Přijetí vyšších 8 bitů
      byte2 = *spi_fifo;                // Přijetí nižších 8 bitů
} while (((0x03&byte1<<8)|byte2) < 800); // Sestavení 10bitu z AD
STOP                                    // Zastavení hodin
```

Zdroj. kód 6.10 Ukázka kódu použitá pro měření čtení z SPI

6.2.2 Testování zápisu

Postup měření byl následující:

- PIN_1 byl nastaven jako vstup.
- Byla provedena počáteční inicializace SPI sběrnice. tj. pinům GPIO 7 - GPIO 11 byly přiřazeny příslušné funkce.
- Byla nastavena frekvence, fáze hodin, polarita a jiné potřebné parametry.
- PIN_1 byl svorkou spojen s výstupem D/A převodníků.
- Hodiny byly spuštěny.
- Na D/A převodník byla odeslána hodnota reprezentující 2,048 V.
- PIN_1 byl čten, do doby než se na něj objevila hodnota logické 1.
- Hodiny byly zastaveny.

Maximální možné napětí z D/A převodníku je 2,048 V. Dle specifikace je napětí na pinu vyhodnoceno jako logická 1, je-li v rozmezí 2 - 3,3 V [3]. Výstup z D/A převodníků je tedy možné použít. Kód použitý na měření je zobrazen viz Zdroj. kód 6.11:

```
START                                   // Spuštění hodin
*spi_fifo = byte1;                      // Zápis prvních 8 bitů viz kap
*spi_fifo = byte2;                      // Zápis druhých 8 bitů viz kap
while (!gpio_lev(PIN));                 // Kontrola pinu
STOP                                    // Zastavení hodin
```

Zdroj. kód 6.11 Ukázka kódu použitá pro měření zápis na SPI

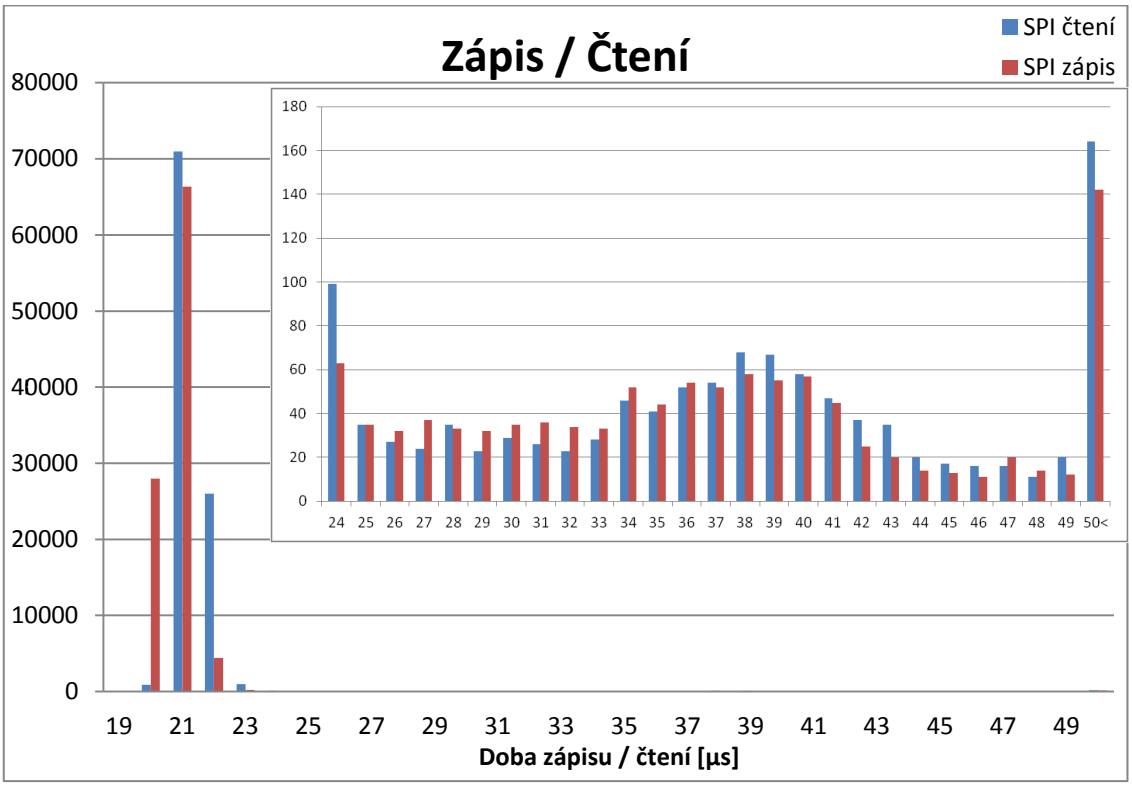
6.2.3 Naměřené hodnoty

Výsledky jsou zobrazeny na Graf 6.3. Test zápisu / čtení byl 100 000x opakován. Procentuální výsledky byly následující:

- 98,9% časů se nacházelo v rozmezí 19-21 µs.
- 99,84% v rozmezí 19-49 µs.
- Zbylých 0,14% bylo větších než je ho hodnota 50 µs.
- Byly naměřeny i hodnoty přesahující 452 µs, a to vlivem přerušení.

[3]http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications

Dosažený čas je závislý na frekvenci SPI. U testovaných čipů byla frekvence nastavena na 1 MHz. Při větší přenosové frekvenci by bylo dosaženo nižšího času. Změřený čas přibližně odpovídá nastavené frekvenci. Délka jednoho taktu u 1 MHz je 1 µs. Je třeba přenést 16 bitů, tj. 16 x 1 µs = 16 µs. V případě použití čipů umožňující rychlejší přenos, by se celková doba zkrátila.



Graf 6.3 Zápis / čtení SPI[4].

6.3 Testování PWM

Mezi další hojně používané technologie patří bezesporu pulzní šířková modulace (PWM). Opět nás bude zajímat, jak dlouho potrvá RPi realizace PWM. Funkce ovládající PWM je ovládána jednou hodnotou. Znaménko určuje směr otáčení a absolutní hodnota udává rychlost. Je-li hodnota měněna pouze v jednom směru, tj. nemění se znaménko, je vykonávaná část kódu zkrácena. (Žlutě zvýrazněný kód není vykonáván Zdroj. kód 6.12). Dojde-li však ke změně směru, je nutné PWM zastavit, změnit část jejího nastavení a opět ji spustit

[4] Jelikož grafy obsahují 100 000 hodnot, je graf koncipován tak, že v pravém rohu je zobrazen detail hodnot, které by z důvodu dominance okolních hodnot byly nepatrné.

Z toho důvodu bylo testování PWM rozděleno na dvě části:

- Hodnota PWM nemění znaménko
- Hodnota PWM mění znaménko

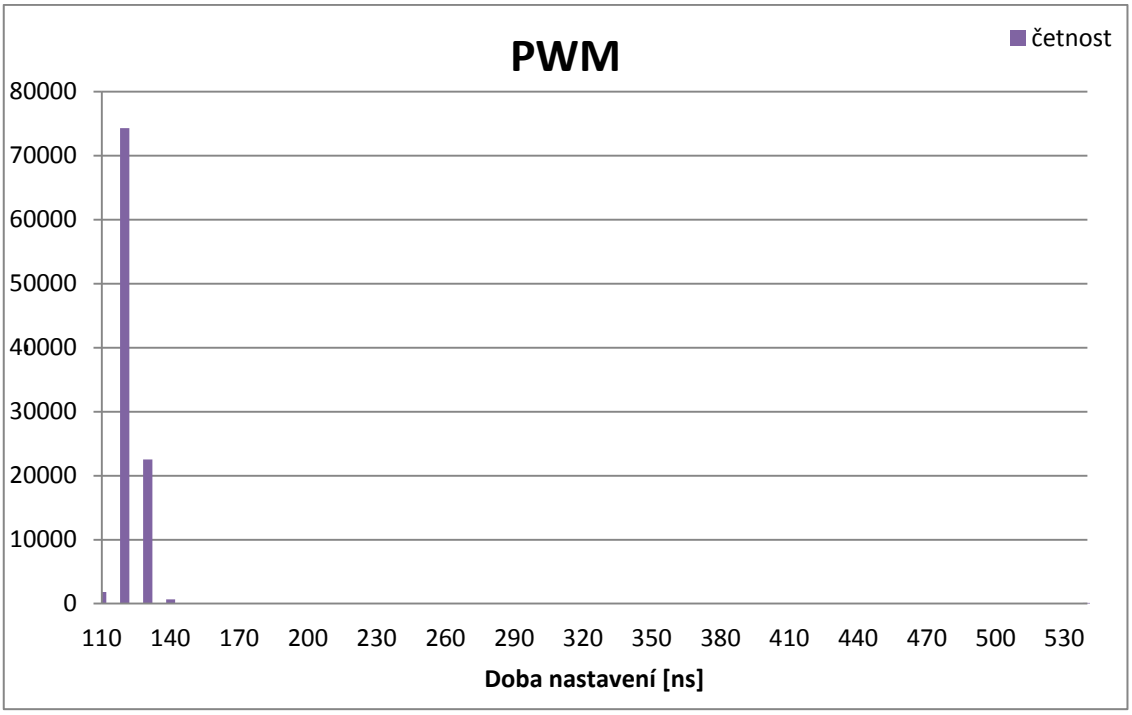
```
START                                          // Spuštění hodin
if((nova_hodnota * predchozi_hodnota) < 0) {   // Kontrola změny směru
        if(nova_hodnota > 0) {                 // Když směr A
                gpio_clr(PIN_1);               // Nastaví MOTB do log 0
                set_pwm_mode(PWM0_ENABLE);
        }else if(nova_hodnota < 0) {           // Když směr B
                gpio_set(PIN_1);               // Nastav MOTB do log 1
                set_pwm_mode(PWM0_ENABLE|PWM0_REVPOLAR);
        }
}

nova_hodnota = abs(nova_hodnota);              // Absolutní hodnot..
*pwm_data = nova_hodnota;                      // Nastaví hodnotu PWM
predchozi_hodnota = nova_hodnota;              // Uloží předchozí hodnotu
STOP                                           // Zastavení hodin
```

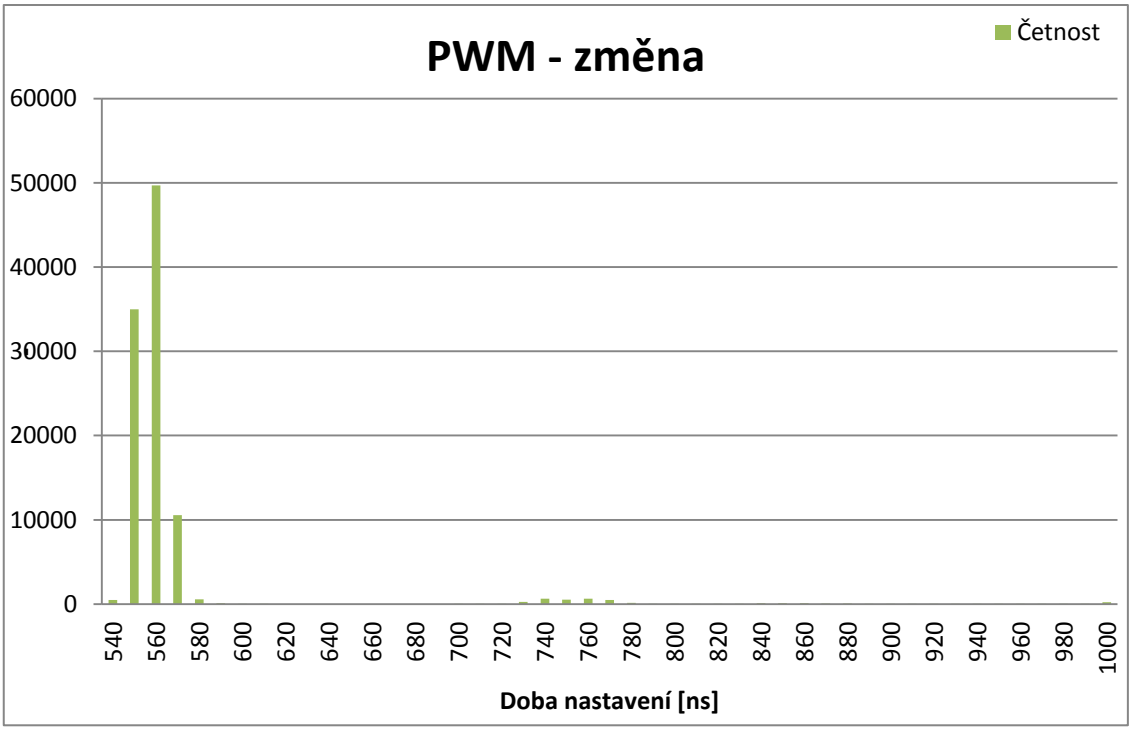
Zdroj. kód 6.12 Ukážka kód pro PWM měření

Test byl 100 000x opakován. Jelikož byl tento časový úsek velmi krátký a hraničil s minimální možnou měřitelnou dobou použitých funkcí, bylo nutné změřit více zápisů, aby byla režie měřící funkce zanedbána. Tím jsme opět odkázáni na průměrnou hodnotu.

Naměřené hodnoty v případě zachování směru jsou zobrazeny na Graf 6.4. Graf 6.5 zobrazuje naměřené hodnoty v případě změny směru.



Graf 6.4 PWM bez změny směru



Graf 6.5 PWM se změnou směru

6.4 Komplexní testy

Limity jednotlivých periferií již byly uvedeny. Nyní se podíváme, jak se RPi vypořádá se čtením a zápisem na větší množství periférií. Testováno bude následující:

- Test čtení z maximálního počtu periférii.
- Test zápisu na maximální počet periférii.
- Test zápisu a čtení.

Testy proběhnou dvakrát. Jednou bude čas měřen programem v C/C++ a podruhé za použití RPiDrv a REXu.

6.4.1 Test čtení

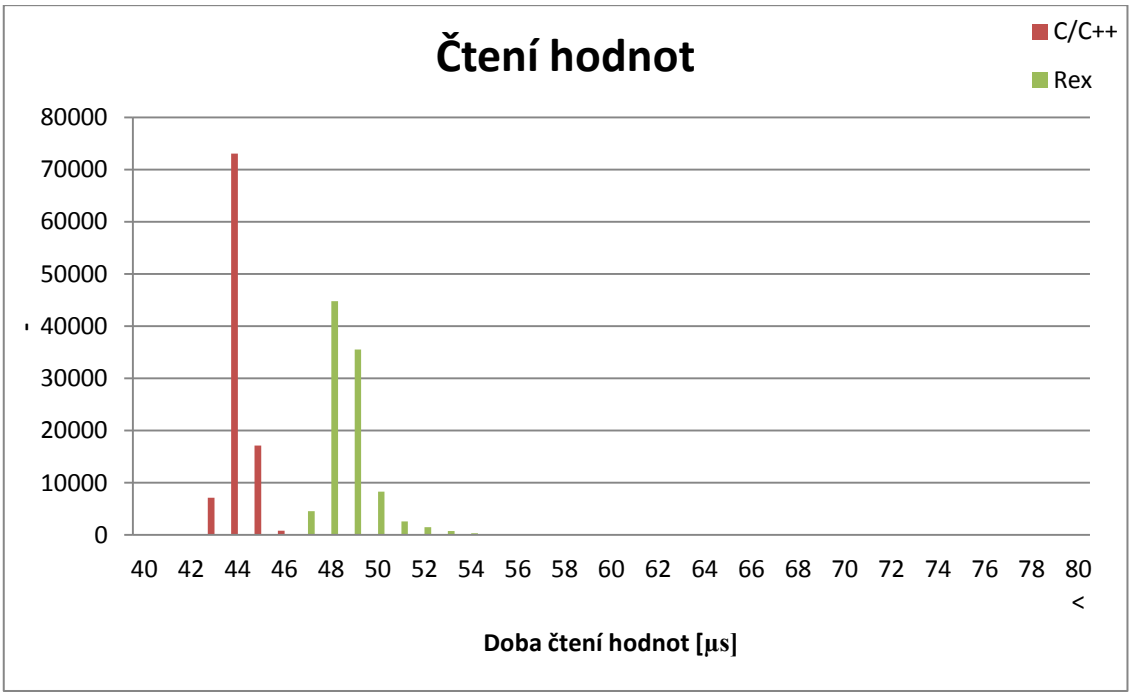
Čteno bylo z následujících periférii:

- GPIO 2, GPIO 3, GPIO 4, GPIO 14, GPIO 15, GPIO 17, GPIO 18, GPIO 22, GPIO 23, GPIO 24, GPIO 25 a GPIO 27 byly nastaveny jako vstupní piny.
- 2x A/D převodník, bylo čteno z obou kanálů.

Test byl měřen 100 000x.

Výsledky měření jsou zobrazeny na Graf 6.6. Z grafu je patrné, že je rychlejší program napsaný v C/C++. Program však nevykonává žádný užitečný kód. Nárůst 7 µs v případě REXu je velmi dobrý výsledek, vzhledem k jeho komplexnosti. REX je schopen získat data z maximálního počtu periférii do 54 µs v 98,5% procentech případů. Pouze 18 ze 100 000 naměřených hodnot bylo vyšších než je 80 µs. Maximální naměřená hodnota byla 256 µs. Naměřený čas přibližně odpovídá použitým periferiím. Vezmeme-li v potaz, že bylo čteno 2x z A/D převodníku tj. 2 x 21µs a 12x z GPIO tj. 12x306ns

$$2 * 21us + 12 * 0{,}306us = 45{,}67\ us$$



Graf 6.6 Čtení z více periférií

6.4.2 Test zápisu

Zapisováno bylo na následující periférie:

- GPIO 3, GPIO 4, GPIO 14, GPIO 15, GPIO 17, GPIO 22, GPIO 23, GPIO 24, GPIO 25 a GPIO 27 byly nastaveny jako výstupní piny.
- 2x D/A převodník, zapisováno bylo na oba kanály.
- 1x PWM, pro změnu směru byl použit GPIO 2. (Změna směru proběhla 1000x.)

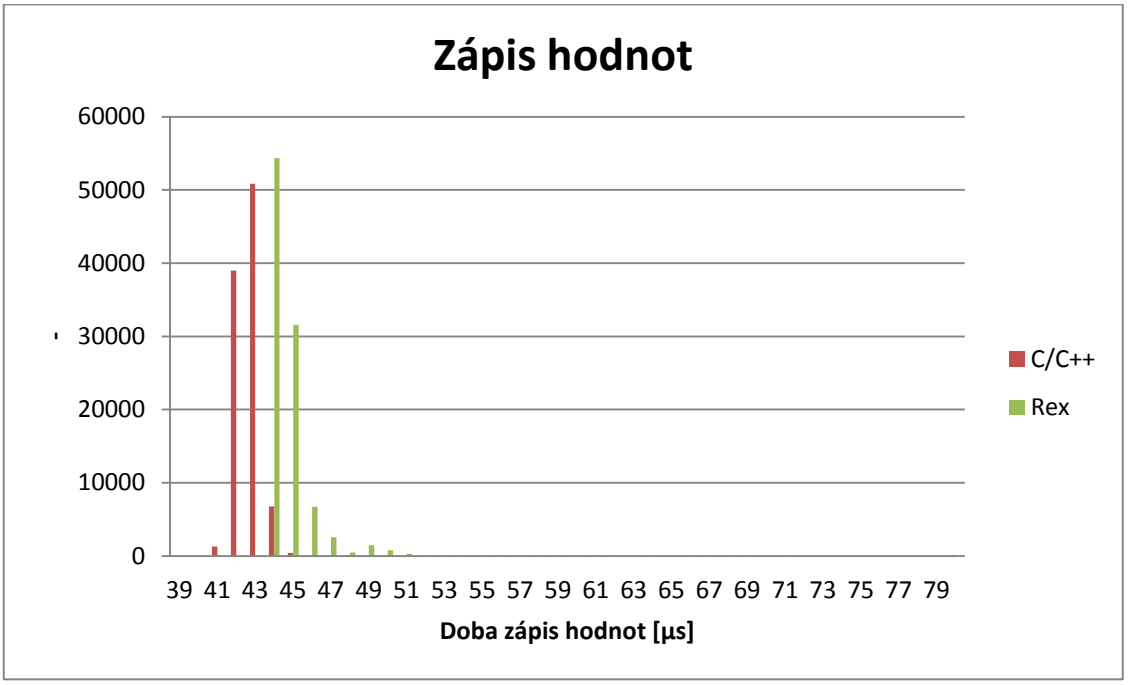
Test byl 100 000x opakován.

Výsledky měření jsou zobrazeny na Graf 6.7. Z grafu je patrné, že je rychlejší program napsaný v C/C++. Vysvětlení je stejné jako u testu čtení. REX je schopen zapsat data na

maximální počet periférii do 51 µs v 98,1% procentech případů. Pouze 13 ze 100 000 naměřených hodnot bylo vyšších než je 80 µs. Maximální naměřená hodnota byla 398 µs. Naměřený čas přibližně odpovídá použitým perifériím. Vezmeme-li v potaz, že bylo zapisováno 2x na D/A převodníku

tj. 2 x 21µs a 12x z GPIO tj. 10x158ns a 1x PWM (110 - 560) ns.

$$2 * 21us + 10 * 0{,}158us + 1x0{,}220 = 43{,}8\ us$$



Graf 6.7 Zápis na více periférií

6.4.3 Test zápisu a čtení

Nastavení bylo následující:

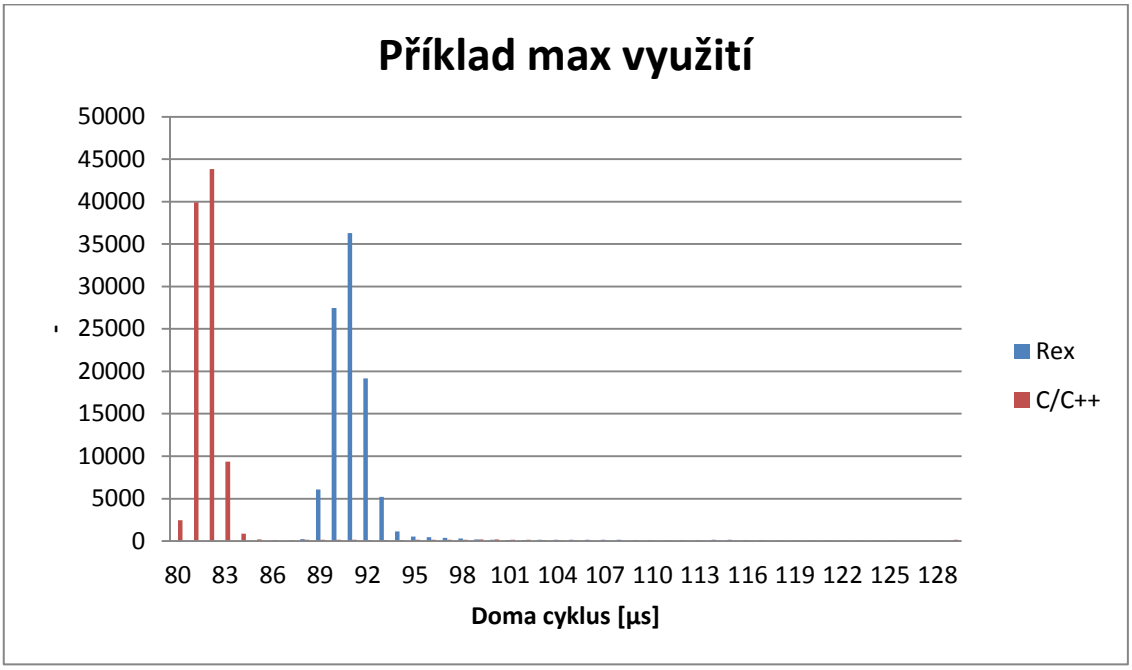
- GPIO 3, GPIO 4, GPIO 14, GPIO 15, GPIO 17 byly nastaveny jako vstupy
- GPIO 22, GPIO 23, GPIO 24, GPIO 25, GPIO 27 byly nastaveny jako výstupy
- 2x D/A převodník, zapisováno bylo na oba kanály.
- 2x A/D převodník, čteno bylo z obou kanálů.
- 1x PWM, pro změnu směru byl použit GPIO 2. (Změna směru proběhla 1000x.)

Test byl opakován 100 000x.

Výsledky jsou zobrazeny na Graf 6.8. Průměrná naměřená hodnota je dokonce lepší než by byl součet jednotlivých průměrných hodnot periférií. Tento jev je způsobem nejspíše optimalizací kódu kompilátorem.

$$4 * 21us + 5 * 0{,}158us + 5 * 0{,}306 + 1x0{,}220 = 86{,}54\ us$$

Výsledky REXu jsou následující. 97,6% případů bylo menších než 100 µs. Pouze 4 ze 100 000 přesahovali 128 µs. Maximální hodnota byla 703 µs.



Graf 6.8 Zápis / čtení z periférií

6.4.4 Shrnutí testů

Testy nám ukázaly, že REX, vzhledem ke své komplexnosti, nijak nezaostává za jednoúčelovým kódem a dosáhl velmi dobrých výsledků. Do 128 µs je schopen v 99,9% případů komunikace se všemi perifériemi. Bohužel se mezi naměřenými časy vyskytují i hodnoty, vlivem přerušení, větší než 700 µs. Je důležité si uvědomit, že i jedna vysoká hodnota nám znehodnotí měření.

Viz následující příklad:

Je třeba získávat data z čidel každých 100 µs, bude-li doba jednoho čtení, vlivem přerušení, 700 µs. Tato hodnota nám znehodnotí následujících šest měření. A zároveň i navrácena hodnota dat bude považována jako neplatná, protože nebyla přijata v časovém limitu. viz Tab. 6.1.

Měření 1	Měření 2	Měření 3	Měření 4	Měření 5	Měření 6	Měření 7	Měření 8	Měření 9	Měření 10
700 µs							100 µs	100 µs	100 µs
CHYBA!	CHYBA!	CHYBA!	CHYBA!	CHYBA!	CHYBA!	CHYBA!	OK!	OK!	OK!

Tab. 6.1 Příklad demonstrující chybu způsobenou přerušením

Dále je potřeba zdůraznit, že naměřené časy odpovídají pouze získání a zapsaní hodnot. V případě, že by REX obsahoval nějaký model, což nejspíše bude, celková naměřená doba by se prodloužila o dobu, jakou by REX příslušný model zpracovával. V předchozím textu byly uvedeny testy. Nyní se podíváme na demonstraci ovladače na laboratorních modelech.

7 Demonstrace ovladače

Ovladač byl odzkoušen na modelu kádinek nacházejících se v laboratoři UL511 a na motoru se zabudovaným inkrementálním rotačním čítačem firmy Maxon.

7.1 Model kádinek

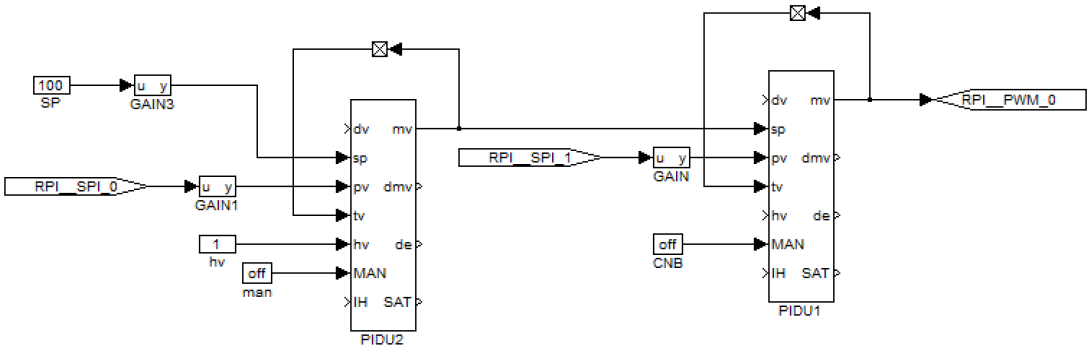
Jedná se o dvě nádoby, které jsou propojeny hadičkou ve spodní části. Každá nádoba obsahuje čidlo na snímání polohy hladiny. Model obsahuje čerpadlo, které je schopné načerpávat a odčerpávat vodu z první nádoby viz Obr. 7.1.

Obr. 7.1 Model kádinek

Cíl je následující: Za pomocí RPi, Gertboard a RExu regulovat hladinu druhé kádinky. Pro úlohu regulace bylo zvoleno kaskádní zapojení. Hladiny obou kádinek jsou na sobě závislé a tím, že budeme regulovat i první kádinku, získáme lepší výsledky, než kdyby byla regulováno pouze hladina druhé kádinky. Druhým důvodem bylo částečně zvýšit výpočetní náročnost úlohy a snažit se tím přiblížit reálným aplikacím.

Model byl připojen k RPi pomocí rozšiřující karty Gert Board. Čidla snímající polohu byla připojena pomocí A/D převodníků připojených k RPi pomocí SPI. Napěťový rozsah čidel byl 0–5V. Bylo třeba použít napěťových děličů pro převedení na rozsah

měřitelný A/D převodníkem. Pro řízení motoru byla použita PWM a motor kontroler na desce Gert Board. Schéma v REXu je zobrazeno na Obr. 7.2. Délka jednoho cyklu byla v REXu nastavena na 1ms.



Obr. 7.2 Schéma modelu kádinek

Výsledky měření jsou zobrazeny v Tab. 7.1. Z tabulky je vidět, že výpočetní doba úlohy je průměrně 87 µs, což je poměrně slušný výsledek, přihlédneme-li k tomu, že přibližně 40 µs trvá získání dat z A/D převodníků. Avšak byla naměřena i doba 491 µs. Vezmeme-li v potaz, že délka cyklu v REXu byla nastavena na 1ms, je zde ještě časová rezerva. Bohužel nastaly i případy, kdy 1ms byla překročena o 316 µs.

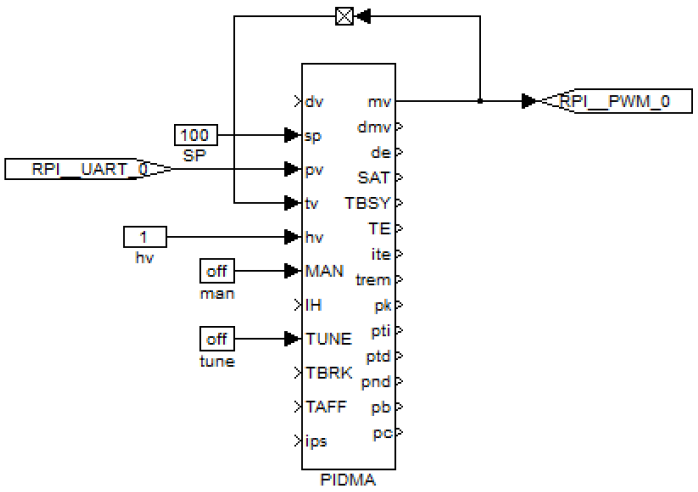
Počet cyklů	2981768
Minimální čas úlohy	76µs
Maximální čas úlohy	491µs
Průměrný čas úlohy	87µs
Maximální překročení cyklu	316µs

Tab. 7.1 Naměřené časy modelu kádinek

7.2 Motor Maxon

Druhým modelem je motor značky MAXON (M061446 001). Motor obsahuje inkrementální rotační čítač a původním cílem bylo řízení otáček motoru dle integrovaného čítače. Čítač disponuje dvěma kanály pro určení směru a třetí kanál určující jednotlivé otáčky. Na jednu otáčku je potřeba zaznamenat 1000 značek, tj. 500 z každého kanálu. Pokud by se motor točil dvěma otáčkami za sekundu, což je poměrné nízká hodnota otáček, bylo by potřeba zaznamenat během sekundy dva tisíce značek. Tedy za každých 500 µs přichází nová hodnota. Z měření v kapitole 6.1 víme, že vlivem přerušení může nastat "hluché místo" o délce několika set µs, během kterých není možné zaznamenat příchozí značku. Z toho vyplývá, že i při velmi nízkých otáčkách

není možné 100% zaručit správnou funkci čítače na RPi. Situace se částečně vylepší využitím ATmega. Avšak je-li zaktivována funkce pro přenesení hodnoty z ATmega na RPi rozhraními I2C, nebo SPi, je opět rychlost snížena. Z těchto důvodů budou otáčky motoru regulovány pouze pomocí třetího kanálu, který výrazně zmenší nároky na použitý hardware. Model v REXu je zobrazen na Obr. 7.3. I nyní byla délka cyklu nastavena na 1ms.



Obr. 7.3 Schéma modelu motoru

Je třeba vzít rovněž v potaz, že běh programu spočíval pouze v čítaní značek, nebyl prováděn žádný smysluplný výpočet, který by režii programu navýšil a konečný výsledek by se zhoršil. Pro testování bylo zvoleno čítání otáček pomocí ATmega.

Motor byl připojen k RPi pomocí přídavné desky Gert Board. K ovládání otáček je použit motor kontroler a jako zpětná vazba slouží IRC připojené k ATmega, jak již bylo uvedeno, byl použit pouze kanál pro otáčky.

Naměřené hodnoty jsou zobrazeny v Tab. 7.2. Průměrný čas běhu úlohy byl 113 µs. I v tomto případě docházelo k překročení doby cyklu a to o 113 µs.

Počet cyklů	1529723
Minimální čas úlohy	97µs
Maximální čas úlohy	563µs
Průměrný čas úlohy	113µs

Tab. 7.2 Výsledky měření motor Maxon

Z měření na reálných objektech jsme získali zajímavé poznatky, a to, že RPi je schopno řídit jednodušší procesy v řádu ms. Za předpokladu, že bychom se smířili s občasným překročením, lze s nadsázkou říci, že u 99% případů, se lze s RPi dostat na hranici 1ms.

8 Možnost připojení senzorů k RPi

K Raspberry Pi je možné připojit celou řadu různých čidel a snímačů. Dalo by se říci, že kdybychom vybrali libovolné čidlo, tak ho nalezneme v provedení, aby bylo schopno komunikovat s RPI. A to z prostého důvodu. RPi je schopné komunikovat pomocí sběrnice SPI, I2C a UART, přičemž drtivá většina senzorů používá právě tyto technologie.

Mezi základní senzory patří bezesporu A/D a D/A převodníky. Většina fyzikálních převodníků převádí měřenou veličinu právě na elektrickou. Ať již vezmeme za příklad čidlo teploty, tlaku, či průtoku, je fyzikální veličina převáděna na elektrickou, a to v analogové podobě. A/D a D/A převodníky je možné pořídit s libovolnou sběrnicí (s větším či menším výběrem) a proto není RPi nijak omezeno co se týče těchto převodníků.

Mezi další senzory bych zařadil inkrementální rotační čítač neboli IRC. IRC je možné zakoupit v několika variantách, podle počtu značek na otáčku. Ale také, a to především, jakým napěťovým úrovním budou odpovídat hodnoty logické. Jelikož RPi využívá 3,3V logiku, bylo by vhodné, aby ji mělo i IRC. Ve své podstatně by zde neměl být žádný problém mezi propojením RPI a IRC. RPi umí rozeznat logickou 0 od logické 1 bez libovolného mezičlenu. Může nastat situace popsaná v kapitole 7.1, kdy RPi nemusí zaznamenávat všechny značky odeslané IRC. K RPi je možné zakoupit enkodér, který čítá značky a aktuální hodnotu je schopen poskytnout pomocí výše uvedených sběrnic. Jako je například v http://www.cui.com/amt-modular-encoders.

Mezi senzory bych ještě zařadil i typy, které mají vlastní řídicí jednotku, jako jsou například řídicí jednotky motorů či měřice vzdáleností. Ať už akustické nebo laserové. S jednotkami je možné komunikovat pomocí UART. Avšak je potřeba dát pozor na souhlasné napěťové úrovně. UART je povětšinou spjat s pěti voltovou logikou. Proto je nutno zařadit převodník úrovní mezi RPi a jednotku s níž je komunikováno.

Velkou výhodou RPi je možnost využití přerušení, které nám může aktivovat libovolnou funkci. Tato výhoda je nesporná, jelikož na přerušení nemusí program aktivně vyčkávat, ale je upozorněn, což vede k vysoké úspoře spotřebované energie, která bude oceněna hlavně při napájení z bateriových článků. Bohužel možnost využití přerušení není v datasheetu (Broadcom 2012) příliš zdokumentována.

Mimo jiné je také možné, jako senzor nebo pro záznam zvukového záznamu, užít PCM, jež je rovněž součástí RPi. Je možné použít i dvoukanálový záznam.

Mimo to je také možné připojit i daleko sofistikovanější zařízení. Jelikož je jádrem RPi operační systém Linux, není problém připojit webkameru nebo kinect, což nám ve spojení s již zmíněnými senzory rozšiřuje pole působnosti.

Praktických uplatnění pro RPi připojeného k senzorům je celá řada. Když si shrneme vlastnosti RPi, kterými jsou malé rozměry, absence mechanických částí, malá hmotnost, jednoduché připojení snímacích senzorů všech druhů, slušný výpočetní výkon, nízká spotřeba, možnosti připojení do sítě Ethernet a operační systém Linux.

Jako příklad bych uvedl řízení osvětlení v domácnosti přes webové rozhraní a mobilní telefon.[5]

[5] http://lifehacker.com/control-your-home-lights-with-a-raspberry-pi-754987546

9 Závěr

V diplomové práci jsme byli seznámeni s platformou Raspberry Pi. Dále jsme se dozvěděli o základech komunikace mezi RPi a jejími periferiemi, přístupu do registrů a možnými úskalími, se kterými bychom se mohli setkat. Dále jsme získali informace o rozšiřující kartě Gert Board, o jejím připojení k RPi a k inovacím, které s sebou tato karta přináší. Následně byly popsány jednotlivé prvky karty a na schématech bylo znázorněno jejich propojení s Raspberry Pi. V případě převodníků byl popsán způsob, jakým se na převodníky získávala požadovaná hodnota. U mikrokontroleru ATmega byly popsány základní kroky k instalaci softwaru umožňujícího programovat pro tento mikrokontroler. Také byl uveden jednoduchý program popisující základní prvky programování pro ATmega.

V další části diplomové práce jsme byli ve stručnosti seznámeni s řídicím systém REX a knihovnou RPiDrv, která umožňuje REXu komunikaci s periferiemi, jimiž disponuje Raspberry Pi. Také jsme se dočetli, jak vytvořit konfigurační soubor s nastavením všech periférií v grafickém programu RPiCfg. Samozřejmostí bylo i představení právě zmíněného RPiCfg, jeho spuštění a detailní ovládání. Jelikož ovladač nebyl převzat, ale byl navržen autorem této diplomové práce, nevěnoval jsem se logicky popisu vývoje ovladače. Namísto toho jsme seznámeni s funkcemi ovladače a možným nastavení, jež ovladač přináší.

Sérií testů byly otestovány jednotlivé periférie. U každého testu nám bylo demonstrováno, jak byl test proveden a následně nám byly předvedeny výsledky měření. V histogramech jsme mohli nahlédnout jak RPi v testech obstálo, tj. byly zobrazeny maximální a minimální naměřené časy s nejčastější hodnotou.

Samozřejmostí bylo předvedení ovladače na reálných modelech. Bylo popsáno, jaké prvky byly využity k propojení s Raspberry Pi, případně s rozšiřující deskou Gert Board.

V závěru práce byly ve stručnosti prodiskutovány možnosti připojení dalších senzorů, které by rozšířily využitelnost Raspberry Pi.

Na úplný závěr bych ještě uvedl celkové shrnutí, co se týče Raspberry Pi a desky Gert Board. Velkou výhodou Raspberry Pi je spojení sofistikovaného operačního systému Linux s periferiemi, které jsou schopny komunikovat nízkoúrovňovými jednotkami. Na druhou stranu jsou zde i nevýhody, které sebou přináší Linux, a to jsou vysoké latence spojené s během tohoto operačního systému. Pro případné nasazení RPi do reálné úlohy je potřeba důkladné zvážit, jak rychlou odpověď od řízeného systému

požadujeme. Co se týče desky Gert Board, jde spíše o demonstrování funkcí Raspberry Pi, než o nasazení v reálné aplikaci.

Obecně vzato, platforma představuje přínos pro řízení jednodušších průmyslových zařízení, která nevyžadují vysoký výkon a mohla by zároveň být použita například při výuce průmyslového řízení, ale i v jiných oblastech využívajících výpočetní technologie.

Seznam literatury

Atmel Corporation (2012). *Atmel 8-bit Microcontroller with 4/8/16/32KBytes In-System Programmable Flash* (http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet.pdf, 20. 5. 2013).

Broadcom Corporation (2012). *BCM2835 ARM Peripherals* (http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf, 20. 5. 2013).

Broadcom (2013). *High Definition 1080p Embedded Multimedia Applications Processor* (http://www.broadcom.com/products/BCM2835, 20. 5. 2013).

Gordons Projects (2013). *Arduino IDE Installation* (https://projects.drogon.net/raspberry-pi/Gert Board/arduino-ide-installation-isp/, 20. 5. 2013).

Microchip Technology Inc. (2010). *MCP4802/4812/4822. 8/10/12-Bit Dual Voltage Output Digital-to-Analog Converter with Internal VREF and SPI Interface* (http://ww1.microchip.com/downloads/en/devicedoc/22249a.pdf, 20. 5. 2013).

ULN2803a (1997). *ULN2803A DARLINGTON TRANSISTOR ARRAY* (http://www.ti.com/lit/ds/symlink/uln2803a.pdf, 1. 1997)

L6203 (1997). *L6201- L6202 - L6203 DMOS FULL BRIDGE DRIVER* (http://pdf.datasheetcatalog.com/datasheet/stmicroelectronics/1373.pdf, 1. 1997)

MCP3002 (2007). Dual Channel 10-Bit A/D Converterwith SPI™ Serial Interface MCP3002 (http://ww1.microchip.com/downloads/en/DeviceDoc/21294C.pdf, 2007)

Gert Board user manunal - Gert van Loo and Myra VanInwegen (Gertboard_User_Manual_Rev_1 0_F.pdf, 2012)

Gertboard Assembly Manual - Element14 (Gertboard_Assembly_Manual_Rev1.1_F.pdf, 2012)