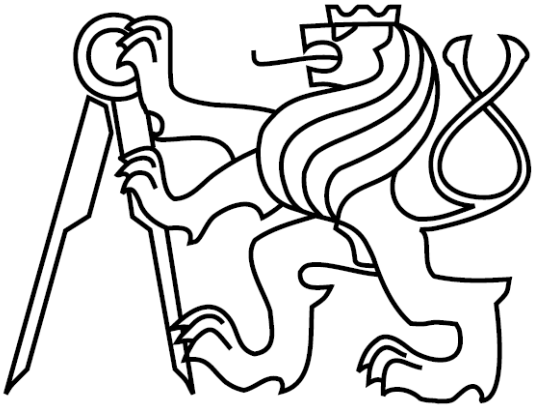
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická



Diplomová práce

Knihovna funkcí pro počítač RASPBERRY PI

Autor: Bc. Tomáš Procházka

Vedoucí práce: Ing. Pavel Kubalík Ph.D.

Praha 2016

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Tomáš Procházka**
Studijní program: Otevřená informatika
Obor: Počítačové inženýrství
Název tématu: **Knihovna funkcí pro počítač RASPBERRY PI**

Pokyny pro vypracování:

1. Vytvořte knihovnu funkcí v programovacím jazyce C pro ovládání jednotlivých periferií počítače Raspberry PI bez operačního systému.
2. Zaměřte se na ovládání univerzálních vstupů a výstupů, rozhraní UART, SPI, I2C, časovačů, řadiče přerušení, pulsně šířkového modulátoru, řadiče SD/MMC karet a ethernetové rozhraní.
3. Implementujte sadu funkcí pro zobrazovaní dat na obrazovce připojené přes HDMI.
4. Pro otestování jednotlivých funkcí knihovny napište demonstrační aplikaci využívající grafický displej, ethernetové rozhraní a SD kartu.

Seznam odborné literatury:

[1] http://www.raspberrypi.org/
[2] http://www.raspi.cz/

Vedoucí: Ing. Pavel Kubalík, Ph.D.
Platnost zadání: do konce letního semestru 2016/2017

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 15. 10. 2015

Čestné prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Datum: 4. 8. 2016

.....

podpis autora práce

Poděkování

Rád bych poděkoval vedoucímu diplomové práce Ing. Pavlu Kubalíkovi, Ph.D. za vedení mé práce a za čas, který mi věnoval při konzultacích. Dále pak celé mé rodině, která mě podporovala.

Abstrakt:

Tato diplomová práce se zabývá vytvořením knihovny pro ovládání jednotlivých periférií počítače Raspberry PI bez použití operačního systému. Důraz je kladen hlavně na ovládání univerzálních vstupů a výstupů, rozhraní UART, SPI, I2C, časovačů, řadiče přerušení, pulsně šířkového modulátoru, řadiče SD/MMC karet a ethernetové rozhraní. Knihovna funkcí bude napsaná v programovacím jazyce C. Pro otestování funkcí knihovny je zapotřebí napsat demonstrující aplikaci, která bude využívat grafický displej, ethernetové rozhraní a SD kartu. Na PC bude vytvořeno grafické uživatelské rozhraní pro ověření komunikace pomocí ethernetového rozhraní. Grafické uživatelské rozhraní bude vytvořeno v jazyce C++. Dále budou vytvořeny i další ukázkové aplikace pro manipulaci s ostatními periférierni na Raspberry PI.

Abstract:

This diploma thesis deals with creating a library for controlling individual peripherals computer Raspberry PI without using the operating system. Emphasis is placed mainly on controlling universal inputs and outputs, UART, SPI, I2C, timers, interrupt controller, pulse width modulator, controller SD/MMC cards and ethernet interfaces. The library functions will be written in the C programming language. For testing the function of the library is needed write demonstrating application that will use graphic display, ethernet interface and SD card. On the PC will be created graphical user interface for checking communication via ethernet interface. The graphical user interface will be created in C++. There will also be created next sample applications for manipulation with other peripherals on Raspberry PI.

Obsah

Seznam tabulek

Seznam obrázků

SEZNAM POUŽÍTÝCH ZKRATEK

ARM	Advanced RISC Machines
BCM	Broadcom
CPU	Central processing unit (Centrální procesorová jednotka)
CRC	Cyclic redundancy check (Cyklický redundantní součet)
DC	Direct current (Stejnosměrný elektrický proud)
DSI	Display Serial Interface (Sériové rozhraní pro displej)
GPIO	General purpose input/output (Univerzální vstupy a výstupy)
GPU	Graphic processing unit (Grafická procesorová jednotka)
HDMI	High Definition Multimedia Interface
I2C	Inter-Integrated Circuit
LSB	Least significant bit (Nejméně významný bit)
MAC	Media Access Control
MMC	MultiMediaCard
MSB	Most significant bit (Nejvíce významný bit)
OS	Operating systém (Operační systém)
PID	Packet identifier (Identifikátor paketu)
PNG	Portable Network Graphics (Přenosná síťová grafika)
PWM	Pulse Width Modulator (Pulzně šířková modulace)
RCA	Radio Corporation of America (Americká rozhlasová společnost)
RISC	Reduced Instruction Set Computing
RPI	Raspberry PI
SD	Secure Digital
SDRAM	Synchronous Dynamic Random Access Memory (Synchronní Dynamická paměť)
SoC	System on a Chip (Systém na čipu)
SOF	Start of frame (Začátek rámce)
SPI	Serial Peripheral Interface (Sériové periferní rozhraní)
SSD	Solid-state drive
TTL	Tranzistor-tranzistor logic (Tranzistorově-tranzistorová logika)
UART	Universal asynchronous receiver/transmitter (Univerzální asynchronní přijímání/ vysílání)
USB	Universal Serial Bus (Univerzální sériová sběrnice)
UTP	Unshielded Twisted Pair (Nestíněná kroucená dvojlinka)

1 Úvod

V dnešní době se stále více setkáváme se zařízeními, které jsou určené pouze pro jeden účel. Takové zařízení se obecně označuje jako vestavěný systém (embedded system). Tyto zařízení jsou určené pro předem definovanou činnost, takže vývojáři zařízení mohou optimalizovat pro konkrétní aplikaci a tak snížit cenu výrobku na minimum. Aby bylo dosaženo minimální ceny pro konečného zákazníka, jsou vestavěné systémy vyráběny sériové ve velkém množství. Z vestavěnými systémy se můžeme setkat v mnoha odvětví průmyslu jako je lékařství, letectví, automotive nebo i třeba v domácnosti. Při návrhu konkrétního vestavěného systému je jeden z důležitých parametrů spolehlivost, nízká spotřeba nebo ochrana v případě závady na zařízení. Nízká spotřeba je důležitá, pokud je zařízení v neustálém chodu (je připojené do sítě nebo je napájené z baterií) a snažíme se tak zredukovat odebíranou energii ze zdroje. Pokud by mělo dojít k tomu, že je ohrožen lidský život, vestavěný systém musí na to pamatovat a ukončit tak chod svého systému. Vestavěné systémy mohou mít různé rozměry a tedy být různě velké.

Příklady vestavěných systémů, které lze nalézt v domácnost:

- Automatická pračka
- WiFi router
- Autíčko na dálkové ovládání
- Videokamera, Fotoaparát

Takový vestavěný systém může být tedy použit i Raspberry PI.

Cílem diplomové práce bude:

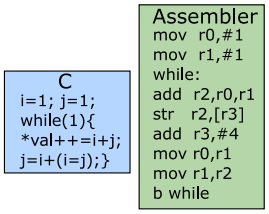
- Seznámením se s Raspberry PI a vybrání vhodné verze aby splňovalo zadání této práce, jelikož v zadání není předem specifické jaká verze Raspberry PI má být použita
- Prostudovat existující knihovny pro RPI v programovacím jazyce C a poté navrhnout jejich vylepšení nebo doplnění
- Vytvoření finální knihovny funkcí v programovacím jazyce C pro zadané periférie bez použití operačního systému pro Raspberry PI
- Napsání ukázkových aplikací pro jednoduché otestování jednotlivých periférii
- Vytvoření finální aplikace a její následné otestování, jenž bude využívat Ethernet, SD kartu a grafický displej připojený přes HDMI.

2 Rešerše

Do této doby jsem se osobně nesetkal s tímto jednodeskovým počítačem. Při školní výuce jsem využíval jiné vývojové kity většinou od firmy STM[1]. Firma STM, která prodává tyto kity, má na svých stránkách i knihovny, které jsou zapotřebí a tak vyvíjení je ulehčené a stačí si pak stáhnout tyto knihovny pro periférii, kterou chceme ovládat. Knihovnu stačí pak zkompilovat a nahrát do kitu, většinou pomocí USB.

U RPI takové knihovny nejsou, na jejich oficiálních stránkách ke stažení. Jedním z možných způsobů je nejdříve nainstalování operačního systému jako je Raspbian (založený na Debianu), který už v sobě má ovladače pro ethernet, USB, SD kartu a displej. Poté si stačí stáhnout knihovnu pro GPIO[2] a zadání je hotové. Ale v mém zadání nelze použít žádný OS a tak tento způsob není možný a musel jsem se ubírat jiným směrem.

Existuje velká komunita vývojářů, kteří se věnují tomuto programování bez pomoci OS (anglicky „bare metal''). Tito vývojáři programují svoje knihovny v programovacím jazyce C, C++ nebo pomocí nízkoúrovňového jazyka Assembly (Assembler), což je programovací jazyk strojových instrukcí, který procesor umí zpracovat. Ukázka takového kódu:



Obrázek 2.1: Ukázka kódu v jiných programovacích jazycích

Existují tedy knihovny, které nabízejí určité funkce. Tyto knihovny, ale buď nejsou dlouhodobě aktualizované (to znamená, jak se vyvíjí verze RPI nelze tyto knihovny, použít pro novější verze RPI) nebo nabízejí pouze určité a tudíž se opakující funkce pro jednoduché periferie (GPIO,UART). Proto bylo dobré vzít nějakou knihovnu a doplnit jí o potřebné periférie aby bylo splněno zadání této práce. Po dlouhém hledání jsem našel knihovnu "USPI", která nabízí funkce pro komunikaci pomocí USB přes který je připojen i ethernet. Dále nabízí i funkce pro komunikaci s jinými zařízeními připojenými přes USB (myš, klávesnice, flashky) tyto funkce ale nejsou součástí mé práce. Uvedená knihovna, která je napsaná programovacím jazyce C (což vyžaduje zadání) sama o sobě nenabízí komunikaci s ostatními periférierni. Knihovna byla přepsaná z jazyka C++ (kvůli jednoduchosti), takže její zdrojové kódy mohou vypadat trochu divně a bylo by dobré je přepsat, aby odpovídaly více programovacímu jazyku C. Výhoda této knihovny je v tom, že pro svoje vlastní funkce nepotřebuje jiné další knihovny.

[1] STM-http://www.st.com/content/st_com/en.html
[2] WiringPi http://wiringpi.com/-knihovna pro GPIO na RPI

3 Teoretický základ

3.1 Timer

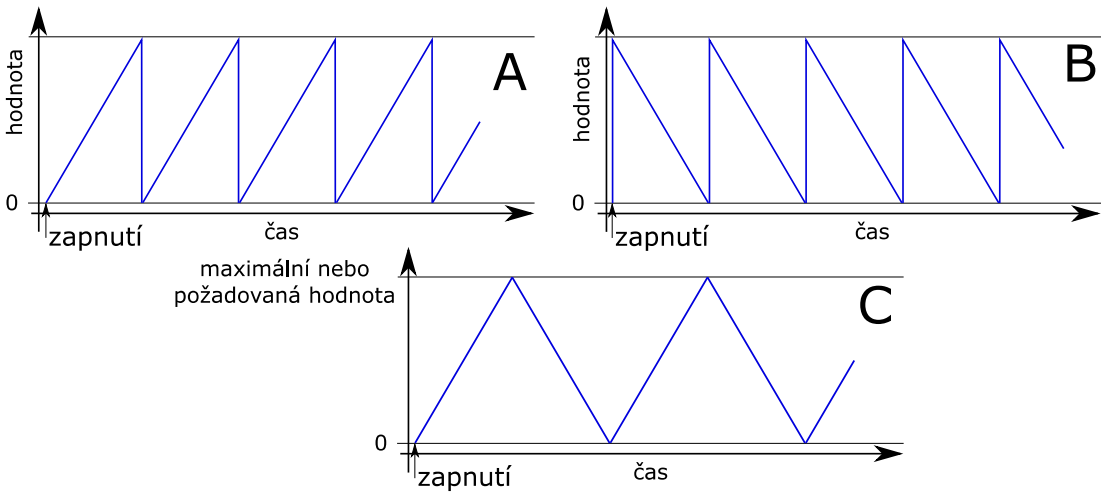
Timer [7] se používá pro odměření určitých časových intervalů nebo pro vytvoření události po uplynutí času. Při odměřování času může buď mikroprocesor čekat a počítat zpoždění, což má za následek, že není prováděný jiný kód programu (při základním nastavení určitého hardwaru je to někdy zapotřebí než lze dále komunikovat s tímto hardwarem) nebo použití druhého způsobu, kdy se nastaví timeru požadovaný čas a až dojde k uplynutí času, pak timer vyvolá přerušení mikroprocesoru a požadovaný kód se provede v obsluze přerušení (například rozsvícení/zhasnutí LED diody). Ve druhém způsobu použití timeru, může mikroprocesor vykonávat jiný kód programu a nemusí tedy čekat.

V této kapitole se bude setkávat se dvěma pojmy:

- Čítač (Counter) – čítá vnější impulzy
- Časovač (Timer) – čítá vnitřní – synchronně přicházející – impulsy (obvykle hodinový signál procesoru), základem pak tvoří čítač

Čítač umožňuje 3 režimy čítání [Obrázek 3.1:]:

- A - čítání nahoru
- B - čítání dolů
- C - střídavě čítání nahoru a dolů



Obrázek 3.1: 3 druhy režimů čítání counteru

U čítače lze někdy nastavit hodnotu od/do jaké má čítat. Jakmile dosáhne této hodnoty nebo nuly v případě, že čítač čítá dolů, dojde k resetování čítače a začne čítat zase od/do požadované hodnoty. Pokud tato hodnota nelze nastavit čítač čítá do své maximální hodnoty (čítání nahoru) nebo nuly (čítání dolů) a poté dojde k přetečení a čítač začne zase od začátku.

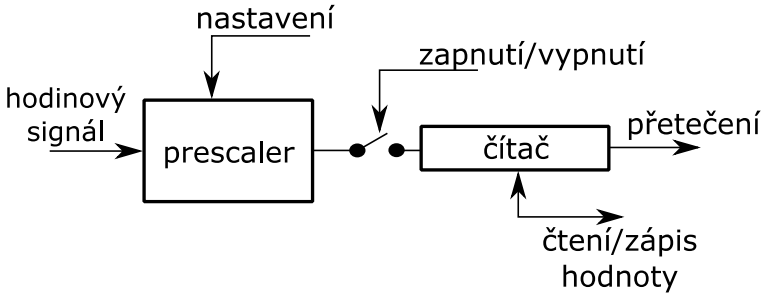
Jak již bylo zmíněné na začátku této kapitoly, základem časovače je čítač. Čítač může být 8 bitový, 16 bitový, 32 bitový nebo 64 bitový. Čítač čítá příchozí pulzy, jakmile dojde k jeho zapnutí. Zapnutí nebo vypnutí lze provádět programově. Dojde-li k jeho zastavení, čítač uchová hodnotu a poté co je opět zapnutý počítá od této hodnoty. Zapojení dále může obsahovat i prescaler, což je předdělička frekvence, která může nastavit jinou rychlost hodinového signálu pro čítač, nezávisle na rychlosti hodin. Hodnoty pro předděličku mohou být pevně dané (obvykle mocnina 2) 1,8,16,64,256 nebo může obsahovat registr, do kterého je možné napsat hodnotu pro předděličku. Tento registr má, ale omezenou svoji velikost.

Pokud systémové hodiny jsou například 1MHz a my potřebujeme frekvenci 1kHz, tak v případě že předdělička obsahuje registr, nastavíme hodnotu na 999.

Neboť výsledná frekvence pro čítač je vypočítaná:

$$\mathrm{tim}_{\mathrm{clock}} = \mathrm{sys}_{\mathrm{clock}}/(\mathrm{prescale} + 1)$$

Pokud ale předdělička obsahuje pouze pevné hodnoty (obvykle mocnina 2), nepodaří se nám nastavit požadovanou frekvenci, ale můžeme nastavit přibližnou.



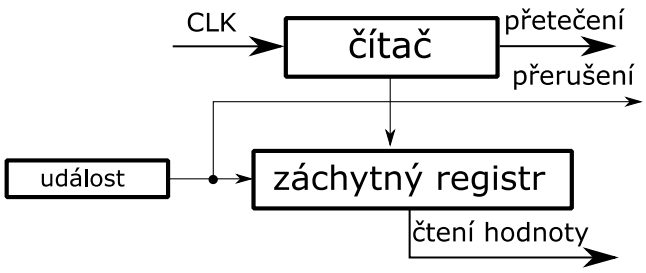
Obrázek 3.2: Základní zapojení časovače

Čítač umožňuje rozšířené režimy:

- Záchytný režim čítače (input capture)
- Čítač v režimu (output compare)

Záchytný režim čítače (input capture)

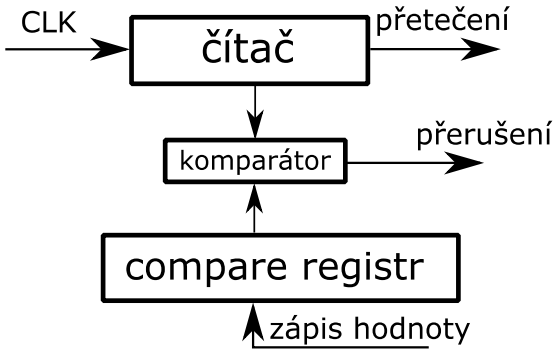
Pomocí události dojde k tomu, že obsah čítače se přepíše do záchytného registru a dojde k přerušení. Čítač se nijak nezastaví a čítá pulzy dále. Příkladem tohoto režimu jsou například stopky, kdy zmáčknutím tlačítka na stopkách se zachytí čas, ale stopky běží dále.



Obrázek 3.3: Čítač v režimu input capture

Čítač v režimu (output compare)

Do compare registru zapíšeme požadovanou hodnotu. Čítač čítá pulzy a porovnává se hodnota čítače s hodnotou v compare registru pomocí komparátoru. Jakmile čítač dosáhne stejné hodnoty jako v registru dojde k přerušení. Tento režim může obsahovat například nastavení času budíku v chytrém telefonu.



Obrázek 3.4: Čítač v režimu output compare

Základní nastavení čítače/časovače:

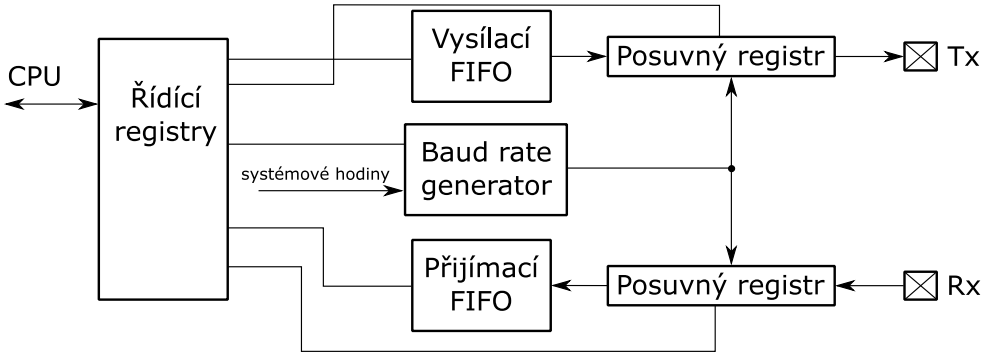
- Nastavení řídících registrů
- Vložení počáteční hodnoty
- Nastavení přerušení (volitelné)
- Zapnutí čítače - od této doby čítač čítá impulzy, z příchozí náběžnou hranou hodinového signálu, a pravidelně se zvětšuje nebo zmenšuje o jedničku

3.2 UART

UART řadič je klíčovou součástí pří sériové komunikaci mezi počítačem (nebo i nějakým modulem) a nějakým vestavěným systémem (například RPI), který obsahuje nízko úrovňové periférie. UART vyjádří jednotlivý bajt (znak) pomocí osmi bitů, které pak postupně vysílá bit za bitem po vodiči. Na koncové straně pak opět UART sestaví bity do kompletního bajtu.

UART je tvořen obvykle následujícími součástkami, jak zobrazuje další obrázek [Obrázek 3.5:].

- Generátor hodin
- Vstupní a výstupní posuvný registr
- Ovládání vysílací a přijímací
- Čtecí a zapisovací řídící logika
- Vysílací a přijímací FIFO



Obrázek 3.5: Základní blokové schéma UART

Existují dva druhy sériového přenosu:

- Asynchronní
- Synchronní

Asynchronní přenos znamená, že pro přenos dat není zapotřebí žádný externí hodinový signál jako v případě synchronního přenosu. Synchronní přenos je, co se týče do rychlosti obvykle efektivnější než asynchronní přenos, jelikož obsahuje pouze datové bity, které jsou přenášeny mezi vysílačem a přijímačem. Asynchronní způsob přenosu je ideální pro minimalizaci požadovaných drátů a I/O pinů, ale znamená to, že musíme zajistit, aby přenos byl spolehlivý. Místo použití hodinového signálu se vysílač a přijímač musí dohodnout předem na tom, kdy daná data jsou platná. Každým posíláním dat (obvykle 8 bitů) se před data vloží start bit a za daty stop bit [Obrázek 3.6:]. Tím vznikne výsledný rám, který se odešle. Startovní bit slouží k přípravě přijímání dat neboli k synchronizaci a stop bit slouží k ukončení přijímání dat.

Obrázek 3.6: Asynchronní přenos dat

Jelikož se jedná o univerzální přenos lze měnit 4 parametry: přenosová rychlost, počet datových bitů, počet stop bitů a jestli obsahuje paritní bit. Přenosová rychlost u asynchronní komunikace se udává v baudech. Baud je jednotka používaná k měření rychlosti přenosu dat, přičemž udává počet změn signálu za sekundu. Čím vyšší je přenosová rychlost, tím rychleji jsou data odeslána nebo přijata. Znaky se posílají od nejméně významného bitu (LSB) po nejvíce významný bit (MSB).

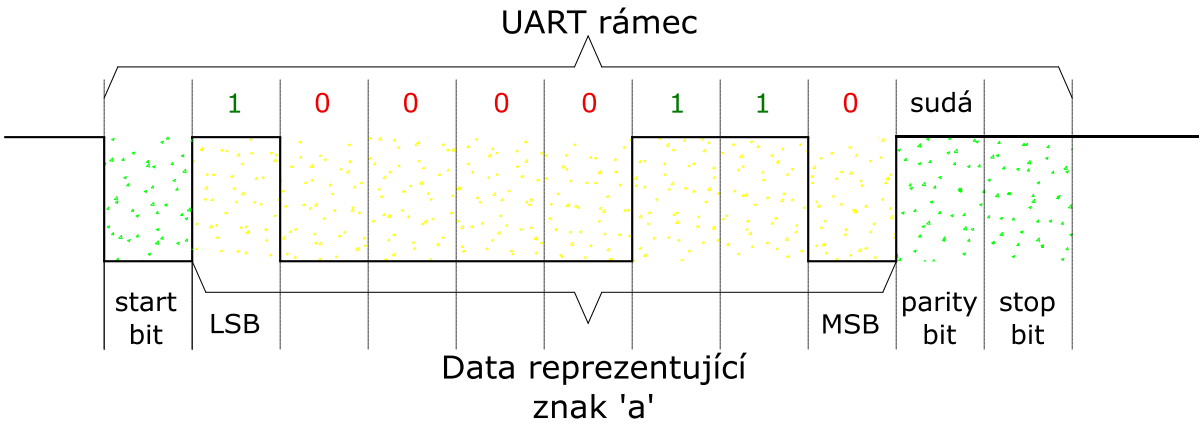
Přenosové rychlosti Bd		
1200	9600	38400
2400	19200	57600
4800	28800	115200

Tabulka 3.1: Seznam přenosových rychlostí

Paritní bit lze nastavit na sudý, lichý nebo nemusí být nataven. Paritní bit slouží jako jednoduchá kontrola, zda došlo v přenosu k chybě.

Je-li parita lichá, pak počet datových bitů logické úrovně 1 + paritní bit musí být liché číslo. Pokud tedy posíláme znak „a“, ten obsahuje tři bity logické úrovně 1, tak potom nastavíme paritní bit na nulu.

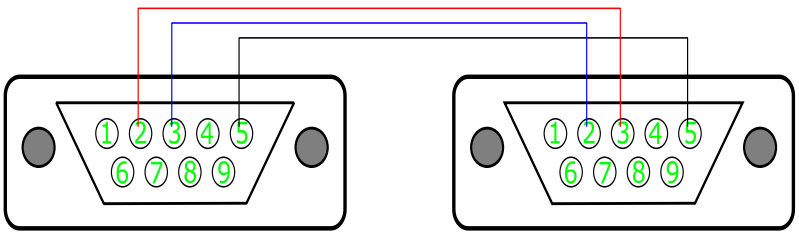
Je-li parita sudá, pak počet datových bitů logické úrovně 1 + paritní bit musí být sudé číslo. Pokud posíláme znak „a“, ten obsahuje tři bity logické úrovně 1, pak potom nastavíme paritní bit na jedničku. Následující obrázek ukazuje, jak vypadá výsledný asynchronní přenos, který posílá znak „a‘‘.



Obrázek 3.7: Asynchronní přenos znaku „a‘‘

Pro sériovou komunikaci můžeme použít RS-232[5], kdy se dají použít pouze některé piny, které se nacházejí na konektoru [Obrázek 3.8:]. Je ale zapotřebí převodník[3] úrovní z TTL na RS-232, jelikož RS-232 má odlišnou úrovňovou logiku.

Ve standartu RS-232 se setkáváme se dvěma pojmy MARK a SPACE, což jsou napěťové úrovně. MARK je negativní napětí a odpovídá logické jedničce. SPACE je pozitivní napětí a odpovídá logické nule. Napětí pro logická jedničku je na vysílači od -5V až -15V a na přijímači -3V až -15V. Napětí pro logickou nulu je na vysílači od +5V až +15V a na přijímači +3V až +15V. Napětí, které je od -3V do 3V není definované.



Pin 1	Data Carrier Detect
Pin 2	Receive Data
Pin 3	Transmit Data
Pin 4	Data Terminal Ready
Pin 5	Signal Ground
Pin 6	Data Set Ready
Pin 7	Request to Send
Pin 8	Clear to Send
Pin 9	Ring Indicator

Obrázek 3.8: Konektor RS-232 společně se signály

[3] Max 232 převodník na 5V logiku, MAX3232 převodník na 3,3V logiku

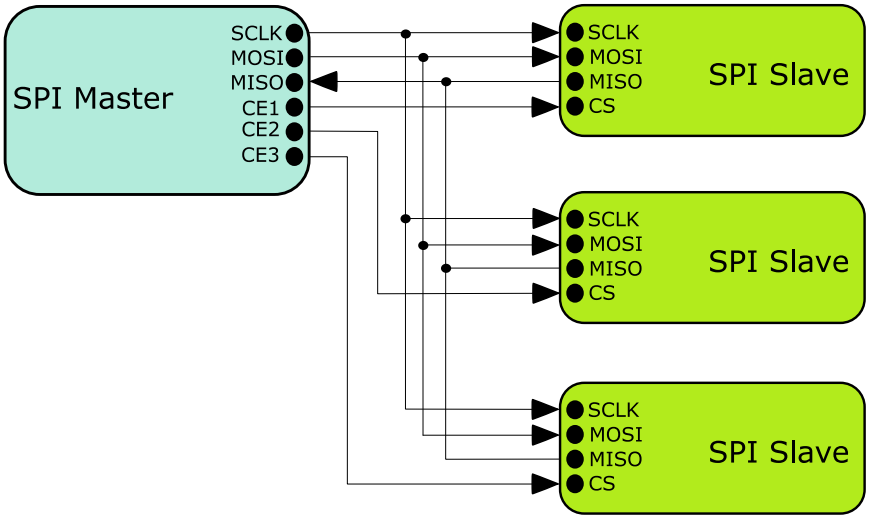
Obrázek 3.9: Převodník z RS-232 na USB

3.3 SPI

SPI je rozhraní typu bod-bod, které bylo vyvinutu firmou Motorola. Má za úkol poskytnout plně duplexní synchronní sériovou komunikaci mezi jedním řídícím zařízením (označený jako Master) a více podřízenými zařízeními (označovaný jako Slave). Master je propojen s každým Slave pomocí čtyř vodičů:

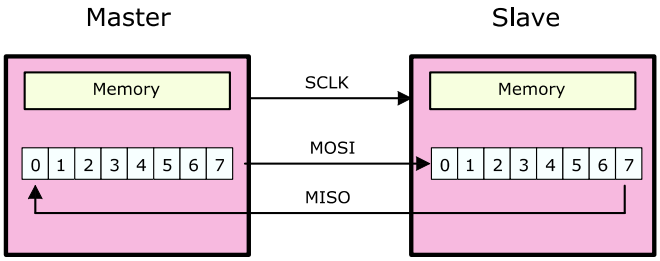
- serial clock (SCK)
- Master Out Slave In (MOSI)
- Master In Slave Out (MISO)
- Slave Select (SS)

Zatímco první tři vodiče jsou sdílené mezi všemi Slave, poslední vodič je vždy propojen s master a jedním Slave [Obrázek 3.10:]. Tento vodič určuje, se kterým Slave bude master komunikovat.



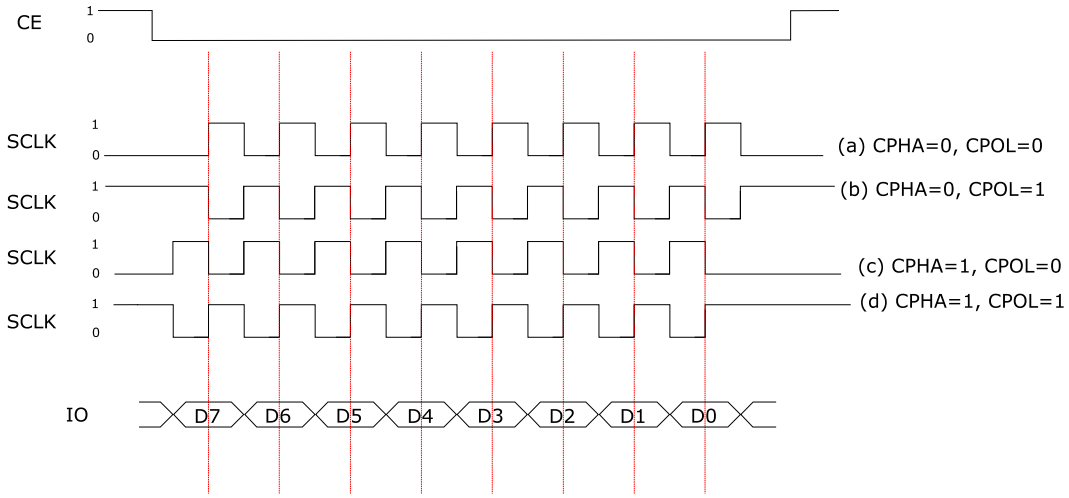
Obrázek 3.10: Konfigurace SPI s jedním master a třemi Slave

Přenos na SPI probíhá vždy mezi Master a některým ze Slave. Oba obsahují posuvné registry, které jsou v okamžiku komunikace propojeny jak je vidět na obrázku [Obrázek 3.11:].



Obrázek 3.11: Propojení master a Slave

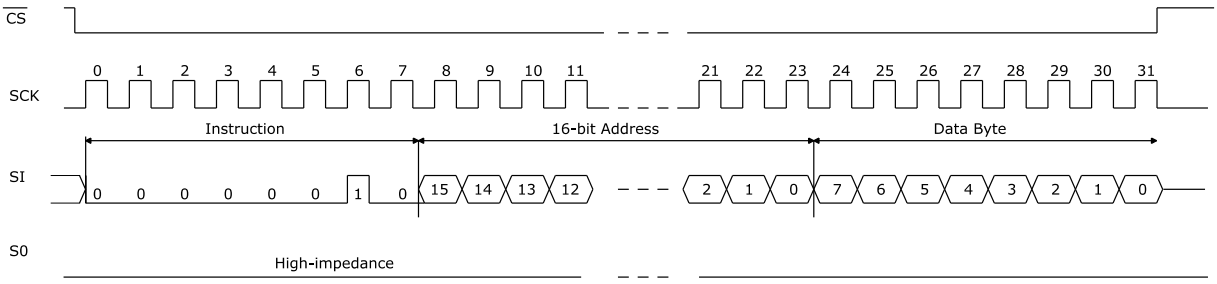
Master pomocí hodinového signálu řídí posouvání obou zmiňovaných posuvných registrů. Pomocí parametrů CPOL a CPHA na SPI, můžeme řídit, kdy se mají vzorkovat data.



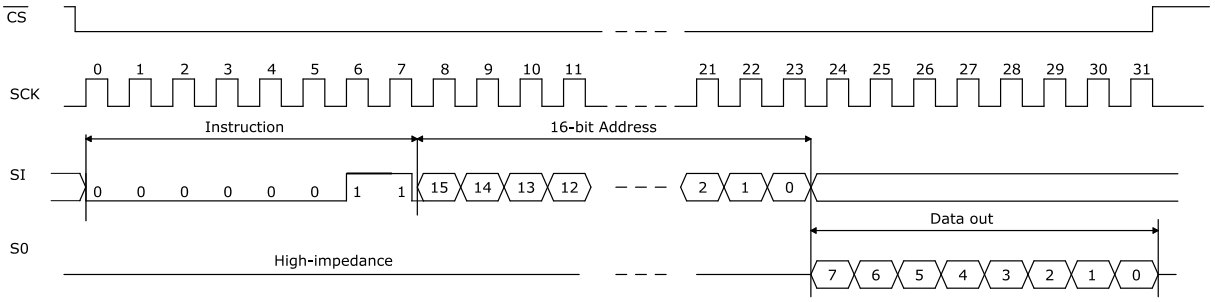
Obrázek 3.12: Vzorkování dat na rozhraní SPI

Pokud je CPHA (clock phase) nastaveno na 0 data se vzorkují na náběžnou hranu hodinového signálu SCK. Pokud je CPHA nastaveno na 1 data se vzorkují až na sestupnou hranu hodinového signálu SCK. Pokud je CPOL (clock polarity) nastaveno na 0 data se vzorkují na náběžnou hranu hodinového signálu SCK. Pokud je CPOL nastaveno na 1 data se vzorkují na sestupnou hodinového signálu hranu SCK. CPOL dále určuje hodnotu SCLK, při tom, kdy nejsou data posílána. Nejběžnější režim, který je použit u SPI je CPHA=0 a CPOL=0.

Při komunikaci master nastaví SS (někdy označený jako CS - Chip Select) vodič požadovaného Slave se kterým chce komunikovat na log 0. Začne generovat hodinový signál na vodiči SCK a v té chvíli vyšlou obě zařízení svoje data na vodičích MOSI a MISO. Délka vyslaných dat je buď 8bit (Byte) a nebo 16bit (Word).



Obrázek 3.13: Zápis dat na adresu pomocí rozhraní SPI



Obrázek 3.14: Čtení dat z adresy pomocí rozhraní SPI

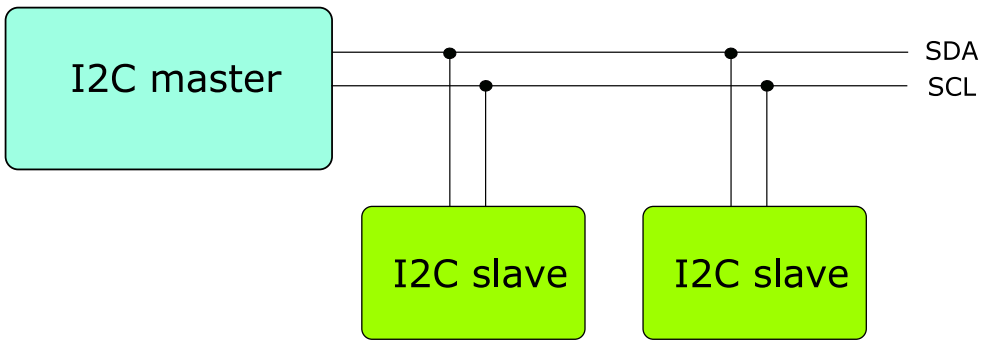
Kromě standardní komunikace pomocí 4 vodičů, SPI rozhraní dovoluje komunikaci po 3 vodičích. Vodiče MOSI a MISO jsou spojeny do jednoho a umožňují oboustrannou komunikaci. Snížení počtu vodičů má za následek polo duplexní režim.

3.4 I2C

Jedná se o sériovou sběrnici typu multi-master kterou vyvinula firma Philips. Řeší proto i arbitraci pro přístup na sběrnici. Každá stanice má přiřazenou svojí vlastní unikátní adresu o délce 7 nebo 10 bitů, která slouží k jejímu výběru.

Jednotlivé stanice jsou spojeny pomocí dvou vodičů:

- datový vodič SDA
- hodinový vodič SCL

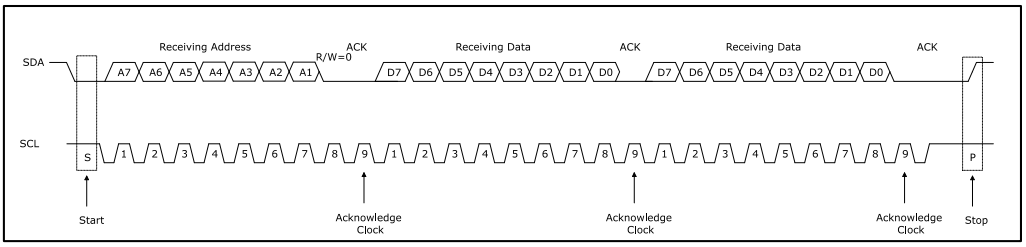


Obrázek 3.15: Zapojení stanic na I2C sběrnici

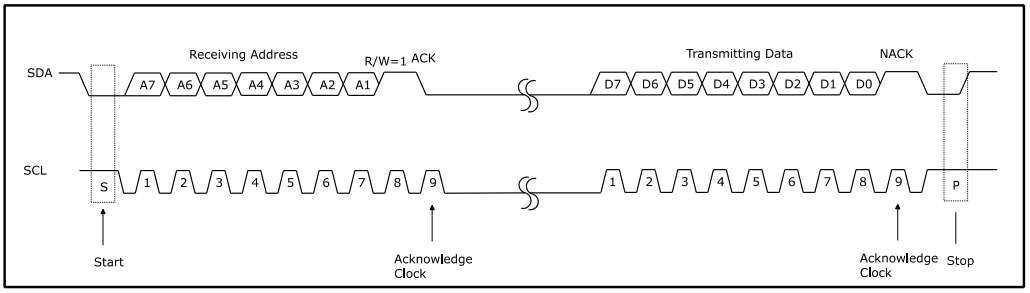
V klidovém stavu, pokud žádná stanice nevysílá, jsou oba vodiče v úrovni HIGH. Komunikaci vždy zahajuje master, kdy vodič SDA dá master do úrovně LOW. Na vodiči SCL se po určitou dobu udržuje úroveň HIGH. Tato doba závisí na zvolené přenosové rychlosti. Tento stav, který se označuje termínem start bit, rozpoznávají všechny připojené stanice na sběrnici. Po start bitu začne master posílat adresu stanice se kterou chce komunikovat. Poté následuje bit, který určuje, zda chce číst data z adresy nebo zapisovat data na adresu.

Po každém bytu následuje bit s názvem ACK, který je vysílán s úrovní HIGH a určuje potvrzení přijímací stanice. Přijímající stanice potvrzuje přijetí tím, že v době vysílání ACK připojí SDA na úroveň L. Po ukončení přenosu je poslán stejně jako na začátku bit označený jako stop.

Pokud Slave zařízení pošle not acknowledges (NACK, $\overline{A}$) bit, znamená to, že zařízení nemá žádná další data k poslání nebo že zařízení není připraveno posílat data. Master musí vygenerovat stop bit nebo opakovat přenos pomocí start bitu.

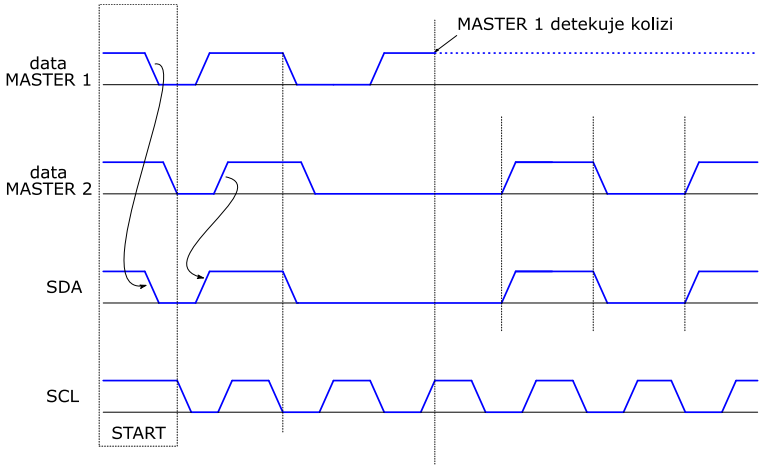


Obrázek 3.16: Zápis dat na adresu pomocí sběrnice I2C



Obrázek 3.17: Čtení dat z adresy pomocí sběrnice I2C

Pro arbitraci se používá metoda s detekcí kolize. Každá ze stanic typu Master může zahájit komunikaci v případě, že je sběrnice v klidovém stavu. Liší-li se bity mezi vysíláním a přijímáním na vodiči SDA, je to indikace, že došlo ke kolize mezi několika stanicemi. Stanice, která zjistí na vodiči SDA úroveň LOW, zatímco sama vysílá HIGH musí vysílání okamžitě ukončit.

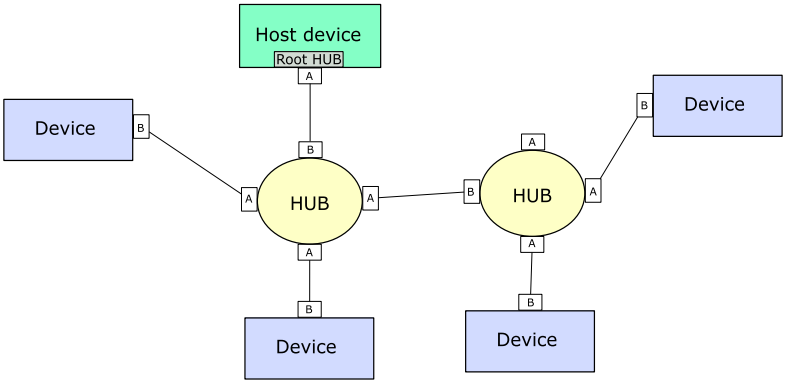


Obrázek 3.18: Detekce kolize na I2C

3.5 USB

V topologii USB systému existují tři typy uzlů:

1) Hostitel (Host)
2) Rozbočovač (Hub)
3) Zařízení (Device)



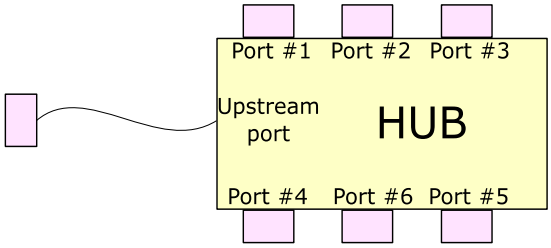
Obrázek 3.19: Topologie USB

Hostitel

Hostitel je jako jediný v systému (má roli master). Jedná se typicky o PC nebo nějaký vestavěný systém (mobilní zařízení nebo RPI). Řídí datové přenosy v celé topologii. Obvykle v sobě integruje řadič hostitele a kořenový rozbočovač (ten může nabízet obvykle 2 USB porty).

HUB

Hub detekuje připojení nebo odpojení zařízení na down-stream portech. Může být napájený ze sběrnice nebo mít svůj vlastní zdroj. Zjišťuje s jakou rychlostí je možné se zařízením komunikovat. Hub umožňuje překlad rychlostí USB (například z High speed na Full speed).

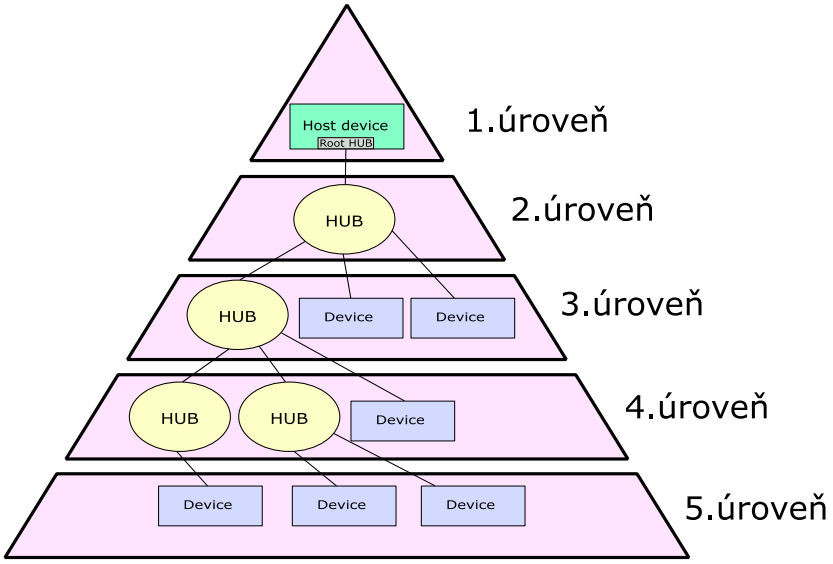


Obrázek 3.20: USB hub

Zařízení

Koncové zařízení (klávesnice, myš), které se připojuje na down-stream portu rozbočovače. Zařízení může být napájeno ze sběrnice případně mít svůj vlastní zdroj (externí disk). Na USB portu je možné maximálně odebírat 500mA, jak definuje norma[15].

Topologie systému USB tvoří tzv. stromovou strukturu [Obrázek 3.21:]. V kořeni stromové struktuře je uzel master a na listech je buď koncové zařízení, nebo HUB. Norma definuje i čas, do kterého musí zařízení odpovědět hostiteli, z tohoto důvodu je kvůli zpoždění vytvořit maximálně 5 úrovní.



Obrázek 3.21: Topologie systému max 5 úrovní

USB systém umožňuje 4 typy přenosů:

- Řídící (Control)

Jako jediný je tento přenos obousměrný. Maximální velikost paketů závisí na rychlosti sběrnice. V případě chyby se přenos paketu opakuje. Skládá se ze dvou, tří nebo více formálně odlišných transakcí. Tento typ se používá při konfiguraci zařízení.

- Izochronní

Jedná se o jednosměrný přenos. Maximální velikost paketů závisí na rychlosti sběrnice. Pro Full speed je velikost paketu 1023 bajtů a pro High speed je 1024 bajtů. V případě chyby se přenos paketu

neopakuje. Obsahuje CRC, které zjišťuje, zda došlo k chybě. Skládá se z formálně totožných transakcí. Tento přenos slouží pro streamy (audio, video).

- Blokový (Bulk)

Stejně jako izochronní přenos je i tento přenos jednosměrný. Maximální velikost paketů závisí na rychlosti sběrnice. Pro Full speed je velikost paketu 64 bajtů a pro High speed je 512 bajtů. V případě chyby se přenos paketu opakuje. Skládá se z formálně totožných transakcí a slouží pro spolehlivý přenos bloků dat.

- Přerušovací (Interrupt)

Jedná se opět o jednosměrný přenos. Pro Full speed je velikost paketu 64 bajtů a pro High speed je 1024 bajtů. V případě chyby se transakce neopakuje.

Transakce se typicky skládá ze tří paketů:

- Token (výzva)
- Data
- Handshake (potvrzení)

Transakci vždy inicializuje master, podle směru hostitele (master) se jedná o transakci vstupní nebo transakci výstupní. Potvrzení generuje příjemce datového paketu.

Pole PID definuje, o jaký typ paketu se jedná. Horní čtyři bity jsou negací spodních čtyř bitů, které slouží jako kontrola.

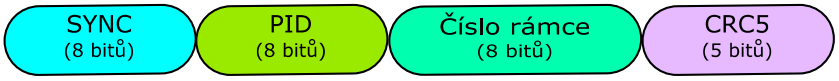
PID_0	PID_1	PID_2	PID_3	$\overline{PID_0}$	$\overline{PID_1}$	$\overline{PID_2}$	$\overline{PID_3}$

Tabulka 3.2: Pole PIDu

Typ PIDů	**Jméno PIDů**	**PID [3:0]**
Token	OUT	0001B
	IN	1001B
	SOF	0101B
	SETUP	1101B
Data	DATA0	0011B
	DATA1	1011B
	DATA2	0111B
	MDATA	1111B
Handshake	ACK	0010B
	NAK	1010B
	STALL	1110B
	NYET	0110B
Special	PRE	1100B
	ERR	1100B
	SPLIT	1000B
	PING	0100B
	Reserved	0000B

Tabulka 3.3: Přehled PIDů

SOF indikuje počátek rámce



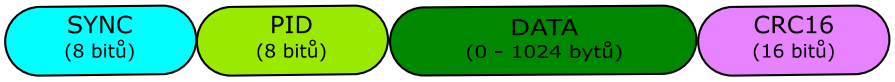
Obrázek 3.22: Struktura paketu SOF

Token (výzva)

- ADDR – adresa koncového zařízení
- ENDP – identifikátor roury

Obrázek 3.23: Struktura paketu Token

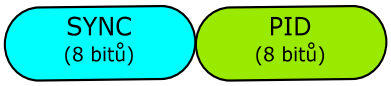
Datový paket slouží k přenosu dat, které odesílá hostitel nebo zařízení



Obrázek 3.24: Struktura paketu Data

Handshake (potvrzení)

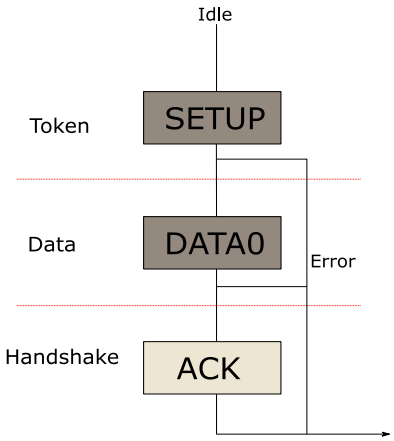
- PID – příjemce potvrzuje pozitivní nebo negativní příjem dat



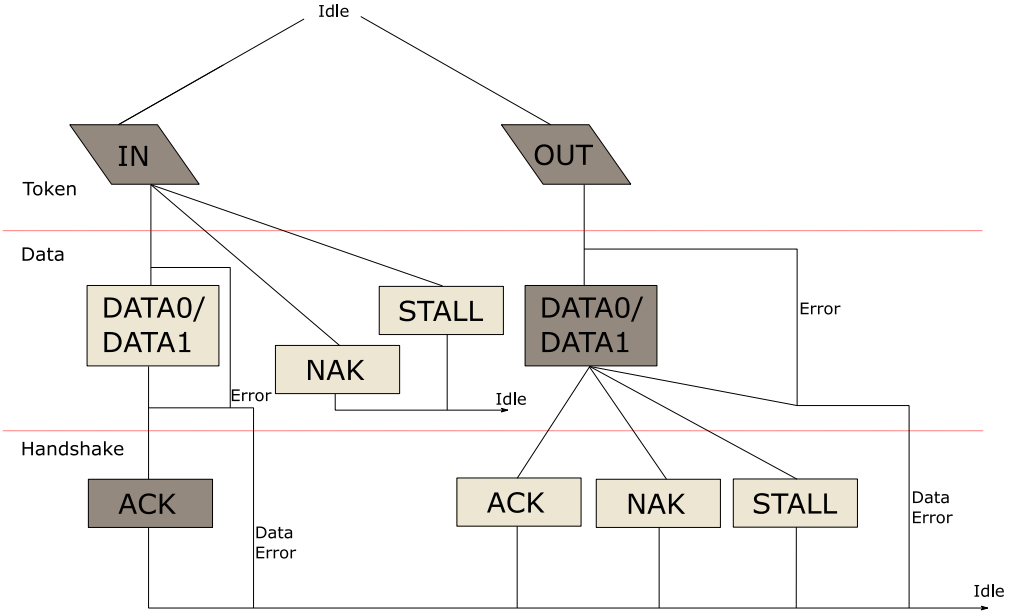
Obrázek 3.25: Struktura paketu Handshake

Pakety, které obsahuje transakce, se liší v závislosti na typu koncového bodu (endpoint). Existují celkem čtyři typy bodů:

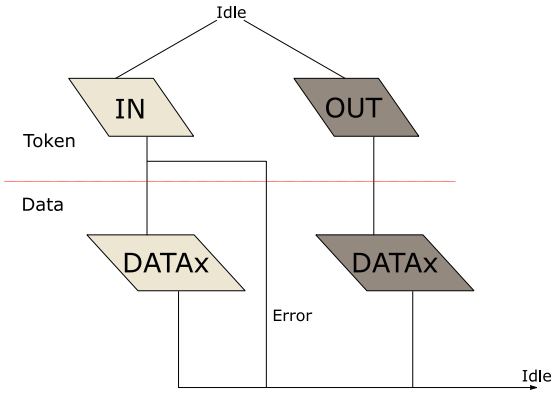
- Řídící
- Přerušení
- Isochronní
- Blokový



Obrázek 3.26: Řídící SETUP transakce



Obrázek 3.27: Transakce přerušovací



Obrázek 3.28: Transakce izochronní

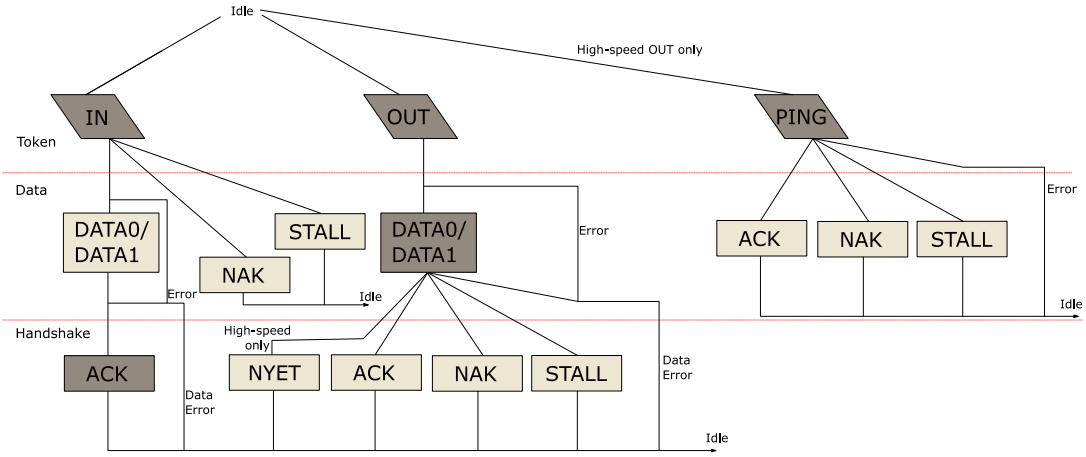
Blokové transakce umožnují bezchybné doručení dat mezi hostitelem a zařízením pomocí detekce chyb a opětovném opakovaní transakce. Bloková transakce se skládá z tokenu, data a handshake paketu jak je zobrazeno na [Obrázek 3.29:]. PING a NYET pakety mohou být použity pouze se zařízeními, které umožnují rychlost High-sped.

Když je hostitel připraven přijímat bloková data pošle IN token zařízení. Zařízení odpoví tím, že vrátí datový paket, ve kterém budou buď DATA, NAK nebo STALL. Paket NAK znamená, že zařízení nemá žádné data připravená v rouře, zatímco STALL znamená, že koncový bod je trvale zastaven. Pokud hostitel obdrží platný datový paket, odpoví ACK handshake. Pokud hostitel detekuje chybu při přijímání dat, nepošle žádný potvrzovací paket zpět zařízení.

Když je hostitel připraven k přenosu blokových dat pošle OUT token paket, následovaný datovým paketem. Pokud jsou data přijata bez chyby, zařízení pošle jeden ze tří potvrzovacích paketů:

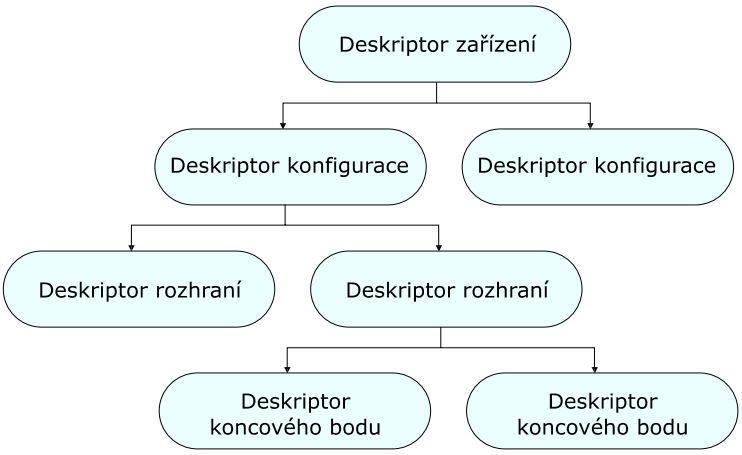
- ACK indikuje, že datový paket byl přijat bez chyby a informuje hostitele, že může poslat další paket v pořadí.
- NAK indikuje, že data byla přijata bez chyby, ale že hostitel by měl znovu odeslat data, jelikož zařízení nebylo ve stavu přijímat data
- Pokud koncový bod byl zastaven, STALL indikuje, že hostitel by neměl opakovat přenos, jelikož je chybový stav na zařízení.

Pokud byl přijat datový paket s chybným CRC, zařízení nepošle žádné potvrzení.



Obrázek 3.29: Transakce bloková

3.5.1 Základní deskriptory pro zařízení



Obrázek 3.30: Struktura návaznosti deskriptorů

Deskriptor zařízení (Device descriptor)

Popisuje obecné informace o zařízení USB. USB zařízení obsahuje pouze jeden deskriptor zařízení. Všechna USB zařízení obsahují řídící (control) rouru. Maximální velikost paketu pro tuto řídící rouru lze nalézt v deskriptoru zařízení. Pole *bNumConfigurations* určuje počet konfigurací, které dané zařízení umožňuje. Formát deskriptoru zařízení je uveden níže.

Pozice	Pole	Velikost	Hodnota	Popis
0	bLength	1	18	délka deskriptoru v bajtech
1	bDescriptorType	1	1	kód typu deskriptoru
2	bcdUSB	2	BCD	podporovaná verze USB
4	bDeviceClass	1		kód třídy
5	bDeviceSubClass	1		kód podtřídy
6	bDeviceProtocol	1		podporovaný protokol třídy
7	bMaxPacketSize0	1		velikost bufferu (a paketu)
8	idVendor	2		ID výrobce
10	idProduct	2		ID výrobku
12	bcdDevice	2	BCD	verze zařízení
14	iManufacturer	1		index do textového deskriptoru
15	iProduct	1		index do textového deskriptoru
16	iSerialNumber	1		index do textového deskriptoru
17	bNumConfigurations	1		počet konfigurací

Tabulka 3.4: Struktura deskriptoru zařízení

Deskriptor konfigurace (Configuration descriptor)

Pro každou konfiguraci jeden. Definuje počet možných rozhraní v dané konfiguraci a spotřebu zařízení v dané konfiguraci.

Pozice	Pole	Velikost	Hodnota	Popis
0	bLength	1	9	délka deskriptoru v bajtech
1	bDescriptorType	1	2	kód typu deskriptoru
2	wTotalLength	2		celková délka všech deskriptorů této konfigurace (včetně interface a endpoint deskriptorů)
4	bNumInterfaces	1		počet rozhraní (interface)
5	bConfigurationValue	1		hodnota pro výběr
6	iConfiguration	1		index do textového deskriptoru
7	bmAttributes	1		bity 0…4 rezervovány bit 5 - remote wakeup (vzbuzení hostitelem) bit 6 - self powered (vlastní zdroj nebo ne) bit 7 - rezervován
8	bMaxPower	1		spotřeba (rozlišení 2mA)

Tabulka 3.5: Struktura deskriptoru konfigurace

Deskriptor rozhraní (Interface descriptor)

Pro každé rozhraní jeden. Definuje počet endpointů, tvořících rozhranní.

Pozice	Pole	Velikost	Hodnota	Popis
0	bLength	1	9	délka deskriptoru v bajtech
1	bDescriptorType	1	4	kód typu deskriptoru
2	bInterfaceNumber	1		pořadové číslo rozhranní
3	bAlternateSetting	1		hodnota pro výběr
4	bNumEndpoints	1		počet endpointů (mimo nultý)
5	bInterfaceClass	1		0xff pro uživatelské
6	bInterfaceSubClass	1		0xff pro uživatelské
7	bInterfaceProtocol	1		0xff pro uživatelské
8	iInterface	1		index do textového deskriptoru

Tabulka 3.6: Struktura deskriptoru rozhraní

Deskriptor koncového bodu (Endpoint descriptor)

Obsahuje jeden pro každý koncový bod. Definuje směr roury, končící v endpointu, podporovaný typ přenosu a číslo endpointu (roury). Definuje taktéž velikost datového buffer a tedy maximální velikost datového paketu. Pro přerušení roury definuje period dotazování. Pro isochronní roury period zasílání dat.

Pozice	Pole	Velikost	Hodnota	Popis
0	bLength	1	7	délka deskriptoru v bajtech
1	bDescriptorType	1	5	kód typu deskriptoru
2	bEndpointAddress	1		bity 0…3 - číslo endpointů bity 4…6 - rezervováno bit 7 - směr (0 - OUT, 1 - IN) z hlediska masteru
3	bmAttributes	1		bity 0…1 - typ přenosu bity 2…3 - synchronizace (isochr.) bity 4…5 - (data, feedback)
4	wMaxPacketSize	2		velikost bufferu
8	bInterval	1		časování pro interrupt a isochr. NAK četnost pro bulk

Tabulka 3.7: Struktura deskriptoru koncového bodu

Textový deskriptor (String descriptor)

Obsahuje textové popisy, které lze číst.

Pozice	Pole	Velikost	Hodnota	Popis
0	bLength	1	N+2	délka deskriptoru v bajtech
1	bDescriptorType	1	3	kód typu deskriptoru
2	bString	N		textový řetězec v Unicode

Tabulka 3.8: Struktura textového deskriptoru

Typ deskriptoru	Hodnota
DEVICE	1
CONFIGURATION	2
STRING	3
INTERFACE	4
ENDPOINT	5

Tabulka 3.9: Typy deskriptorů

Pozice	Pole	Velikost	Hodnota	Popis
0	bmRequestType	1		bity 0…4 – příjemce (0 – device, 1 – interface, 2 – endpoint, 3 – jiný, 4 až 31 rezervováno) bity 5…6 – typ (0 – standartní, 1 – specifický, 2 - specifický výrobce, 3 – rezervováno) bit 7 – směr (0 - host to device, 1 - device to host)
1	bRequest	1		kód typu žádosti
2	wValue	2		význam dle typu žádosti
4	wIndex	2		význam dle typu žádosti, typicky index nebo ofset
6	wLength	2		velikost dat pro datovou fázi (kolik Bytu chceme vyčíst nebo zapsat)

Tabulka 3.10: Datový paket v SETUP transakci řídícího přenosu

4 Analýza

Základním předpokladem pro splnění mého zadání je zvolení správné verze RPI. V této kapitole se seznámíme se s tím co je to vůbec Raspberry PI, jaká je jeho historie a základní informace, které se budou hodit pro začátek vědět. V poslední části se pak zaměřím konečně na to, jaké jsou verze RPI a jak se od sebe tyto verze jednotlivě liší. V další podkapitole zde budou popsané i informace o periférií.

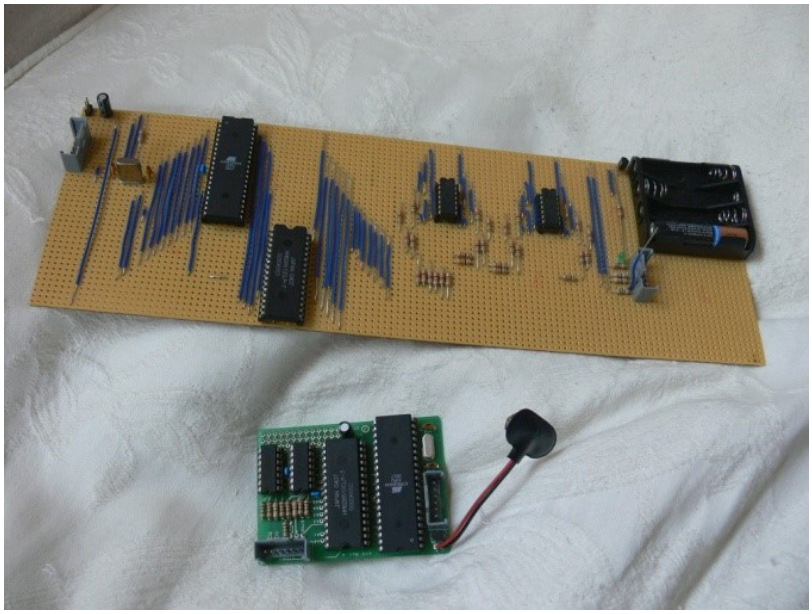
4.1 Co to je Raspberry PI?

RPI je jednodeskový počítač co se týče do velikosti zhruba platební karty, který vyniká svojí nízkou cenou a velkou škálou použitelnosti. Veškeré součástky jsou osazeny na desce plošných spojů, to znamená, že není zapotřebí nic „skládat‘‘ dohromady jako u běžného počítače.RPI má bohaté možnosti rozšíření o další hardware, zejména o vstupně/výstupní moduly. Udržení nízkých nákladů na desku bylo docíleno za použití čipu typu SoC, což je technika, u níž se paměť, mikroprocesor a grafický procesor umísťuje do jediného čipu a tím se minimalizuje potřeba místa na desce plošných spojů. RPI neobsahuje žádné rozhraní pro pevný disk nebo SSD. Pro Operační systém a pro uchování trvalých dat se používá SD karta. RPI bylo primárně určeno pro podporu výuky programování studentů ve škole ale zájem o něj však předčilo všechna očekávání jeho tvůrců. Je vyvíjeno nadací Raspberry Pi Foundantion se sídlem v Británii. Veškeré informace o RPI lze nalézt zde [1].

4.2 Historie RPI

Nápad na vytvoření cenově dostupného počítače pro děti vznikl v roce 2006, kdy Dr. Eben Upton a jeho spolupracovníci v počítačové laboratoři Cambridgeské univerzity v Anglii byly znepokojeni nad stále klesajícím počtem a úrovní studentů hlásících se ke studiu informatiky na škole. Jako malá skupina lidí se domnívaly, že problém je buď v nedostatečném školním vzdělání, nebo ve financích. S prvním problémem nemohly nic dělat a tak se chtěly pokusit vyřešit druhý problém, kdy se počítače stávají dražšími a tím pádem rodiče nemají dostatek financí pro jeho koupi.

S tím se muselo něco udělat a tak mezi roky 2006 až 2008 bylo navrženo několik verzí toho, co známe dnes jako RPI. Jeden z prvních prototypů je zobrazen na nadcházejícím obrázku:



Obrázek 4.1: Jeden z prvních prototypů[4]

Po roce 2008 se procesory vyvíjené pro mobilní zařízení staly více cenově dostupné a výkonné, a proto realizace projektu vypadala uskutečnitelná. Dr. Eben Upton (v současnosti návrhář obvodu u společnosti Broadcom), Rob Mullins, Jack Lang a Alan Mycroft (jeho bývalí kolegové z počítačové laboratoře University of Cambridge) se spojili s Petem Lomasem (z firmy Norcott Technologies) a Davidem Brabenem (spoluautorem hry Elite pro mikropočítač BBC Micro) a společně založili nadaci Raspberry Pi Foundation a nápad mohl být realizován. O tři roky později byla vydaná první verze RPI Model B, která byla hromadně vyráběna. Během dvou let se prodalo přes dva miliony kusů.

4.3 Základní informace o RPI

Jak už bylo zmíněno RPI používá dva procesory. Centrální procesor rodiny ARM a grafický procesor VideoCore IV kompatibilní s OpenGL ES 2.0. Nadace RPI proto uzavřela spojenectví se společností Broadcom, aby mohla využívat jejich návrhů mikroprocesoru i grafických procesorů v čipu SoC. Firma Broadcom se specializuje na procesory pro přenosná zařízení, které se používají i v mobilních telefonech. Mikroprocesor BCM je navržen pro přenosná zařízení, musí tedy pracovat s minimálním příkonem, aby se co nejvíce prodloužila životnost baterie. Poměrně nízká frekvence hodin mikroprocesoru pomáhá snižovat příkon. Nižší frekvence hodin znamená, že procesor může pracovat při nízkém napětí, což má za následek snížení celkového množství uvolněného tepla a zvýšení životnosti procesoru.

RPI obsahuje dva druhy paměti: dynamická paměť s přímým přístupem DRAM a flash paměť SD.

Důležité u SD karet kromě kapacity je i třída karty. Ta značí, čím vyšší číslo třídy SD karta obsahuje tím je SD karta výkonnější, ale za cenu vyšší pořizovací ceny.

[4] https://www.raspberrypi.org/blog/raspberry-pi-2006-edition/

Třída	Minimální přenosová rychlost
Třída 2	2 MB/s
Třída 4	4 MB/s
Třída 6	6 MB/s
Třída 10	10 MB/s

Tabulka 4.1: Označení tříd SD karet

Jako napájecí zdroj pro RPI slouží microUSB, které přivádí napětí +5V DC. K napájení lze využít jakýkoliv nabíječku pro chytré telefony nebo powerbanku, která má konektor microUSB. Lze se připojit k monitoru nebo televize za pomoci HDMI nebo RCA[2], zvukový výstup je přes 3.5 mm jack nebo HDMI. Pomocí rozhraní DSI nacházející se na každé verzi RPI je možné připojit display, ale toto rozhraní není potřeba pro tuto práci.

4.4 Verze RPI

4.4.1 Raspberry Pi 1

V integrovaném obvodu SoC se používá čip BCM 2835 a grafická procesorová jednotka. Jedna součást integrovaného obvodu SoC čip BCM2835 se vyrábí s procesoru ARM1176JZF běžící na frekvenci 700MHz a grafického procesoru Broadcom VideoCore IV.

Model A

Tento model byl vydaný v únoru roku 2012. Prodával se za 25$. Obsahuje pouze jeden USB2.0 port připojený přímo na BCM2835, 256 MB RAM (sdílená s GPU), RCA konektor, slot pro SD nebo MMC kartu. Na RPI lze nalézt 26 pinů GPIO ale umožňuje i dodatečné vývody GPIO pro rozšíření.

Obrázek 4.2: RPI model A[5]

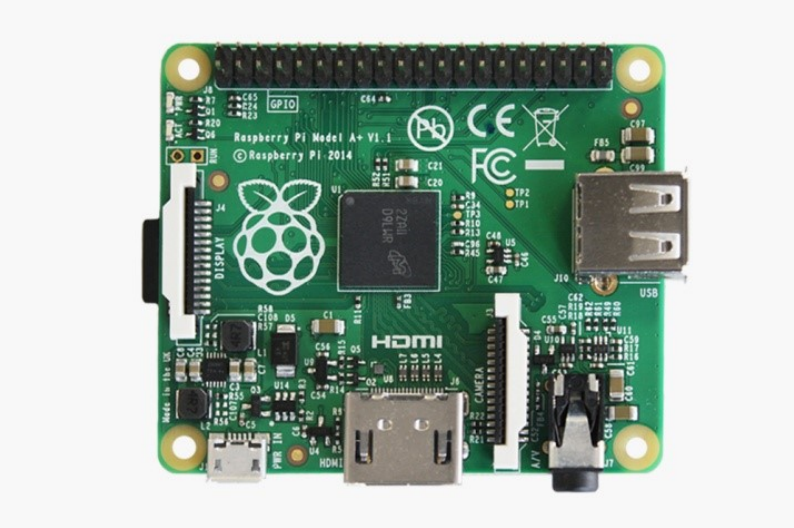
[5] https://upload.wikimedia.org/wikipedia/commons/4/45/Raspberry_Pi_-_Model_A.jpg

Model A+

Verze, která byla vydána v listopadu roku 2014. Jedná se o vylepšení předchozího modelu. Tento model se doporučuje pro vestavěné projekty nebo pro projekty s nižší spotřebou, které nepožadují ethernet nebo více USB portů.

Ve srovnání s RPI1 model A bylo:

- přidáno více GPIO, z původních 26 pinů na 40 pinů
- slot pro SD kartu byl nahrazen slotem micro SD karty
- vylepšení audia, nízká hlučnost napájení
- snížená spotřeba energie
- zarovnání USB konektoru s okrajem desky
- odstranění RCA konektoru
- zmenšení rozměrů desky

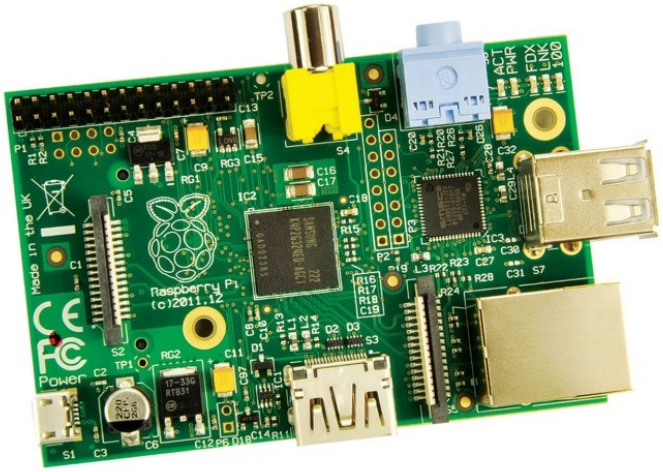


Obrázek 4.3: RPI model A+[6]

Model B

Stejně jako model A i tento model obsahuje 26 pinů GPIO včetně dodatečných vývodů GPIO pro rozšíření. Oproti modelu A byl přidán další USB2.0 port a hlavně přidaný ethernetový adaptér 10/100 s konektorem RJ45. Nelze tedy už připojit přímo na BCM2835, ale přes integrovaný rozbočovač USB. Došlo i ke zvýšení RAM paměti z původních 256MB na 512 MB (sdílená s GPU).

[6] http://diit.cz/clanek/raspberry-pi-model-je-nova-levna-prtava-varianta



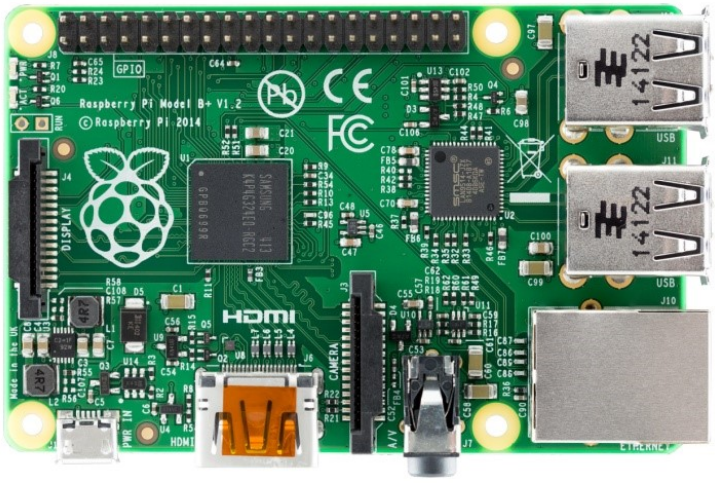
Obrázek 4.4: RPI model B[7]

Model B+

Jde o poslední verzi RPI 1, která byla vydána v červenci roku 2014. Jedná se o vylepšení předchozího modelu.

Ve srovnání s RPI1 model B bylo:

- přidáno více GPIO, z původních 26 pinů na 40 pinů
- přidáno více USB2.0, z původních 2 na 4 a vylepšení hotplugu
- slot pro SD kartu byl nahrazen slotem pro micro SD kartu
- vylepšení audia, nízká hlučnost napájení
- snížená spotřeba energie
- zarovnání USB konektoru s okrajem desky
- odstranění RCA konektoru



Obrázek 4.5: RPI model B+[8]

[7] http://www.raspishop.cz/wp-content/uploads/2013/03/Raspberry_Pi.jpg
[8] http://www.postavrobota.cz/fotky46704/fotos/_vyr_182raspberrybplus.jpg

4.4.2 Raspberry Pi 2 Model B

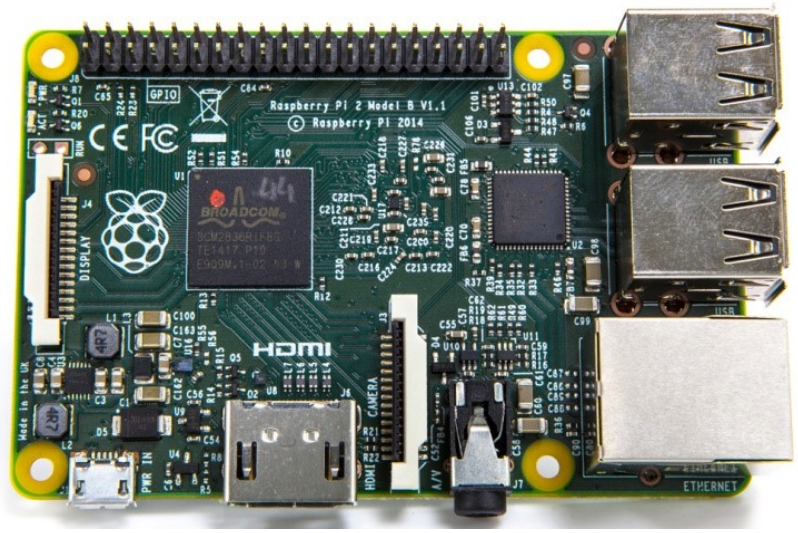
Jedná se o druhou generaci RPI, která byla vydaná v únoru roku 2015. Rozměrově odpovídá RPI 1 Model B+. Je zpětně kompatibilní s RPI1.

Ve srovnání s RPI1 má:

- Čtyř-jádrový SoC Broadcom BCM2836 z rodiny ARM Cortex-A7 taktovaný na 900 MHz
- 1GB RAM (sdílená s GPU)

Stejně jako RP1 model B+ obsahuje:

- 4xUSB2.0 porty
- 40 GPIO pinů
- HDMI port
- ethernet port
- slot pro micro SD kartu
- rozhraní pro připojení kamery
- rozhraní pro připojení display
- Grafický procesor VideoCore IV

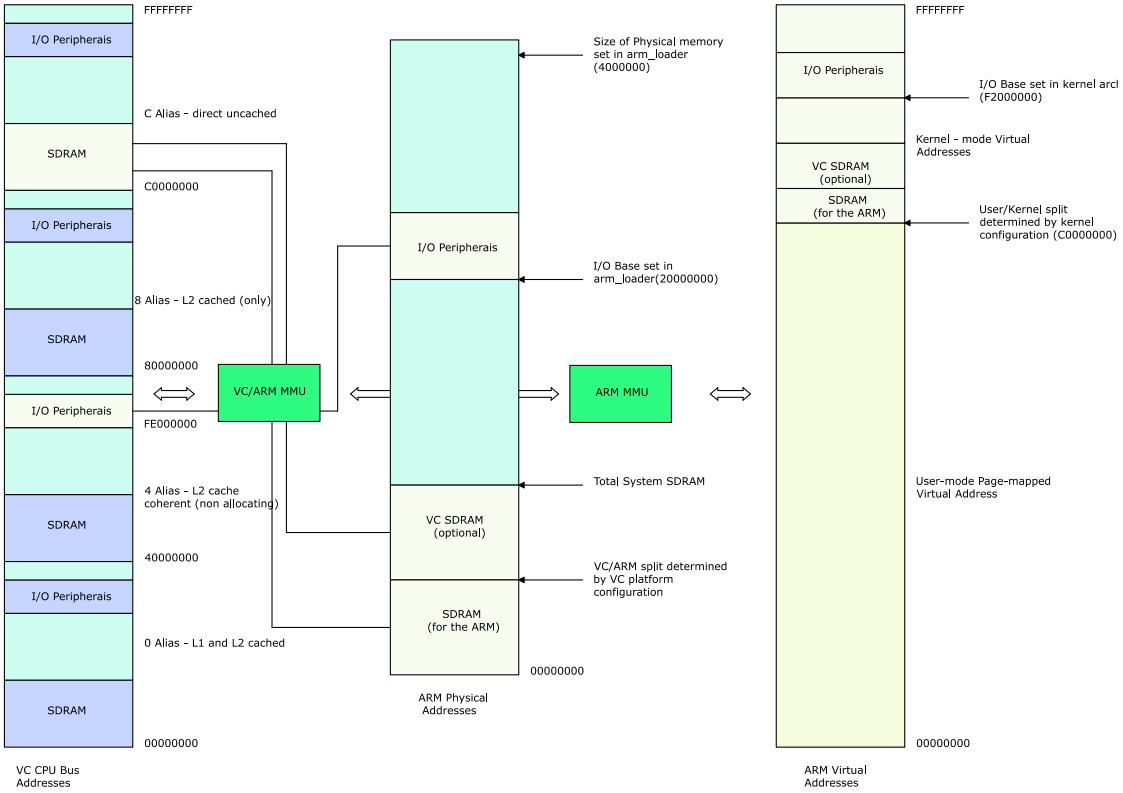


Obrázek 4.6: RPI 2 model B[9]

4.5 Popsaní periférié

Tato podkapitola popisuje základní infomace, které jsou potřebné pro zacházení s periériemi, které jsou na RPI. Pro RPI1 jsou periférie na fyzické adrese v rozsahu od 0x20000000 až 0x20FFFFFF. Pro RPI2 jsou periférie na adrese 0x3F000000 až 0x3FFFFFFF [6].

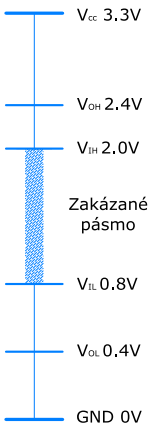
[9] http://www.root.cz/zpravicky/raspberry-pi-2-ma-sestkrat-rychlejsi-ctyrjadro-a-1-gb-ram/



Obrázek 4.7: Ukázka adres periférií na čipu BCM2835

4.5.1 GPIO

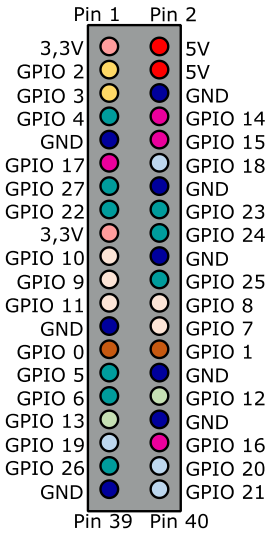
RPI GPIO používá 3.3V úrovňovou logiku. Při zapojení pinu na 5V úrovňovou logiku může dojít k tomu, že poškodí zařízení. Neexistuje tedy žádná ochrana proti přepětí a je nutné na to pamatovat. Vstupní napětí přivedené na GPIO vyšší než 2.0V bude vyhodnoceno jako log 1 čili úroveň HIGH. Vstupní napětí menší než 0.8V bude vyhodnoceno jako log 0 čili úroveň LOW.



Obrázek 4.8: Úrovňová logika

RPI celkem obsahuje 54 GPIO a všechny mohou být nastaveny do režimu jako vstup, výstup nebo speciální funkce jako je UART, SPI nebo I2C. Ale ne všechny tyto GPIO jsou vyvedené a lze tak k nim připojit jiné zařízení. Pouze prvních 28 GPIO jsou vyvedené z RPI2 a pro cíle této práce

obsahují tyto GPIO periférie, které jsou potřeba. U RPI dochází občas k nepřesnostem a zaměňují se pojmy jako GPIO 5 a Pin 5. Pokud tedy mluvíme o Pinu 5, jedná se číslo pinu, které je fyzicky vyvedené z RPI ven. V našem případě tedy Pin 5 odpovídá (GPIO 3).



Obrázek 4.9: GPIO na RPI 1 modelu A+,B+ a RPI2

Na GPIO nastavené jako vstup může být např. připojené tlačítko a číst jestli je tlačítko sepnuté nebo rozepnuté.

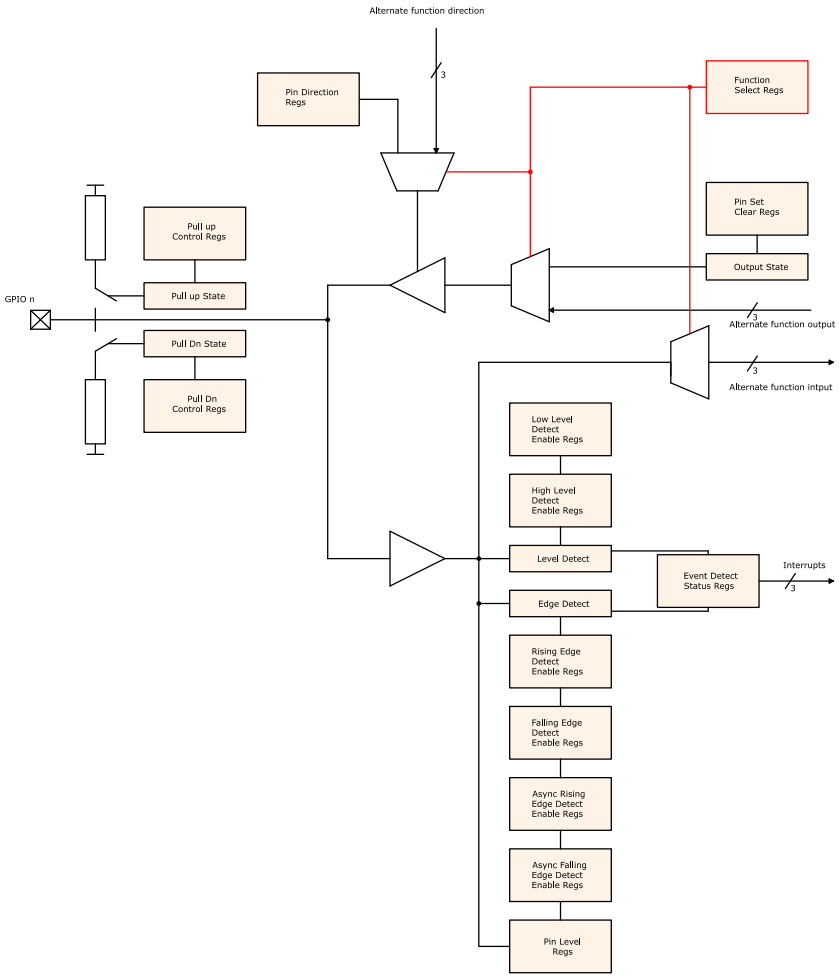
Na GPIO 47 je připojená Led dioda nacházející se na RPI. Tato LED dioda je využita k tomu, že při nahrání jádra do RPI informuje programátora, zda bylo nahrané jádro správně (problikne). Pokud LED dioda neproblikne, musí se RPI vypnout a poté zapnout znovu. LED dioda může být i dále použita k otestování jedné z prvních aplikací, které jsem využil i já osobně. GPIO 47 sloužil původně k tomu, že detekoval, zda je připojená SD karta, ale už tomu tak není a slouží pouze pro LED diodu (nazývá se aktivní). Kromě této LED diody na GPIO 47 je i na RPI další LED dioda na GPIO 35, která informuje o tom, zda je RPI napájené.

Na GPIO portech je možnost zapnutí a vypnutí interních pull-up nebo pull-down rezistorů.

- Hodnota pull-up rezistoru je 50 kOhm - 65 kOhm
- Hodnota pull-down rezistoru je 50 kOhm - 60 kOhm

Pomocí každého GPIO portu lze vyvolat přerušení. Přerušení je možné vyvolat několika způsoby:

- Při úrovni HIGH
- Při úrovni LOW
- Při změně z úrovně HIGH na LOW
- Při změně z úrovně LOW na HIGH



Obrázek 4.10: Základní blokové schéma GPIO

4.5.2 Řadič přerušení

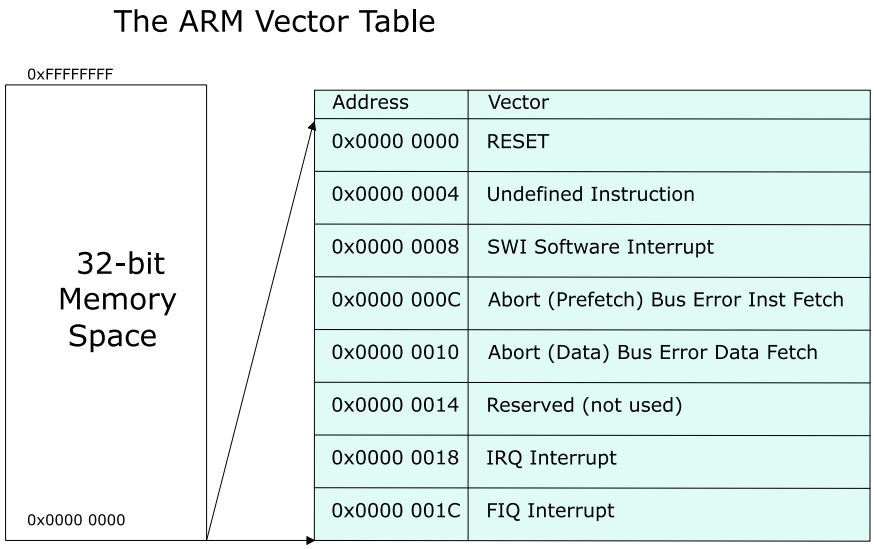
Řadič přerušení (anglicky interrupt controller) slouží k tomu, aby detekoval a obsloužil přerušení, do kterého vstoupí procesor.

Procesor ARM se může nacházet v jednom ze sedmi režimech (modech)[12]:

- User mode (číslo módu-0b10000) – režim normálního běhu programu
- Fast interrupt mode (číslo módu-0b10001) – režim rychlého přerušení
- Interrupt mode (číslo módu-0b10010) – režim normálního přerušení
- Supervisor mode (číslo módu-0b10011) – chráněný režim pro operační systém
- Abort mode (číslo módu-0b10111) – implementuje virtuální paměť a ochranu paměti
- Undefined mode (číslo módu-0b11011) – provedení neznámé instrukce
- System mode (číslo módu-0b11111) – spouští priviligované tásky OS

Přerušení a výjimky

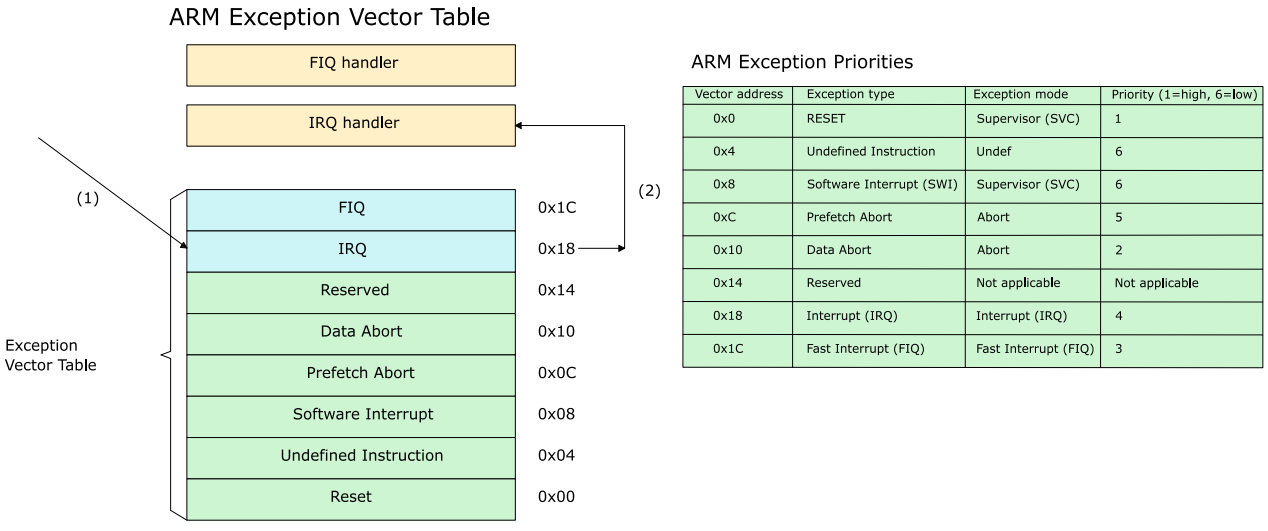
Pokud dojde k výjimce nebo přerušení, procesor potřebuje vědět, kde je umístěn kód pro obsluhu přerušení. U ARM procesoru je to realizované pomocí tabulky vektorů [Obrázek 4.11:]. Tato vektorová tabulka je umístěna na adrese 0x0000 0000.



Obrázek 4.11: ARM Vector table

V ARM jádrech může dojít k osmi typům přerušení (vyjímek). Jedná se o:

1. Reset – při hardwarovém nebo softwarovém resetu
2. Undefined – při pokusu o provedení neznámé instrukce
3. Software interrupt – softwarové přerušení
4. Prefetch Abort – vykonání instrukce na neznámé adrese
5. Data Abort – získání dat z neznámé adresy
6. Reserved – rezervované pro pozdější použití
7. Interrupt request – přerušení od periférií
8. Fast interrupt request – rychlé přerušení



Obrázek 4.12: Priority ARM Vector table

Před použitím řadiče přerušení se musí pomocí strojové instrukce CPS (Change Processor State)[12] povolit nebo zakázat jedno ze dvou (irq,fiq) přerušení, které chceme používat. Obsah tohoto registrů může měnit pouze kód, který je prováděn v privilegovaném režimu (obvykle User).

- ***__asm volatile ("cpsie i") //instrukce pro povolení irq***
- ***__asm volatile ("cpsie f") //instrukce pro povolení fiq***
- ***__asm volatile ("cpsid i") //instrukce pro zakázaní irq***
- ***__asm volatile ("cpsid f") //instrukce pro zakázaní fiq***

Adresa offsetu	Velikost	Typ	Jméno registru	Popis
0x200	8	R/W	IRQ basic pending	určuje, které přerušení je třeba obsloužit
0x204	32	R/W	IRQ pending 1	určuje, které přerušení je třeba obsloužit
0x208	32	R/W	IRQ pending 2	určuje, které přerušení je třeba obsloužit
0x20C	8	R/W	FIQ control	nastavení a povolení fast přerušení
0x210	32	R/W	Enable IRQs 1	povolení přerušení
0x214	32	R/W	Enable IRQs 2	povolení přerušení
0x218	8	R/W	Enable Basic IRQs	povolení přerušení
0x21C	32	R/W	Disable IRQs 1	zakázání přerušení
0x220	32	R/W	Disable IRQs 2	zakázání přerušení
0x224	8	R/W	Disable Basic IRQs	zakázání přerušení

Tabulka 4.2: Registry pro řadič přerušení

#	IRQ_ID	#	IRQ_ID	#	IRQ_ID
IRQ 1	TIMER_1	IRQ 29	AUX	IRQ 54	SPI
IRQ 3	TIMER_3	IRQ 52	GPIO_3	IRQ 57	UART
IRQ 9	USB	IRQ 53	I2C	IRQ basic 0	ARM_TIMER

Tabulka 4.3: Přerušení

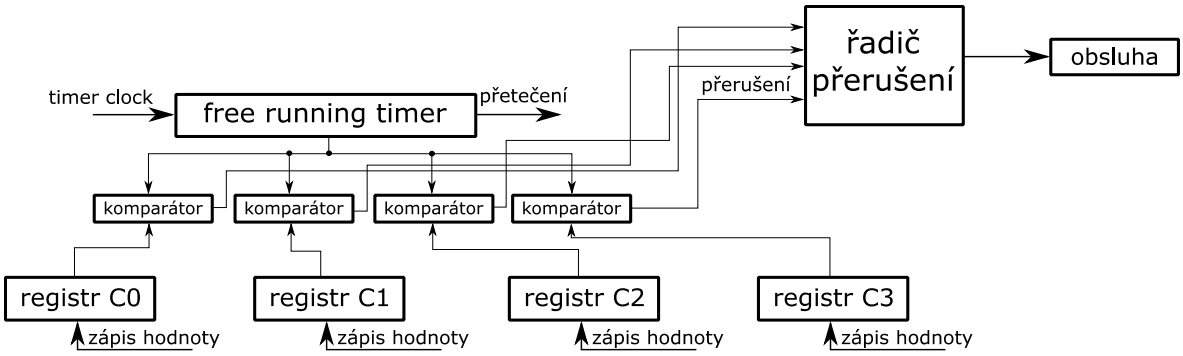
4.5.3 Timer

RPI obsahuje dva odlišné druhy timeru. První z těchto dvou je systémový timer a druhý je ARM timer.

Systémový timer obsahuje čtyři oddělené 32 bitové registry pro porovnání („output compare") a obsahuje 64-bitový free running timer což v překladu znamená volně běžící časovač, který běží na frekvenci 1MHz. Tento časovač neustále běží od začátku do konce, aniž by se zastavil. Počítá vstupní pulzy od 0 až do své maximální hodnoty což je $2^{64} - 1$. Následně dojde k přetečení a počítá opět od nuly. Free running timer je řízen timer clockem a dojde k jeho zastavení, jakmile se procesor zastaví v režimu ladění (debug mode). Každý z výstupních porovnávacích registrů je porovnávaný s nejméně významnými 32 bity free running timeru. Pomocí komparátoru lze zjistit, zda se tyto hodnoty shodují a pokud se shodují, systémový timer generuje signál pro příslušný registr. Tento signál je přiveden pak do řadiče přerušení. Nicméně lze použít pouze 2 výstupní porovnávací registry, jelikož VideoCore GPU používá už 2 registry. Dají se použít pouze registry C1 a C3. Tento timer neobsahuje žádnou předděličku.

Adresa offsetu	Velikost	Jméno registru	Popis
0x00	32	CS	registr pro řízení a stav system timer
0x04	32	CLO	registr pro počítání (LSB)
0x08	32	CHI	registr pro počítání (MSB)
0x0C	32	C0	registr pro zápis hodnoty (output compare 0)
0x10	32	C1	registr pro zápis hodnoty (output compare 1)
0x14	32	C2	registr pro zápis hodnoty (output compare 2)
0x18	32	C3	registr pro zápis hodnoty (output compare 3)

Tabulka 4.4: registry pro systémový timer



Obrázek 4.13: schéma čítače v režimu output compare

ARM timer je založen na ARM SP804[Obrázek 4.14:], který bude popsaný níže, ale má několik rozdílů oproti standartnímu SP804[8]:

- obsahuje pouze 1 timer, SP804 obsahuje 2 timery
- timer může běžet pouze v režimu free running, jak je napsané v dokumentaci [6] ale timer se chová spíše jako by běžel v režimu periodickém
- má navíc registr, který slouží jako před dělička hodin pro timer, která je 10 bitová
- obsahuje bit, který slouží k tomu aby ARM timer běžel nebo byl zastaven, když je procesor v režimu ladění (debug mode)
- má také 32- bitový free running counter

Hodiny pro ARM timer jsou odvozeny od systémových hodin. Systémové hodiny mají frekvenci 250MHz. Tyto hodiny lze měnit v případě, že systém například přejde do úsporného režimu (low power mode). Pro přesné načasování je ale lepší použít systémový časovač [předchozí podkapitola].

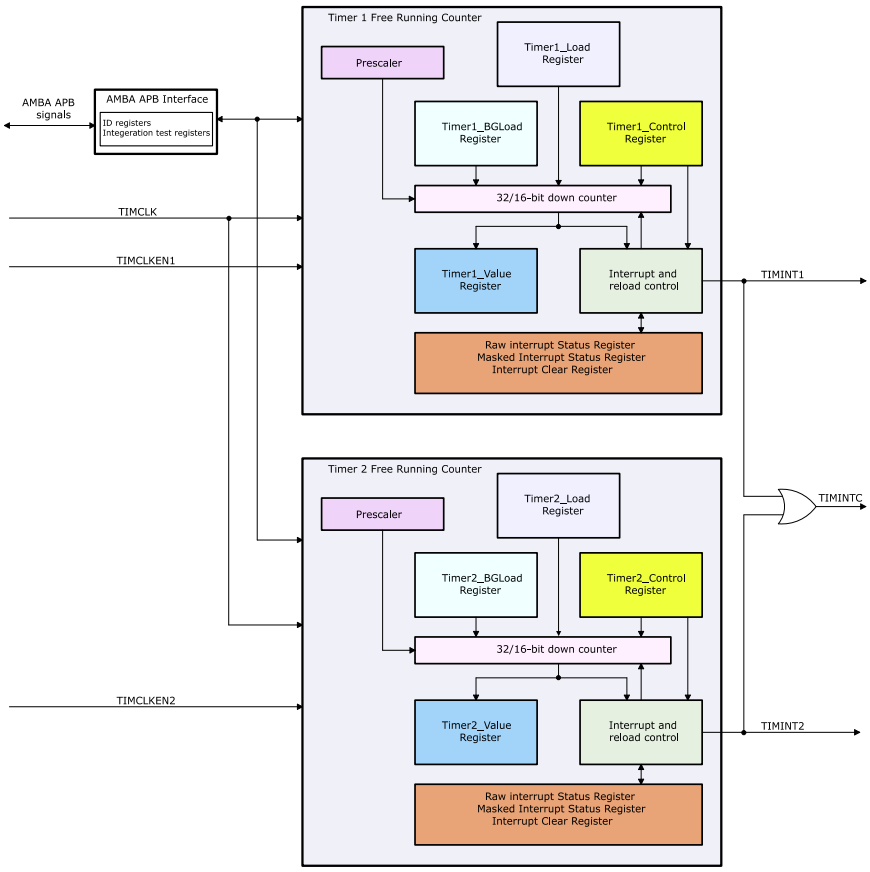
Free running counter není přítomen v SP804. Řídící (Control) registr v SP804 je 8 bitový, ale jelikož RPI obsahuje tento čítač je proto registr 32 bitový. Tento čítač lze povolit, pokud je nastaven 9 bit v řídícím registru. Jakmile je čítač povolen, dochází k jeho spuštění a následnému navyšování hodnoty. Čítač nelze nijak resetovat. Čítač má svoji vlastní před děličku, která je řízena pomocí řídícího registru. Před dělička tohoto čítače je 8 bitová. Čítač ale neumožňuje přerušení, jakmile dojde k jeho přetečení, proto lze tento čítač použít pouze, jako zpoždění to znamená, že mikroprocesor bude čekat a žádný kód nebude vykonaný.

Adresa offsetu	Velikost	Typ	Jméno registru	Popis
0x400	32	R/W	Load	nastaví hodnotu pro časovač
0x404	32	RO	Value	aktuální hodnota v časovači
0x408	32	R/W	Control	řídící registr časovače a čítače
0x40C	32	WO	IRQCLR	smazání interrupt pending bit
0x410	32	RO	RAWIRQ	nastavení bitu (interrupt pending bits), pokud čítač dosáhne nuly, samo o sobě negeneruje přerušení
0x414	32	RO	MASKIRQ	-
0x418	32	R/W	Reload	kopie Load registru
0x41C	32	R/W	Prediv	před dělička pro časovač
0x420	32	R/W	CNTR	aktuální hodnota v free runnig counteru

Tabulka 4.5: registry pro ARM timer

Rozdíl mezi Load registrem a Reload registrem je v tom, že pokud je hodnota zapsána do Load registru dojde k tomu, že čítač čítá okamžitě od této hodnoty dolů (dojde k resetu čítače). Ale pokud je hodnota zapsána do Reload registru, čítač bude čítat od této hodnoty až poté co čítač dosáhne nuly (nedochází k resetu).

ARM timer označený jako SP804 [Obrázek 4.14:] obsahuje 2 stejné časovače. Každý časovač má programovatelný 32/16-bitový čítač, který umí generovat přerušení, jakmile dosáhne 0. Oba časovače mají společné hodiny, ale hodiny se dají povolit pro každý časovač odděleně.

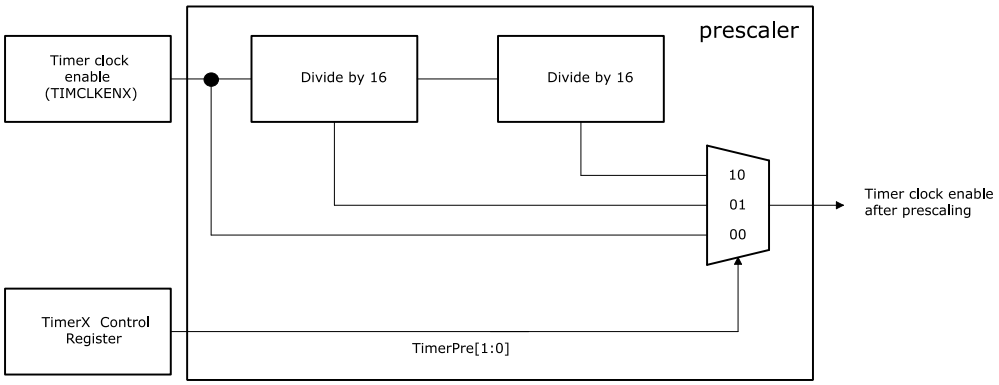


Obrázek 4.14: blokové schéma ARM SP804 modulu

Název	Typ	zdroj/cíl	Popis
TIMCLK	Input	Clock generator	Timer clock enable
TIMCLKEN1	Input	Clock generator	Timer1 clock enable
TIMCLKEN2	Input	Clock generator	Timer2 clock enable
TIMINT1	Output	Interrupt controller	Timer1 interrupt, active HIGH
TIMINT2	Output	Interrupt controller	Timer2 interrupt, active HIGH
TIMINTC	Output	Interrupt controller	Combined interrupt, active HIGH

Tabulka 4.6: význam signálů na ARM SP804 modulu

SP804 má svoji vlastní před děličku, která umožňuje nastavit pevné hodnoty 1,16,256. V řídícím registru, se nachází 2 bity (TimerPre), které se nastaví na hodnotu. Tato hodnota je přivedena na vstup multiplexoru a ten vybere hodnotu před děličky.



Obrázek 4.15: Před dělička generující výsledný hodinový signál pro čítač

Následující parametry lze naprogramovat:

- free running režim, periodický nebo one-shot režim
- 32-bitový nebo 16-bitový čítač
- před dělička s dělícím poměrem 1,16,256
- povolení nebo zakázání generování přerušení
- maskování přerušení

Aktuální hodnotu čítače můžeme kdykoliv vyčíst z registru Value.

Timer umožňuje 3 režimy:

- free running – čítač běží od své maximální hodnoty až do nuly. Dojde k přetečení a čítač počítá zase od maximální hodnoty.
- periodický - čítač si načte hodnotu, která je uložena v Load registru. Čítač tuto hodnotu snižuje až na nulu. Poté co dosáhne čítač nuly, načte si znovu hodnotu z registru Load a celý postup se opakuje.
- one-shot - čítač si načte hodnotu, která je uložena v Load registru. Čítač tuto hodnotu snižuje až na nulu a poté se zastaví.

Přerušení je generované, pokud je nastaven odpovídající bit v řídícím registru a čítač dosáhne hodnoty 0x00000000 pokud je čítač 32 bitový nebo hodnoty 0xXXXX0000 pokud je čítač 16 bitový. Nejvýznamnějších 16 bitů čítače jsou v 16 bitovém režimu ignorovány.

4.5.4 UART

RPI obsahuje dva UARTy. Mini UART (označovaný jako UART1) a PL011 UART [10] (označený jako UART0). Blokové schéma PL011 lze nalézt na [Obrázek 4.17:]. Jelikož UART1 je mini UART neumožňuje některé vlastnosti jako UART0. UART1 a UART0 se nachází na stejných vyvedených pinech z RPI proto je lepší používat UART0.

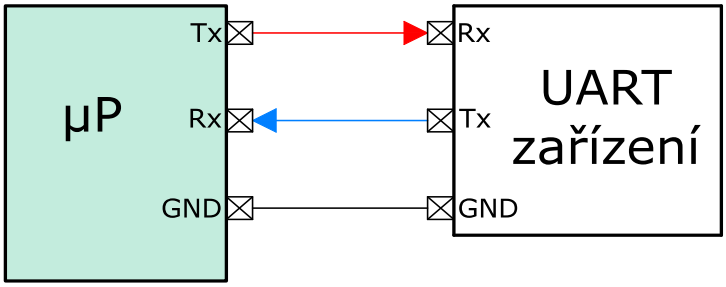
UART0 umožňuje následující funkce:

- Oddělené FIFO paměti pro vysílání (16x8) a přijímání (16x12) dat, kde 4 bity slouží k informaci o chybách v přijmu (Overrun error, Break error, Parity error, Framing error)
- Povolení FIFO pamětí pro přijímání nebo vysílání dat
- Programovatelný generátor přenosová rychlosti
- Falešné detekce start bitu
- Nastavení a zjištění přerušení
- Programovatelné hardwarové řízení toku
- Plně programovatelné sériové rozhraní:
    - Počet start bitů 1
    - Počet datových bitů 5-8
    - Paritní bit sudý, lichý nebo není nastaven
    - Počet stop bitů 1-2
    - Přenosová rychlost až do UARTCLK/16

UART1 umožňuje následující funkce:

- Počet start bitů 1
- Počet datových bitů 7 nebo 8
- Počet stop bitů 1
- Bez parity
- Přenosová rychlost je vypočítaná za pomoci systémových hodin
- Mini UART používá 8krát pře vzorkování

UART posílá data na pinu označovaný jako TX a přijímá data na pinu RX. Na RPI lze tyto piny najít na GPIO 14 UART0_TX(vysílání) a GPIO 15 UART0_RX(přijímání) nastavené v režimu speciálních funkci. UART1_TX a UART1_RX se nacházejí na stejných pinech. Je důležité si ale uvědomit, že při propojení RPI s nějakým modulem, který má v sobě UART se propojují piny, které jsou označeny jako Tx a Rx, jak je vidět na následujícím obrázku.



Obrázek 4.16: Schéma zapojení UART zařízení s mikroprocesorem

UART0 řadič obsahuje dva registry, které se používají jako hodnota dělitele pro výpočet přenosové rychlosti. Registr UART_IBRD je celé část přenosové rychlosti hodnoty dělitele. Registr UART_FBRD obsahuje zlomkovou část přenosové rychlosti hodnoty dělitele.

Pro výpočet hodnoty dělitele musíme znát hodnotu UARTu clocku a výslednou přenosovou rychlost, kterou chceme nastavit. Hodiny pro UART, který je na RPI, běží na frekvenci 3MHz. Přenosovou rychlost budeme uvažovat, že chceme nastavit typicky na 115200Bd.

Pak výpočet hodnoty dělitele je dán vztahem:

$$Baud\ rate\ divisor = \left(\frac{3 * 10^6}{16 * 115200}\right) = 1.6276$$

UART_IBRD=1, UART_FBRD=0.6276 ale registr UART_FBRD nemůže obsahovat desetinou část, proto je nutné ještě tento vztah upravit:

$$UART\_FBRD = \Big((int)\big((0.6276 * 2^6) + 0.5\big)\Big) = 40$$

Dělička přenosové rychlosti je pak:

$$1 + \frac{40}{64} = 1.625$$

Výsledná přenosová rychlost je nastavena na:

$$\left(\frac{3 * 10^6}{16 * 1.625}\right) = 115384$$

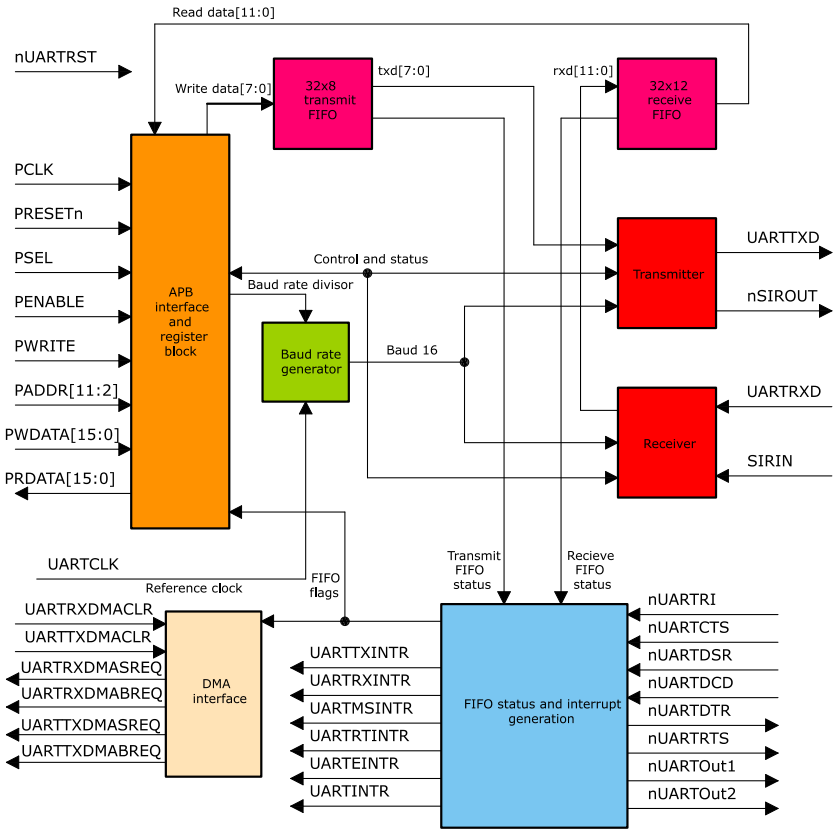
A chybovost je možné určit tímto vztahem:

$$\frac{115384 - 115200}{115200} * 100\% = 0.1597\%$$

Maximální chybovost, při použití 6-bitového registru UART_FBRD je:

$$\frac{1}{2^6} * 100\% = 1.5625\%$$

Registr UART_IBRD musí být vždy větší nebo rovný 1 a menší než 65536. Z toho vyplývá, že maximální přenosová rychlost, kterou lze nastavit je 187500Bd. Větší přenosová rychlost nelze nastavit. Obdobným způsobem lze určit, jakou minimální přenosovou rychlost, lze nastavit a to je 3Bd. Tato přenosová rychlost nebude v praxi nikdy použita. Čím větší je tedy nastavená přenosová rychlost, tím větší je chybovost.



Obrázek 4.17: Blokové schéma PL011 UARTu

Data (znaky), které chceme posílat (přijímat) pomocí UARTu můžeme dvěma způsoby:

- Pomocí FIFO pamětí
- Pomocí holding registru

Rozdíl v těchto způsobech je v tom, že pomocí holding registru lze vložit pouze jeden znak, zatímco do FIFO paměti lze vložit více znaků a nemusíme tedy čekat, až se znak pošle (přijme). Velikost této paměti je však omezena. Tento druh způsobu lze povolit v registru UART_LCRH nastavením čtvrtého bitu, který určuje, jaký způsob se má použít. Pokud je bit nastaven na jedničku použijí se FIFO paměti.

Adresa offsetu	Velikost	Typ	Jméno registru	Popis
0x00	32	R/W	UART_DR	Data register
0x18	32	RO	UART_FR	Flag register
0x24	32	R/W	UART_IBRD	Integer Baud rate divisor
0x28	32	R/W	UART_FBRD	Fractional Baud rate divisor
0x2C	32	R/W	UART_LCRH	Line Control register
0x30	32	R/W	UART_CR	Control register
0x34	32	R/W	UART_IFLS	Interupt FIFO Level Select Register
0x38	32	R/W	UART_IMSC	Interupt Mask Set Clear Register
0x3C	32	RO	UART_RIS	Raw Interupt Status Register
0x40	32	RO	UART_MIS	Masked Interupt Status Register
0x44	32	WO	UART_ICR	Interupt Clear Register

Tabulka 4.7: Registry UART0u

Přenosová rychlost pro UART1 je pak vypočítaná jako:

$$Buad\ rate = \left(\frac{250 * 10^6}{8 * (AUX\_MU\_BAUD\_REG + 1)}\right)$$

Kde 250MHz jsou hodiny pro UART. Minimální přenosovou rychlost, kterou můžeme u tohoto UARTu nastavit je 476 Baudu, kdy nastavíme registr na jeho maximálního hodnotu, tedy 65535. Maximální přenosovou rychlost, kterou můžeme u tohoto UARTu nastavit je 31,25 MBd. Uvedený vzorec lze upravit, jelikož chceme nastavit požadovanou přenosovou rychlost a ne registr. Vzorec pro nastavení hodnoty pro registr AUX_MU_BAUD_REG vypadá pak takto:

$$\text{AUX\_MU\_BAUD\_REG} = \frac{250 * 10^6}{8 * Baud\ rate} - 1$$

V registru AUX_MU_LCR_REG je možné nastavit kolik je počet datových bitů (7 nebo 8). Oproti UART0u mají FIFO paměti velikost (8x8) dat. Data jsou teda přímo zapisovaná do FIFO pamětí, které jsou oddělené. Stejně jako UART0 i UART1 umožňuje přerušení.

Adresa offsetu	Velikost	Typ	Jméno registru	Popis
0x00	3	RO	AUX_IRQ	Auxiliary Interrupt status
0x04	3	R/W	AUX_ENABLES	Auxiliary enables
0x40	8	R/W	AUX_MU_IO_REG	Mini Uart I/O Data
0x44	8	R/W	AUX_MU_IER_REG	Mini Uart Interrupt Enable
0x48	8	R/W	AUX_MU_IIR_REG	Mini Uart Interrupt Identify
0x4C	8	R/W	AUX_MU_LCR_REG	Mini Uart Line Control
0x50	8	R/W	AUX_MU_MCR_REG	Mini Uart Modem Control
0x54	8	RO	AUX_MU_LSR_REG	Mini Uart Line Status
0x58	8	RO	AUX_MU_MSR_REG	Mini Uart Modem Status
0x5C	8	R/W	AUX_MU_SCRATCH	Mini Uart Scratch
0x60	8	R/W	AUX_MU_CNTL_REG	Mini Uart Extra Control
0x64	32	RO	AUX_MU_STAT_REG	Mini Uart Extra Status
0x68	16	R/W	AUX_MU_BAUD_REG	Mini Uart Baudrate

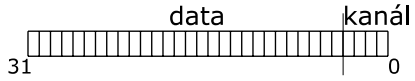
Tabulka 4.8: Registry UART1u

4.5.5 Mailboxes

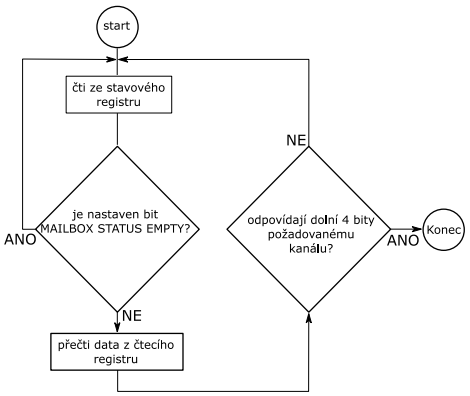
Mailboxes[11] nebo-li poštovní schránky umožňují komunikaci mezi ARM procesorem a VideoCore. Každý mailbox má své vlastní registry, které lze nalézt v tabulce 5.7, ale nás nejvíce bude zajímat mailbox0, proto adresy registrů pro další mailboxy nebudou zmíněné. Mezi funkcemi, který budeme potřebovat je čtení z mailboxu nebo zápis do mailboxu.

Postup při čtení z mailboxu:

- čteme stavový registr mailboxu pokavaď je nastaven příznak EMPTY
- poté přečteme data z čtecího registru mailboxu
- pokud nejméně významné 4 bity přečtených dat neodpovídají číslu kanálu, musíme celý proces opakovat od začátku
- horních 28 bitů z přečtených dat jsou naše požadovaná data



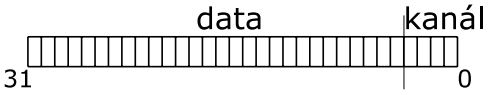
Obrázek 4.18: čtecí registr mailboxu



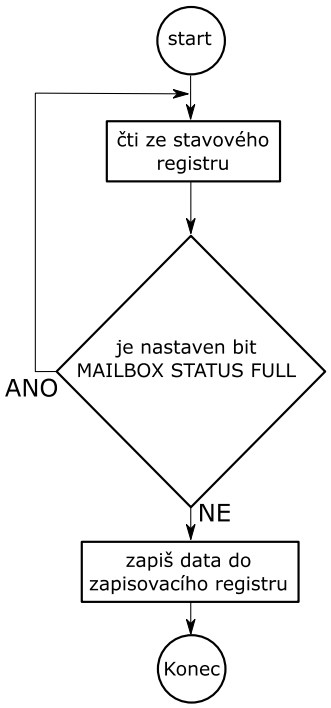
Obrázek 4.19: proces při čtení z mailboxu

Postup pro zápis do mailboxu:

- čteme stavový registr mailboxu pokavaď je nastaven příznak FULL
- horních 28 bitů opět tvoří data, která chceme zapsat, nejméně 4 významné bity slouží k určení kanálu. Takto vytvořený 32 bitový registr zapíšeme do zapisovacího registru mailboxu



Obrázek 4.20: zapisovací registr mailboxu



Obrázek 4.21: proces při zápisu do mailboxu

Mailbox registry	Adresa	Popis
BASE	0xB880	základní adresa pro mailbox registry
READ	0xB880+0x00	mailbox registr pro čtení
WRITE	0xB880+0x20	mailbox registr pro zápis
STATUS	0xB880+0x18	mailbox registr pro kontrolování stavů
STATUS registr	**Bit**	**Popis**
EMPTY	0x40000000	tento bit je nastaven ve stavovém registru, pokud není možné nic číst z mailboxu
FULL	0x80000000	tento bit je nastaven ve stavovém registru, pokud není místo k zápisu do mailboxu

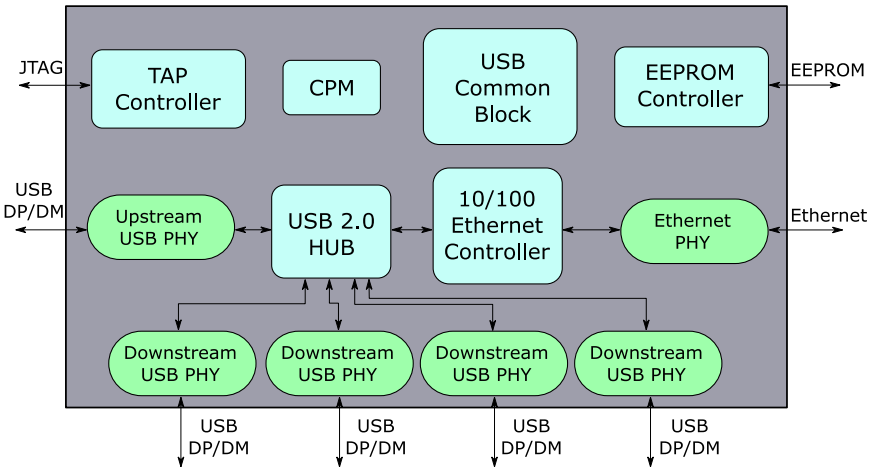
Tabulka 4.9: registry mailboxu0

Seznam kanálů pro mailbox0, které jsou potřeba, ostatní kanály nejsou uvedeny, jelikož v této práci nejsou potřeba:

1. FrameBuffer – kanál číslo 1
2. Property Tags – kanál číslo 8

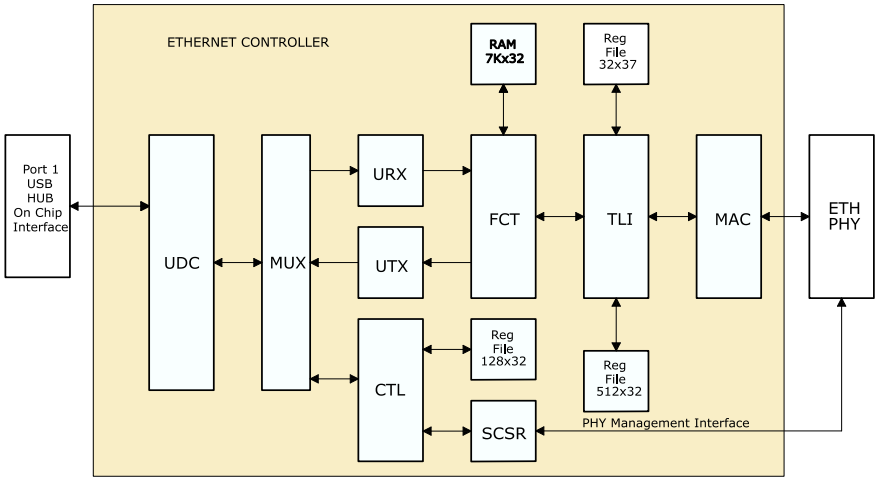
4.5.6 Ethernet

RPI 2 obsahuje čip LAN9514, který v sobě zahrnuje ethernet řadič. Komunikace s tímto ethernet řadičem probíhá pomocí USB. Jak je vidět na následujícím obrázku [Obrázek 4.22:]. Ethernet řadič je připojený k HUBu a na dalších downstream portech není nic připojené.



Obrázek 4.22: Vnitřní blokové schéma čipu LAN9514

Ethernet[17] řadič je připojen na portu 1 USB rozbočovače. USB subsystém umožňuje čtyři koncové body: řídící, vstupní blokový, výstupní blokový a přerušovací. Vstupní a výstupní blokový endpoint dovoluje ethernetu přijímat a vysílat rámce najednou.



Obrázek 4.23: Blokové schéma ethernet řadiče

Offset	Pole	Hodnota
0h	bmRequestType	C0h
1h	bRequest	A1h
2h	wValue	00h
4h	wIndex	{0h, CSR[10] Address[11:0]}
6h	wLength	04h

Tabulka 4.10: Formát čtení z registru v SETUP transakci

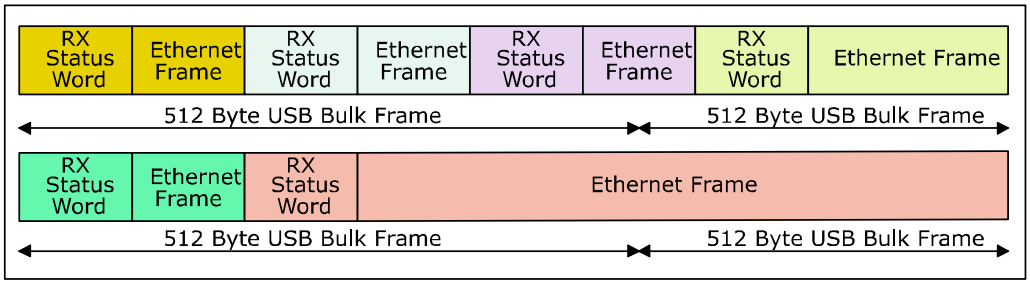
Offset	Pole	Hodnota
0h	bmRequestType	40h
1h	bRequest	A0h
2h	wValue	00h
4h	wIndex	{0h, CSR Address[11:0]}
6h	wLength	04h

Tabulka 4.11: Formát zápisu do registru v SETUP transakci

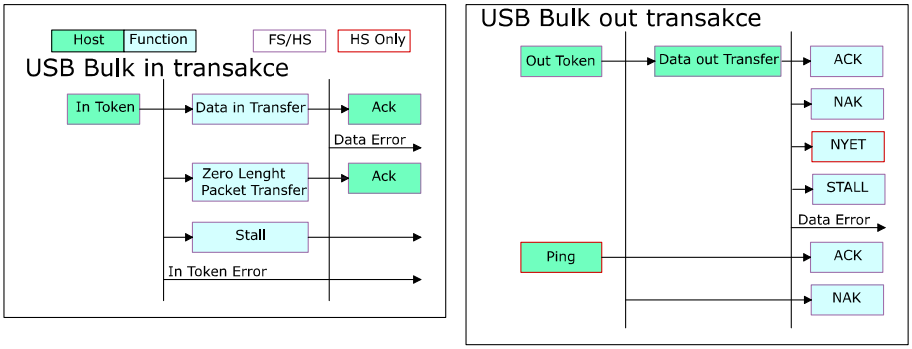
FIFO řadič používá 28 KB interní SRAM buffer. Pro Rx buffer je alokované 20 KB a pro Tx buffer je alokované 8 KB.

Ethernetové rámce jsou přímo ukládány do Rx bufferu a poté je možné tyto rámce číst jako vstupní blokové pakety pomocí USB. FIFO řadič předá uložená data do vstupní blokové roury typicky 512 nebo 64 bytů v závislosti na aktuální rychlosti (HS nebo FS) USB. Hostitel si pak tyto data může vyčíst.

[10] CSR – Control a Status registr



Obrázek 4.24: Ethernetové rámce převedené (zapouzdřené) na USB pakety



Obrázek 4.25: USB Hromadné vstupní a výstupní transakce

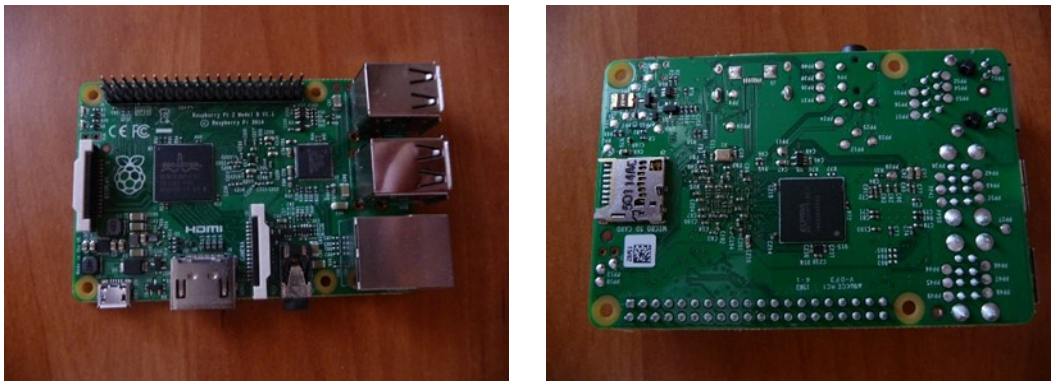
BITS	19	18		17	16	
DESCRIPTION	MACRTO_INT	RX FIFO Has Frame		TXSTOP_INT	RXSTOP_INT	
BITS	15	14	13	12	11	10:0
DESCRIPTION	PHY_INT	TXE	TDFU	TDFO	RXDF_INT	GPIO_INT

Obrázek 4.26: Formát přerušovacího paketu

5 Návrh řešení

5.1 Základní stavební bloky

Z analýzy vyplývá, že z uvedených verzí pouze tři splňují zadání této práce. RPI1 model B nebo model B+ a RPI2 model B. RPI1 model B nebyl v době psaní této práce už na českých trzích k dispozici. Tak jsem se rozhodoval pouze mezi dvěma verzemi. RPI2 se v březnu 2015 prodávala skoro za stejnou cenu jako RPI1 model B+, ale ne v každém internetovém obchodě měli RPI2 na skladě, jelikož byla tato verze oznámena před měsícem. Proto jsem se rozhodl chvíli počkat (1-2 týdny) a objednat si RPI2, který je ve všech směrech lepší než RPI1 model B.



Obrázek 5.1: Přední a zadní strana RPI

Obrázek 5.2: Osazení RPI v krabička

Kromě samostatného RPI jsou potřeba i další součástky nebo moduly, které budou demonstrovat ukázky aplikací. Tyto moduly a obrázky budou popsané až v kapitole Testování.

5.2 Nahrání jádra do RPI

Soubory, které budou potřeba, se nahrávají do RPI z SD karty, to má ovšem za následek, že pokud je soubor modifikovaný musí se opět nahrát na SD kartu. Toto je jeden z největších doposud nalezených problémů a při vývoji této knihovny by měl být tento problém určitě odstraněn.

5.2.1 Potřebné soubory

Při zapnutí RPI jako první začne pracovat GPU jádro. GPU jádro je hlavní části a umožňuje první krok k nahrání systému do RPI. GPU jádro používá určitý vnitřní firmware k zajištění přístupu na SD kartu. SD karta by měla být ve formátu FAT32 a měla by obsahovat určité soubory, které jsou potřeba pro spuštění systému.

Níže je uvedený seznam těchto souborů, které lze zdarma stáhnout zde[3]:

- bootcode.bin
- loader.bin (není už tento soubor potřeba)
- start.elf
- fixup.dat
- kernel.img

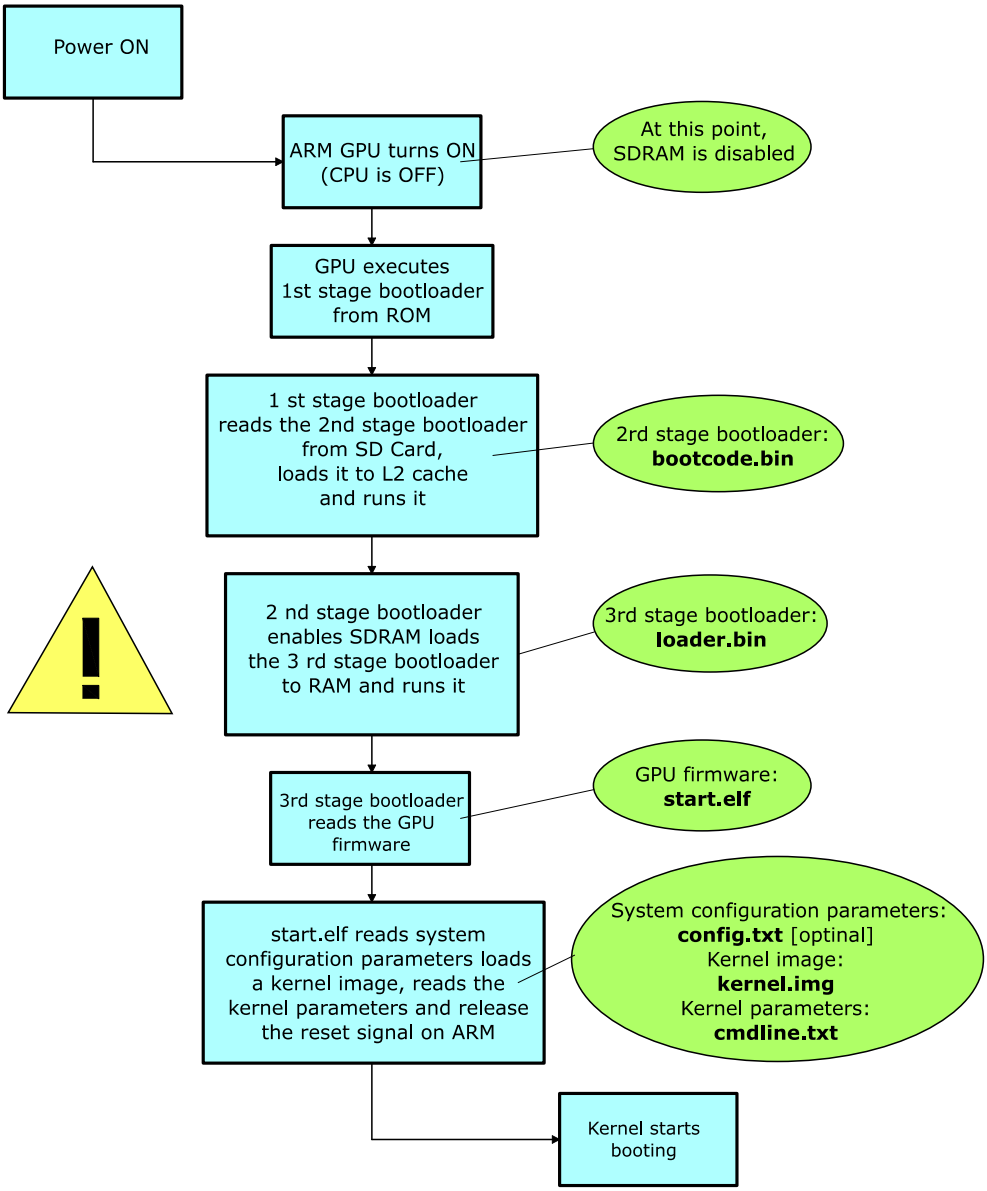
bootcode.bin – jedná se o první soubor, který je načten z SD karty do GPU jádra. Tento soubor je načten do vyrovnávací paměti L2 a má povinnost povolit SDRAM paměť systému (do této doby byla zakázána) a nahrát další fázi bootloaderu.

loader.bin – soubor obsahující kód, pomocí kterého lze načíst binární soubory s koncovkou „.elf“. Tato fáze pak může načíst soubor start.elf, který je poslední a nejzajímavější fází procesu spouštění. Soubor loader.bin už ale není zapotřebí.

start.elf – tento soubor je poslední fází bootloaderu. Načte soubor s názvem kernel.img v paměti SDRAM na adrese 0x8000. Soubor start.elf umožňuje některé možnosti konfigurace. Je-li soubor s názvem config.txt přítomen v kořenovém adresáři SD karty potom soubor start.elf načte konfigurace systému, které se nacházejí v tomto souboru. Seznam možností konfigurace lze nalézt zde [4]. Poté, co soubor start.elf tohle všechno provede, resetuje jádro ARM a ARM procesor spustí z načteného jádra startovní adresu.

fixup.dat – soubor je použit ke konfiguraci SDRAM oddílu mezi GPU a CPU.

kernel.img – jádro, které je načteno do ARM procesoru. Tento soubor obsahuje zdrojový kód. Pro RPI2 lze použít i soubor označený jako kernel7.img.



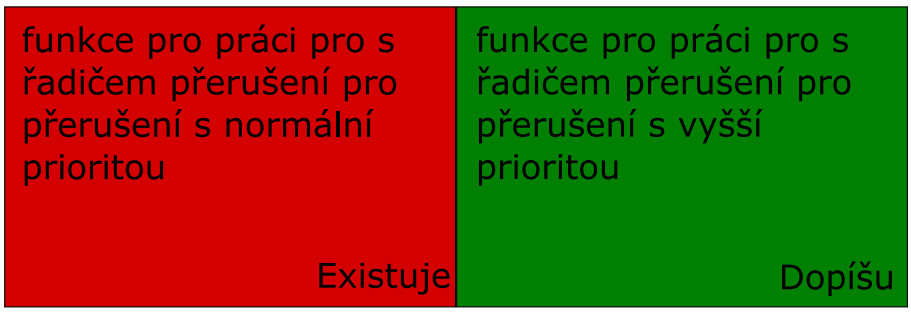
Obrázek 5.3: Proces zavedení jádra do RPI

5.2.2 Křížový překladač

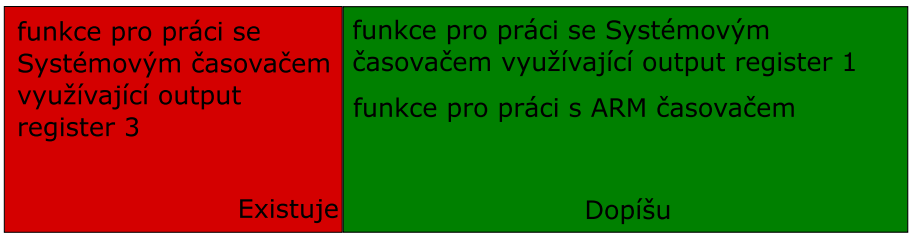
Slouží k vygenerování spustitelného kódu pro jinou platformu, než na jaké platformě je překlad zdrojových kódů spuštěn. Jako křížový překladač byl použit arm-none-eabi, který je zdarma zde ke stažení [5]. Překladač je dostupný jak pro OS Windows, tak Linux, na kterých jsem si tento překladač vyzkoušel a otestoval tím, že jsem vygeneroval a nahrál soubor kernel.img do RPI. Na Mac OS jsem tento překladač nezkoušel, jelikož nevlastním počítač Macintosh ale měl by fungovat i pod tímto OS.

5.3 Návrh knihoven

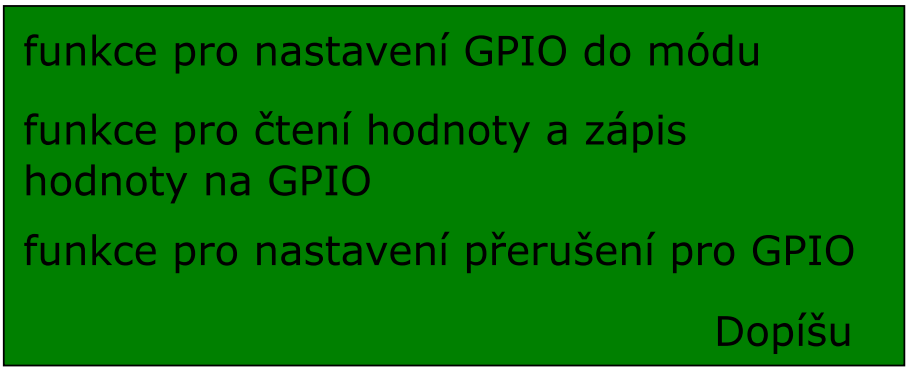
Jak bylo zmíněné v kapitole rešerše, tato práce bude postavena na knihovně USPI, kerá je open-sourcová a umožňuje práci se zařizeními připojenými pomocí USB. Tato knihovna sama o sobě nemůže fungovat, a proto uživatel napsal i další rozšířenou knihovnu pro ovládání displeje, časovače (systémový časovač output register 3) a řadiče přerušení. Na následujících obrázcích je vidět jaké knihovny a funkce budu muset dopsat, jaké knihovny modifikuji a jaké knihovny existují pro jednotlivé periférie.



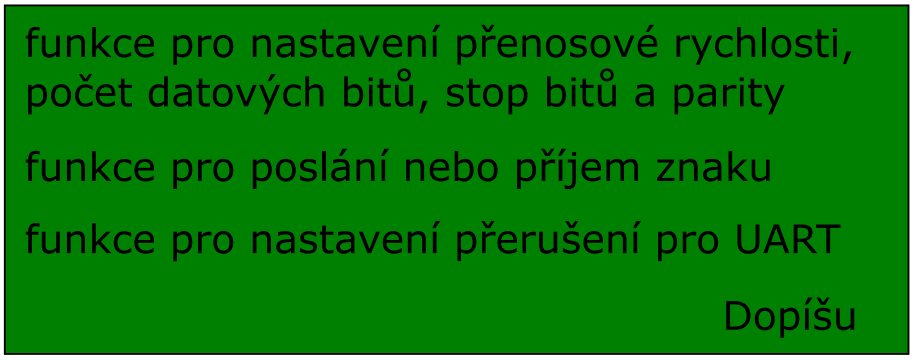
Obrázek 5.4: Návrh knihovny pro řadič přerušení



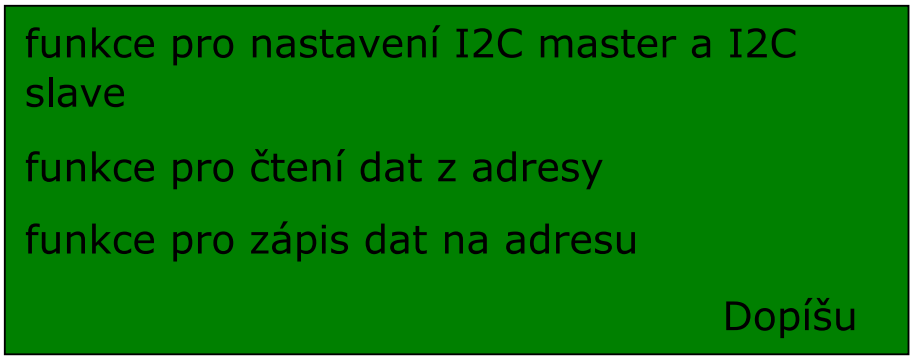
Obrázek 5.5: Návrh knihovny pro časovač



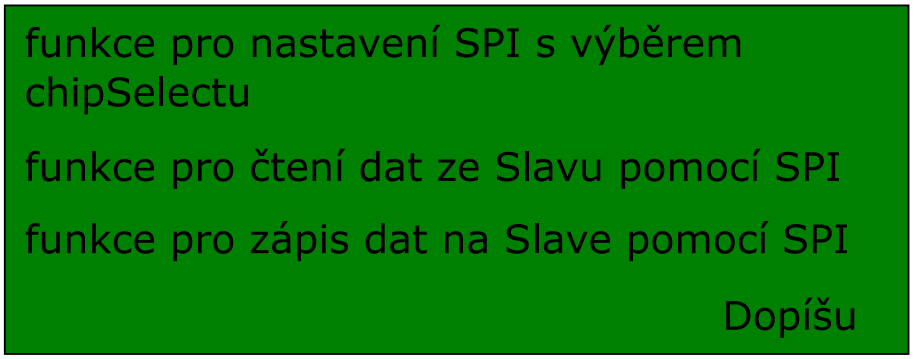
Obrázek 5.6: Návrh knihovny pro GPIO



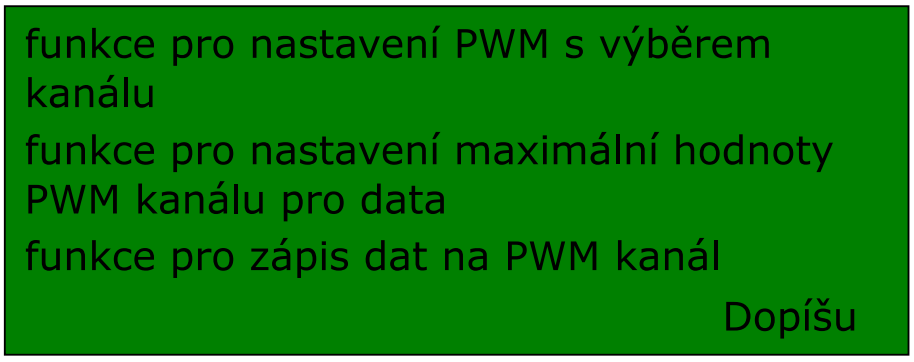
Obrázek 5.7: Návrh knihovny pro UART



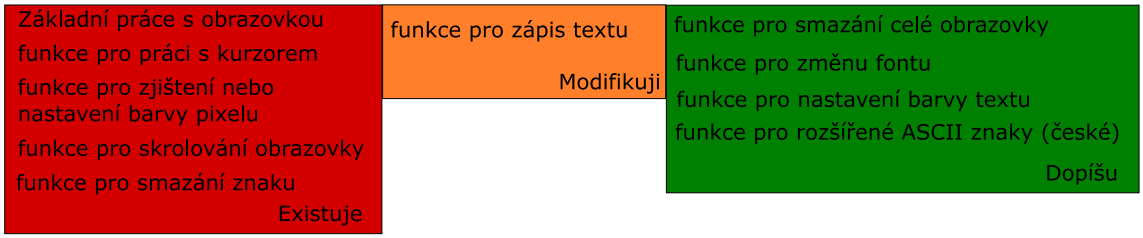
Obrázek 5.8: Návrh knihovny pro I2C



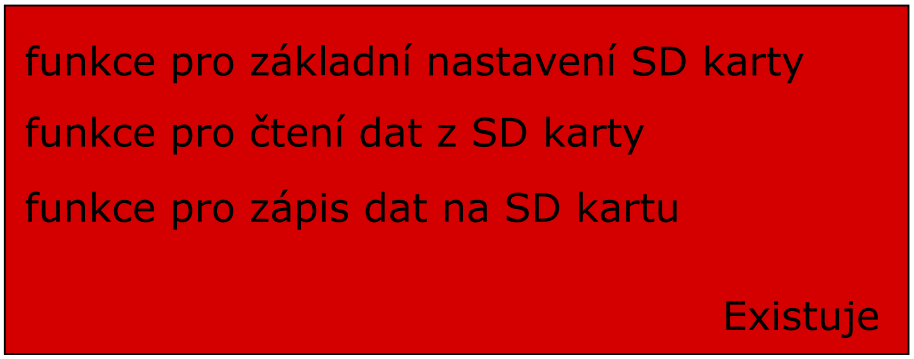
Obrázek 5.9: Návrh knihovny pro SPI



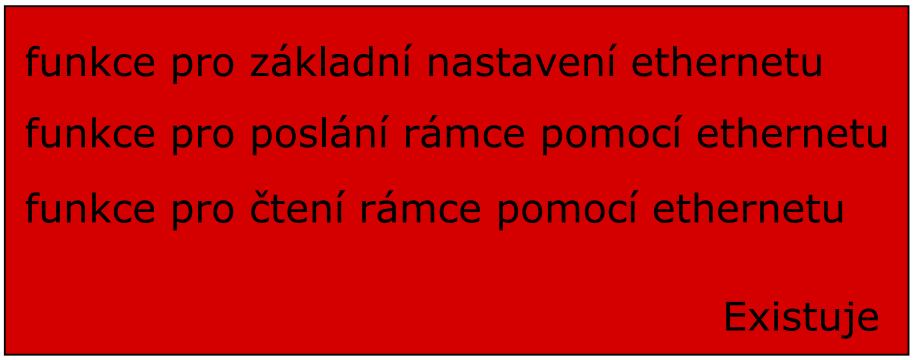
Obrázek 5.10: Návrh knihovny pro PWM



Obrázek 5.11: Návrh knihovny pro HDMI



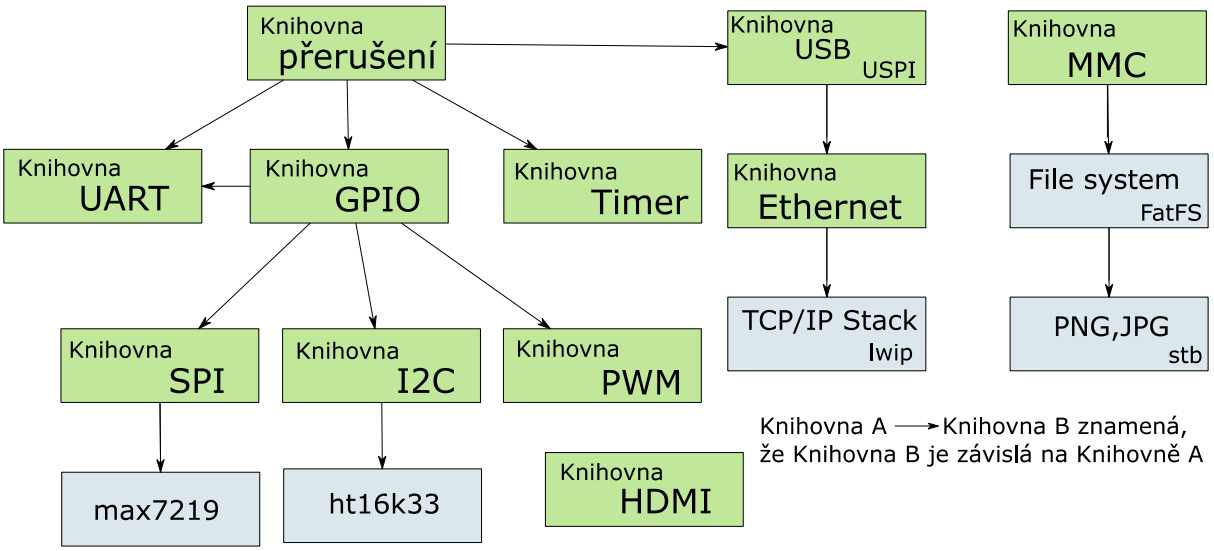
Obrázek 5.12: Návrh knihovny pro MMC



Obrázek 5.13: Návrh knihovny pro Ethernet

6 Řešení

Tato kapitola popisuje, jaké knihovny jsou potřebné k splnění diplomové práce. Ukazují, jaké funkce (potřebné struktury) jsou implementovany a jak jsou na sobě jednotlivé knihovny závislé či nezávislé. Součástí jsou i další knihovny napsané také v programovacím jazyce C. Tyto knihovny nebyly mnou implementovany, ale slouží k tomu, aby výsledná ukázková aplikace a jiné ukázkové aplikace byly zajímavější a tyto knihovny jsou tedy nad rámec této práce. Knihovny se dají použít i pro jiný vestavěný systém, jako je například používá firma STMicroelectronics u svých vývojových kitů se kterými jsem se setkal při školní výuce. Proto jsem tedy usoudil, že tyto knihovny by byly dobré do projektu zapojit. Jedná se o knihovny, které jsou zobrazené na následujícím obrázku [Obrázek 6.1:], označené jako lwip,stb a FatFS. Tyto knihovny jsou všechny open-sourcové.



Obrázek 6.1: Závislost jednotlivých knihoven

Zelená barva na obrázku [Obrázek 6.1:] značí, že tyto knihovny jsou určené přímo pro RPI. Modrá barva pak značí, že drobnými úpravami v kódu mohou být tyto knihovny použité i pro jený vestavěný systém.

6.1 Knihovna Přerušení

Tato podkapitola popisuje, jaká má tato knihovna struktury a funkce, které lze použít (interní - jdou použít mimo tuto knihovnu nebo externí - jdou použít pouze v této knihovně).

InterruptController_T - jedná se o strukturu, která v sobě zahrnuje pro každé přerušení (celkem 72 možných přerušení ale, potřeboval jsem pouze 9 přerušení), handler, který se má vykonat, pokud dojde k přerušení a parametr pro tento handler, obsahuje i číslo přerušení z vyšší prioritou (FIQ), které bude mít přednost před normálním přerušení. Pro toto přerušení s vyšší prioritou je také možné nastavit handler a parametr pro tento handler. Zde je ukázka této struktury:

- ***InterruptController_interrupt_handler *handler[72];***
- ***void *parameters[72];***
- ***InterruptController_interrupt_handler *handler_FIQ;***
- ***void *parameters_FIQ;***
- ***unsigned FIQ;***

Name function	**void InterruptController_init (InterruptController_T *pointerIT);**
Description	Základní nastavení řadiče přerušení
Parameters	• žádné další parametry kromě ukazatele na strukturu pro InterruptController
Return values	• žádná
Notes	• žádné

Name function	**void InterruptController_disconnect_IRQ (InterruptController_T *pointerIT, unsigned IRQ);**
Description	Zrušení handleru, který se má vykonat, jakmile dojde k přerušení
Parameters	• **IRQ**: číslo přerušení
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **InterruptController_init**

Name function	**void InterruptController_connect_IRQ (InterruptController_T *pointerIT, unsigned IRQ, InterruptController_interrupt_handler *handler, void *parameters);**
Description	Nastavení handleru, který se má vykonat, jakmile dojde k přerušení
Parameters	• **handler**: handler, který se vykonat, jakmile dojde k přerušení • **parameters**: parametr, který je možné nastavit pro handler • **IRQ**: číslo přerušení
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **InterruptController_init**

Name function	**void InterruptController_enable_disable_IRQ (InterruptController_T *pointerIT,unsigned IRQ,MODE_IRQ mode);**
Description	Povolení nebo zakázání přerušení
Parameters	• **IRQ**: číslo přerušení • **mode**: jedná se o výčtový typ, který určuje jetli se má povolit nebo zakázat přerušení
Return values	• žádná
Notes	• Tato funkce už je volána ve funkcích **InterruptController_connect_IRQ** nebo **InterruptController_disconnect_IRQ**

Name function	**static void InterruptController_enable_FIQ(void);**
Description	Povolení přerušení s vyšší prioritou
Parameters	• žádné
Return values	• žádná
Notes	• Tato funkce už je volána ve funkci **InterruptController_connect_FIQ**

Name function	**static void InterruptController_disable_FIQ(void);**
Description	Zakázání přerušení s vyšší prioritou
Parameters	• žádné
Return values	• žádná
Notes	• Tato funkce už je volána ve funkci **InterruptController_disconnect_FIQ**

Name function	**void InterruptController_connect_FIQ(InterruptController_T *pointerIT,unsigned FIQ,InterruptController_interrupt_handler *handler, void *parameters);**
Description	Nastavení handleru, který se má vykonat, jakmile dojde k přerušení s vyšší prioritou
Parameters	• **handler**: handler, který se vykonat, jakmile dojde k přerušení s vyšší prioritou • **parameters**: parametr, který je možné nastavit pro handler • **FIQ**: číslo přerušení s vyšší prioritou
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **InterruptController_init**

Name function	**void InterruptController_disconnect_FIQ(InterruptController_T *pointerIT,unsigned FIQ);**
Description	Zrušení handleru, který se má vykonat, jakmile dojde k přerušení s vyšší prioritou
Parameters	• **FIQ**: číslo přerušení s vyšší prioritou
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **InterruptController_init**

Důležité funkce této knihovny:

- Inicializace
- Nastavení handleru pro přerušení

6.2 Knihovna Časovač

Tato knihovna popisuje jaké struktury a funkce lze použít pro práci s časovačem. Knihovna je rozdělena na Systémový a ARM časovač. Systémový časovač může použít dva porovnávací registry, pomocí kterých lze nastavit přerušení. ARM časovač obsahuje i další free running counter, který je použit k zpoždění. Mezi nejdůležitější funkce, které jsou potřeba, je inicializace časovače, přerušení způsobené časovačem a generování určitého zpoždění. Další funkce uživatel ocení, ale nejsou zdaleka nejdůležitější.

Nejdříve budou popsaná struktura a funkce pro ARM časovač společně s free running counterem.

ARMTimer_T- jedná se o strukturu, která obsahuje informaci, zda je zapnutí timer (enable), handler, který se má vykonat, pokud dojde k přerušení, parametr pro tento handler a počet mikrosekund nastavených pro tento timer.

- ***boolean enable;***
- ***ARM_timer_interrupt_handler*handler;***
- ***void *paraSemeters;***
- ***u32 mikroSecond;***

Name function	**void ARM_timer_init(ARMTimer_T *pointer_ARM_timer, u32 mikroSecond);**
Description	Základní nastavení ARM časovače
Parameters	• **mikroSecond**: udává počet mikrosekund
Return values	• žádná
Notes	• žádné

Name function	**void ARM_timer_start(ARMTimer_T *pointer_ARM_timer);**
Description	Zapnutí ARM časovače, od této doby začíná čítat pulzy
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ARMTimer
Return values	• žádná
Notes	• Tato funkce musí být volána až po funkci **ARM_timer_init**

Name function	**void ARM_timer_stop(ARMTimer_T *pointer_ARM_timer);**
Description	Vypnutí ARM časovače
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ARMTimer
Return values	• žádná
Notes	• Tato funkce musí být volána až po funkci **ARM_timer_init**

Name function	**void ARM_timer_set_second(ARMTimer_T *pointer_ARM_timer,u32 second);**
Description	Nastavení času pro ARM časovač
Parameters	• **second**: udává počet sekund
Return values	• žádná
Notes	• Tato funkce musí být volána až po funkci **ARM_timer_init**

Name function	**void ARM_timer_set_miliSecond(ARMTimer_T *pointer_ARM_timer,u32 miliSecond);**
Description	Nastavení času pro ARM časovač
Parameters	• **miliSecond**: udává počet mili sekund
Return values	• žádná
Notes	• Tato funkce musí být volána až po funkci **ARM_timer_init**

Name function	**void ARM_timer_set_mikroSecond(ARMTimer_T *pointer_ARM_timer,u32 mikroSecond);**
Description	Nastavení času pro ARM časovač
Parameters	• **mikroSecond**: udává počet mikro sekund
Return values	• žádná
Notes	• Tato funkce musí být volána až po funkci **ARM_timer_init**

Name function	**void ARM_timer_connect_interrupt(ARMTimer_T *pointer_ARM_timer, InterruptController_T *pointerIT, ARM_timer_interrupt_handler *handler, void *parameters);**
Description	Nastavení handleru, který se má vykonat po odpočtu časovače k nule a zapnutí přerušení pro ARM časovač
Parameters	• **handler**: handler, který se vykoná, jakmile dojde k přerušení • **parameters**: parametr, který je možné nastavit pro handler
Return values	• žádná
Notes	• Tato funkce musí být volána až po funkci **ARM_timer_init** a po funkci **InterruptController_init**

Name function	**void ARM_timer_disconnect_interrupt(ARMTimer_T *pointer_ARM_timer, InterruptController_T *pointerIT);**
Description	Zrušení handleru, který se má vykonat po odpočtu časovače k nule a vypnutí přerušení pro ARM časovač
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ARMTimer a na strukuru InterruptController
Return values	• žádná
Notes	• Tato funkce musí být volána až po funkci **ARM_timer_init** a po funkci **InterruptController_init**

Name function	**void ARM_FRC_timer_delay_second(u32 second);**
Description	Nastavení zpoždění, ve kterém procesor čeká v USER modu
Parameters	• **second**: udává počet sekund, kolik bude procesor čekat
Return values	• žádná
Notes	• Free running counter

Name function	**void ARM_FRC_timer_delay_miliSecond(u32 miliSecond);**
Description	Nastavení zpoždění, ve kterém procesor čeká v USER modu
Parameters	• **miliSecond**: udává počet mili sekund, kolik bude procesor čekat
Return values	• žádná
Notes	• Free running counter

Name function	**void ARM_FRC_timer_delay_mikroSecond(u32 mikroSecond);**
Description	Nastavení zpoždění, ve kterém procesor čeká v USER modu
Parameters	• **mikroSecond**: udává počet mikro sekund, kolik bude procesor čekat
Return values	• žádná
Notes	• Free running counter

Name function	**static void ARM_timer_manager_interrupt_handler (void * parameters);**
Description	Handler,který se vykoná, jakmile dojde k přerušení a poté se pak až zavolá handler, který si nastavil uživatel
Parameters	• **parameters**: parametrem je zde ukazatel na strukturu ARMTimer
Return values	• žádná
Notes	• Jedná se o interní funkci, takže uživatel ji nezavolá

Jako druhý je zde popsaný systémový časovač a jeho porovnávací registr (Outpout compare 1) dále označený jako Timer1.

Timer1_T- struktura, která má stejně jako struktura ARMTimer_T handler a parametr pro tento handler a počet mikrosekund nastavených pro tento porovnávací registr. Ale co má navíc je, jestli má být periodický, tj pokud dojde jednou k přerušení tak jestli má být hodnota uložena znovu do porovnávacího registru. Přerušení je zde způsobené tedy tím, že je porovnaná hodnota v porovnávacím registru s hodnotou systémového časovače.

- ***Timer1_interrupt_handler* handler;***
- ***void *parameters;***
- ***boolean periode;***
- ***u32 micro_second;***

Name function	**void Timer1_init(Timer1_T *pointerTimer,u32 micro_second,boolean periode);**
Description	Základní nastavení Timer1
Parameters	• **micro_second**: udává počet mikrosekund • **periode**: určuje, jestli má být zapsaná znovu hodnota (**micro_second**) při přerušení do porovnávacího registru
Return values	• žádná
Notes	• žádné

Name function	**void Timer1_connect_interrupt (Timer1_T *pointerTimer,InterruptController_T *pointerIT,Timer1_interrupt_handler*handler, void *parameters);**
Description	Nastavení handleru, který se má vykonat při shodě porovnávacího registru a Systémového časovače a povolení přerušení pro Timer1
Parameters	• **handler**: handler, který se vykoná, jakmile dojde k přerušení • **parameters**: parametr, který je možné nastavit pro handler
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **Timer1_init** a funkci **InterruptController_init**

Name function	**void Timer1_disconnect_interrupt (Timer1_T *pointerTimer,InterruptController_T *pointerIT);**
Description	Zrušení handleru a zakázání přerušení pro Timer1
Parameters	• žádné další parametry kromě ukazatele na strukturu pro Timer1 a InterruptController
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **Timer1_init** a funkci **InterruptController_init**

Name function	**void Timer1_change_time(Timer1_T *pointerTimer,u32 micro_second);**
Description	Změna hodnoty v output compare1
Parameters	• **micro_second**: udává počet mikrosekund
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **Timer1_init**

Name function	**void Timer1_stop_period(Timer1_T *pointerTimer);**
Description	Vypnutí periodického stavu
Parameters	• žádné další parametry kromě ukazatele na strukturu pro Timer1
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **Timer1_init**

Name function	**void Timer1_start_period(Timer1_T *pointerTimer);**
Description	Zapnutí periodického stavu
Parameters	• žádné další parametry kromě ukazatele na strukturu pro Timer1
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **Timer1_init**

Důležité funkce této knihovny:

- Inicializace
- Počítání zpoždění
- Vytvoření události po uplynutí času za pomoci přerušení

6.3 Knihovna GPIO

Tato knihovna popisuje GPIOPiny její strukturu a funkce, které lze použít. Pokud je GPIOPin nastaven na speciální funkci, nelze ovšem všechny funkce použít a musí se použít knihovna pro tuto speciální funkci (alternativní funkce). Speciální funkce se zde rozumí to, že některé GPIO piny lze použít jako například UART nebo I2C, které jsou připojeny přímo k těmto řadičům a práce s nimi je jednodušší a stará se o ně jiná knihovna (Knihovna UART nebo Knihovna I2C).

GPIOPin_T - je struktura, která obsahuje handler, který se vykoná, pokud dojde k přerušení a parametr pro tento handler. Přerušení je možné způsobit různými způsoby (náběžná, sestupná hrana).

Dále obsahuje číslo GPIOPinu (tedy nejedná se o číslo pinu který je vyveden z RPI [Obrázek 4.9:]). Hodnotu, která se nachází na tomto GPIOPinu. Jako poslední obsahuje režimy do kterých GPIOPin muže být nastaven.

- ***GPIO_interrupt_handler*handler;***
- ***void *parameters;***
- ***GPIOInterrupt interrupt;***
- ***u8 pin;***
- ***u8 value;***
- ***GPIOMode mode;***

Name function	**void GPIOPin_init (GPIOPin_T *pointerGPIO, u8 pin, GPIOMode mode);**
Description	Základní nastavení GPIOPinu
Parameters	• **pin**: číslo GPIO (celkem 54 GPIO) • **mode**: výčtový typ, který umožňuje nastavit GPIO do režimu input, output, input pull up, input pull down, alternative function 0-5
Return values	• žádná
Notes	• žádné

Name function	**boolean GPIOPin_write (GPIOPin_T *pointerGPIO,u8 value);**
Description	Zápis hodnoty na GPIOPin
Parameters	• **value**: hodnota, která se má zapsat na GPIO buď 0 nebo 1
Return values	• Návratová hodnota určuje, zda došlo k zápisu hodnoty na GPIO
Notes	• Tato funkce musí být zavolána až po funkci **GPIOPin_init,** hodnota je zapsaná pokud GPIOPin je nastaven do režimu output

Name function	**boolean GPIOPin_read (GPIOPin_T *pointerGPIO,u8 *value);**
Description	Čtení hodnoty z GPIOPinu
Parameters	• **value**: hodnota, která je přečtena
Return values	• Návratová hodnota určuje, zda došlo k přečtní hodnoty z GPIO
Notes	• Tato funkce musí být zavolána až po funkci **GPIOPin_init,** hodnota je přečtena pokud GPIOPin je nastaven do režimu input

Name function	**void GPIOPin_set_alternate_function (GPIOPin_T*pointerGPIO,u8 alternateF);**
Description	Nastavení GPIO jako speciální funkce
Parameters	• **alternateF**: hodnota, alternativní funkce 0-5
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **GPIOPin_init**

Name function	**void GPIOPin_connect_interrupt (GPIOPin_T*pointerGPIO,GPIO_interrupt_handler *handler,void *parameters, GPIOInterrupt interrupt);**
Description	Nastavení handleru, který se má vykonat, jakmile dojde k přerušení
Parameters	• **handler**: handler, který se vykoná, jakmile dojde k přerušení • **parameters**: parametr, který je možné nastavit pro handler • **interrupt**: jedná se o výčtový typ, který určuje, při jaké události bude detekované přerušení. Může teda být nastaveno na RisingEdge, FallingEdge, HighLevel, LowLevel, AsyncRisingEdge, AsyncFallingEdge
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **GPIOPin_init** a po zavolání funkce **GPIOManager_init**

Name function	**void GPIOPin_disconnect_interrupt (GPIOPin_T*pointerGPIO);**
Description	Zrušení handleru, který se má vykonat, jakmile dojde k přerušení
Parameters	• žádné další parametry kromě ukazatele na strukturu pro GPIOPin
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **GPIOPin_init** a po zavolání funkce **GPIOManager_init**

Name function	**static void GPIOPin_enable_interrupt (GPIOPin_T*pointerGPIO, GPIOInterrupt interrupt);**
Description	Nastavení GPIOPinu na událost, při které bude detekované přerušení
Parameters	• **interrupt**: jedná se o výčtový typ, který určuje, při jaké události bude detekované přerušení. Může teda být nastaveno na RisingEdge, FallingEdge, HighLevel, LowLevel, AsyncRisingEdge, AsyncFallingEdge
Return values	• žádná
Notes	• Tato funkce je volána ve funkci **GPIOPin_connect_interrupt**

Name function	**static void GPIOPin_disable_interrupt (GPIOPin_T*pointerGPIO);**
Description	Zrušení události GPIOPinu, při které bude detekované přerušení
Parameters	• žádné další parametry kromě ukazatele na strukturu pro GPIOPin
Return values	• žádná
Notes	• Tato funkce je volána ve funkci **GPIOPin_disconnect_interrupt**

Jelikož všechny GPIOPiny sdílý stejný Interrupt Request Line, tak při nastavení handleru pro GPIOPin se musí zavolat před tím funkce, pro inicializaci GPIOManageru, který se stará o to, že zjistí jaký GPIOPin si vyžádal přerušení.

GPIOManager_T - struktura, která obsahuju pole ukazatelů na každou strukturu GPIOPin_T, dále ukazetel na na strukturu řadiče přerušení a zda je zapnuté přerušení.

- ***InterruptController_T *pointerIT;***
- ***boolean enable;***
- ***GPIOPin_T *allPin[54];***

Name function	**void GPIOManager_init (GPIOManager_T *pointerGPIOManager, InterruptController_T *pointerIT);**
Description	Základní nastavení GPIOManageru
Parameters	• žádné další parametry kromě ukazatele na strukturu pro GPIOManager a řadiče přerušení
Return values	• žádná
Notes	• Tato funkce musí být zavolána, pokud chceme používat přerušení pro GPIOPiny. Před touto funkcí musí být zavolána funkce pro inicializaci řadiče přerušení **InterruptController_init**

Důležité funkce této knihovny:

- Inicializace GPIOPinu
- Čtení hodnoty z GPIOPinu
- Zápis hodnoty na GPIOPin
- Nastavení GPIOPinu jako speciální funkce
- Vyvolání přerušení pro vstupní GPIOPin

6.4 Knihovna UART

Tato knihovna popisuje jaké má struktury a funkce a které lze použít, kromě přerušení, které je způsobené přijetím znaku lze i vyvolat další přerušení ty ale nejsou moc používané. Nejdůležitější funkce pro práci z touto knihovnou je tedy inicializace, poslaní znaku (řetězce znaků) a příjem znaku (způsobené přerušením nebo čekáním v hlavní smyčce).

UART_T – struktura, které obsahuje pro každé možné přerušení handler a parametr pro handler, dále obsahuje, zda je povolené přerušení a zda se pro data použijí holding registr nebo FIFO paměti;

- ***UART_interrupt_handler*handler[11];***
- ***void *parameters[11];***
- ***boolean setFIFO;***
- ***boolean enable_IC;***

Name function	**boolean UART_init(UART_T *pointerUART,BaudRate rate,DataBit data,StopBit stop,ParityBit parity);**
Description	Základní inicializace UARTu
Parameters	• **rate**: jedná se o výčtový typ, který dovoluje nastavit standartní přenosovou rychlost 9600,19200, 38400, 57600,115200 • **data**: výčtový typ, který umožňuje nastavit 5,6,7,8 datových bitů (ale většinou je potřeba 8) • **stop**: výčtový typ, který umožňuje nastavení počtu stop bitů (1,2) • **parity**: výčtový typ, který umožňuje nastavení, zda je paritní bit a jaká je parita (sudá nebo lichá)
Return values	• Návratová hodnota určuje, zda proběhla inicializace v pořádku
Notes	• žádné

Name function	**void UART_destroy(UART_T *pointerUART);**
Description	Ukončení práce s UARTem
Parameters	• žádné další parametry kromě ukazatele na strukturu pro UART
Return values	• žádná
Notes	• žádné

Name function	**void UART_send_string(UART_T *pointerUART,const void *buffer);**
Description	Poslání řetězce znaků pomocí UARTu
Parameters	• **buffer**: pole, které obsahuje jednotlivé znaky, které se mají poslat
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po **UART_init**

Name function	**void UART_send_char(UART_T *pointerUART, unsigned char byte);**
Description	Poslání znaku pomocí UARTu
Parameters	• **byte**: znak, který bude poslán
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po **UART_init**

Name function	**unsigned char UART_receive_char(UART_T *pointerUART);**
Description	Čtení znaku pomocí UARTu
Parameters	• žádné další parametry kromě ukazatele na strukturu pro UART
Return values	• Znak, který je přečten
Notes	• Tato funkce musí být zavolána až po **UART_init**

Name function	**void UART_set_FIFO(UART_T *pointerUART);**
Description	Nastavení použití FIFO pamětí
Parameters	• žádné další parametry kromě ukazatele na strukturu pro UART
Return values	• žádná
Notes	• žádné

Name function	**void UART_connect_interrupt (UART_T *pointerUART,InterruptController_T *pointerIT,UART_interrupt_handler *handler,void *parameters,UART_INTERRUPT bit);**
Description	Nastavení handleru, který se má vykonat, jakmile dojde k přerušení pro konkrétní situaci
Parameters	• **handler**: handler, který se vykoná, jakmile dojde k přerušení • **parameters**: parametr, který je možné nastavit pro handler • **bit**: jedná se o výčtový typ, který udává, pro jaké situace lze vyvolat přerušení, ale nejpoužívanější je pro příjem znaku (UART_INTERRUPT_RX=4//receive interrupt)
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **UART_init** a funkci **InterruptController_init**

Name function	**void UART_disconnect_all_interrupt (UART_T *pointerUART,InterruptController_T *pointerIT);**
Description	Zrušení všech přerušení pro UART
Parameters	• žádné další parametry kromě ukazatele na strukturu pro UART a ukazatele na strukturu pro řadiče přerušení
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **InterruptController_init**

Důležité funkce této knihovny:

- Inicializace
- Poslání znaku
- Přijmutí znaku za pomoci přerušení

6.5 Knihovna I2C

Tato knihovna umožňuje pracovat s periférii I2C jak v roli Master tak v roli Slavu. Pro Slave nebyla implantované funkce přerušení a taky nebylo odzkoušeno na reálném zařízení. Pro Master bylo odzkoušeno na reálném zařízení. RPI umožňuje pracovat se dvěma perifériemi I2C v roli Master. Nejdříve bude popsaná struktura a funkce pro I2C v roli Master.

I2CMaster_T – struktura, které obsahuje pouze jednu datovou položku a to základní adresu periférie, kterou chceme použít, jelikož RPI umožňuje použít dvě oddělené peiférie, které mají své vlastní registry.

- ***unsigned base_address***

Name function	**void I2C_master_init (I2CMaster_T *pointerI2C,u8 device, I2C_Mode mode);**
Description	Základní inicializace I2C master
Parameters	• **device**: určuje jakou periférii I2C master má být použita, buď 0 nebo 1 • **mode**: jedná se o výčtový typ, který umožňuje nastavit rychlý režim nebo normální režim tj. frekvence hodin je 400kHz nebo 100kHz
Return values	• žádná
Notes	• žádné

Name function	**boolean I2C_master_read (I2CMaster_T *pointerI2C,u8 address, void *buffer, unsigned count,unsigned *number_read);**
Description	Čtení dat ze sběrnice I2C
Parameters	• **address**: adresa slave zařízení • **buffer**: pole do kterého se ukládají přečtená data • **count**: počet dat kolik má být přečteno • **number_read**: počet přečtených dat
Return values	• Návratová hodnota určuje, zda nedošlo při čtení k chybě
Notes	• Tato funkce musí být zavolána až po **I2C_master_init**

Name function	**boolean I2C_master_write (I2CMaster_T *pointerI2C,u8 address, void *buffer, unsigned count,unsigned *number_write);**
Description	Zápis dat na sběrnici I2C
Parameters	• **address**: adresa slave zařízení • **buffer**: pole které obsahuje data k zápisu • **count**: počet dat kolik má být zapsáno • **number_write**: počet zapsaných dat
Return values	• Návratová hodnota určuje, zda nedošlo při zápisu k chybě
Notes	• Tato funkce musí být zavolána až po **I2C_master_init**

Zde bude popsané funkce pro I2C v roli Slave. RPI obsahuje pouze jednu periférii pro I2C Slave a tudíž není potřebná žádná struktura a volání funkcí pak má méně parametru.

Name function	**void I2C_slave_init (u8 address_slave);**
Description	Základní inicializace I2C slave
Parameters	• **address_slave**: nastavení Slave adresy pro tuto periférii
Return values	• žádná
Notes	• žádné

Name function	**boolean I2C_slave_write (void *buffer, unsigned count,unsigned *number_write);**
Description	Zápis dat na sběrnici I2C
Parameters	• **buffer**: pole které obsahuje data k zápisu • **count**: počet dat kolik má být zapsáno • **number_write**: počet zapsaných dat
Return values	• Návratová hodnota určuje, zda nedošlo při zápisu k chybě
Notes	• Tato funkce musí být zavolána až po **I2C_slave_init**

Name function	**boolean I2C_slave_read (void *buffer, unsigned count,unsigned *number_read);**
Description	Čtení dat ze sběrnice I2C
Parameters	• **buffer**: pole do kterého se ukládají přečtená data • **count**: počet dat kolik má být přečteno • **number_ read**: počet přečtených dat
Return values	• Návratová hodnota určuje, zda nedošlo při čtení k chybě
Notes	• Tato funkce musí být zavolána až po **I2C_slave_init**

Důležité funkce této knihovny:

- Inicializace Masteru a Slavu
- Čtení dat z adresy
- Zápis dat na adresu

6.6 Knihovna SPI

Tato knihovna popisuje práci s řadičem pro SPI. Je možné připojit až dvě Slave zařízení, pomocí chipSelect se pak vybere, s kterým zařízení se má komunikovat.

SPI_T – struktura, které obsahuje chip_select a clock_rate což je frekvence hodin pro SPI.

- ***u8 chip_select***
- ***u32 clock_rate***

Name function	**void SPI_init(SPI_T *pointerSPI,u32 clock_rate,u8 chip_select);**
Description	Základní inicializace SPI
Parameters	• **clock_rate**: frekvence hodin • **chip_select**: nastavení chip selectu v rozmezí od 0 do 1
Return values	• žádná
Notes	• žádné

Name function	**void SPI_set_clock_rate(SPI_T *pointerSPI,u32 clock_rate);**
Description	Základní inicializace SPI
Parameters	• **clock_rate**: frekvence hodin
Return values	• žádná
Notes	• Jelikož hodiny pro oba chip selecty jsou sdílené, pak pokud potřebujeme jinou frekvenci hodin

Name function	**void SPI_write(SPI_T *pointerSPI,void *bufferTx, unsigned count);**
Description	Zápis dat na SPI rozhraní
Parameters	• **bufferTx**: pole které obsahuje data k zápisu • **count**: počet dat kolik má být zapsáno
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **SPI_init**

Name function	**void SPI_read(SPI_T *pointerSPI,void *bufferRx, unsigned count);**
Description	Čtení dat z SPI rozhraní
Parameters	• **bufferRx**: pole do kterého se ukládají přečtená data • **count**: počet dat kolik má být přečteno
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **SPI_init**

Důležité funkce této knihovny:

- Inicializace
- Čtení dat ze Slavu pomocí SPI rozhraní
- Zápis dat na Slave pomocí SPI rozhraní

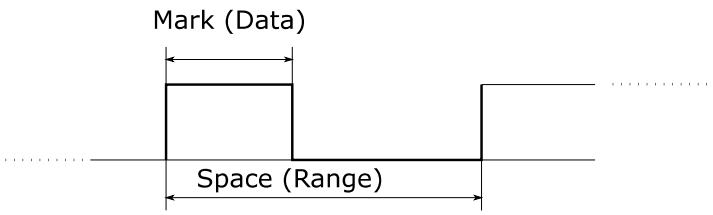
6.7 Knihovna PWM

Tato knihovna umožňuje práci s PWM řadičem. PWM řadič má dva oddělené kanály. Oba PWM kanály jsou řízení stejným PWM clockem.

PWM_T – struktura, které obsahuje číslo PWM kanálu, se kterým chceme pracovat. Maximální hodnotu dovolenou pro zápis dat. Jestli je použit Mark-Space nebo Balanced mód a zda je povolený kanál.

- ***boolean channel***
- ***unsigned range***
- ***boolean mark_space***
- ***boolean enable***

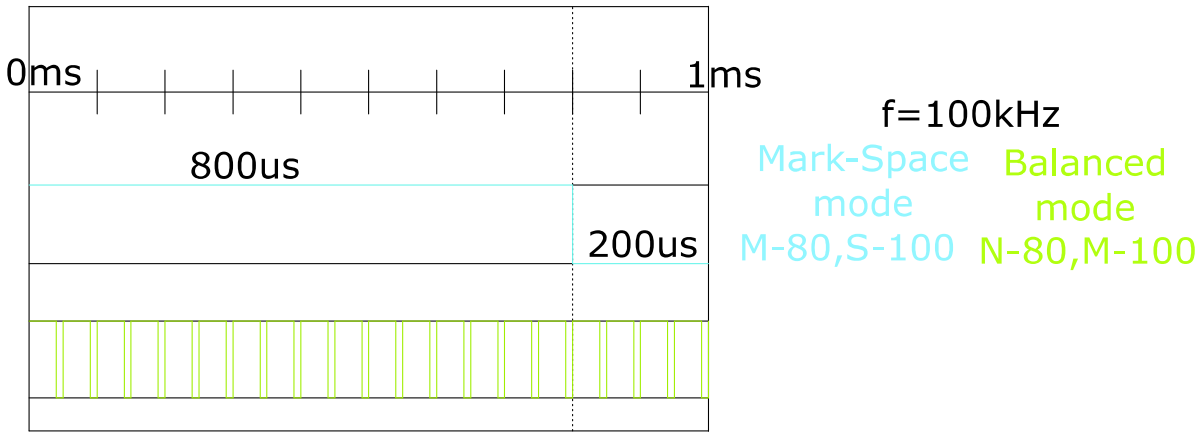
V Mark-Space módu se nastaví hodnota Mark na logickou jedničku a Space na logickou nulu. Délka těchto hodnot je daná zápisem do příslušných registrů (Data, Range) a ta pak udává počet hodinových pulzů pro PWM.



Obrázek 6.2: Nastavení PWM do mark-space modu

V Balanced módu je střída reprezentovaná jako (N/M). Kde hodnota N je reprezentovaná jako data zapsaná na PWM kanál a hodnota M je reprezentovaná jako maximální hodnota dovolená pro zápis (range). Pulzy jsou vysílány tak aby byla splněna střída.

Rozdíl v těchto dvou modech je vidět na následujícím obrázku:



Obrázek 6.3: Rozdíl mezi Mark-Space a Balanced modem

Name function	**void PWM_init (PWM_T *pointerPWM,boolean channel, boolean mark_space, unsigned range, unsigned clock_rate);**
Description	Základní inicializace PWM kanálu
Parameters	• **channel**: jaký kaál má být použit, buď kanál 0 nebo kanál 1 • **mark_space**: použití mark-space módu • **range**: maximální hodnota povolená pro data • **clock_rate**: nastavení frekvence pro PWM
Return values	• žádná
Notes	• Frekvence pro PWM je daná pomocí vzorce clock_rate=clock_rate/range

Name function	**void PWM_destroy (PWM_T *pointerPWM);**
Description	Ukončení práce s PWM kanálem
Parameters	• žádné další parametry kromě ukazatele na strukturu pro PWM
Return values	• žádná
Notes	• žádné

Name function	**void PWM_set_range(PWM_T *pointerPWM, unsigned range);**
Description	Nastavení maximální hodnoty kanálu pro data
Parameters	• **range**: maximální hodnota povolená pro data
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **PWM_init**

Name function	**void PWM_set_mode(PWM_T *pointerPWM, boolean mark_space);**
Description	Nastavení, zda se má použít MARK SPACE mód
Parameters	• **mark_space**: použití mark-space módu
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **PWM_init**

Name function	**void PWM_set_data (PWM_T *pointerPWM, unsigned data);**
Description	Zapsání dat na PWM kanál
Parameters	• **data**: hodnota, která se zapíšou na kanál. Tyta hodnota nesmí být větší jak hodnota **range**
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **PWM_init**

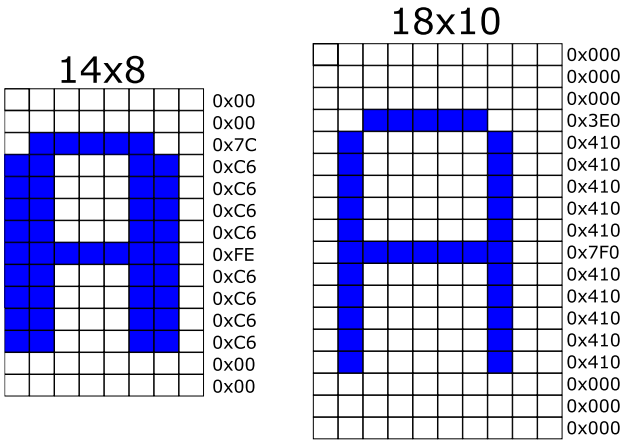
Důležité funkce této knihovny:

- Inicializace
- Zápis dat na PWM kanál

6.8 Knihovna HDMI

Tato podkapitola popisuje, jaká má tato knihovna struktury a funkce, které lze použít (externí - jdou použít mimo tuto knihovnu nebo interní - jdou použít pouze v této knihovně, pak jsou označené klíčovým slovem static).

Tato knihovna obsahuje devět odlišných fontů písma, které se liší velikostí jednotlivého písma (12x6,14x8,16x8,18x10,20x10,22x11,24x12,28x14,32x16).



Obrázek 6.4: Použití různých fontů pro zobrazení znaku

Každý font písma navíc obsahuje buď tučné písmo, nebo normální písmo. Kromě standartních ascii znaků je implementované i rozšíření ascii znaků (kvůli českým znakům). Dále tato knihovna obsahuje předem definované hodnoty pro jednotlivé barvy (celkem obsahuje 141 barev). Od základních barev jako je červená, modrá, bílá až po méně používané jako je růžová, žlutozelená nebo olivová.

Funkce, které zde budou vypsané, nebyly mnou implementované ale byly doplněné tak, aby fungovaly pro fonty a pro barvu.

ScreenDevice_T – struktura, která obsahuje několik datových položek. Kromě ukazatele na další dvě struktury (**FrameBuffer_T** a **CharGenerator_T**) obsahuje i další potřebné informace, které bude popsané přímo v kódu kvůli lepšímu přehledu:

- ***FrameBuffer_T *frameBuffer;*** ***//struktura pro Videocore***
- ***CharGenerator_T *charGenerator;*** ***// struktura informace o znaku***
- ***ScreenColor_T *buffer;*** ***// buffer displeje pro zobrazení pixelu***
- ***unsigned size;*** ***//velikost bufferu***
- ***unsigned width_display;*** ***//šířka displeje***
- ***unsigned height_display;*** ***//výška displeje***
- ***unsigned used_height;*** ***//počet řádků, které lze použít pro text***
- ***unsigned cursor_X;*** ***// pozice kurzoru na ose x***
- ***unsigned cursor_Y;*** ***// pozice kurzoru na ose y***
- ***boolean visible_cursor;*** ***//kurzor je vidět***
- ***ScreenColor_T color_text;*** ***//barva textu***
- ***FONT font;*** ***//použitý font***
- ***unsigned char previous_char;*** ***//kvůli rozšířenému českému fontu***

CharGenerator_T – struktura, která obsahuje pouze dvě datové položky. Šířku a výšku jednotlivých znaků.

- ***unsigned width_char;*** ***//šířka znaku***
- ***unsigned height_char;*** ***//výška znaku***

Pomocí mailboxu, lze získat adresu bufferu (ve struktuře framebufferu), která slouží k zobrazení pixelů na obrazovce. Máme přístup ke každému pixelu na obrazovce a je možné tedy měnit barvu každého pixelu na obrazovce. GPU očekává v mailboxu adresu struktury, která je uvedená níže.

Zde je seznam datových položek této struktury, včetně jednotlivých velikostí (každá datová položka musí mít velikost 32 bitů):

- ***unsigned int width;*** ***//Physical width of display in pixel***
- ***unsigned int height;*** ***//Physical height of display in pixel***
- ***unsigned int virt_width;*** ***//Width of the virtual Framebuffer***
- ***unsigned int virt_height;*** ***//Height of the virtual Framebuffer***
- ***unsigned int pitch;*** ***//Number of bytes between each row of the Framebuffer***
- ***unsigned int depth;*** ***// Number of bits per pixel***
- ***unsigned int offset_X;*** ***//X offset of the virtual Framebuffer***
- ***unsigned int offset_Y;*** ***//Y offset of the virtual Framebuffer***
- ***unsigned int buffer_ptr;*** ***//Address of buffer allocated by VC***
- ***unsigned int buffer_size;*** ***//Size of buffer allocated by VC***

Barevná hloubka (anglicky depth) určuje kolik různých barev lze použít při vykreslování pixelu na obrazovce. Čím více bitů umožňuje barevná hloubka, tím „lépe‘‘ může být obrázek vykreslen na obrazovku pro lidské oko.

Obrázek 6.5: Ukázka obrázků v různých barevných hloubkách, převzato z[11]

Name function	**void ScreenDevice_init (ScreenDevice_T *pointerScreenD, unsigned width, unsigned height);**
Description	Základní inicializace obrazovky s možností nastavení šířky a výšky
Parameters	• **width**: šířka obrazovky • **height**: výška obrazovky
Return values	• žádná
Notes	• Pokud je šířka a výška nastavena na nulu, tak se pomocí VideoCore určí jaká je šířka a výška připojené obrazovky

Name function	**void ScreenDevice_clear_line_end (ScreenDevice_T *pointerScreenD);**
Description	Smazání řádku na obrazovce od pozice kurzoru na ose x
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**boolean ScreenDevice_clear_whole_line_end(ScreenDevice_T *pointerScreenD,unsigned line);**
Description	Smazání celého řádku na displeji
Parameters	• **line**: řádek, který se smaže
Return values	• Návratová hodnota určuje, zda řádek byl smazaný
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

[11] https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/screen01.html

Name function	**void ScreenDevice_clear_display_end (ScreenDevice_T *pointerScreenD);**
Description	Smazání obrazovky od pozice kurzoru na ose x a od pozice kurzoru na ose y
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_erase_char (ScreenDevice_T *pointerScreenD, unsigned pos_x, unsigned pos_y);**
Description	Smazání znaku na obrazovce
Parameters	• **pos_x**: pozice x na obrazovce • **pos_y**: pozice y na obrazovce
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_display_scroll (ScreenDevice_T *pointerScreenD);**
Description	Skrolování obrazovky
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**
Name function	**int ScreenDevice_display_write_text (ScreenDevice_T *pointerScreenD, void *buffer, unsigned count);**
Description	Zápis textu na obrazovku
Parameters	• **buffer**: buffer s textem • **count**: velikost bufferu
Return values	• Návratová hodnota určuje, kolik znaků bylo zapsané na obrazovku
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_display_write_char (ScreenDevice_T *pointerScreenD,unsigned char write_char);**
Description	Zápis znaku na obrazovku
Parameters	• **write_char**: znak, který se má zapsat
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_cursor_up (ScreenDevice_T *pointerScreenD);**
Description	Nastavení kurzoru na novou pozici na ose y (kurzor se nastaví na předchozí řádku)
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• Návratová hodnota určuje, zda lze zvětšit font písma
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_cursor_down (ScreenDevice_T *pointerScreenD);**
Description	Nastavení kurzoru na novou pozici na ose y (kurzor se nastaví na další řádku)
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• Návratová hodnota určuje, zda lze zvětšit font písma
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_cursor_home (ScreenDevice_T *pointerScreenD);**
Description	Nastavení kurzoru na pozici na nulu na ose x i na ose y
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• Návratová hodnota určuje, zda lze zvětšit font písma
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_cursor_move (ScreenDevice_T *pointerScreenD, unsigned row, unsigned column);**
Description	Přesunutí kurzoru na zvolenou pozici
Parameters	• **row**: přesune kurzor po ose y • **column**: přesune kurzor po ose x
Return values	• Návratová hodnota určuje, zda lze zvětšit font písma
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_cursor_right (ScreenDevice_T *pointerScreenD);**
Description	Nastavení kurzoru na novou pozici na ose x (kurzor se nastaví na další sloupec)
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• Návratová hodnota určuje, zda lze zvětšit font písma
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_cursor_left (ScreenDevice_T *pointerScreenD);**
Description	Nastavení kurzoru na novou pozici na ose x (kurzor se nastaví na předchozí sloupec)
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• Návratová hodnota určuje, zda lze zvětšit font písma
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_tabulator (ScreenDevice_T *pointerScreenD);**
Description	Nastavení odsazení kurzoru na novou pozici na ose x na obrazovce
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• Návratová hodnota určuje, zda lze zvětšit font písma
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_new_line (ScreenDevice_T *pointerScreenD);**
Description	Nastavení kurzoru na novou řádku na obrazovce
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• Návratová hodnota určuje, zda lze zvětšit font písma
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**ScreenColor_T ScreenDevice_get_pixel (ScreenDevice_T *pointerScreenD,unsigned pos_x, unsigned pos_y);**
Description	Zjištění barvy pixelu
Parameters	• **pos_x**: pozice na ose x • **pos_y**: pozice na ose y
Return values	• Návratová hodnota určuje barvu pixelu
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_set_pixel (ScreenDevice_T *pointerScreenD, unsigned pos_x, unsigned pos_y, ScreenColor_T Color);**
Description	Nastavení barvy pixelu
Parameters	• **pos_x**: pozice na ose x • **pos_y**: pozice na ose y • **Color**: barva pixelu
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Funkce mnou implantované do této knihovny:

Name function	**boolean ScreenDevice_increase_font_size(ScreenDevice_T *pointerScreenD);**
Description	Zvětšení fontu písma
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• Návratová hodnota určuje, zda lze zvětšit font písma
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**boolean ScreenDevice_decrease_font_size(ScreenDevice_T *pointerScreenD);**
Description	Zmenšení fontu písma
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• Návratová hodnota určuje, zda lze zmenšit font písma
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_change_font_size(ScreenDevice_T *pointerScreenD,FONT_SIZE font_size);**
Description	Zmenšení fontu písma
Parameters	• **font_size**: výčtový typ, které obsahuje uvedené fonty, které byly zmíněné na začátku této podkapitoly (12x6,14x8,16x8,18x10,20x10,22x11,24x12,28x14,32x16)
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init.** Při této funkci dochází k tomu, že je obrazovka smazaná pomocí funkce **ScreenDevice_clear_all_display**

Name function	**void ScreenDevice_BackSpace(ScreenDevice_T *pointerScreenD);**
Description	Smazání předchozího znaku na obrazovce
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**boolean ScreenDevice_delete_char(ScreenDevice_T *pointerScreenD,u8 startLine,u16 delete_char,u32 count);**
Description	Smazání vybraného znaku na obrazovce
Parameters	• **startLine**: od jakého řádku se mají znaky mazat • **delete_char**: znak který se má smazat • **count**: počet znaku kterých se má smazat
Return values	• Návratová hodnota určuje, jestli bylo zmazaných počet znaků **count**
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**boolean ScreenDevice_change_char(ScreenDevice_T *pointerScreenD,u8 startLine,u16 previous_char,u16 change_char,u32 count);**
Description	Nahrazení vybraného znaku jiným znakem na obrazovce
Parameters	• **startLine**: od jakého řádku se mají znaky měnit • **previous_char**: znak který se má měnit • **change_char**: znak kterým se mění • **count**: počet znaku kterých se má změnit
Return values	• Návratová hodnota určuje, jestli bylo změněno počet znaků **count**
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_set_color_text(ScreenDevice_T *pointerScreenD,u32 color);**
Description	Nastavení barvy textu
Parameters	• **color**: barva textu, lze použít předem definovné hodnoty pro barvu (141 barev)
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_set_bold_font (ScreenDevice_T *pointerScreenD,BOLD bold);**
Description	Nastavení zda se má použít tučný font pro písmo
Parameters	• **bold**: výčtový typ, který udává, zda se použije tučný font nebo ne
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Name function	**void ScreenDevice_clear_all_display (ScreenDevice_T *pointerScreenD);**
Description	Smazání celého displeje
Parameters	• žádné další parametry kromě ukazatele na strukturu pro ScreenDevice
Return values	• žádná
Notes	• Tato funkce musí být zavolána až po funkci **ScreenDevice_init**

Důležité funkce této knihovny:

- Inicializace
- Zapsaní textu na obrazovku
- Nastavení barvy textu
- Změna fontu
- Nastavení nebo zjištění barvy pixelu

6.9 Knihovna MMC

Zde budou popsané funkce pro práci z SD kartou. Všechny uvedené funkce nebyly mnou implantovany. Pouze byly upravené a doplněné tak, aby fungovaly i pro RPI 2.

Name function	**int sd_card_init();**
Description	Inicializace SD karty
Parameters	• žádná
Return values	• Vrací, zda došlo při inicializaci k chybě nebo ne
Notes	• žádné

Name function	**size_t sd_read(uint8_t *buf, size_t buf_size, uint32_t block_no);**
Description	Čtení dat z SD karty
Parameters	• **buff:** buffer pro uložení přečtených dat • **buf_size**: velikost bufferu pro data • **block_no**: číslo logický bloku **(**sektoru**)**
Return values	• Vrací, zda došlo při čtení k chybě nebo ne
Notes	• Tato funkce musí být zavolaná až po **sd_card_init**

Name function	**size_t sd_write(uint8_t *buf, size_t buf_size, uint32_t block_no);**
Description	Zápis dat na SD kartu
Parameters	• **buf**: buffer s daty, které mají být zapsané • **buf_size**: velikost bufferu z daty • **block_no**: číslo logický bloku **(**sektoru**)**
Return values	• Vrací, zda došlo při zápisu k chybě nebo ne
Notes	• Tato funkce musí být zavolaná až po **sd_card_init**

Důležité funkce této knihovny:

- Inicializace
- Čtení dat z SD karty
- Zápis dat na SD kartu

6.10 Knihovna Ethernet

Tato podkapitola popisuje, jaká má tato knihovna struktury a její funkce, které jsou potřebné pro práci s tímto zařízením. Ethernet je připojen k RPI pomocí USB. Jelikož knihovna USPI umožňuje práci se zařízeními připojných pomocí USB, byly tyto funkce už implementované v knihovně a mnou jenom patřičně upravené.

Struktura **SMSC951x_T** obsahuje tyto datové položky:

- ***TUSBDevice deviceUSB***
- ***TUSBEndpoint *endpointBulkIn***
- ***TUSBEndpoint *endpointBulkOut***
- ***TUSBEndpoint *endpointInterrupt***
- ***u8 MAC_address[6]***
- ***u8 *bufferTx***
- ***TUSBRequest *m_pURB***
- ***u8 *report_buffer***

Name function	**void SMSC951x_init(SMSC951x_T *pointer, TUSBDevice * deviceUSB);**
Description	Základní inicializace ethernetu
Parameters	• **deviceUSB:** ukazatel na funkce tohoto zařízení
Return values	• žádná
Notes	• žádné

Name function	**int SMSC951x_send_frame (SMSC951x_T *SMSC951x, const void *bufferTx, u32 length);**
Description	Poslání rámce pomocí ethernetu
Parameters	• **bufferTx**: buffer s daty k poslání pomocí ethernetu, velikost bufferu je maximálně 1600 bytů • **length**: velikost bufferu
Return values	• Vrací, zda rámec byl uspěšné poslaný nebo došlo k chybě v poslání
Notes	• Tato funkce může být použita až po inicializaci ethernetu

Name function	**int SMSC951x_receive_frame (SMSC951x_T *SMSC951x, void *bufferRx, u32 *length);**
Description	Čtení rámce pomocí ethernetu
Parameters	• **bufferRx**: buffer pro uložení přečteného rámce, velikost bufferu je maximálně 1600 bytů • **length**: počet bytu uložených v přečteném rámci
Return values	• Vrací, zda je rámec ke čtení k dispozici nebo ne
Notes	• Tato funkce může být použita až po inicializaci ethernetu

Důležité funkce této knihovny:

- Inicializace
- Čtení rámce pomocí ethernetu
- Poslání rámce pomocí ethernetu

7 Testování

V této kapitole bude popsané, jak jednotlivé knihovny byly testovany a jaký další hardware byl dokoupen, aby bylo ověřená funkčnost jednotlivých knihoven. Dále zde budou uvedeny i další knihovny (napsané mnou) pro ovládání tohoto hardwaru. Při vybírání hardwaru, který by otestoval knihovny, byl důraz kladen na jednoduché zařízení a na nízkou cenu těchto zařízení.

Seznam rozšířeného hardwaru:

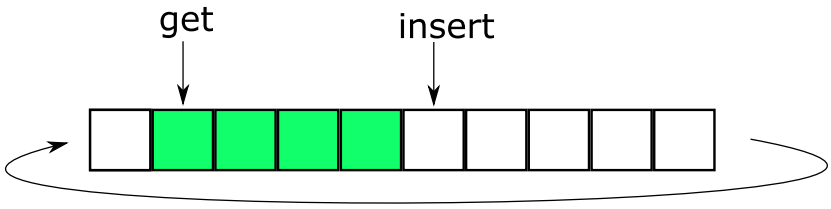
- Převodník z UARTu na USB CP2102 (sériová linka)
- Jtron 8-Digital Display Module (SPI)
- Keyestudio 8×8 Matrix I2C LED Displej module (I2C)
- mircoSD karta a redukce (nahrání souborů, uložení, otevření)
- Monitor z vstupem na HDMI nebo DVI (odpovídající kabel)
- UTP kabel (ethernet)

Seznam dalších součástek:

- Nepájivé pole
- Led dioda a tlačítka (odpory)
- Drátové propojky typu Female-Male a Female-Female
- Převodník napěťových úrovní (kvůli bezpečnosti)
- Klávesnice s USB připojením (výsledná ukázková aplikace)

7.1 Ukázkové aplikace

Při ukázkových aplikacích, které využívají například přerušení, byla použita datová struktura Kruhový buffer (Circle buffer) k ukládání dat. Jedná se o frontu s fixní délkou. Kruhový buffer obsahuje dva ukazatele. Ukazatel kam se můžou vkládat prvky (insert) a ukazatel odkud se můžou vybírat prvky (get). Ukázka kruhového bufferu je na obrázku 7.1.



Obrázek 7.1: Kruhový buffer

CircleBuffer_T - struktura kruhového bufferu, která má 5 členů. Obsahuje tedy informace o paměti, velikost paměti, kam se má uložit do paměti prvek, odkud se má vzít z paměti prvek a počet prvků které jsou v paměti uložené.

- ***void *buffer***
- ***u32 size_buffer***
- ***u32 insert***
- ***u32 get***
- ***u32 number_element***

Tato datová struktura má taky svoje vlastní funkce:

Name function	**void CircleBuffer_init(CircleBuffer_T *CB,u32 size_buffer);**
Description	Základní nastavení kruhového bufferu
Parameters	• **size_buffer**: velikost kruhového bufferu
Return values	• žádná
Notes	• Dochází k alokování paměti pro **void *buffer**

Name function	**void CircleBuffer_insert(CircleBuffer_T *CB,void *buffer);**
Description	Vložení prvku do kruhového bufferu
Parameters	• **buffer**: prvek, který se má vložit
Return values	• žádná
Notes	• Tato funkce může být použita až po inicializaci **CircleBuffer_init**, před zavoláním této funkce by mělo být zkontrolované, jestli kruhový buffer není plný **CircleBuffer_full**

Name function	**void* CircleBuffer_get(CircleBuffer_T *CB);**
Description	Vyjmutí prvku z kruhového bufferu
Parameters	• žádné další parametry kromě struktury kruhového bufferu
Return values	• žádná
Notes	• Tato funkce může být použita až po inicializaci **CircleBuffer_init**, před zavoláním této funkce by mělo být zkontrolované, jestli kruhový buffer není prázdný **CircleBuffer_empty**

Name function	**boolean CircleBuffer_full(CircleBuffer_T *CB);**
Description	Testování zda je kruhový buffer plně obsazený
Parameters	• žádné další parametry kromě struktury kruhového bufferu
Return values	• Vrací, zda je kruhový buffer plně obsazený nebo ne
Notes	• Tato funkce může být použita až po inicializaci **CircleBuffer_init**

Name function	**boolean CircleBuffer_empty(CircleBuffer_T *CB);**
Description	Testování zda je kruhový buffer prázdný
Parameters	• žádné další parametry kromě struktury kruhového bufferu
Return values	• Vrací, zda je kruhový buffer prázdný nebo ne
Notes	• Tato funkce může být použita až po inicializaci **CircleBuffer_init**

Name function	**void CircleBuffer_destroy(CircleBuffer_T *CB);**
Description	Ukončení práce s kruhovým bufferem
Parameters	• žádné další parametry kromě struktury kruhového bufferu
Return values	• žádná
Notes	• Dochází k dealokování paměti pro **void *buffer**

7.1.1 První ukázková aplikace

První ukázková aplikace testuje knihovny GPIO, Timer a Přerušení, takže nejsou potřebné žádné součástky navíc a je pouze použité to, co se nachází na RPI. V aplikaci lze zvolit, jaký timer má být použit (ARM nebo Systémový) pro přerušení. V hlavní smyčce pak každou sekundu dochází k tomu, že je LED dioda (aktivní - zelená LED dioda) rozsvícena nebo zhasnuta. Pomocí přerušení od timeru

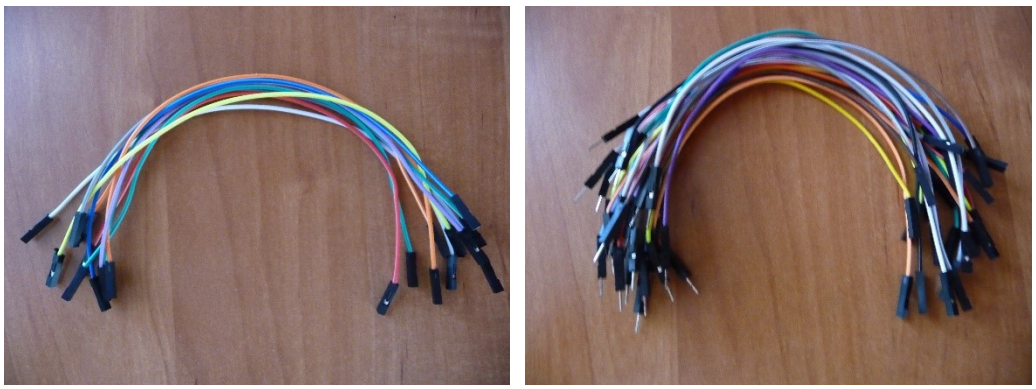
pak dochází každou sekundu k tomu, že je druhá LED dioda (červená LED dioda) rozsvícena nebo zhasnuta. To má za následek to, že v jeden čas svítí obě LED diody, nebo jsou zhasnuté.

Tím došlo k správnému otestování knihovny GPIO, Timeru a Přerušení.

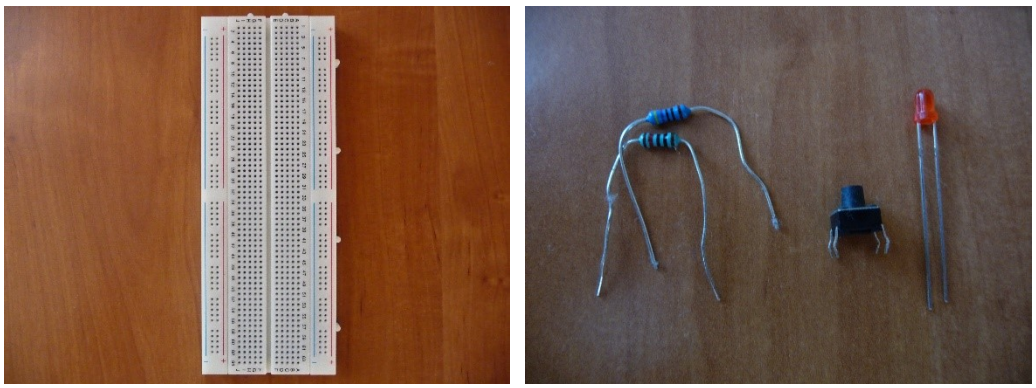
7.1.2 Druhá ukázková aplikace

Pro druhou ukázkovou aplikaci už je potřebné použít součástky, které nejsou součástí RPI. Tyto součástky pak testují piny, které jsou vyvedené z RPI[Obrázek 4.9:]. Součástky (LED dioda, tlačítko), které jsem zvolil, jsou jednoduché a nepotřebují žádné složité zapojení a cena je v řádech jednotkách korun. Kromě těchto součástek je potřeba i nepájivé pole [Obrázek 7.3:], do kterého se osadí tyto součástky. Dále je potřeba i drátové propojky[Obrázek 7.2:] typu Female-Male, které slouží k tomu, aby vybrané piny na RPI byly propojené z nepájivým polem. Pak stačí zapojit vybrané součástky podle obrázku 7.4.

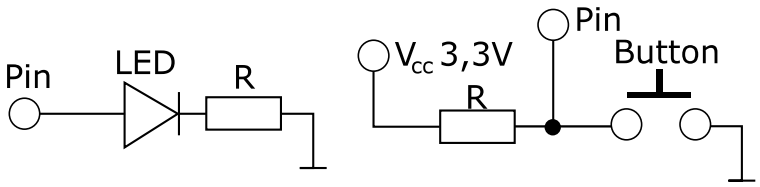
Tato aplikace pak testuje opět knihovny GPIO, Timer a Přerušení, ale s tím rozdílem, že přerušení je způsobeno stiskem tlačítka. V hlavní smyčce dochází k rozsvícení (zhasnutí) LED diody (aktivní - zelená LED dioda) v intervalu jedné sekundy. Pokud dojde k přerušení, které je zde způsobené stiskem tlačítka, dojde k obsluze handleru ve kterém se rozsvítí LED dioda (připojená přes pin). Při dalším stisku tlačítka je pak LED dioda zhasnuta. Takto se střídá rozsvícení a zhasnutí LED diody.



Obrázek 7.2: Drátové propojky Female-Female a Female-Male



Obrázek 7.3: Nepájivé pole a součástky pro ověření pinů



Obrázek 7.4: Schéma zapojení

Tím došlo k správnému otestování knihovny GPIO, Timeru a Přerušení.

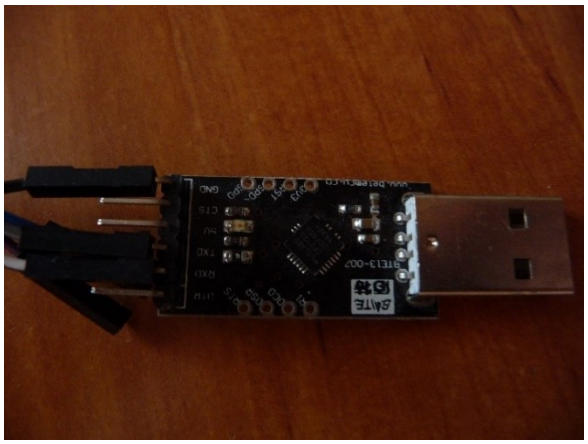
7.1.3 Třetí ukázková aplikace

Tato ukázková aplikace si klade důraz na otestování knihovny UART. Kromě této knihovny je zapotřebí také knihovny Přerušení a Timer jako v předešlých ukázkových aplikací.

Nejjednodušší způsob jak otestovat tuto periférii je pomocí převodníku z UARTu na USB [Obrázek 7.5:], který bude sloužit jako sériová linka. Pomocí programu Advanced Serial Port Terminal[12] lze ověřit funkčnost. Tento program má jednoduché grafické rozhraní, na kterém lze nastavit 4 parametry potřebné pro UART. Tyto parametry musejí být nastavené na stejné hodnoty, jak v tomto programu, tak v knihovně UART pro RPI. V tomto případě jsem zvolil tyto parametry:

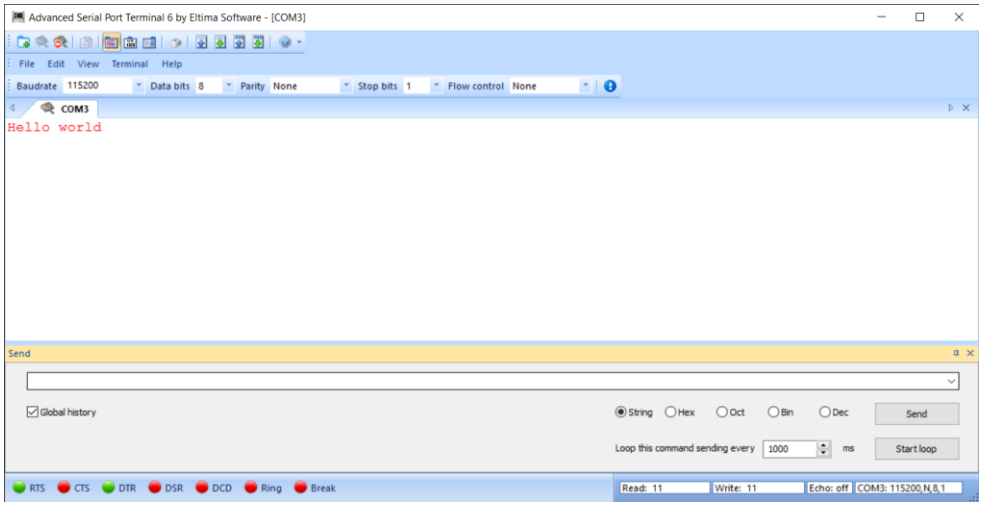
- Baudrate=115200
- Data Bit=8
- Parity=None
- Stop Bit=1

Při této aplikaci už dojde k použití Kruhového bufferu, který byl zmíněný na začátku této kapitoly. Tento kruhový buffer bude sloužit k tomu, že bude ukládat znaky poslané pomocí sériové linku z PC do RPI. Přerušení je zde způsobené pomocí UARTu tím, že přijde nový znak. Tento znak se uloží do kruhového bufferu v případě, že není plný. Hlavní smyčka pak slouží k tomu, že se vezmou všechny znaky, které jsou uložené v kruhovém bufferu a opět se pošlou zpět do PC pomocí UARTu. Timer zde slouží pouze ke zpoždění, které jsem nastavil na 500ms.



Obrázek 7.5: Převodník z UARTu na USB CP2102

[12] http://www.eltima.com/products/serial-port-terminal/

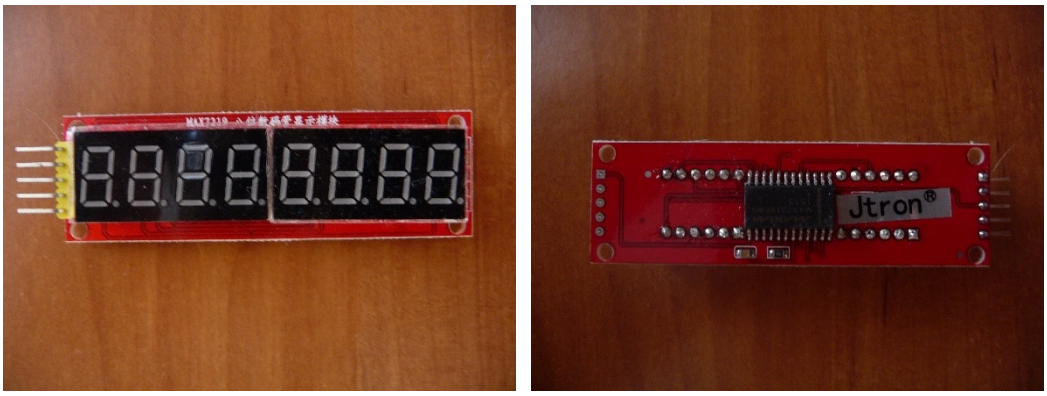


Obrázek 7.6: Ukázka programu Advanced Serial Port Terminal

Tím došlo k správnému otestování knihovny UART, Timeru a Přerušení.

7.1.4 Čtvrtá ukázková aplikace

Tato ukázková aplikace ověřuje hlavně funkčnost knihovny SPI. Z důvodů použití dalšího hardwaru bylo zapotřebí ještě doplnit tuto ukázkovou aplikaci o další knihovnu pro řadič max7219[18]. Tento řadič je připojen k displeji, který umožňuje práci až z osmi sedmisegmentovkami [Obrázek 7.7:].



Obrázek 7.7: Jtron 8-Digital Display Module (SPI)

Zde je popis základních funkcí pro tento displej:

Name function	**boolean max7219_init_display(SPI_T *pointerSPI);**
Description	Základní inicializace displeje s řadičem max7219
Parameters	• žádné další parametry kromě ukazatele na strukturu pro SPI
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí SPI nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **SPI_init**

Name function	**boolean max7219_clear_all_display(SPI_T *pointerSPI);**
Description	Smazání celého displeje (nastavení na nuly)
Parameters	• žádné další parametry kromě ukazatele na strukturu pro SPI
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí SPI nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **max7219_init_display**

Name function	**boolean max7219_clear_digit_display(SPI_T *pointerSPI,u8 rank_digit);**
Description	Smazání pouze jedné sedmisegmentovky (nastavení na nuly)
Parameters	• **rank_digit**: pozice sedmisegmentovky, která lze nastavit od 1 až po 8
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí SPI nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **max7219_init_display**

Name function	**boolean max7219_turn_Off_display(SPI_T *pointerSPI);**
Description	Vypnutí displeje ani jedna segmisegmentovka nesvítí
Parameters	• žádné další parametry kromě ukazatele na strukturu pro SPI
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí SPI nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **SPI_init**

Name function	**boolean max7219_turn_On_display(SPI_T *pointerSPI);**
Description	Zapnutí displeje segmisegmentovka svítí
Parameters	• žádné další parametry kromě ukazatele na strukturu pro SPI
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí SPI nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **SPI_init**

Name function	**boolean max7219_scan_limit_register(SPI_T *pointerSPI,Scan_Limit display_digit);**
Description	Nastavení s kolika sedmisegmentovkami je možné pracovat
Parameters	• **display_digit**: výčtový typ, který umožňuje nastavit od 1 do 8 sedmisegmentovek, které mohou být rozvícení
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí SPI nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **SPI_init**

Name function	**boolean max7219_intensity_display(SPI_T *pointerSPI,u8 intensity);**
Description	Nastavení jakou intenzitu světla má mít sedmisegmentovka
Parameters	• **intenzity**: intenzita světla nebo-li jas segmisegmentovky, umožňuje nastavit až 16 odlišných jasů v rozmezí hodnot od 0x0 do 0xF
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí SPI nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **max7219_turn_On_display**

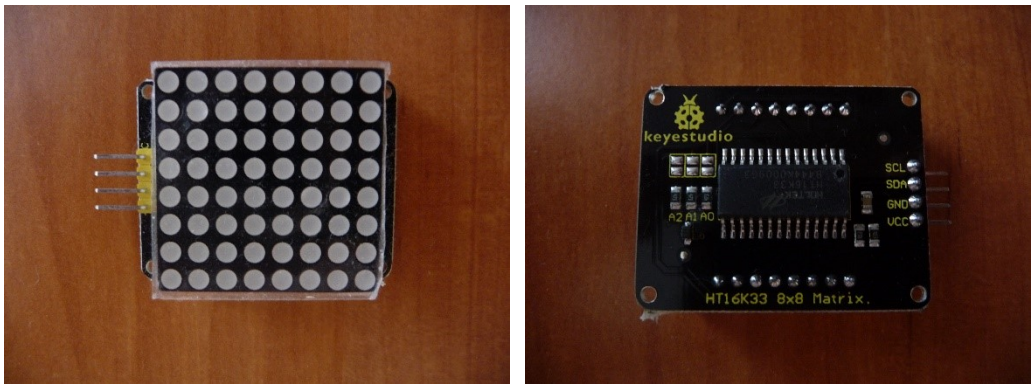
Name function	**boolean max7219_show_digit_display(SPI_T *pointerSPI,u8 rank_digit,u8 digit);**
Description	Zobrazení čísla na displeji pomocí segmentů
Parameters	• **rank_digit**: pozice sedmisegmentovky v rozmezí od 1 do 8 • **digit**: číslo, které je zobrazeno na sedmisegmentovce v rozmezí od 0 do 9
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí SPI nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **max7219_init_display**

Kromě knihovny SPI, přes které je připojen tento modul, je zapotřebí i knihovna Timer. Pomocí displeje je pak zobrazen čas, jak dlouho je RPI spuštěno. V hlavní smyčce pak dochází k tomu, že tento čas zobrazený na displeji je aktualizován každou sekundu, tím že timer čeká v hlavní smyčce. Displej tedy používá jenom 6 sedmisegmentovek s tím, že první dvě slouží jako hodina, druhé dvě jako minuta a třetí dvě jako sekunda času. Displej zobrazuje 24 hodinový formát času (00:00:00-23:59:59).

Tím došlo k správnému otestování knihovny SPI, max7219 a Timeru.

7.1.5 Pátá ukázková aplikace

Demonstrující aplikace ověřuje funkčnost knihovny I2C. Jelikož jsem použil další hardware, bylo zapotřebí ještě doplnit o další knihovnu pro řadič ht16k33[19], který tento hardware používá. Jedná se o displej s 8x8 LED diodami [Obrázek 7.8:]. Tento



Obrázek 7.8: Keyestudio 8×8 Matrix I2C LED Displej module

Zde je popis základních funkcí pro tento displej:

Name function	**boolean ht16k33_init_display (I2CMaster_T *pointerI2C);**
Description	Základní inicializace displeje s řadičem ht16k33
Parameters	• žádné další parametry kromě ukazatele na strukturu pro I2CMater
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí I2C nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **I2C_master_init**

Name function	**boolean ht16k33_show_digit(I2CMaster_T *pointerI2C,u8 digit);**
Description	Zobrazení čísla (pomocí LED) na displeji
Parameters	• **digit**: číslo, které je zobrazeno na displeji v rozsahu od 0 do 9
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí I2C nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **ht16k33_init_display**

Name function	**boolean ht16k33_LED_all_Off(I2CMaster_T *pointerI2C);**
Description	Vypnutí všech LED diod na displeji
Parameters	• žádné další parametry kromě ukazatele na strukturu pro I2CMater
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí I2C nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **ht16k33_init_display**

Name function	**boolean ht16k33_LED_all_On(I2CMaster_T *pointerI2C);**
Description	Zapnutí všech LED diod na displeji
Parameters	• žádné další parametry kromě ukazatele na strukturu pro I2CMater
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí I2C nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **ht16k33_init_display**

Name function	**boolean ht16k33_LED_On(I2CMaster_T *pointerI2C,u8 row, u8 column);**
Description	Zapnutí (rozsvícení) LED diody na zvoleném místě
Parameters	• **row:** řádek displeje v rozmezí od 1 do 8 • **column:** sloupec displeje v rozmezí od 1 do 8
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí I2C nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **ht16k33_init_display**

Name function	**boolean ht16k33_LED_Off(I2CMaster_T *pointerI2C,u8 row, u8 column);**
Description	Vypnutí (zhasnutí) LED diody na zvoleném místě
Parameters	• **row:** řádek displeje v rozmezí od 1 do 8 • **column:** sloupec displeje v rozmezí od 1 do 8
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí I2C nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **ht16k33_init_display**

Name function	**boolean ht16k33_turn_On_display(I2CMaster_T *pointerI2C);**
Description	Zapnutí displeje
Parameters	• žádné další parametry kromě ukazatele na strukturu pro I2CMater
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí I2C nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **I2C_master_init**

Name function	**boolean ht16k33_dimming_set(I2CMaster_T *pointerI2C,u8 dimming);**
Description	Nastavení jasu LED diod
Parameters	• **dimming**: umožňuje nastavit až 16 odlišných jasů v rozmezí hodnot od 0x0 do 0xF
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí I2C nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **ht16k33_turn_On_display**

Name function	**boolean ht16k33_turn_Off_system_oscilator(I2CMaster_T *pointerI2C);**
Description	Vypnutí oscilátoru, který slouží pro blikání LED diod
Parameters	• žádné další parametry kromě ukazatele na strukturu pro I2CMater
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí I2C nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **ht16k33_turn_On_display**

Name function	**boolean ht16k33_turn_On_system_oscilator(I2CMaster_T *pointerI2C);**
Description	Zapnutí oscilátoru, který slouží pro blikání LED diod
Parameters	• žádné další parametry kromě ukazatele na strukturu pro I2CMater
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí I2C nebo ne
Notes	• Tato funkce musí být zavolána až po funkci **ht16k33_turn_On_display**

Name function	**boolean ht16k33_blik_freq (I2CMaster_T*pointerI2C, Blik_Frequency freq);**
Description	Nastavení zda mají LED diody blikat a z jakou frekvencí
Parameters	• **freq**: jedná se o výčtový typ s možností nastavení frekvence blikání: None,2Hz,1Hz,0.5Hz
Return values	• Vrací, zda došlo k chybě při posílání dat pomocí I2C nebo ne
Notes	• Při blikání LED diody musí být zapnutý oscilator pomocí funkce **ht16k33_turn_On_system_oscilator**

Kromě knihovny I2C, přes které je připojen tento modul je zapotřebí i knihovna Timer, která umožňuje, aby bylo vidět okem jednotlivé změny na displeji. Pomocí displeje jsou zobrazené nejprve jednotlivé čísla od 0 do 9. V hlavní smyčce pak dojde k tomu, že je displej kompletně smazaný a následně jsou rozsvícené a zhasnuté jednotlivé LED diody na displeji. Na konci smyčky jsou rozsvícené všechny LED diody.

Tím došlo k správnému otestování knihovny I2C, max7219 a Timeru.

7.1.6 Šestá ukázková aplikace

Šestá ukázková aplikace testuje knihovnu HDMI. Ověřuje, zda je správně vypsaný text na obrazovku připojenou přes HDMI. Na začátku programu je vypsaný na obrazovku text, jaká je

maximální dovolená teplota na čipu a jaká je aktuální teplota na čipu. Kromě těchto informací je i na obrazovku vypsané jaká je verze RPI, velikost paměti a typ procesoru. V hlavní smyčce dochází k tomu, že každých 500ms je zobrazený na obrazovku další znak ‚a'. Jako základní barva zobrazeného textu je nastavena Bílá barva (White color) ale tato barva je každých 500ms také změněna na Červenou (Red color), Modrou (Blue color) nebo Zelenou (Green color). Jednotlivé barvy se tedy mění a znak ‚a' je zobrazený na obrazovce vždy jednou z těchto čtyř barev.

Tím došlo k správnému otestování knihovny HDMI.

7.1.7 Sedmá ukázková aplikace

Otestování sedmá ukázková aplikace používá hlavně knihovnu MMC a pro ní knihovnu File System[FatFs od Chan[13]]. Pro tuto aplikaci je potřeba i knihovna pro HDMI, která slouží k vypisování zpráv na obrazovku. Na začátku této aplikace je testované, zda je SD karta připojena. Pokud ano dojde ke kladné zprávě na obrazovku. Pokud ne dojde k záporné zprávě na obrazovku. V hlavní smyčce dochází k tomu, že pokud existuje soubor s názvem ‚soubor.txt' obsah tohoto souboru je vypsaný na obrazovku. V případě že tento soubor neexistuje, pak je vytvořen na SD kartě a do tohoto souboru je zapsaný určitý text.

Tím došlo k správnému otestování knihovny MMC.

7.1.8 Osmá ukázková aplikace

Tato aplikace testuje knihovnu pro ethernet a knihovnu lwip[14], která byla do tohoto projektu přidaná. RPI zde slouží jako webový server. Pomocí webového prohlížeče spuštěného na PC lze tedy komunikovat s RPI. Komunikace probíhá promocí protokolu HTTP. Ukázka webové aplikace je vyobrzena na nadcházejícím obrázku [Obrázek 7.9:]. Nachází se zde tlačítko, které slouží k rozsvícení nebo zhasnutí aktivní LED diody na RPI. Dále je zde textové pole, do kterého uživatel zapíše text a pak pomocí tlačítka SendText pošle tento text, který se zobrazí na displeji ke kterému je připojeno RPI pomocí HDMI vstupu. Pokud je k RPI připojena klávesnice, lze i vyčíst jaký znaky byly stisknuté. Tyto stisknuté znaky pomocí klávesnice jsou ukládány do kruhového bufferu, který byl zmíněný na začátku této kapitoly. Každou sekundu pošle klient (PC) požadavek serveru (RPI), zda byly stisknuté nějaké klávesy na klávesnici připojenou k RPI. Tyto znaky jsou pak zobrazené v posledním textovém poli webové aplikace. Pokud je LED dioda na RPI zaplá (svítí) a dojde k obnovení (refresh) webové aplikace, barva tlačítka je přizpůsobena stavu této LED diody (svítí-barva zelená, nesvítí-barva šedá).

Pro spuštění webové aplikace je nutný prohlížeč (zvolil jsem od Googlu prohlížeč Chrome) a hlavně IP adresa serveru:

- ***#define SERVER_IP_ADDRESS "192.168.0.225"***

[13] http://elm-chan.org/fsw/ff/00index_e.html
[14] http://savannah.nongnu.org/projects/lwip/

Tím došlo k správnému otestování knihovny ethernet a knihovny lwip.

Obrázek 7.9: Ukázka webové aplikace přes webový prohlížeč Chrome

7.1.9 Devátá ukázková aplikace

Nejjednodušší způsob jak otestovat PWM knihovnu bez použití dalších součástek je pomocí dvou LED diod, jelikož PWM knihovna obsahuje 2 kanály. Pomocí těchto kanálu pak je možné ovlivňovat, jak moc jednotlivé led diody svítí. První z kanálů je vyveden na GPIO12 (nebo GPIO18) a druhý na GPIO13 (nebo GPIO19). V hlavní smyčce teda dochází k tomu, že je řízen jas jednotlivých LED diod za pomocí střídy. Pokud jedna LED dioda svítí maximálně, pak druhá LED dioda nesvítí.

Tím došlo k správnému otestování knihovny PWM.

7.2 Výsledná ukázková aplikace

Jelikož výsledná ukázková aplikace má používat knihovnu pro ethernet, tak jsem se rozhodl, že vytvořím aplikaci, která pomocí ethernetu bude posílat obrázky jako bitmapu a na RPI bude tento obrázek zobrazený přes připojený grafický displej. RPI komunikuje tedy s PC pomocí ethernetu. Na straně PC bylo zapotřebí vytvořit jednoduchou GUI aplikaci pro posílání obrázků.

Rozhodl jsem se, že tedy použiju pro aplikaci, která poběží na PC, programovací jazyk C++ a grafickou knihovnu Qt[15] napsanou také v programovacím jazyce C++ (původně jsem zvolil grafickou knihovnu wxWidgets[16], ale z touto knihovnou jsem měl v pozdější době problémy pod OS Windows, pod Linuxem tato knihovna byla též testovaná a problém nikdy nenastal). Knihovna Qt lze použít pod OS Windows nebo i Linux. Obrázky jsou posílané pomocí raw socketů. Z tohoto důvodu bylo potřebné pro Windows najít knihovnu, která by umožňovala přímé posílání a čtení síťových paketů

[15] https://www.qt.io/
[16] https://www.wxwidgets.org/

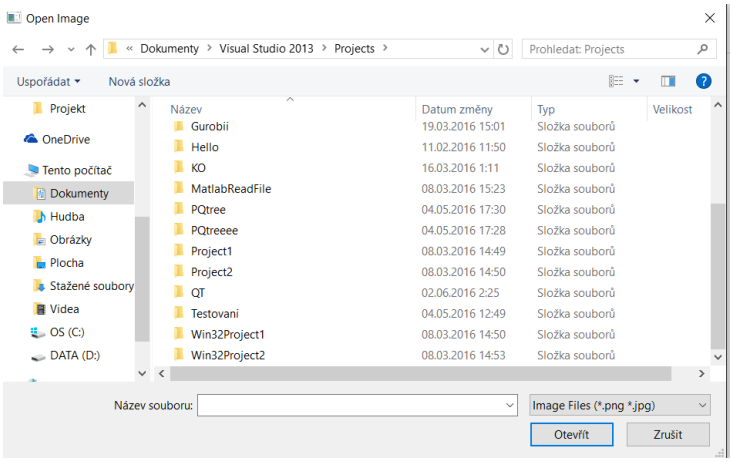
bez použití standartního síťového zapouzdření TCP/IP stacku. Na Linuxu už taková knihovna přímo existuje a není nic potřeba stahovat nebo hledat, a tudíž je i vývoj jednoduší než na Windows. Při hledání jsem narazil na knihovnu Winpcap, která umožnuje obejít standartní síťový protokol (TCP nebo UDP) a posílat pouze data, která potřebuji.

7.2.1 GUI pro PC

Grafické uživatelské rozhraní nové aplikace se skládá z pěti tlačítek na panelu nástrojů.

První tlačítko („Připojit“) umožňuje se připojit k síťovému zařízení. Pokud není vybrané síťové zařízení, dojde k zobrazení varovného okna se zprávou. Po úspěšném připojení se změní text z „Nepřipojeno“ na „Připojeno“ a vedle textu kruh změní z původní červené barvy na zelenou barvu jak je vidět na [Obrázek 7.13:].

Druhé tlačítko („Otevřít“) slouží k tomu, že otevře standartní dialogové okno souboru[Obrázek 7.10:] a uživatel si vybere obrázek, který chce poslat (pouze obrázky ve formátu png a jpg jsou zobrazené). Obrázek je zobrazen i v aplikaci a uživatel ho tak vidí v orámovaném čtverci [Obrázek 7.13:] (zde v této ukázce je posílaný obrázek pouze zelený).



Obrázek 7.10: Standartní dialogové okno souboru

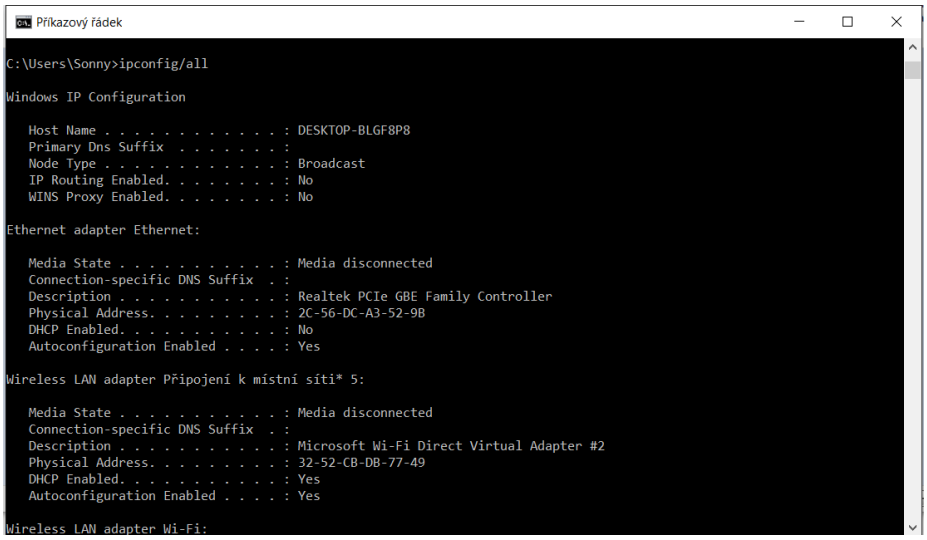
Třetí tlačítko („Poslat“) posílá vybraný obrázek pomocí zvoleného síťového rozhraní. Pokud není zvolené síťové rozhraní nebo na začátku není vybraný žádný obrázek, uživateli se zobrazí varovné okno se zprávou. Pokud ale je vybrané síťové rozhraní i obrázek, pak dojde k poslání obrázku v jeho původní velikosti a ne jak bylo přizpůsobeno uživateli vidět v orámovaném čtverci. Mezitím, kdy se posílá obrázek, je možné otevřít i jiný obrázek (pomocí tlačítka „Otevřít“) ale tento obrázek nebude poslaný až do doby kdy je poslaný předchozí. Uživatel je tímto stavem zase informovaný pomocí varovného okna se zprávou.

Čtvrté tlačítko („Nastavení“) pomocí tohoto tlačítka dojde k tomu, že je zobrazené nové dialogové okno, kde je zobrazený seznam síťových zařízení nacházejících se na dotyčném PC. Uživatel si vybere

pomocí, kterého síťového zařízení chce data posílat. To je docela problém pro běžného uživatele a nelze určit, jaké síťové zařízení je ethernet. Ukázka seznamu sítových zařízení na mém PC:

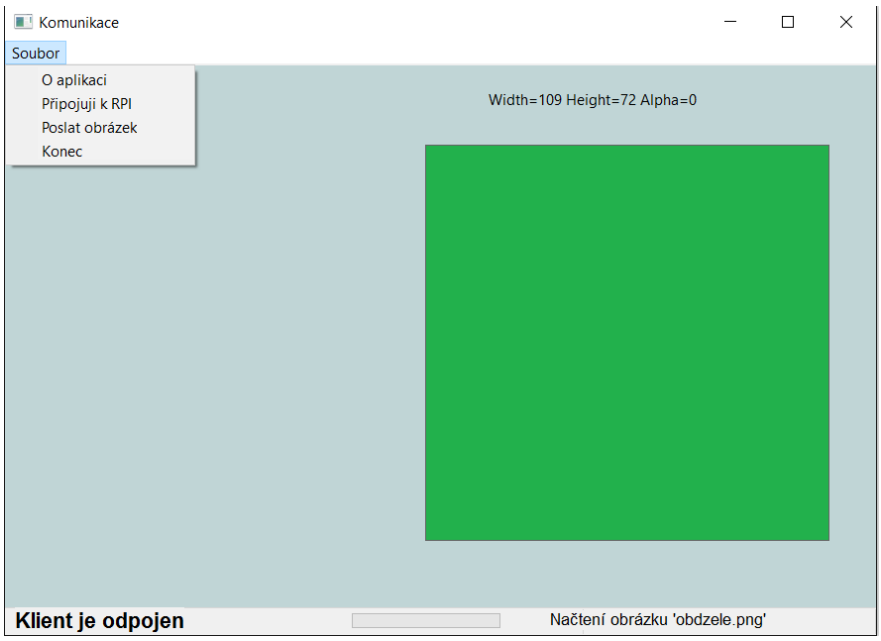
- ***\\Device\\NPF_{41B0AF95-F139-4FEA-BCFE-9BBF2F88AFA4}" Microsoft***
- ***\\Device\\NPF_{0497A953-4A82-4F77-81CA-3CDB1B39268E}" Microsoft***
- ***\\Device\\NPF_{768777CA-E1A9-4CB9-B56A-D41D437618C9}" Realtek PCIe GBE Family Controller***

Pomocí příkazového řádku a použitím příkazu *ipconfig/all* lze snadno ověřit, zda bylo vybrané správné síťové zařízení [Obrázek 7.11:].

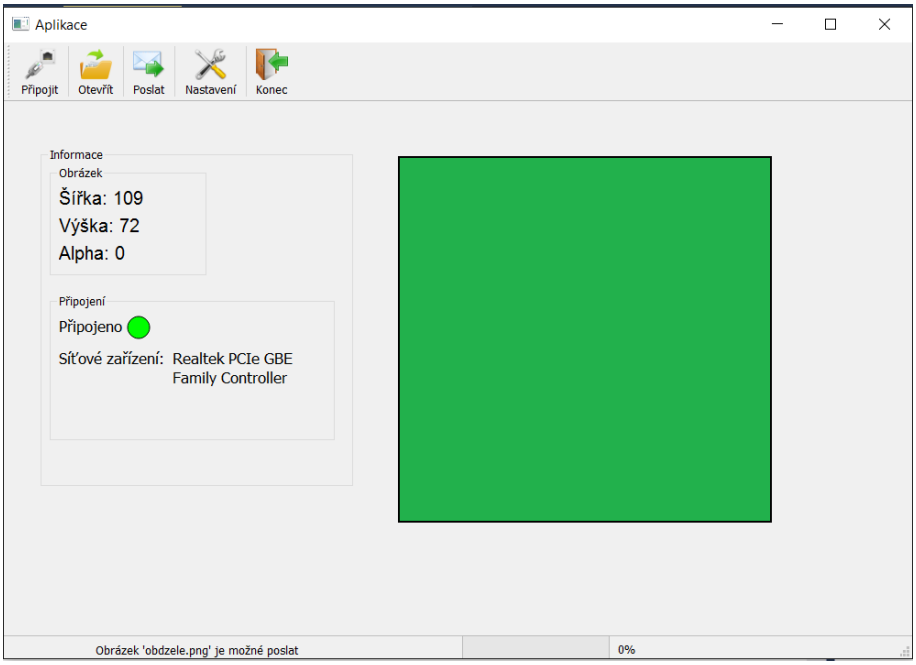


Obrázek 7.11: Ověření síťového zařízení pomocí příkazového řádku ve Windows

Páté tlačítko („Konec“) je velice jednoduché a stará se ukončení aplikace. Aplikace není ale ukončena pokud nejsou poslány všechny data rozpracovaného obrázku.



Obrázek 7.12: Ukázka původní aplikace GUI (knihovna wxWidgets) pro PC



Obrázek 7.13: Ukázka nové aplikace GUI (knihovna Qt) pro PC

7.2.2 Vlastní síťový protokol

Kromě posílání dat (jednotlivé složky RGBA obrázku) bylo zapotřebí také vytvořit svůj vlastní paket s hlavičkou, aby se zabránilo tomu, že pokud OS Windows pošle nějaký svůj paket pomocí protokolu, aby tento paket nebyl přijatý na straně RPI. Proto výsledný paket obsahuje i hlavičku a poté až data. Na straně RPI se zkontroluje, jestli hlavička (`DESTINATION_MAC`, `SOURCE_MAC`, `EtherType`) odpovídá a pokud ano paket je uložený do bufferu.

Výsledná struktura pro ethernet paket pak vypadá takto:

- ***uint8_t DESTINATION_MAC[6]***
- ***uint8_t SOURCE_MAC[6]***
- ***uint16_t EtherType***
- ***uint32_t IDCode***
- ***uint32_t IDPacket***
- ***uint32_t DataLength***
- ***uint8_t Data[0]***

Kde DESTINATION_MAC je cílová adresa (pokud posílá PC paket, jedná se o adresu RPI), které znají obě zařízení před započnutím komunikace a tato adresa pak slouží k tomu, aby se ověřilo, že se jedná o paket, který chci uložit.

Tato adresa pak vypadá takto:

- ***#define RASPBERRY_MAC {0xb8, 0x27, 0xeb, 0xb6, 0xb7, 0x9a}***

Opakem cílové adresy je zdrojová adresa SOURCE_MAC (pokud posílá PC paket, jedná se o jeho adresu). Obě zařízení znají tuto adresu před započnutím komunikace a tato adresa taky slouží k tomu, aby se ověřilo, že se jedná o paket, který chci uložit.

Tato adresa pak vypadá takto:

- ***#define COMPUTER_MAC {0x2c,0x56,0xdc,0xa3,0x52,0x9b}***

Poslední v hlavičce ethernetu je šestnáctibitové číslo, které určuje protokol EtherType. Toto číslo není náhodné a musel jsem zkontrolovat, zda nějaký protokol nepoužívá toto číslo[17]. Číslo, které jsem použil, nepoužívá žádný protokol.

- ***#define MY_PROTOCOL 1152***

IDCode určuje, co se má s paketem udělat, jestli paket přišel nebo jaké data obsahuje paket. Seznam těchto kódů lze nalézt zde:

- ***#define CODE_ACK 0x0001 //potvrzení paketu lze poslat další paket***
- ***#define CODE_NACK 0x0002 //nelze posílat další paket***
- ***#define CODE_NONE 0x0003 //uvedený kód neznám***
- ***#define CODE_CONNECT 0x0011 //paket testuje připojení RPI***
- ***#define CODE_PICTURE_FIRT_DATA 0x0012 //paket obsahuje šířku, výšku a jestli pakety budou obsahovat i alfa kanál***
- ***#define CODE_PICTURE_OTHER_DATA 0x0013 //paket obsahuje RGB nebo RGBA složky obrázků***

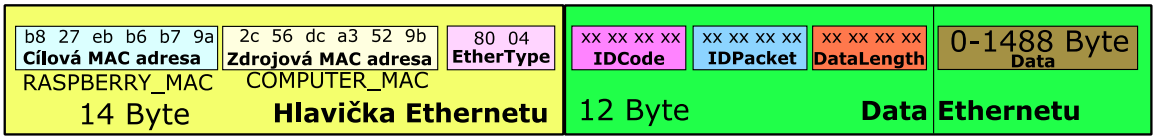
První tři kódy posílá RPI a slouží pouze k tomu, že buď potvrdí paket pozitivně (`CODE_ACK`) a PC může posílat další paket nebo potvrdí negativně (`CODE_NACK`) a další paket není poslaný a musí se posílat znovu od začátku. Tato situace může nastat, pokud je RPI vypnuté a hned zapnuté. Pokud pošle RPI paket s IDCode (`CODE_NONE`), znamená to, že paket, který přišel, obsahoval IDCode, který RPI nezná. Paket s IDCode (`CODE_CONNECT`) posílá PC a slouží k ověření, že RPI je připojené a schopné komunikovat s PC. Poslední dva kódy posílá PC a určuje tím, jaké data jsou obsažena v paketu.

Člen struktury EthernetPacket IDPacket slouží hlavě k tomu, pokud jedno ze zařízení se vypne (u PC se shodí aplikace) pak druhé zařízení přestane vysílat nebo přijímat rozpracovaný obrázek a dojde k tomu, že obrázek posílaný jako bitmapa se zahodí. Při posílání paketů je na straně PC toto číslo inkrementované a testuje se, zda se příchozí IDPacket shoduje s IDPacketem RPI.

DataLength je délka dat a Data[0] je ukazatel na tyto data. V těchto datech jsou většinou jednotlivé složky RGB nebo RGBA obrázku nebo v prvním paketu při komunikaci mezi PC a RPI, pokud posílá obrázek, obsahuje tento paket šířku, výšku, a jestli obsahuje obrázek alfa kanál.

Každý paket, který je vyslaný z PC, je na straně RPI potvrzen a to znamená, že žádná data nejsou ztracena a buď je obrázek přijatý celý, nebo ne. Z tohoto důvodu je tento protokol spolehlivý. Pokud paket, který pošle PC, není potvrzen paketem od RPI dojde k tomu, že PC pošle opět stejný paket. Tento interval je nastaven na 500ms do této doby musí RPI paketem odpovědět.

[17] http://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml



Obrázek 7.14: Ukázka paketu, který pošle PC

7.2.3 Výsledky

Původní aplikace používala TCP protokol a grafickou knihovnu wxWidgets [Obrázek 7.12:]. Pro tuto aplikaci jsem použil vývojové prostředí Code::Blocks.

Nová aplikace používá vlastní protokol pomocí raw socketů a grafickou knihovnu Qt [Obrázek 7.13:]. Pro aplikaci jsem použil vývojové prostředí Microsoft Visual Studio 2013 a Qt Designer pro grafiku. Výhoda oproti původní aplikaci spočívá v hezčím GUI a také v menším posílání dat pomocí ethernetu díky raw socketů a tak obrázky můžou být poslané rychleji. Nevýhoda pak je těžší implementace a potřebná knihovna Winpcap pro Windows. Oproti původní aplikaci se musí taky vybrat síťové rozhraní, ke kterému se uživatel chce připojit. Další výhodou na straně RPI je, že už není potřeba knihovny lwip.

7.2.4 Aplikace pro RPI

Tato aplikace umožňuje pomocí počítače RPI zobrazovat obrázky na displeji připojené přes HDMI. Zobrazování obrázků je možné dvěma způsoby. Buď pomocí posílání obrázků přes ethernet jako bitmapa nebo načtení obrázku z SD karty ve formátu png. Pro práci s SD kartou je nutné mít připojenou klávesnici. Klávesnice i displej musejí být připojené už při zapnutí RPI. I zde je využitá datová struktura kruhový buffer, který umožňuje uchovat v paměti celkem 5 obrázků, které jsou poslané pomocí ethernetu pro možné pozdější uložení na SD kartu ve formátu png. Pro ukládání a načítání obrázků ve formátu png byla zapotřebí knihovna stb[18].

Před hlavní smyčkou probíhá inicializace jednotlivých knihoven, které jsou potřeba.

V hlavní smyčce se kontroluje:

Zda je nějaký paket k dispozici. Pokud je paket k dispozici, tak se přečte a zkontroluje se, jestli tento paket je pro RPI. Kontrola probíhá pomocí ověření hlavičky ethernet paketu (`DESTINATION_MAC, SOURCE_MAC, EtherType`). Pokud hlavička souhlasí, zkontroluje se, jestli paket, který přišel, obsahuje jeden z uvedených IDCode. Pakliže IDCode nesouhlasí RPI pošle paket zpět z uvedeným IDCode (*`CODE_NONE`*). Jestliže paket obsahuje IDCode (*`CODE_CONNECT`*) pak RPI pošle paket s IDCode (*`CODE_ACK`*). Poslední možností je, že se jedná už o data (obrázek) a tak se zkontroluje, jestli číslo v IDPacket odpovídá číslu, které má uložené RPI a takové číslo IDPacketu očekává. Poté se uloží zbytek paketu (bez hlavičky paketu tedy pouze data) do bufferu a pošle se zpět paket s IDCode (*`CODE_ACK`*).

Ukázka seznamu datových položek ve struktuře, která uchovává obrázek:

[18] https://github.com/nothings/stb

- ***unsigned char *buffer;*** ***//jednotlivé pixely obrázku vyjádřené pomocí RGB nebo RGBA***
- ***unsigned int height;*** ***//výška obrázku***
- ***unsigned int width;*** ***//šířka obrázku***
- ***unsigned int offset;*** ***// kam se mají příchozí pixely uložit do bufferu***
- ***char isAlpha;*** ***//obsahuje obrázek alfa složku obrázku***

Poté co je testován, zda je paket k dispozici, probíhá testování, zda je obrázek už poslaný celý. Když je obrázek poslaný celý, dojde k jeho uložení do Kruhového bufferu. Smaže se celá obrazovka a je zobrazen obrázek na obrazovce. Je-li obrázek větší než velikost displeje, pak dojde k tomu, že obrázek se přizpůsobený velikosti obrazovku. To znamená, že nějaké pixely z původní obrázku nejsou zobrazeny na displeji. Obrázek je zobrazený na středu obrazovky.

Poslední kontrola spočívá v tom, že pokud je připojena klávesnice, tak se kontroluje, jaké klávesy byly stisknuté. Stisknuté klávesy se v přerušení ukládají do kruhového bufferu a v hlavní smyčce se pouze jenom čtou. Podle toho jaké byly stisknuté klávesy se RPI nachází v jednom ze tří stavů, které slouží jako menu:

- *BASIC_IMAGE*
- *READ_IMAGE*
- *SAVE_IMAGE*

V těchto uvedených stavech dochází k tomu, že je možné použít opět stisknuté klávesy k rozšířenějším funkcí.

Stav *BASIC_IMAGE* – základní stav, který pomocí kláves *F2* smaže celou obrazovku a načte všechny soubory v adresáři png a jména těchto souborů zobrazí na obrazovku. Pomocí klávesy *F3* smaže celou obrazovku. Klávesou *F4* se přepne do stavu *READ_IMAGE* a klávesou *F5* se přepne do stavu *SAVE_IMAGE*. Dále je pomocí klávesnice stisknutím *šipky doprava* nebo *šipky doleva* zobrazit na displeji obrázek, který byl poslaný přes ethernet.

Stav *READ_IMAGE* – slouží k tomu, aby bylo možné načíst obrázek ve formátu png z SD karty.

- ***Zadejte název obrázku k načtení z SD karty:***

Pomocí stisknutých znaků, které jsou zobrazené na displeji, je možné zadat tento název obrázku. Dojde-li k překlepu tak pomocí klávesy *BackSpace* je možné smazat předchozí znak. Při stisku klávesy *Enter* se pak pokusí načíst obrázek z SD karty. Pokud takový obrázek s názvem existuje na SD kartě, je tento obrázek zobrazen na displeji. Pokud neexistuje nebo se nepovedlo načíst obrázek z SD karty uživatel je informovaný pomocí displeje textem:

- ***Obrázek se nepovedlo načíst buď je poškozený nebo neexistuje***

Klávesou *Esc* je možné kdykoliv se vrátit do stavu *BASIC_IMAGE* stejně jako při stisku klávesy *Enter*.

Stav *SAVE_IMAGE* – slouží k tomu, aby bylo možné uložit obrázek ve formátu png na SD kartu.

- ***Zadejte název obrázku k uložení na SD kartu:***

Pokud ale žádný obrázek nebyl poslaný pomocí ethernetu, nejde žádný obrázek uložit.

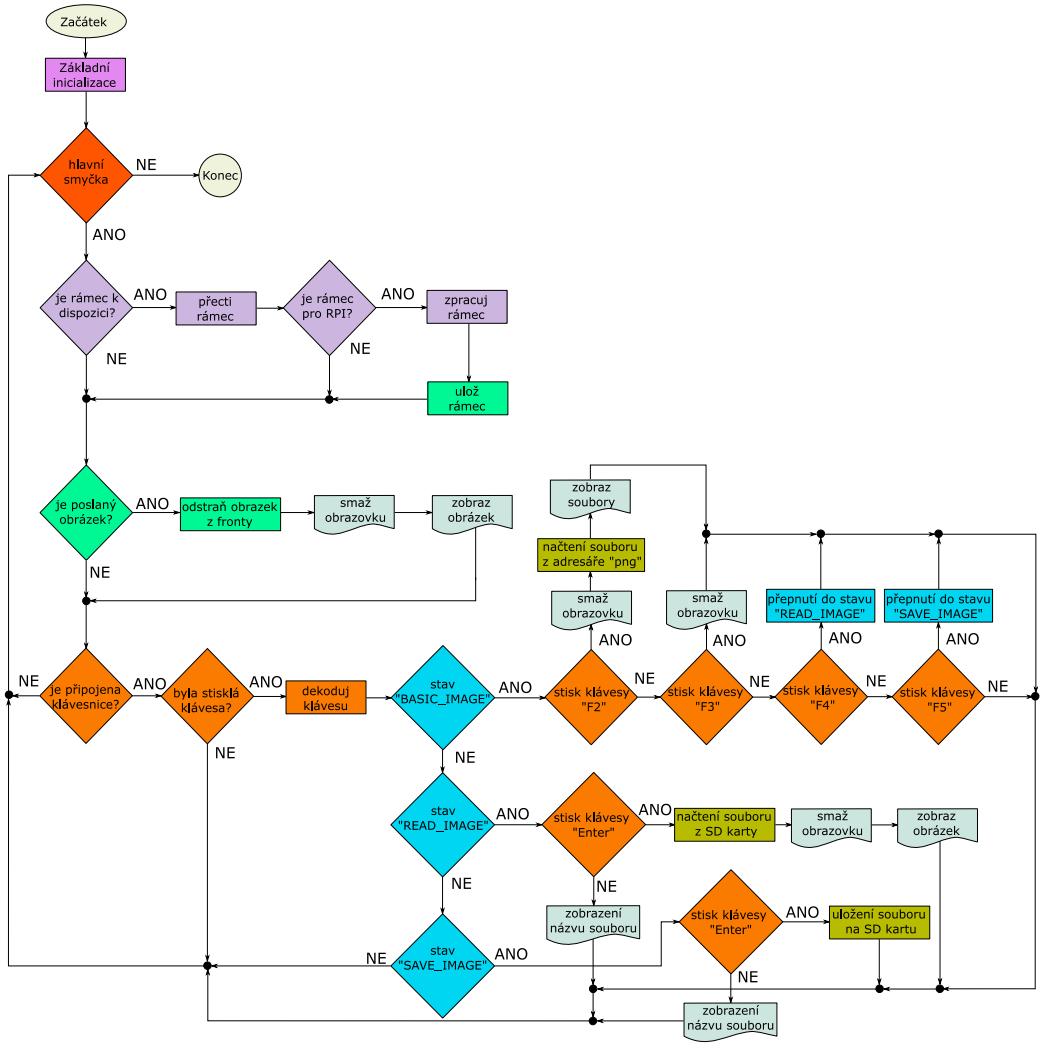
- ***Není vybraný žádný obrázek!!***

Pokud ale je poslaný obrázek pomocí ethernetu, tak je možné ho uložit. Stejně jako ve stavu *READ_IMAGE* jsou znaky názvu souboru zobrazené na displeji. Pomocí klávesy *Enter* se obrázek uloží na SD kartu. Jestliže se uloží obrázek, je uživatel o tom informovaný.

- ***Obrázek byl uložen.***

Klávesou *Esc* je možné kdykoliv se vrátit do stavu *BASIC_IMAGE* stejně jako při stisku klávesy *Enter*.

Ukázka vývojového diagramu finální aplikace je na zobrazen na následujícím obrázku [Obrázek 7.15:].



Obrázek 7.15: Vývojový diagram finální aplikace pro RPI

8 Závěr

Úkolem této Diplomové práce bylo vytvořit knihovny funkcí v programovacím jazyce C pro ovládání periférií na počítači Raspberry Pi bez použití operačního systému. Pro předem zadané periférie GPIO, UART, SPI, I2C, PWM, řadiče přerušení, řadiče SD/MMC karet a ethernetové rozhraní. Další úkolem bylo zapotřebí implementovat sadu funkcí pro zobrazení dat na obrazovce připojené přes HDMI a následné vytvoření finální demonstrující aplikace, která využívá knihovny pro grafický displej, ethernetové rozhraní a SD kartu.

V této práci byly tedy vytvořené knihovny funkcí v programovacím jazyce C pro zadané periférie. Pro jednotlivé knihovny fukcí byly napsané ukázkové aplikace, které byly otestované na reálných zařízeních, které jsem si zvolil, a tyto zařízení bylo potřeba dokoupit.

Při otestování finální demonstrující aplikace, byl použitý počítač (s OS Windows), který byl připojen pomocí ethernetu k Raspberry Pi. Komunikace zde probíhá pomocí modelu Master/Slave. Kde roli Masteru přebírá PC a roli Slavu byla daná Raspberry Pi. Na PC bylo vytvořený program s grafickým uživatelským rozhraní v programovacím jazyce C++ s grafickou knihovnou Qt. Program umožnuje posílání obrázků ve (formátu png nebo jpg) pomocí ethernetu z PC do RPI jako bitmapu. Jako vývojové prostředí jsem zvolil Microsoft Visual Studio 2013 pro aplikaci napsanou pro PC. Obrázky byly posílané pomocí raw socketů. Z tohoto důvodu bylo potřebné pro Windows najít knihovnu, která by umožňovala přímé posílání a čtení síťových rámců bez použití standartního síťového zapouzdření TCP/IP stacku. Taková knihovna není součástí OS Windows, a proto jsem zvolil knihovnu Winpcap, která toto umožňuje.

Při poslání všech pixelů, z kterých se obrázek skládá, je na straně RPI zobrazen na grafickém displeji připojeném přes HDMI. Pomocí klávesnice lze pak uložit tento obrázek na SD kartu ve formátu png. Dále je pomocí stisku klávesy možné načíst obrázek z SD karty ve formátu png a tento obrázek zobrazit na displeji. Obrázek je zobrazen na středu grafického displeje, a pokud nelze celý zobrazit na grafickým displeji dojde k jeho zmenšení.

Rozšíření této práce vidím v implementaci knihovny pro použití externího USB Wifi adaptéru, který by sloužil místo ethernetu a tak by RPI nemuselo být připojené fyzicky k PC pomocí UTP kabelu. Dále je možné implementovat knihovnu funkcí pro periférii I2S, která slouží k přehrávání zvuku a v této práci nebyla tato knihovna potřeba.

9 Literatura

[1] Raspberry PI. Dostupné z: www.raspberrypi.org

[2] „Wikipedie.org“. Dostupné z: https://en.wikipedia.org/wiki/RCA_connector

[3] GitHub. Dostupné z: https://github.com/raspberrypi/firmware/tree/master/boot

[4] RPiconfig. Dostupné z: http://elinux.org/RPiconfig

[5] Křížový překladač. Dostupné z: https://launchpad.net/gcc-arm-embedded

[6] BCM2835 datasheet. Dostupné z: https://cdn-shop.adafruit.com/product-files/2885/BCM2835Datasheet.pdf

[7] J. Fischer, Výukové materiály předmětu A4B38NVS: Čítače, ČVUT FEL

[8] ARM SP804 datasheet. Dostupné z: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0271d/DDI0271.pdf

[9] Sériová linka RS-232. Dostupné z: http://vyvoj.hw.cz/rozhrani/hw-server-predstavuje-seriova-linka-rs-232.html

[10] UART PL011 datasheet. Dostupné z: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0183g/DDI0183G_uart_pl011_r1p5_trm.pdf

[11] Mailboxes. Dostupné z: https://github.com/raspberrypi/firmware/wiki/Mailboxes

[12] ARM Architecture Reference Manual. Dostupné z: https://www.scss.tcd.ie/~waldroj/3d1/arm_arm.pdf

[13] ARM Deveoper Suite,http://infocenter.arm.com/help/topic/com.arm.doc.dui0056d/DUI0056.pdf

[14] J. Novák, Výukové materiály předmětu A4M38KRP: Sběrnice USB, ČVUT FEL

[15] Universal Serial Bus Specification revision 1.1 datasheet

[16] Universal Serial Bus Specification revision 2.0 datasheet

[17] SMSC LAN9514/LAN9514i datasheet. Dostupné z: http://ww1.microchip.com/downloads/en/softwarelibrary/man-lan95xx-dat/lan9514_lan9514i%20databook%20rev.%201.2%20(03-01-12).pdf

[18] MAX7219C datasheet Dostupné z: https://www.sparkfun.com/datasheets/Components/General/COM-09622-MAX7219-MAX7221.pdf

[19] Ht16k33 datasheet. Dostupné z: https://cdn-shop.adafruit.com/datasheets/ht16K33v110.pdf

Příloha A

Obsah přiloženého CD

Součástí této diplomové práce je i přiložené CD, na kterém se nacházejí jednotlivé adresáře, ve kterých jsou uloženy zdrojové kódy a katalogové listy součástek:

- Adresář Bootloader:
  - Bootloader_PC.zip – Program pro nahrání kódu do RPI přes ethernet pro OS Windows.
  - Bootloader_RPI.zip – Program pro nahrání kódu do RPI přes ethernet pro RPI.

- Adresář *Datasheet*:
  - *ARM.pdf* – Informace o ARM procesorech.
  - *ARM_prirucka.pdf* – Uživatelská příručka pro ARM procesory.
  - *ARM_SP804.pdf* – Katalogový list ARM Timeru SP804.
  - *ARM1176JZF.pdf* – Katalogový list ARM procesu ARM1176JZF na čipu BCM2835.
  - *ARMCortex-A7.pdf* – Katalogový list ARM procesu Cortex-A7 na čipu BCM2836.
  - *ARMv7.pdf* – Architektura ARM7 pro procesor ARM Cortex-A7
  - *BCM2835.pdf* – Katalogový list čipu BCM2835.
  - *HT16K33.pdf* – Katalogový list řadiče HT16K33.
  - *MAX7219.pdf* – Katalogový list řadiče MAX7219.
  - *SMSC_LAN9514.pdf* – Katalogový list čipu LAN9514 pro ethernet.
  - *UART_PL011.pdf* – Katalogový list UARTu PL011.
  - *USB_1.1.pdf* – USB norma 1.1.
  - *USB_2.0.pdf* – USB norma 2.0.

- Adresář *Kod_PC*:
  - *GUI Qt.zip* – Program vytvořený v jazyce C++ v prostředí Microsoft Visual Studio 2013 určený pro OS Windows, součástí je i knihovna Winpcap pro Windows
  - *GUI Widgets.zip* – Program vytvořený v jazyce C++ v prostředí Code::Blocks určený pro OS Windows i Linux

- Adresář *Kod_RPI*:
    - *boot.zip* – Soubory potřebné na SD kartě.
    - *demo.zip* – Ukázky demo aplikací pro RPI.
    - *final.zip* – Finální aplikace pro RPI.
    - *library.zip* – Knihovna pro RPI.

- Adresář *Programy*:
    - *aspt.exe* – instalační soubor programu Advanced Serial Port Terminal
    - *gcc-arm-none-eabi-4_9-2015q3-20150921-win32.exe* – křížový překladač

- Adresář *Obrázky*:
    - *obrázky.zip* – Obrázky v této práci.

- Soubor *Knihovna_funkcí_pro_RPI.pdf* – tento dokument