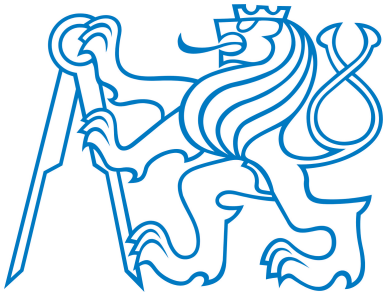
České Vysoké Učení Technické v Praze
Fakulta Elektrotechnická
Katedra měření



DIPLOMOVÁ PRÁCE

Časově synchronní systém sběru dat

Autor: Vojtěch Eliáš
Vedoucí práce: doc. Ing. Jan Fischer, CSc.
Praha, 2012

zadani prace (CZ)

Čestné prohlášení autora

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

Tato práce vznikla v laboratoři videometrie, katedry měření ČVUT - FEL v Praze pod vedením doc. Ing. Jana Fischera, CSc. Navazuje též na výzkum v rámci MSM6840770015 - "Výzkum metod a systémů pro měření fyzikálních veličin a zpracování naměřených dat", jehož některé poznatky a výstupy také využívá.

V Praze dne _____ _____

podpis

Anotace

Diplomová práce se zabývá návrhem a implementací univerzálního systému synchronního sběru dat. Systém je tvořen množinou nezávislých procesorových modulů osazených 32-bitovými mikrořadiči z rodiny STM32 určených pro distribuovaný sběr dat a ovládání akčních členů. Komunikačním médiem je síť Ethernet umožňující nadřazenému počítačovému systému vzdálenou konfiguraci a zpracování dat.

Hlavními rysy navrženého systému jsou podpora sady protokolů TCP/IP pro datovou komunikaci, využití standardu IEEE-1588 Precise Time Protocol (PTP) pro precizní synchronizaci a využití real-time operačního systému pro embedded systémy FreeRTOS pro realizaci aplikační logiky jednotlivých distribuovaných zařízení.

Součástí práce je též návrh kompaktního procesorového modulu určeného pro experimentální vývoj aplikací a pro nasazení při výuce předmětů orientovaných na problematiku vestavěných systémů, jejichž výuka probíhá na Katedře měření Fakulty elektrotechnické, ČVUT v Praze.

Klíčová slova

Embedded systém, jednočipový mikrořadič, ARM Cortex-M3, STM32, FreeRTOS, PTP, IEEE-1588, Ethernet, distribuovaný systém

Annotation

This thesis describes design and implementation of general-purpose time synchronous data acquisiton system. The system consists of a set of independent embedded devices (modules) equipped with 32-bit microcontrollers from the STM32 device family used for distributed data acquisition and control. Ethernet network is applied as the communication medium thus enabling remote supervision of the system using personal computer.

The main features of the designed solution are utilization of TCP/IP protocol suite ensuring data communication, application of standard IEEE-1588 for precise time synchronisaton over the networked devices and utilization of embedded real-time operating system FreeRTOS.

Part of this thesis is dedicated to design and development of a compact proccessor module that is intended to be used in experimental application designs and for educational purposes in courses focused on embedded systems within the Department of measurement, Faculty of Electrical Engineering, CTU in Prague.

Keywords

Embedded system, microcontroller, ARM Cortex-M3, STM32, FreeRTOS, PTP, IEEE-1588, Ethernet, distributed system

Poděkování

Na tomto místš děkuji doc. Ing. Janu Fischerovi CSc. za důkladné vedení mé práce, ochotu obětovat cenný čas společným konzultacím, a množství praktických připomínek a rad při návrhu a realizaci řešení. Dále děkuji odborným pracovníkům Katedry měření, především Ing. Janu Breuerovi za poskytnutí velmi cenné odborné pomoci a Radkovi Řípovi za poskytnutí technického zázemí v prostředí laboratoře. Konečně děkuji svým nejbližším za trpělivost a podporu, především Zuzaně, mým rodičům a přátelům.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Elektronické a softwarové technologie prochází již několik desetiletí extémně dynamickým vývojem. V oblastech průmyslové automatizace a metrologie se v současné době široce uplatňuje decentralizace řízení technologických procesů. V decentralizovaných systémech se využívá sítí distribuovaných zařízení (senzorů a akčních členů), která disponují určitou mírou autonomie a obsahují vlastní řídící logiku umožňující vykonání algoritmů sběru a zpracování dat reprezentujících hodnoty fyzikálních veličin na nejnižší úrovni technologického procesu.

Elementy distribuovaného systému (uzly sítě) jsou zároveň vybaveny některým standartním komunikačním rozhraním umožňujícím jak jejich vzájemnou interakci na jedné straně, tak i komunikaci s nadřazeným počítačovým systémem na straně druhé (např. poskytování výstupů své činnosti a vzdálená konfigurace).

Významnou výhodou takto koncipovaných systémů je jejich modularita. Distribuovaný systém lze poměrně snadno upravovat konkrétním potřebám například při změně výrobního postupu, či zahájení výroby nového produktu. Požadované úpravy lze dosáhnout buď vzdálenou konfigurací parametrů jednotlivých uzlů sítě pomocí nadřazeného počítače, softwarovou úpravou konkrétního zařízení, nebo změnou topologie přidáním dalších prvků. Neméně důležitou vlastností je distribuce spolehlivosti - při poruše některého subsystému nemusí nutně dojít k velkým škodám na řízené technologii, její příčiny mohou být zároveň snáze lokalizovány. V případě centralizovaného systému, ve kterém jsou řídící algoritmy systému prováděny centrálním počítačem, je riziko fatálních následků poruchy řídícího počítače mnohem vyšší.

Existuje velké množství praktických aplikací, které vyžadují ke své činnosti synchronizaci. Sami přirozeně využíváme měření času pro časové sladění svých aktivit s procesy a lidmi ve svém okolí. Na základě synchronizace s ostatními je možné definovat a provádět

společné plány, následky jejího výpadku jsou téměř vždy více, či méně nepříjemné. Na stejném principu funguje velké množství technických aplikací, které v reálném čase provádí sekvence nějakých definovaných kroků. Je-li algoritmus řešení prováděn distribuovaným systémem - tedy množinou nezávislých počítačových strojů - je nezbytné, aby tyto stroje byly v reálném čase synchronizovány. Pro příklad uveďme analýzu seismické aktivity geograficky rozlehlých oblastí při lokalizaci epicenter zemětřesení nebo synchronizaci funkce robotických montážních strojů s množstvím mechanických akčních členů. V těchto aplikacích je nutné synchronizovat jednotlivé funkční bloky a jejich akce v reálném čase napříč celým, často prostorově rozsáhlým systémem. Tato diplomová práce se soustředí na problematikou synchronizace v distribuovaných systémech.

1.1 Stanovení cílů práce

Předmětem práce je návrh a realizace univerzálního synchronního systému distribuovaných zařízení, umožňujícího sběr dat i ovládání akčních členů, a dále komunikaci s nadřazeným počítačovým systémem.

Jádro výsledné realizace tvoří jednotlivé elementy distribuovaného systému reprezentované univerzálními mikropočítačovými moduly s potřebným programovým vybavením realizujícím vlastní aplikační logiku, komunikaci v počítačové síti a synchronizaci podle standardu IEEE-1588. V rámci práce bude třeba jednotlivé moduly fyzicky zkonstruovat a implementovat jejich software.

Klíčovou součástí návrhu je využití real-time operačního systému (konkrétně FreeRTOS) při implementaci programového kódu modulů. Využití operačních systémů při návrhu aplikací pro vestavěná zařízení umožňuje flexibilní tvorbu software, jehož části lze snadno udržovat, přenášet do jiných projektů a rozšiřovat funkcionalitu existujícího kódu.

Pro zajištění uživatelského přístupu k síti modulů bude vytvořena aplikace určená pro PC. Pomocí ní bude možno připojit se ke konkrétnímu zařízení sítě a stahovat z něj data reprezentující výstupy provedených měření. Jednotlivé uzly sítě (moduly) tak realizují funkci senzoru a datového serveru. Implementace protokolu IEEE-1588 navíc umožní distribuovaný sběr časově korelovaných dat. Toho lze prakticky využít při dalším počítačovém zpracování naměřených dat.

Kapitola 2

Teoretický Rozbor

Tato kapitola obsahuje teoretický rozbor zadaného problému a popis jednotlivých dílčích komponent jeho řešení.

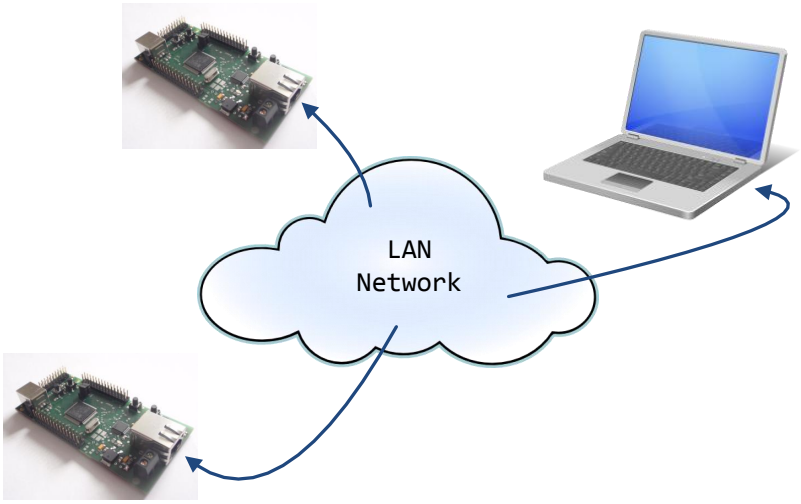
V úvodní části kapitoly jsou stanoveny základní rysy výsledného systému a důležité kroky na cestě k jeho realizaci. Následuje popis technologií, které budou při práci využity, a se kterými bude třeba se blíže seznámit.

2.1 Obecný popis navrhovaného systému

Jak bylo stručně nastíněno v kapitole 1, předmětem návrhu je distribuovaný systém sběru dat složený ze synchronizovaných modulů. Strukturu systému se dvěma moduly a řídícím počítačem je možno ilustrovat pomocí obrázku 2.1. V konkrétní praktické aplikaci může mít distribuovaný systém výrazně rozsáhlejší topologii s větším počtem modulů a připojených počítačů.

Konfiguraci distribuovaného systému obecně tvoří následující hardwarové komponenty:

- **Množina IO modulů**
  - embedded zařízení vybavené jednočipovým mikrořadičem (dále také μC)
  - zajišťuje interakci s reálným prostředím
  - umožňuje sběr/zpracování dat a síťovou komunikaci
- **Infrastruktura sítě**
  - soustava zařízení a kabeláže zajišťující funkci sítě Ethernet



Obrázek 2.1: Základní struktura navrhovaného systému pro sběr dat

- **Síť nadřazených počítačů**
  - Desktop/laptop zařízení s uživatelským rozhraním pro konfiguraci a ovládání IO modulů

Pro dosažení cílů stanovených v 1.1 je nejprve potřeba určit, na jakém hardwaru bude distribuovaný systém sběru dat postaven. Řešení zadaného úkolu se odvíjí především od výběru použitého μC.

Druhým krokem je studium real-time operačního systému FreeRTOS, a jeho implementace na konkrétním μC.

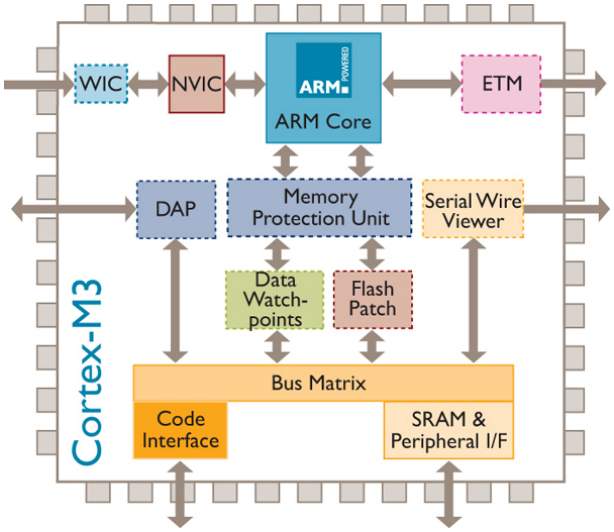
Jakmile se podaří připravit funkční operační prostředí FreeRTOS běžící na testovacím hardware, bude možné systém dále rozšířit o podporu komunikace v síti Ethernet, a nad ní implementovat prostředky synchronizace pomocí protokolu IEEE-1588.

Konečně bude k dispozici distribuovaný systém sestávající z navržených komponent propojených sítí LAN pomocí Ethernetu a bude možné připravit vlastní aplikační program jednotlivých zařízení a uživatelské rozhraní pro PC v roli nadřazeného počítačového systému.

2.2 Klíčové komponenty návrhu

2.2.1 Mikrořadiče STM32

V procesorových modulech využitých v této diplomové práci budou osazeny 32-bitové mikrořadiče rodiny STM32F s jádrem ARM Cortex-M3 dodávané společností STMicroelectronics. Procesory s tímto jádrem jsou určeny pro využití ve vestavěných aplikacích jako je digitální zpracování signálu, řešení real-time úloh a také low-power aplikacích (přenosná zařízení napájená bateriemi s nároky na nízký příkon). Blokové schéma jádra ARM Cortex-M3 je znázorněno na obrázku 2.2. Podrobný popis jádra lze nalézt v [1].



Obrázek 2.2: Struktura procesorového jádra ARM Cortex-M3 (převzato z www.arm.com)

V průběhu řešení bude využit nejprve μC STM32F105RBT6 (viz. [2]), pro který bude vyvinuta vlastní experimentální platforma. Tato poslouží k získání zkušeností s prostředím FreeRTOS. Při následné realizaci systému s podporou síťové komunikace bude použit μC STM32F207VGT6 (viz. [3]) osazený v modulu vyvinutém v Laboratoři videometrie na Katedře měření ČVUT v Praze (podrobné informace o tomto modulu lze nalézt v [15]). Následuje velmi stručný přehled základních vlastností a parametrů dvojice použitých mikrořadičů:

- **STM32F105RBT6**
  - 72 MHz
  - 128 KB Flash

  - 32 KB SRAM
  - SPI, I2C, ADC, DAC
  - DFU, SWD, JTAG

- **STM32F207VGT6**
  - 120 MHz
  - 1024 KB Flash
  - 128 KB SRAM
  - SPI, I2C, ADC, DAC, Ethernet
  - DFU, SWD, JTAG

2.2.2 Embedded RTOS

Využití operačního systému při návrhu embedded aplikací přináší řadu výhod. Vývojář má k dispozici funkční nástroje pro dekompozici řešeného problému a systémovou realizaci řešení. Zároveň je možné více abstrahovat od použitého hardware a realizovat platformě nezávislé programové konstrukce. Systémově navržený program je pak snáze udržitelný, rozšiřitelný a jeho funkcionalitu lze relativně snadno použít v jiných aplikacích a i na jiných platformách (pro které byl daný RTOS portován).

Na druhou stranu běžící RTOS prostředí klade další nároky na výpočetní zdroje (RAM, Flash, procesorový čas) a přináší tak určitý overhead. Další nevýhodou je nutnost zvládnutí nové technologie. Toto je však diskutabilní vzhledem k přirozené vlastnosti člověka učit se novým věcem.

Na trhu je k dispozici celá řada komerčních, i open-source operačních systémů pro embedded aplikace. Jmenujme například **eCos**, síťově orientovaný systém **Contiki**, **ChibiOS/Rt** orientovaný na hluboce vestavěné systémy (deeply embedded real time applications), úsporný preemptivní plánovač **ScmRTOS** a konečně **FreeRTOS** a jeho komerční mutace **OpenRTOS** a **SafeRTOS** (určený pro safety-critical aplikace). Všechny uvedené operační systémy byly portovány na jádro ARM Cortex-M3.

V této práci bude využit systém FreeRTOS. Důvodem k této volbě je jeho široké rozšíření v technické praxi a tudíž početná uživatelská základna schopná poskytnout silnou podporu při řešení potíží. Systém FreeRTOS byl dosud portován na 30 různých architektur a 17 vývojových prostředí, a dále se rozvíjí. S ohledem na tento fakt je čas vynaložený k získání znalosti tohoto prostředí jistě výhodnou investicí.

2.3 Experimentální platforma pro FreeRTOS

Základními předpoklady pro tržní úspěch a široké rozšíření konkrétního mikrořadiče (μC) v průmyslové praxi jsou společně s jeho cenou a množstvím inovativních prvků také uživatelské vlastnosti μC, především snadné a rychlé osvojení programovacích technik na dané platformě, snadnou tvorbu aplikací a minimální čas potřebný k vývoji aplikace (Time To Market).

Pro výrobce procesorové techniky a zejména mikrořadičů pro vestavěné aplikace je typické, že společně s integrovanými obvody dodávají na trh různé (softwarové i hardwarové) podpůrné nástroje, například kompilátory, knihovny funkcí, programátory/debuggery a vývojové kity (tzv. evaluation boards). Vývojový kit je typicky univerzální modul osazený daným μC umožňující vývojáři provádět praktické experimenty a tím si rychle osvojit práci s novým hardwarem.

Na Katedře měření Fakulty elektrotechnické probíhá výuka procesorové techniky a embedded systémů. S ní spojený experimentální vývoj embedded aplikací je realizován právě na vývojových kitech zmíněných v předchozím odstavci. Vzhledem k pořizovací ceně vývojových kitů dodávaných velkými výrobci je však často nelze využít jako reálné vestavěné systémy v pevných aplikacích, a zároveň jejich použití značně prodražuje výuku.

Předmětem této diplomové práce je využití operačního systému FreeRTOS v synchronizovaných embedded systémech. Nasazení platformy FreeRTOS se shodou okolností současně připravuje také při výuce předmětů orientovaných na problematiku pokročilých embedded systémů. Pro studium prostředí FreeRTOS je nutné mít k dispozici vhodnou vývojovou platformu. Součástí této diplomové práce bude proto konstrukce vlastního laboratorního modulu.

2.3.1 Podpora síťové komunikace

Zřejmě základní vlastností modulů tvořících distribuovaný systém je podpora síťové komunikace pomocí sady protokolů TCP/IP. Prostřednictvím sítě probíhá automatická synchronizace, i datová komunikace s nadřazeným systémem.

Pro operační systém FreeRTOS byly oficiálně implementovány dva nástroje pro síťovou komunikaci: μIP a lwIP. Oba zásobníky byly vyvinuty Adamem Dunkelsem na Swedish Institute of Computer Science.

Výhodou zásobníku μIP jsou jeho velmi nízké nároky na systémové zdroje (ROM, RAM), tyto jsou však vyváženy jeho omezenými možnostmi. LwIP má oproti němu širší

podporu síťových protokolů a může být využit pro implementaci protokolu PTP (viz. níže). Zároveň je implementace lwIP pro μC rodiny STM32 k dispozici na stránkách STMicroelectronics. LwIP je tak ideálním kandidátem pro implementaci synchronního distribuovaného systému.

2.3.2 Synchronizace zařízení v síti Ethernet

Pro účely synchronizace zařízení připojených k síti Ethernet v této diplomové práci bude využit Precision Time Protocol definovaný standardem IEEE-1588 ([16] a [17]). První verze PTP byla standardizována v roce 2002, později byl PTP standard upraven v roce 2008 a tento je označován jako PTP V2. Poznámky k alternativním možnostem řešení této problematiky (například využití protokolu NTP, SNTP nebo časovacího signálu ze satelitní sítě GPS a jiných) lze nalézt například v [15] nebo v [18].

PTP využívá ke své funkci protokol UDP, pomocí kterého se připojená zařízení automaticky synchronizují. V kontextu této práce bude protokol implementován nad funkčním TCP/IP stackem, tedy v aplikační vrstvě.

2.3.3 Průmyslové standardy pro Ethernet

Rozhraní Ethernet je v současné době masově rozšířeným standardem pro komunikaci v počítačových sítích. V průmyslu se tato technologie vzhledem ke své přímočaré implementaci (snadná dostupnost síťových komponent s tímto rozhraním a možnost integrace s prostředím Internet) prosazuje i v aplikacích dosud využívajících průmyslové sběrnice jako je CAN, RS232, RS485 a jiné.

Podstatnou nevýhodou tohoto rozhraní, která tlumí možnosti jeho aplikačního využití v průmyslu, je nedeterminismus komunikace plynoucí z využití mechanismu CSMA/CD. Z tohoto důvodu vzniklo množství průmyslových komunikačních protokolů pro Ethernet, které tento fakt různými technikami a na různých úrovních obchází.

Následuje stručný přehled vybraných průmyslových standardů pro Ethernet a jejich charakteristických vlastností:

- **Ethernet/IP**
  - Ethernet Industrial Protocol
  - v současnosti vyvíjený společností ODVA
  - funguje na aplikační vrstvě ISO/OSI modelu

  - využívá sadu protokolů TCP/IP
  - podpora IEEE-1588 (PTP je přímo mapován do objektového modelu CIP)
  - www.odva.org

- **EtherCAT**
  - vyvinut společností Beckhoff
  - real-time Ethernet
  - efektivní využití přenosového pásma pomocí 'on the fly' zpracování telegramů pro jednotlivá zařízení v rámci jediného rámce
  - master-slave uspořádání
  - pro přenos se využívá logická kruhová topologie přenosu telegramů
  - synchronizace pomocí IEEE-1588
  - www.ethercat.org

- **Ethernet Powerlink (EPL)**
  - 'CANopen over Ethernet'
  - hardwarově nezávislý real-time protokol ('software-only concept')
  - podpora IEEE-1588
  - existuje oficiální open-source implementace EPL
  - www.ethernet-powerlink.org

- **ModBus TCP**
  - implementace sběrnice MODBUS pro Ethernet
  - využívá sadu protokolů TCP/IP
  - klient-server komunikace
  - www.modbus.org

Vyčerpávající informace o těchto i jiných real-time Ethernet řešeních lze nalézt například v seriálu článků časopisu AUTOMA [22], v [23] a [24], a dále například na stránkách Industrial Ethernet University.

2.3.4 Tvorba uživatelského rozhraní

Zamýšlené uživatelské rozhraní celého systému představuje GUI aplikace pro bežný osobní počítač. Ta bude sloužit pro komunikaci s jednotlivými moduly zapojenými do distribuované sítě a získávání měřených dat. Implementace bude provedena v prostředí jazyka Java SE.

Kapitola 3

Precision Time Protocol (IEEE-1588)

Jak bylo uvedeno v části 2.3.2, je pro účely synchronizace distribuovaných modulů využit Precision Time Protocol IEEE-1588. V této kapitole bude nejprve popsán princip činnosti PTP, poté bude následovat popis jeho open-source implementace PTP deamon, která bude využita v této práci.

3.1 Stručný popis protokolu

PTP byl vyvinut za účelem velmi přesné synchronizace distribuovaných počítačových systémů v sítích LAN. Za velmi přesnou je v tomto případě považována synchronizace v sub-mikrosekundovém rozsahu (jednotky až stovky nanosekund). Protokol lze využít v systémech komunikujících v lokálních sítích (LAN), které podporují multicast messaging - typicky v Ethernetu. K přenosu synchronizačních zpráv využívá protokol UDP.

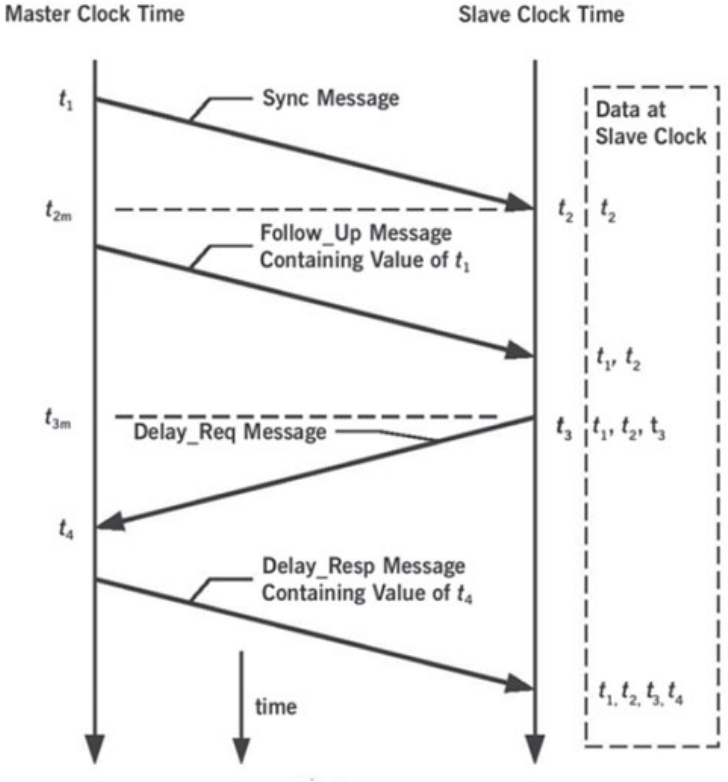
Vzhledem k tomu, že PTP poskytuje funkční nástroj pro exaktní synchronizaci distribuovaných systémů, stává se standard IEEE-1588 - jak uvádí [18] - součástí mnoha moderních průmyslových komunikačních protokolů (Ethernet/IP, PROFInet, Ethernet POWERLINK).

3.2 Princip synchronizace PTP

Kompletní specifikaci standardu IEEE-1588 [16] lze objednat v digtální nebo tištěné podobě na stránkách asociace IEEE http://standards.ieee.org.

V sytému synchronizovaném pomocí PTP jsou hodinová zařízení typu Master a Slave. Master zařízení (podle kterého se Slave zařízení synchronizují) je typicky časováno pomocí mechanismu (například pomocí GPS nebo radiového signálu), který považujeme za dostatečně přesný.

Proces synchronizace se skládá ze dvou fází. V první fázi je určen časový rozdíl (offset) mezi Master a Slave dvojicí pomocí synchronizační zprávy Sync vyslané Master zařízením. V okamžiku odeslání *Sync* zprávy Master sejme časovou značku (timestamp) - záznam vlastního systémového času. Tuto hodnotu následně odešle ve zpráve *Follow up* Slave zařízením. Informační obsah zprávy *Follow up* později Slave využije pro výpočet časového odstupu (offset) synchronizující se dvojice. Druhá fáze synchronizace slouží k určení aktuálních přenosových vlastností komunikačního média. V této fázi dojde k výměně dvojice zpráv *Delay Request* (vyslanou Slave zařízením) a *Delay Response* (vyslanou Master zařízením), která poskytne potřebné informace pro výpočet dopravního zpoždění šíření zpráv. Situaci ilustruje obrázek 3.1. V okamžiku přijetí zprávy *Follow up* má Slave



Obrázek 3.1: Proces synchronizace pomocí PTP (zdroj evaluationengineering.com)

zařízení k dispozici časové údaje t_1 a t_2 a může formulovat následující rovnici

$$delay + offset = t_2 - t_1, \tag{3.1}$$

kde $offset$ je časový odstup zařízení Slave od zařízení Master a *delay* představuje dopravní zpoždění přenosu zprávy komunikačním médiem. Po přijetí zprávy *Delay Response* může Slave zařízení zkonstruovat rovnici

$$delay - offset = t_4 - t_3, \tag{3.2}$$

Uvedené vztahy tvoří soustavu lineárních rovnic, jejíž řešení má podobu

$$delay = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \tag{3.3}$$

$$offset = \frac{(t_2 - t_1) - (t_4 - t_3)}{2}. \tag{3.4}$$

Přehledný popis mechanismu lze nalézt též v [19].

Algoritmus výpočtu předpokládá symetrické zpoždění zpráv, tedy že komunikace ve směru od Master k Slave zařízení probíhá stejně rychle. Toto však nemusí platit v případě větších sítí obsahujících přepínače a routery. Pro prvky síťové infrastruktury bylo definováno zařízení Boundary clock, které se synchronizuje k připojenému Master zařízení a na portech, ke kterým jsou připojena Slave zařízení vystupuje jako Master. V této konfiguraci jednotlivá spojení vykazují symetrické vlastnosti.

Klíčovou vlastností ovlivňující přesnost synchronizace je přesnost odečítaní časových značek při příjmu a odeslání synchronizačních zpráv, kterou musí zajistit precizní design hardwaru.

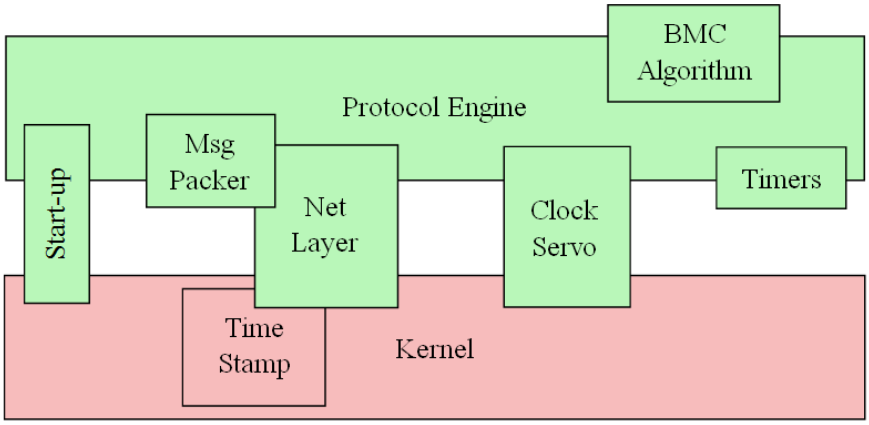
3.3 PTP deamon

PTP deamon (PTPd) je kompletní implementací protokolu PTP definovaného standardem IEEE1588 v obou existujících verzích (IEEE-1588-2002, resp. IEEE-1588-2008).

PTPd je open-source projekt s licencí typu BSD a jeho zdrojové kódy jsou k dispozici společně dalšími informacemi o projektu na stránkách projektu [20]. Na stránkách projektu je rovněž volně přístupná stručná dokumentace zdrojových kódů a architektury PTPd, ze které budeme v následujícím textu vycházet.

3.3.1 Architektura PTPd

Zdrojový kód PTPs se skládá z několika funkčních komponent. Situaci popisuje blokový diagram viz. obrázek 4.5.



Obrázek 3.2: Organizace zdrojového kódu projektu PTPd (převzato z [20])

Zdrojový kód je rozdělen na část platform-independent (soubory umístěné v kořenovém adresáři PTPd) a část platform-dependent (její zdrojové soubory jsou umístěné v podadresáři `dep/`).

3.3.2 Hlavní komponenty systému

Protocol Engine,`protocol.c`

Implementuje stavový automat definovaný IEEE1588. Stavový automat je implementován poměrně nepřehledným příkazem `switch`pracujícím nad stavy protokolu (MASTER, SLAVE, PASSIVE, FAULTY a dalších). Mechanismus je implementován procedurou `doState()`, která je periodicky volána v nekonečné smyčce. Přechody mezi stavy automatu jsou zajišťované procedurou `toState()`a jsou primárně výsledkem činnosti BMC (Best Master Clock) algoritmu.

Základní události a akce jsou:

- **Události**
  - Message recieves (příjem zpráv) zpracované procedurou `handle()`
  - Timer Expiration
- **Akce**
  - Message sends (vysílání zpráv) prováděné procedurou `issue()`
  - reset časovačů

- periodické spouštění BMC algoritmu
- update a synchronizace systémového času

BMC, `bmc.c`

Implementace Best Master Clock algoritmu (funkce `bmc()`) definovaného specifikací. Funkce `bmc()`na základě synchronizačních zpáv (Sync messages) získaných od ostatních MASTER zařízení v systému vrací stav do kterého by měl stavový automat v daném běhu přejít (stavy: MASTER, SLAVE, LISTENING nebo PASSIVE).

Clock Servo, `dep/servo.c`

Clock servo provádí výpočet časového odstupu (offset-from-master) ze známých hodnot master-to-slave delay a slave-to-master delay. Vypočtená hodnota je využita k parametrizaci systémového času.

Message Packer, `dep/msg.c`

Provádí zpracování PTP zpráv ve formátu definovaném specifikací.

Network Layer, `dep/net.c`

Tato komponenta PTPd zajišťuje inicializaci spojení, příjem a odesílání dat protokolu. Tato vrstva rovněž extrahuje časové značky (timestamp) z přijatých Event zpráv.

Time Stamp

Záznam času přijetí/odeslání dat. Zaznamenané časové údaje jsou využity při výpočtu synchronizace. Tato komponenta je implementována hardwarově pro minimalizaci jitter (časový posun) při odečitání časů přijetí/odeslaní rámců.

Timer, `dep/timer.c`

Časovače využívané komponentou Protocol Engine. Slouří k řízení periodického vysílání Sync message (v případě MASTER zařízení), vysílání Delay Request (v případě SLAVE zařízení), periockého spouštění BMC algoritmu a Sync Recieve timeouts.

Start-up, `dep/startup.c`

Inicializace stavu systému a jeho uživatelská konfigurace.

Other code

Soubor `ptpd.c`obsahuje vstupní bod PTPd.

Kapitola 4

Vývoj laboratorního modulu pro FreeRTOS

V této kapitole bude popsán procesorový modul, který (jak bylo uvedeno v části 2.3) byl v rámci této diplomové práce navržen a zkonstruován jako testovací platforma pro operační systém FreeRTOS a pro následné využití při výuce předmětů orientovaných na problematiku embedded systémů vyučovaných na Katedře měření Fakulty elektrotechnické, ČVUT v Praze. Pro snazší identifikaci a úsporu místa v textu nese tento modul označení **microLab** (v dalším textu μLab).

Úvodní část kapitoly definuje základní požadavky a charakteristické vlastnostnosti, které od realizovaného systému budeme očekávat. V dalších častech pak bude postupně popsáno elektrické zapojení systému, vlastní fyzická realizace, závěr kapitoly je věnován uživatelskému popisu přístroje.

4.1 Systémové požadavky

Výstupem návrhu by měl být malý univerzální procesorový stroj - vývojový modul - použitelný jak pro studijní experimentální vývoj, tak pro nasazení v reálných embedded aplikacích.

4.1.1 Kompaktní návrh

Student by při práci s přípravkem neměl mít pocit, že pracuje se složitým uzavřeným systémem (black box). Naopak, měl by mít kompletní přehled o všech prvcích, které má na plošném spoji modulu k dispozici. Tento požadavek se s rostoucí složitostí systému

stává neudržitelným, proto byl při návrhu kladen důraz na co nejjednodušší fyzickou realizaci s malým množstvím součástek. Splnění tohoto kritéria navíc příznivě ovlivňuje cenu a umožňuje navrhnout elegantní modul s kompaktními rozměry.

Pro zachování malých rozměrů plošného spoje je zároveň nutné vhodně redukovat počet uživatelsky přístupných špiček μC. Podmínkou ale je, aby tato redukce funkční potenciál μC příliš neomezila, a uživatel mohl maximálně využít periferie, které jsou přítomné na čipu osazeného mikrořadiče.

4.1.2 Snadné programování paměti Flash

Klíčovým požadavkem, který měl vyvíjený přípravek splňovat, je komfortní způsob jeho programování pomocí rozhraní dostupného na běžném současném (2011) PC či notebooku, bez nutnosti použití přídavného hardwaru. Zároveň však musí být zachována možnost programovat a ladit kód pomocí rozhraní SWD široce podporovaného vývojovými prostředími všech velkých výrobců (Atollic, KEIL, IAR). Toto umožňuje jednak realizaci studentských prací mimo prostory laboratoře/školy, a současně pohodlný firmware update v případě reálné embedded aplikace.

4.2 Použité komponenty a technologie

4.2.1 Mikrořadič

Modul μLab je primárně určen pro osazení mikrořadičem STM32F105R [2] s jádrem ARM Cortex-M3 [5], který dodává společnost STMicroelectronics. Vzhledem k pinové kompatibilitě rodiny STM32F je s jistými omezeními možné osadit libovolný jiný μC rodiny STM32F v pouzdře LQFP64.

4.2.2 Dostupné periferie

V rámci zachování přehledného layoutu DPS a kompaktních rozměrů modulu byla zvolena následující konfigurace uživatelsky přístupných špiček/periferií:

I/O rozhraní modulu

- **Vyvedené piny**
  - PA0 - PA10

  - PB10 - PB15

- **Komunikační rozhraní**
  - USB OTG
  - 3x USART
  - 2x SPI
  - 1x I2C
  - 1x CAN
- **Další funkce**
  - 8x 12b AD převodník
  - 2x 12b DA převodník
  - 1x aplikační LED

4.2.3 Komunikace s PC

Komunikaci modulu s PC je možné realizovat pomocí sběrnice USB připojené k modulu μLab prostřednictvím konektoru Mini-B.

Druhou možností je komunikace pomocí rozhraní RS232. Její implementace je sice výrazně snazší, avšak toto rozhraní již není typicky instalováno na současných přenosných počítačích. V takovém případě je třeba použít převodník USB/RS232. Rozhraní RS232 nicméně dosud patří mezi základní komunikační prostředky v procesorové technice.

4.2.4 Programovací rozhraní

Existují minimálně dvě možnosti programování programové paměti (Flash) osazeného mikrořadiče:

- SWD (Serial Wire Debugging)
- DFU (Device Firmware Update)

Výhodou rozhraní SWD je možnost ladění programu za běhu. K jeho použití je potřeba připojit k modulu ladící sondu (programátor) ST-link nebo vývojový kit VL Discovery, který sondu obsahuje a zároveň podporuje ladění externích procesorových modulů. ST-link i modul VL Discovery vyrábí společnost STMicroelectronics.

Protokol DFU [8] sice neumožňuje runtime ladění programového kódu, k jeho použití však není potřeba žádný přídavný hardware, stačí pouze připojení modulu μLab k počítači pomocí USB.

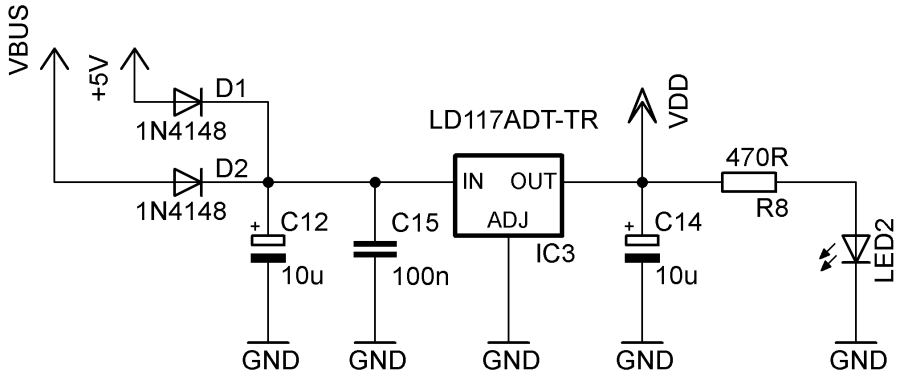
V případě, že designer aplikace nemá k dispozici programátor s podporou runtime ladění aplikace, autor doporučuje použít pro získávání debugging informací (chybové výpisy, stavy proměnných apod.) ověřenou funkční komunikaci pomocí rozhraní RS232 nebo USB a programovat systém pomocí DFU.

4.3 Popis elektrického zapojení vývojového modulu

V této části budou popsány jednotlivé bloky elektrického zapojení sytému.

4.3.1 Obvod napájení

Schéma zapojení obvodu napájecího zdroje modulu je na obrázku 4.1.



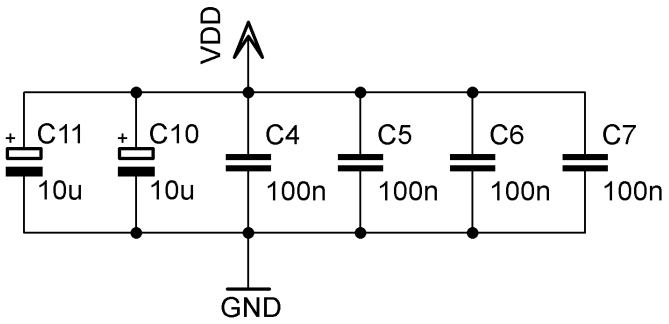
Obrázek 4.1: Schéma obvodu napájení

Systém je navržen tak, aby bylo možné použít pro jeho napájení jak externí zdroj napětí +5V (klasický adaptér) připojený pomocí konektoru PWR_5V, tak i napájecí napětí distribuované sběrnicí USB. Toto umožňuje zapojení dvojice diod D1 a D2 na vstupu zdroje. Tyto zároveň slouží jako ochrana proti přepólování.

Pro stabilizaci napájecího napětí osazeného procesoru je použit stabilizátor LD1117 s výstupním napětím 3,3V v katalogovém zapojení. Přípustné napájecí napětí procesoru

však je v rozsahu 2,0 až 3,6V (2,4 až 3,6V v případě využití funkce ADC, viz. [2]), proto může být pro napěťouvou stabilizaci použit i obvod s jinou úrovní výstupního napětí v tomto rozsahu (samozřejmě nutně v kompatibilním pouzdře).

Pro blokování napájecího rozvodu celého systému jsou použity kapacitory podle schématu 4.2. Použité kapacitory jsou určeny výhradně k filtraci a blokování napájecího napětí μC.



Obrázek 4.2: Blokování napájení

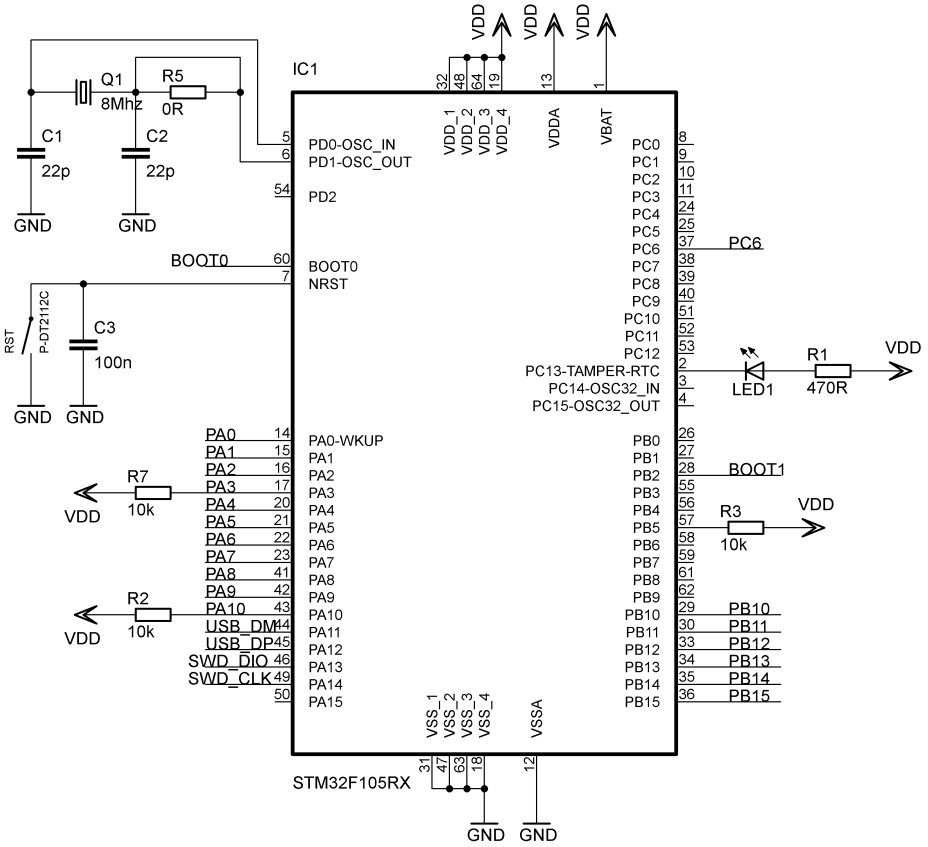
4.3.2 Připojení mikrořadiče

Na obrázku 4.3 je znázorněno připojení mikrořadiče.

Napájecí piny pro analogovou i digitální část μC jsou připojeny na společný rozvod, který je dostatečně blokován sadou kapacitorů (viz. výše). Z důvodu úspornosti výsledného návrhu není podporována funkce Battery Backup a vstup VBAT je připojen k napájecímu rozvodu VDD [9]. Odpovědnost za stabilitu napájecího napětí nese designer nadřazené aplikace.

Pro potřeby testování funkce systému a ladění kódu je k pinu PC13 μC přes sériový rezistor R1 připojena luminiscenční dioda LED1.

Obvod externího krystalového oscilátoru je tvořen krystalovým výbrusem Q1, dvojicí keramických kapacitorů C1 a C2 a přemostěným rezistorem R5 (podrobněji v 4.4.1 a [9]).



Obrázek 4.3: Okolní obvody mikrořadiče

Obvod aktivace Bootloaderu

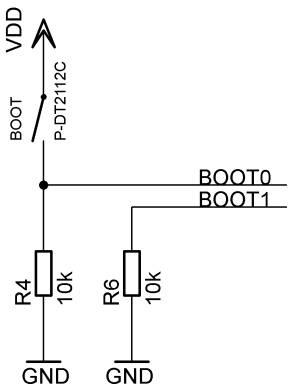
Modul μLab podporuje protokol DFU pro instalaci/update paměti Flash. Pro jeho funkci je třeba aktivovat bootloader (zavaděč programu), který je uložený v systémové paměti μC. K tomu slouží obvod ilustrovaný na obrázku 4.4.

Rezistory R2, R3 a R7 slouží k definici napěťové úrovně na periferiích, které umožňují bootování programu, ale jejich použití k tomuto účelu se nepředpokládá. Konstrukční poznámky k jejich funkci a případnému osazení jsou v části 4.4.1.

4.3.3 Obvody rozhraní

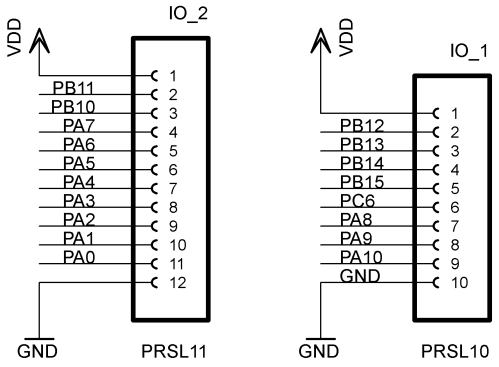
Vstupně-výstupní linky modulu

Vybrané piny (a jim přiřazené periferie) μC jsou vyvedeny na postranní konektorové lišty společně s napájecími rozvody modulu. Tyto konektorové lišty tvoří aplikační rozhraní systému, které umožňuje snadnou integraci modulu do jiné nadřazené aplikace. Konfigu-



Obrázek 4.4: Obvod aktivace bootloaderu

race konektorů je na obrázku 4.5. Rozteč postranních konektorů odpovídá standartnímu



Obrázek 4.5: Aplikační rozhraní

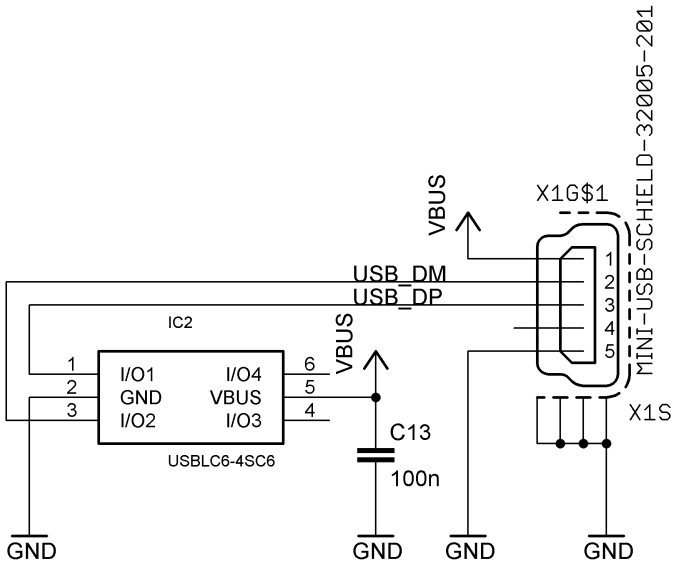
rastru (100 mil) a proto může být modul použit i pro laddění aplikací na rastrovaných kontaktních polích.

USB

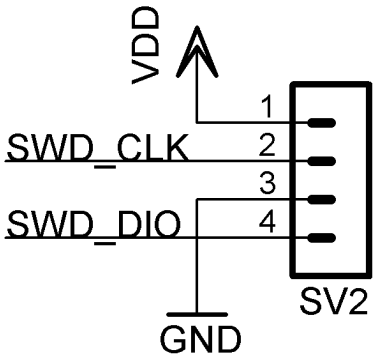
Sběrnice USB je k μC připojena podle schématu na obrázku 4.6. Pro ochranu proti elektrostatickému výboji (ESD) slouží obvod IC2. Jeho přítomnost nemá vliv na praktickou funkci systému.

Programovací rozhraní

Rozhraní SWD použitelné pro programování modulu a jeho ladění je vyvedeno na čtyřpinový konektor. Připojení jeho kontaktů je na obrázku 4.7.



Obrázek 4.6: Připojení USB



Obrázek 4.7: Rozhraní SWD

4.4 Popis fyzického návrhu (hardware)

4.4.1 Návrh plošného spoje

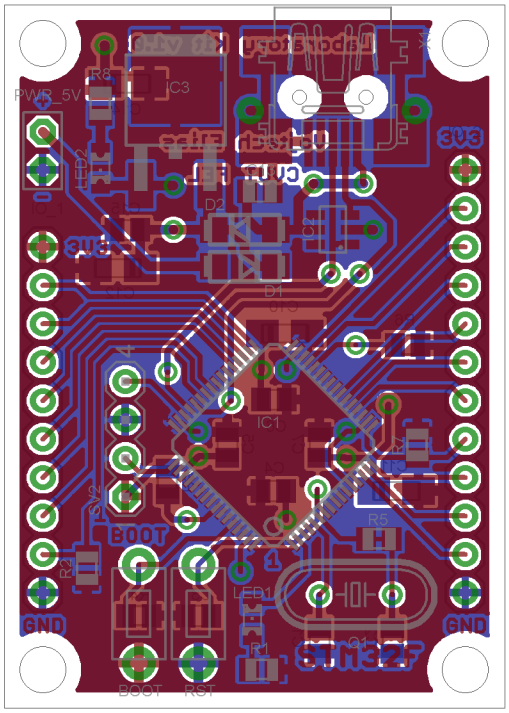
Dvouvrstvý plošný spoj modulu μLab, na kterém je výše popsané elektrické zapojení realizováno, byl navržen v prostředí Eagle od společnosti CadSoft. Toto prostředí je s určitými omezeními dostupné zdarma [1].

DPS má obdélníkový tvar o šířce 1300 mil (přibližně 33 mm) a výšce 1830 mil

[1] Více informací lze nalázt na stránkách společnosti http://www.cadsoft.de/.

(přibližně 46.5 mm). Plošný spoj byl navržen v palcovém rastru (1 palec = 2.54 cm) a platí, že jeden mil je tisícinou palce.

Na obrázku 4.8 je komplexní pohled na provedení plošného spoje modulu μLab. Pro zjednodušení návrhu a zlepšení elektrických vlastností systému je po obou stranách

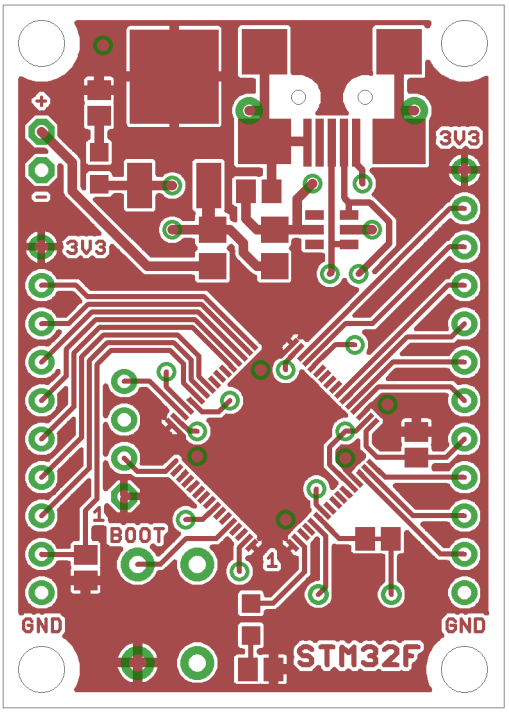


Obrázek 4.8: Provedení DPS (pohled shora)

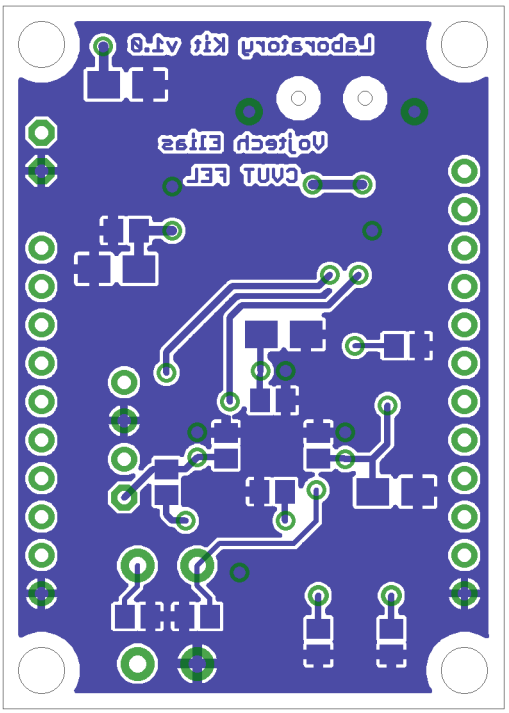
plošného spoje rozlitá měď. Tato je na horní straně připojena k napájecímu napětí desky VDD a na spodní straně je připojena k zemi GND.

Modul je možné vestavět do jiné aplikace pomocí čtveřice montážních otvorů o průměru 120 mil (3.048 mm) umístěných v rozích DPS. Rozteč postranních konektorů odpovídá standartnímu rastru (100 mil) a proto může být modul použit i pro ladění aplikací na rastrovaných kontaktních polích.

Přehledný pohled na horní (TOP) vrstvu plošného spoje je vyobrazen v 4.9. Jak již bylo zmíněno výše, v místech kterými nevedou žádné signální spoje je využito rozlité mědi připojené k napájecímu napětí. V horní části desky jsou situované obvody stabilizace napájecího napětí obvod pro připojení rozhraní USB společně s konektorem USB Mini-B. Po stranách plošného spoje jsou umístěny aplikační datové konektory. Mikrořadič je umístěn ve střední části desky, v prostoru mezi spodními dvěma montážními otvory jsou umístěna tlačítka a uživatelská LED. Přehledný pohled na spodní (BOT) vrstvu plošného



Obrázek 4.9: Horní strana DPS (TOP)



Obrázek 4.10: Spodní strana DPS (BOTTOM)

spoje je vyobrazen v 4.10.

Z obrázku je zřejmé, že významná část plošného spoje je na spodní straně DPS propojena rozlitím mědi. Tato je připojena k zemnícímu vodiči GND. Na spodní straně DPS jsou umístěny filtrační kapacitory a některé rezistory (viz. dále). Postranní konektorové lišty (aplikační rozhraní) se též pájí ze spodní strany DPS.

Konstrukční instrukce

Osazení pull-up/down rezistory

Rezistory R2, R3 a R7 slouží k definici napěťové úrovně na periferiích, které umožňují bootování programu, ale jejich použití k tomuto účelu se nepředpokládá. Algoritmus bootloaderu v cyklu testuje přítomnost dat na bootovatelných periferií, proto je potřeba na vstupních pinech nepoužitých periferií úroveň definovat. Podrobnější informace o bootloaderu μC z rodiny STM32F lze nalézt v [7].

Mikrořadič STM32F105, který je osazený v modulu μLab umožňuje bootování pomocí periferií USART1, USART2, CAN2 a DFU (USB). Protože modul je primárně navržen pro bootovaní pomocí DFU, jsou u ostatních periferií (které jsou na čipu procesoru implementovány) umístěny pull-up/down rezistory.

POZNÁMKA: Přítomnost pull-up/down rezistorů omezuje možnosti využití daných pinů μC. Pro laboratorní účely je proto možné tyto rezistory neosadit a ověřit funkci bootloaderu. Jejich osazení potom provést až v případě, že DFU modulu nebude pracovat správně.

Oscilátor

Rezistor R5 je součástí designu DPS pro případ, že by obvod externího oscilátoru vyžadoval pro správnou funkci další úpravy. Toto není typicky třeba, proto se osazení R5 nepředpokládá. V opačném případě je nutné před osazením R5 nožem odstranit zkrat který je veden mezi pájecími pady R5. Další informace o problematice návrhu obvodu oscilátoru pro μC z rodiny STM32 jsou k dispozici v [10] a [9].

Napěťový stabilizátor

Je-li pro stabilizaci napětí μLab využit integrovaný stabilizátor v pouzdře SOT-223, je pro zajištění správné funkce nutné odskřípnout jeho prostřední vývod (vývod číslo 2), ten

je totiž spojen s výstupem stabilizátoru a mohlo by dojít ke zkratu a zničení stabilizátoru. Přímo pod tímto vývodem je totiž na DPS umístěn prokovený otvor spojený se zemí.

4.5 Uživatelský popis systému

4.5.1 Typický odběr modulu

Oživování nově osazeného procesorového modulu může být někdy provázeno problémy. Tyto bývají nejčastěji způsobené nedokonalým pájením. Pro snaz39 lokalizaci problému je proto výhodné mít k dispozici typické hodnoty odběru elektrického proudu v určitých módech činnosti procesoru.

Tabulka 4.1 uvádí naměřené hodnoty proudu tekoucího napájecím přívodem modulu v daných módech činnosti. Při měření byl modul napájen externím zdrojem 5 V a byl

Mód činnosti	**I [mA]**
Aplikační program	21
Reset	12
Bootloader	27

Tabulka 4.1: Typické hodnoty proudového odběru modulu μLab

použit digitální multimetr Mastech M-832.

4.5.2 Možnosti napájení modulu

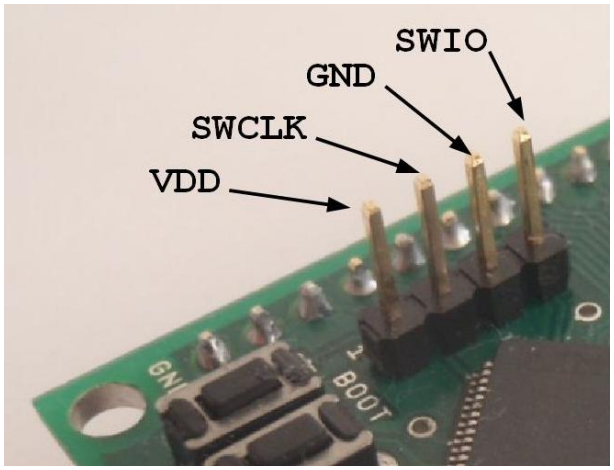
Modul je napájen externím napájecím zdrojem stejnosměrného napětí 5-15 V (konektor PWR_5V, polarita připojeného napětí je na desce vyznačena) a/nebo napájecím rozvodem připojené sbernice USB, přítomnost napájecího napětí je indikována svitem LED2 umístěné v horní části DPS. Krajní špičky postranních konektorových lišt jsou vždy 3V3 a GND - napájecí napetí 3,3V. To umožnuje použití modulu pro napájení vyvíjené aplikace. Proud dodávaný do připojeného obvodu musí odpovídat možnostem osazeného stabilizátoru a ochranných diod na vstupu napěťové stabilizace. Napájecí piny na postranních lištách jsou přehledně označeny přímo na DPS jako 3V3 a GND.

4.5.3 Způsoby nahrávání programu do vnitřní paměti Flash

Jak bylo uvedeno v části 4.2.4, pro programování vnitřní paměti Flash a ladění programu modulu μLab může být využito rozhraní SWD a DFU.

Rozhraní pro ladění programu Serial Wire Debug

Toto ladící rozhraní může být využito v případě, že je k dispozici k dispozici programátor s tímto rozhraním a vývojové prostředí, které tento programátor podpruje. Na obrázku 4.11 je vyznačeno uspořádání pinů konektoru SV2 pro připojení rozhraní SWD.



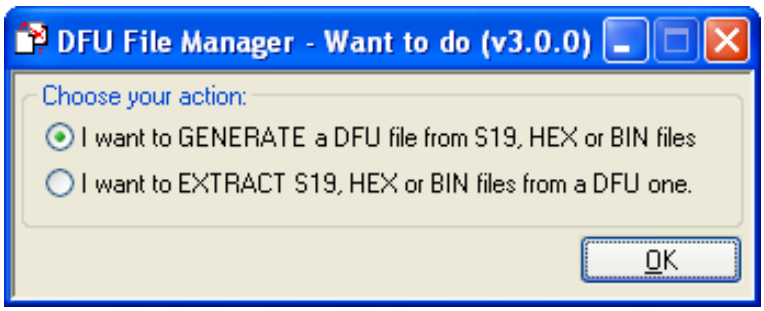
Obrázek 4.11: Uspořádání pinů konektoru pro připojení ladícího rozhraní SWD

Rozhraní pro programování Flash paměti Device Firmware Upgrade

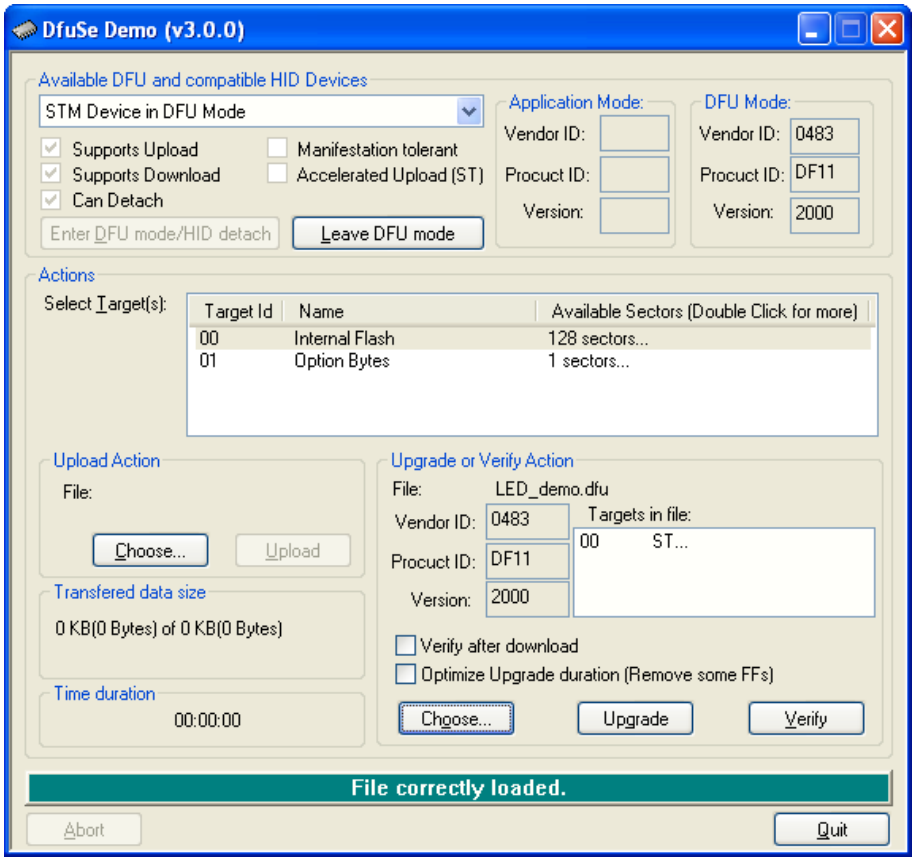
Aby bylo možné programovat modul pomocí rozhraní DFU, je nejprve třeba ze stránek STMicroelectronics stáhnout softwarový balík DfuSe. Ten obsahuje všechny potřebné utility pro programování cílové platformy.

Pomocí programu **DFU File Manager** (viz. obrázek 4.12) vygenerujeme binárního nebo hexadecimálního výstupu kompilace soubor ve formátu `dfu`.

Poté můžeme pomocí USB kabelu propojit μLab s počítačem, spustit program **DfuSe Demonstration** a aktivovat bootloader stiskem a pridržením tlačítka označeného na DPS jako BOOT a následným stiskem tlačítka RST umístěného vedle něj. Pro korektní funkci musí být tlačítko BOOT stisknuto ještě bezprostředně po resetu obvodu. Program **DfuSe Demonstration** použijeme k vlastnímu natažení programu do interní paměti Flash (viz. obrázek 4.13).



Obrázek 4.12: Okno programu DFU File Manager



Obrázek 4.13: Okno programu DfuSe Demonstration

4.6 Errata

V této části se nachází odhalené návrhové chyby modulu μLab.

4.6.1 OTG VBUS

Je-li osazen μC z řady Connectivity Line, je třeba pro správnou funkci periferie OTG vytvořit na DPS zvláštní propojku mezi napájecím vodičem sběrnice USB a pinem PA9. Zároveň tento pin nesmí být v aplikaci konfigurován jako výstup. Není tak umožněno využít periferii USART1.

V případě přesto omylem dojde k této nefunkční konfiguraci, dojde k zablokování funkce μC. V tomto stavu nelze k procesoru přistoupit pomocí rozhraní SWD. Je nutné uvézt μC do boot módu a vymazat jeho Flash paměť pomocí programu **STM32 ST-Link Utility**.

4.7 Závěrečná poznámka

V průběhu tvorby této diplomové práce byla provedena revize (redesign) návrhu plošného spoje. V rámci této revize byly odstraněny všechny známé návrhové chyby, kterými byl původní modul zatížen. Nová verze plošného spoje byla odeslána do výroby a následně byla vyrobena série v počtu 35-ti kusů. Z časových důvodů však nebyl tento nový modul osazen a proto nemohl být využit k realizaci experimentů v této práci.

Kapitola 5

Real-time operační systém FreeRTOS

Tato část je věnována real-time operačnímu systému FreeRTOS a jeho využití v embedded systémech.

Operační systém bude nejprve krátce představen výčtem svých charakteristických vlastností. Následuje rozbor činnosti FreeRTOS a popis jeho portace pro procesory s jádrem ARM Cortex-M3. Informace popisující funkci systému FreeRTOS byly částečně čerpány v [26].

5.0.1 Stručný popis systému FreeRTOS

FreeRTOS je v podstatě jádro real-time operačního systému (kernel) obsahující preemptivní plánovač, nástroje pro synchronizaci výpočetních procesů a jejich vzájemnou komunikaci a správu paměti. Přestože je navržen tak, aby byl co nejjednodušší s ohledem na nízké hardwarové nároky, snadné použití a rozšiřitelnost na nové procesorové platformy, poskytuje uživateli velmi užitečné nástroje pro realizaci aplikací.

V současnosti existují oficiálně podporované portace pro 30 architektur.

Následuje stručný výčet základních vlastností systému:

- preemptivní, kooperativní a hybridní plánovač úloh (scheduler)
- nízké nároky na paměť (instalace kernelu v laboratorním modulu zabírá přibližně 6kB paměti Flash)
- tasks (procesy), co-routines

- prostředky meziprocesní synchronizace/komunikace (mutexy, semafory, fronty zpráv)
- správa paměti + stack overflow detekce

FreeRTOS je open-source projektem vyvíjeným společností Real Time Engineers Ltd. a je zdarma dostupný pod modifikovanou licencí GPL. Pod touto licencí je možné používat software i v komerčních aplikacích. Podrobnosti týkající se modifikace GPL licence jsou k dispozici na stránkách projektu (http://www.FreeRTOS.org).

Vedle open source verze systému existují dvě příbuzné platformy dostupné pod komerční licencí:

- **OpenRTOS**
  - komerční verze systému FreeRTOS
- **SafeRTOS**
  - verze systému určená pro safety-critical aplikace
  - dokumentovaná a testovaná verze systému odpovídající standardu IEC 61508 SIL 3

5.0.2 Funkce scheduleru

Jádrem systému je preemptivní plánovač úloh - scheduler. Zajišťuje současný běh výpočetních procesů tak, že v daném okamžiku jsou zdroje procesoru přiděleny právě jednomu procesu. Scheduler zároveň realizuje algoritmus výběru procesu pro přidělení zdrojů procesoru. Vysvětlení pojmu proces dává následující odstavec.

Procesy

Proces (task) je izolovaný segment programového kódu, který v rámci aplikace řeší nějaký konkrétní podproblém. Aplikace je tvořena množinou procesů.

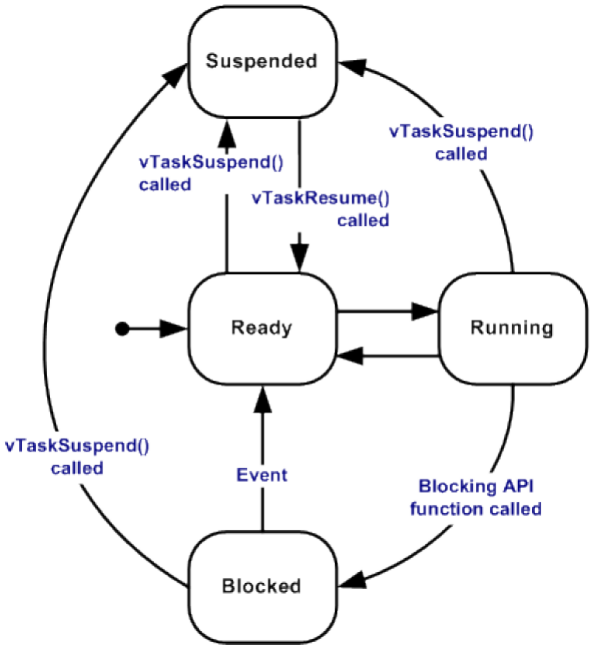
V systému FreeRTOS může proces být v jednom ze čtyř stavů:

- `running`
  - Proces je aktivní a vykonává svojí činnost.
- `ready`

  - Proces je připraven k přechodu do stavu `running`, nemá však přiděleny výpočetní zdroje, protože v daném okamžiku je aktivní jiný proces se stejnou nebo vyšší prioritou.

- `blocked`
  - proces čeká na nějakou externí (např. pokus o čtení dat z prázdné fronty) nebo časovou událost (volání funkce `vTaskDelay()` blokuje proces na určitou dobu). Procesy v tomto stavu mají definovaný timeout, po kterém přechází do stavu `ready` .

- `suspended`
  - tento stav je aktivován resp. deaktivován pouze voláním funkce `vTaskSuspend()` resp. `vTaskResume()`



Obrázek 5.1: Stavový diagram výpočetního procesu v systému FreeRTOS (převzato z [26])

Stavový diagram výpočetního procesu je znázorněn na obrázku 5.1. Každý proces v systému má danou prioritu - nazáporné celé číslo - a platí, že vyšším číslům odpovídá vyšší priorita. Planávač přiděluje procesorový čas přednostně procesu, který má nejvyšší prioritu, ze všech procesů, které jsou ve stavu **ready**.

5.0.3 Způsob časování jádra OS

Měření času v kernelu operačního systému FreeRTOS se odvíjí od počtu tiknutí systémového časovače. Hodnota aktuálního času, jak jej vnímá FreeRTOS je uložená v proměnné `xTickCount` která je v průběhu inicializace scheduleru nastavena na nulu pak inkrementována v obsluze přerušovací rutiny nakonfigurovaného časovače.

V případě portace na procesory rodiny STM32F je jako generátor systémového času použit Cortex System Timer (SysTick), což je periferie Cortexového jádra procesoru.

Frekvence generátoru systémového času FreeRTOS je nastavitelná pomocí kombinace hodnot `configCPU_CLOCK_HZ` a `configTICK_RATE_HZ` v souboru `FreeRTOS_config.h`.

Při každém obnovení hodnoty systémového času kernel ověřuje, zda není potřeba přerušit činnost běžícího procesu a předat řízení jinému v rámci použité rozvrhovací strategie.

5.1 Portace pro jádro ARM Cortex-M3

Portace pro procesory s jádrem ARM Cortex-M3 je součástí oficiální distribuce FreeRTOS. Tvoří ji dvojice souborů `portmacro.h` a `port.c` , které jsou umístěné v tzv. Portable Layer, tedy podadresáři /Source/Portable obsahujícím všechny oficiální portace systému. Portable layer rovněž obsahuje adresář MemMang s trojicí schémat pro správu paměti.

Pro instalaci systému na konkrétní platformě je třeba vybrat portaci odpovídající použitému procesoru a kompilátoru. V případě použití vývojového prostředí Attolic TrueSTUDIO a procesoru STM32F105, který je osazen na modulu μLab je použit GCC/ARM_CM3.

5.1.1 Mapování přerušení

Aby bylo možné program zkompilovat, je třeba namapovat rutiny přerušení **SVCall**, **PendSV** a **SysTick** do portable vrstvy FreeRTOS (soubor `port.c`).

V případě použití mikrořadiče STM32 a knihovny Standard Peripherals Library s podporou CMSIS (Cortex Microcontroller Software Interface Standard) lze toto provést následující úpravou ve startup souboru `startup_stm32fxx_x.s` projektu:

- `SVC_Handler` → `vPortSVCHandler`
- `PendSV_Handler` → `xPortPendSVHandler`

- `SysTick_Handler` → `xPortSysTickHandler`

SVCall

Supervisor call (SVC) je výjimka vyvolaná instrukcí SVC. Aplikace může použít tuto instrukci pro přechod do privilegovaného módu procesoru. Obslužná rutina této výjimky obsahuje následující kód:

```
ldr        r3, pxCurrentTCBConst2 /* Restore the context. */
ldr r1, [r3] /* Use pxCurrentTCBConst to get the pxCurrentTCB address. */
ldr r0, [r1] /* The first item in pxCurrentTCB is the task top of stack. */
ldmia r0!, {r4-r11} /* Pop the registers that are not automatically saved on
msr psp, r0 /* Restore the task stack pointer. */
mov r0, #0
msr        basepri, r0
orr r14, #0xd
bx r14

.align 2
pxCurrentTCBConst2: .word pxCurrentTCB
```

PendSV

Přerušení PendSV je využíváno v prostředí operačního systému pro přepínání kontextu.

SysTick

Přerušení od systémového časovače je ošetřeno rutinou:

```
void xPortSysTickHandler( void )
{
unsigned long ulDummy;

        /* If using preemption, also force a context switch. */
        #if configUSE_PREEMPTION == 1
                *(portNVIC_INT_CTRL) = portNVIC_PENDSVSET;
        #endif

        ulDummy = portSET_INTERRUPT_MASK_FROM_ISR();
        {
                vTaskIncrementTick();
        }
        portCLEAR_INTERRUPT_MASK_FROM_ISR( ulDummy );
}
```

Inicializace systému

Před spuštěním jádra FreeRTOS je třeba inicializovat aplikační procesy. K tomu slouží makro `xTaskCreate( pvTaskCode, pcName, usStackDepth, pvParameters, uxPriority, pxCreatedTask )`. Význam parametrů je následující:

- **pvTaskCode**
  - Ukazatel na funkci s kódem procesu
  - Funkce musí být implementována jako nekonečná smyčka
- **pcName**
  - Jméno procesu
- **usStackDepth**
  - velikost zásobníku procesu
  - počet proměnných, které lze do zásobníku uložit

- **pvParameters**
  - ukazatel na pole paramterů, který bude předán vzniklému procesu
- **uxPriority**
  - priorita procesu
- **pxCreatedTask**
  - zpětný ukazatel na nově vzniklý proces

Plánovač systému je spuštěn voláním funkce `vTaskStartScheduler()`. V této funkci je nejprve inicializován defaultní proces Idle, poté je po zakázání všech přerušení jádra volána funkce `xPortStartScheduler()` portable vrstvy. Tato funkce inicializuje systémový časovač SysTick a spouští první proces. Programové řízení je nadále předáno plánovači procesů.

Kapitola 6

Návrh modelové aplikace s podporou FreeRTOS

V této kapitole bude popsána realizace projektu využívajícího platformu FreeRTOS jako základní programovací nástroj pro tvorbu embedded aplikací. Předmětem projektu je nárvh reálného embedded systému a jeho implementace pro hardware μLab, jehož technický popis byl předložen v kapitole 4. Pomocí řešení praktické úlohy - jednotka digitálních hodin - budou demonstrovány aspekty návrhu aplikací v systému FreeRTOS.

Úvodní část kapitoly definuje zadání projektu. Následuje popis hardwarových komponent, ze kterých se bude fyzická realizace skládat, a uživatelský popis výsledné aplikace. V dalším textu bude proveden popis aplikační architektury s podporou FreeRTOS a tento bude porovnán s hypotetickou konvenční implementací z hlediska náročnosti implementace, přehlednosti kódu, rozšiřitelnosti a možnosti údržby. V závěru kapitoly bude provedená práce stručně shrnuta.

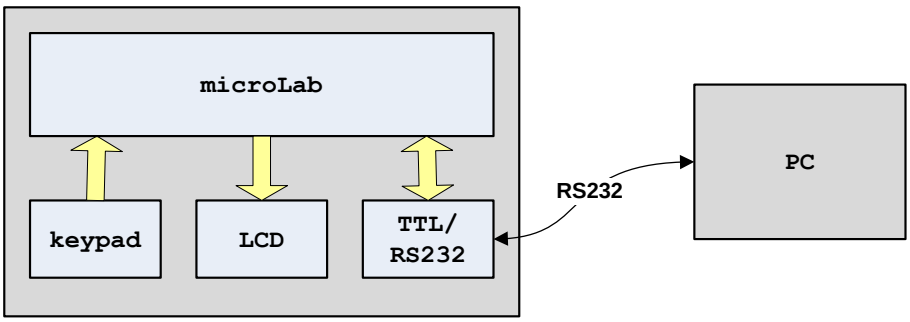
6.1 Zadání projektu

Návrh a realizace řídící jednotky digitálních hodin s procesorem STM32. Systém bude vybaven maticovou klávesnicí a LCD panelem pro zadávání dat a jejich zobrazení. Hodiny jsou uživatelsky nastavitelné prostřednictvím maticové membránové klávesnice a aktuální čas je zobrazen na připojeném LCD panelu. Systém zároveň umožní uživatelský přístup (odečítání aktuálního času a stavových informací systému) pomocí počítače prostřednictvím rozhraní RS232.

6.2 Projektová studie

6.2.1 Hardwarové komponenty

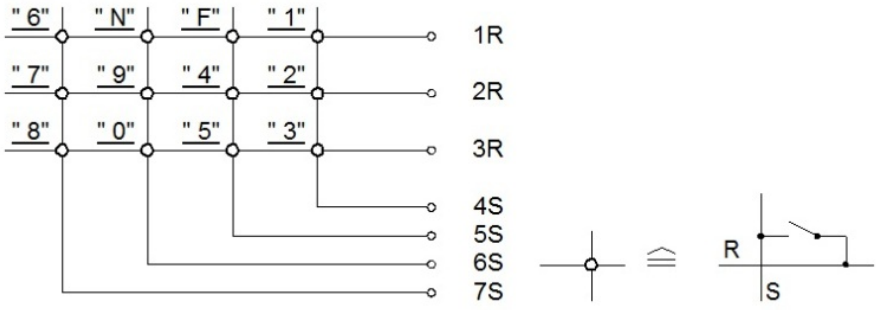
Blokové schéma systému je znázorneno na obrázku 6.1. Řídící jednotku tvoří vývojový modul μLab osazený mikrořadičem STM32F105RBT6. Tento modul slouží k obsluze vstupně-výstupních periferních zařízení (klávesnice, LCD) a realizuje datovou komunikaci po RS232.



Obrázek 6.1: Blokové schema jednotky digitálních hodin

Vstupní periferie

Pro zadávání údajů uživatelem slouží panelová maticová klávesnice se znaky ’0’ až ’9’ a dále ’N’ a ’F’ (viz. obrázek 6.3). K obvodu se připojuje sedmi vodiči (spínače odpovídající klávesám jsou vnitřně uspořádány do matice o rozměru 3x4 prvků viz. schéma zapojení na obrázku 6.2).



Obrázek 6.2: Elektrické schéma klávesnice (prevzato z [14])

Obrázek 6.3: Membránová klávesnice (převzato z [14])

Výstupní periferie

Výstup systému je zobrazován na LCD panelu WM-C1602N (2x16 znaků) s integrovaným řadičem HD44780. Displej je napájen z externího zdroje stejnosměrného napětí 5 V. K procesoru bude připojen sedmi vodiči: čtyřmi datovými pro multiplexovaný prenos datových bytů a třemi řídícími signály (RS, R/W a E). Podrobné informace o ovládání displeje jsou v [11] a v [12].

Datové rozhraní

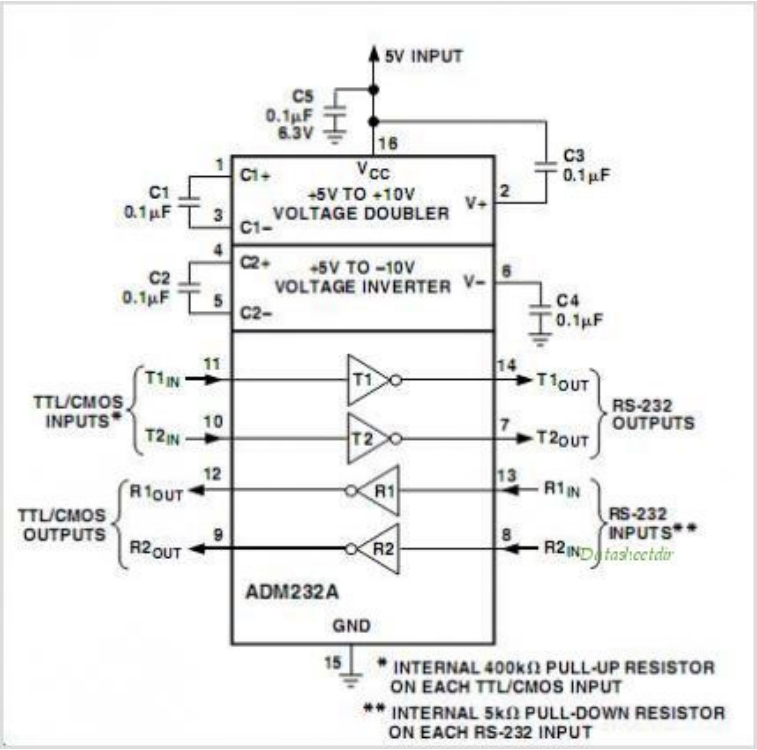
Pro datovou komunikaci je využito rozhraní RS232 a periferie mikrořadiče USART3, jejíž datové linky (TX, RX) jsou vyvedeny na piny PB10 a PB11. Konverzi signálu rozhraní RS232 a TTL zajišťuje obvod ADM232 v katalogovém zapojení dle obrázku 6.4, podrobnosti lze nalézt v [13].

6.2.2 Uživatelský popis přístroje

Na LCD panelu přístroje bude přehledně zobrazen aktuální čas ve 12-ti nebo 24-hodinovém formátu společně se stručnou legendou popisující funkci tlačítek pro intiuitivní ovládání přístroje připojenou klávesnicí.

Ovládání pomocí klávesnice

Stiskem tlačítka 'F' bude možno přepínat formát zobrazení aktuálního času. Stisk tlačítka 'N' provede vstup do módu nastavení času. V tomto módu uživatel postupně zadá požadovaný čas. Pro přehlednost nastavení bude aktuálně nastavovaná pozice blikat. Systém nedovolí uživateli zadat nekorektní čas.



Obrázek 6.4: Schéma zapojení prevodníku úrovní TTL/RS232 [13]

Vzdálený přístup

K systému bude možné se připojit vzdáleně postřednictvím rozhraní RS232 a vyčítat hodnoty aktuálního času. Toto bude provedeno odesláním znaku 't' nebo 'T' do zařízení. Systém zároveň bude po tomto rozhraní odesílat stavové informace.

Pro komunikaci s přístrojem je možno využít některého volně dostupného RS232 terminálu pro PC, například programu **AccessPort** nebo **Free Serial Port Terminal**. Požadované parametry připojení jsou uvedeny v tabulce 6.1.

Přenosová rychlost	9600 Bd
Počet datových bitů	8
Počet stop bitů	1
Parita	–

Tabulka 6.1: Nastavení parametrů UART pro komunikaci s jednotkou digitálních hodin

6.3 Popis realizace jednotky digitálních hodin

6.3.1 Dekompozice úlohy

Při návrhu aplikace je třeba řešený problém vhodně dekomponovat, tj. rozdělit jej na množinu podproblémů, které lze řešit odděleně. Tyto podproblémy potom typicky lze implementovat pomocí nezávislých výpočetních procesů systému FreeRTOS. Úlohu definovanou zadáním 6.1 lze rozčlenit na následující základní komponenty:

- **Obsluha klávesnice**
  - provádí mechanismus detekce a čtení stisknutých kláves
- **Generátor reálného času**
  - pro generování času je možné využít přerušení generované periferií RTC (Real Time Clock)
- **LCD zobrazovač**
  - přehledné zobrazení údajů
- **Datové rozhraní**
  - obsluha komunikace po sériové lince (RS232)

Obsluha klávesnice

Stav klávesnice musí být periodicky ověřován tak, aby sytém korektně reagoval na zásahy uživatele. V případě, že je například frekvence čtení stavu klávesnice nízká, získá uživatel pocit, že zařízení nefunguje spolehlivě a má tendenci klávesy ntisknout nepřiměřenou silou.

Řídící program bude periodicky ověřovat stav klávesnice v intervalu 10ms. Pro odstranění zákmitů tlačítka vzniklých odskoky spínaných kontaktů bude stav vyhodnocen jako aktivní pokud bude alespoň trojice po sobě jdoucích skanů klávesnice shodná.

Klávesnice bude připojena k pinům PA1 až PA7.

Generátor reálného času

Generátor reálného času je klíčovým subsytémem. K jeho realizaci bude využita periferie RTC (Real Time Clock) mikrořadiče. Systémový čas bude aktualizován s vysokou prioritou na základě přerušení od RTC.

LCD zobrazovač

Ovladač LCD poskytne potřebné API všem ostatním komponentám systému, jejichž výstup je zobrazován na displeji. Pro jeho obsuluhu platí podobné principy, jaké byly popsány v části 6.3.1 k obsluze klávesnice.

Datové rozhraní

Komunikace prostřednictvím RS232 není založena na periodické bázi a systémové zdroje nebude příliš zatěžovat. Priorita její obsluhy v porovnání s real-time komponentami (RTC, klávesnice) může být nižší.

6.3.2 Přístupy k řešení úlohy

Konvenční způsob programování

Výše popsaný problém lze řešit konvenčním způsobem pomocí cyklického vykonávání programu v nekonečné smyčce a obsluze zdrojů přerušení v příslušných přerušovacích rutinách.

Přerušovací rutina RTC, která se periodicky vykonává v intervalu 1 s musí obsahovat kód pro aktualizaci systémových času a jeho zobrazení na displeji. Obsluhu klávesnice realizuje časovač, v jehož obslužné rutině se provádí detekce stisku klávesy a případně její zpracování. Celá aplikace pak funguje jako stavový automat, jehož přechody jsou prováděny v reakci na konkrétní uživatelský vstup. Majoritní část programu je implementována v ISR (Interrupt Service Routine).

Výhody:

- Malá velikost zdrojového souboru
- Nízké nároky na výpočetní zdroje (RAM)
- Stačí znalost použitého μC bez dalších technogií

Nevýhody:

- Obtížná přenositelnost kódu (těžko lze oddělit aplikační kód od kódu přímo vázaného na hardware)

- Komplikovaná údržba aplikace (například provádění servisních operací s časovým odstupem)
- Problémy při přidávání další funkcionality
- Selhává s rostoucí složitostí systému (kritické časovací požadavky)

Aplikace s podporou FreeRTOS

Aplikace v operačním systémeu FreeRTOS je tvořená množinou paralelních procesů. Každému procesu je při jeho inicializaci přidělena priorita, na jejímž základě preemptivní plánovač systému rozděluje procesorový čas.

Řešení zadané úlohy spočívá v rozvržení funkcionality mezi procesy a přidělení odpovídajících priorit.

Výhody:

- Flexibilní návrh odstraňuje nevýhody konvenčního řešení
- Preempce procesů je z pohledu vývojáře transparentní
- Procesům, které nejsou připraveny k běhu (např. čekají na data) není přidělen procesorový čas, což vede k jeho úspoře (*Event-driven approach*)

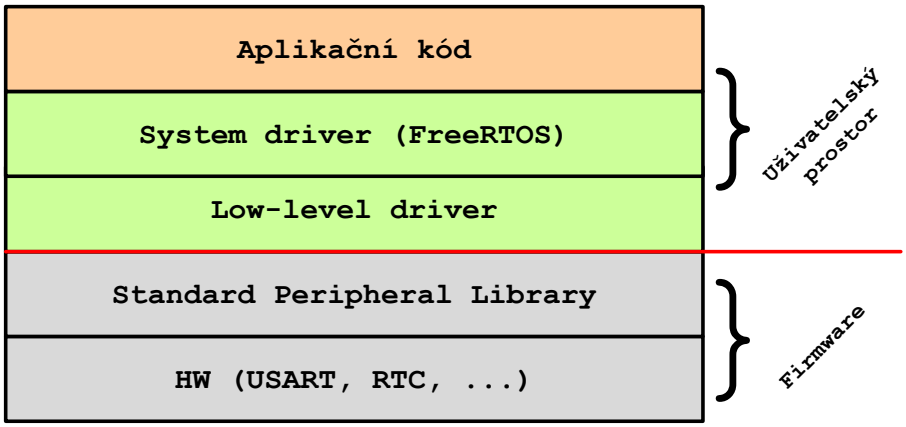
Nevýhody:

- Využití operačního systému nutně přináší určité nároky na systémové zdroje (*overhead*)

6.3.3 Architektura programu

Pro všechny použité vnitřní i vnější periferie (RTC, LCD, USART, klávesnice) byly vytvořeny nízko-úrovňové ovladače (drivery) poskytující API pro snadnou inicializaci a programový přístup k jejich funkcím. Nad nimi byla vytvořena systémová vrstva, která reprezentuje aplikační rozhraní. Kód na této úrovni proto neprovádí přímé zásahy do řízení periferií, ale obsahuje pouze aplikační logiku. Jednotlivé úrovně (vrstvy) softwarové realizace dokládá obrázek 6.5.

Popsaný způsob strukturování kódu aplikace má následující příjemnou vlastnost. Změna zdrojového kódu některé z vrstev (například přechod na novou verzi knihovny



Obrázek 6.5: Uspořádání vrstev zdrojového kódu aplikace

Standard Peripheral Library) vynucuje provedení odpovídajících úprav pouze ve vrstvě bezprostředně nad ní. Pro vyšší vrstvy software je tato změna transparentní.

6.3.4 Implementační detaily

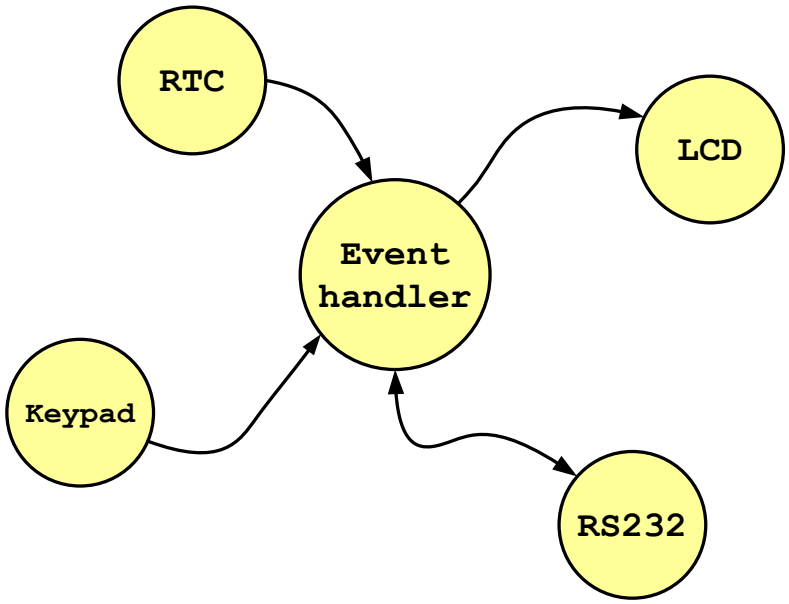
Proces	Priorita	Velikost zásobníku [B]
Event handler	2	1024
RTC	3	128
Keypad	1	128
LCD	1	128
RS232	1	128

Tabulka 6.2: Parametry procesů aplikace (konfigurace priorit)

Aplikaci tvoří pět procesů, jejichž funkce je synchronizována prostředky meziprocesní komunikace (fronty, semafory). Parametry jednotlivých procesů (zejména priorita a velikost paměti alokované pro zásobník, viz. tabulka 6.2) jsou umístěny v souboru `FreeRTOSConfig.h`, který je pro optimálním místem pro jejich definici. Následující programový úsek ilustruje inicializaci procesu pro čtení klávesnice:

```
void Keypad_task_init() {
        /* Configure keypad hardware */
        Keypad_init();

        /* Start LCD task */
        xTaskCreate(Keypad_task, "Keypad_task",
                KEYPAD_TASK_STACK_SIZE, NULL,
                KEYPAD_TASK_PRIORITY, NULL);
}
```

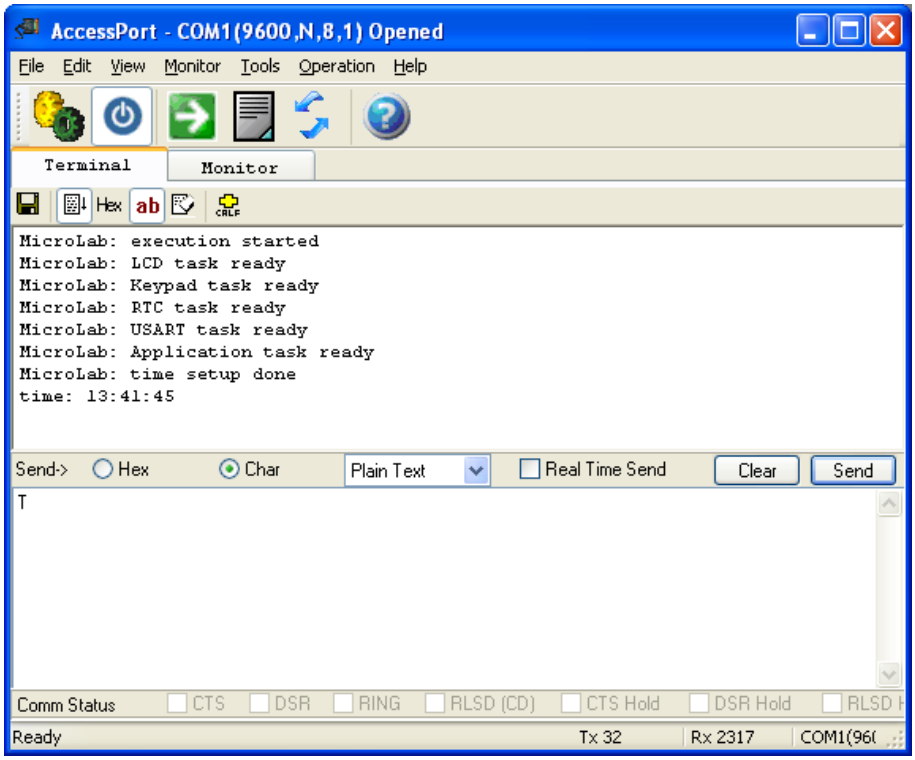


Obrázek 6.6: Orientace datových toků mezi procesy aplikace

Obrázek 6.6 znázorňuje funkční propojení jednotlivých procesů. Kruhy představují aplikační procesy systému FreeRTOS, orientace šipek odpovídá směru meziprocesní komunikace realizované datovými frontami. Jádrem aplikace je proces s označením `Event handler`, který zpracovává události (*events*) generované procesy `Keypad`, `RTC` a `RS232`. Tyto události přijímá na společné frontě. V případě, že je fronta prázdná, je `Event handler` blokován a je mu odebráno řízení. Zdrojový kód procesu `Event handler` vypadá následovně:

```
static void Application_task(void *pvParmeters) {
        AppEvent_typedef event_id;
        while (1) {
                if (xQueueReceive(event_queue,
                                &event_id, portMAX_DELAY) == pdTRUE) {
                        /* Process event */
                        event_processor(event_id);
                }
        }
}
```

Jednotka digitálních hodin s připojenými periferiemi byla realizována na kontaktním poli. Jádro systému představuje modul μLab popsaný v kapitole 4. Obrázek 6.7



Obrázek 6.7: Okno terminálu se záznamem výstupu sériové komunikace

znázorňuje okno sériového terminálu s datovým výstupem modulu.

Kapitola 7

Implementace distribuovaného systému

Kapitola obsahuje popis realizace distribuovaného systému představeného v kapitole 2, složeného z komponent popsaných v předchozím textu práce.

Nejprve bude popsána realizace hardwarové části řešení. Následuje rozbor softwarové implementace systému, skládající se z části popisující programovou implementaci použitých embedded zařízení a části popisující uživatelské rozhraní systému. Závěr kapitoly obsahuje výstupy měření a experimentů, kterými byla funkce systému ověřena.

7.1 Hardware systému

Jako hardwarová platforma pro realizaci uzlů distribuovaného systému byl zvolen procesorový modul s podporou rozhraní Ethernet navržený v Laboratoři videometrie Katedry měření. Charakteristické vlastnosti lze shrnout následovně:

- μC STM32F207VGT6
- časování krystalovým výbrusem 25 MHz
- rozhraní Ethernet, USB 2.0
- programovací rozhraní JTAG, SWD
- uživatelsky přístupné špičky mikrořadiče
- napájení externím zdrojem, nebo přes USB/Ethernet rozhraní

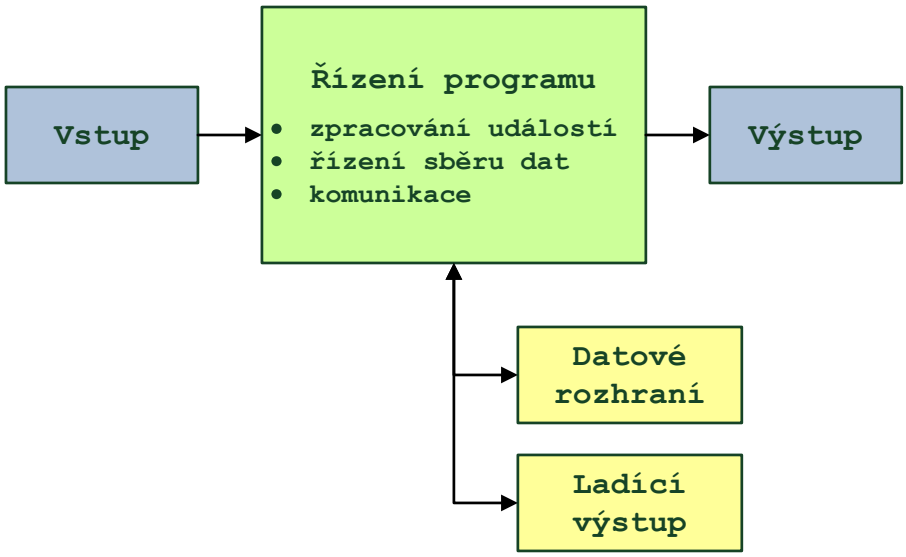
- kompaktní rozměry

Pro potřeby ověření funkce implementovaného software bylo provedeno ruční osazení dvojice desek plošných spojů. Podrobné informace týkající se použitého modulu lze nalézt v Bakalářské práci Adama Bařtipána [15].

Dvojice modulů společně s osobním počítačem tvoří síť LAN. Centrálním prvkem sítě je ethernetový switch (konkrátně model **CQ point CQ-C005**).

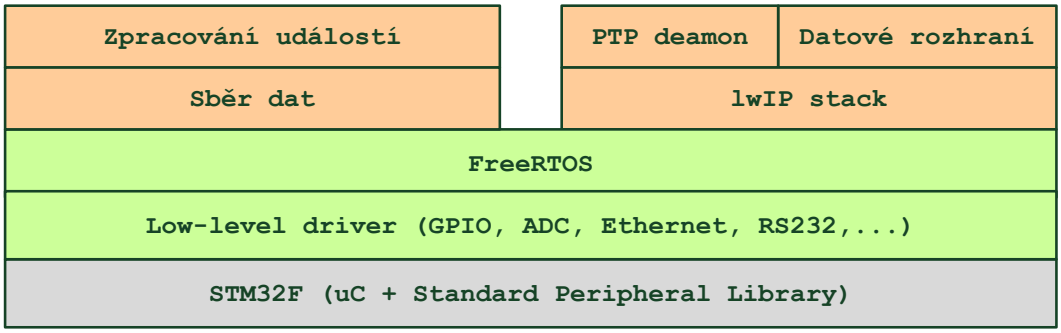
7.2 Software modulů pro sběr dat

7.2.1 Operační struktura modulu



Obrázek 7.1: Blokové schéma modulu pro distribuovaný sběr dat

Blokové schéma modulu je na obrázku 7.1. Vstupem je integrovaný AD převodník, který slouží pro měření napětí v rozsahu 0-3,3 V (referenční napětí ADC periferie procesoru je v případě použitého modulu připojeno k napájecímu rozvodu). Výstup tvoří LED osazená na desce modulu určená k indikaci stavu systému (LED svítí v případě Master zařízení). Datové rozhraní realizuje obousměrnou komunikaci v síti Ethernet, pro servisní účely slouží výstup na rozhraní RS232. Představu o struktuře navrhovaného softwaru poskytuje obrázek 7.2. V následujícím textu budou popsány implementační detaily jednotlivých jeho částí.



Obrázek 7.2: Struktura vrstev zdrojového kódu modulu pro distribuovaný sběr dat

7.2.2 Inicializace mikrořadiče

Základní inicializace μC je provedena ve funkci `SystemInit()` (`system_stm32f2xx.c`). Tabulka 7.1 uvádí konfigurované hodnoty taktovacích frekvencí jednotlivých hodinových

Hodinový signál	frekvence [MHz]
SYSCLK	120
HCLK	120
PCLK1	30
PCLK2	60

Tabulka 7.1: Nastavení taktovacích frekvencí hodinových sigbálů mikrořadiče

signálů.

7.2.3 Implementace ovladače PTP

Síťovou komunikaci zajišťuje LwIP stack. Implementace lwIP verze 1.3.2 pro mikrořadiče stm32F2x7 a operační systém FreeRTOS je k dispozici ke stažení na stránkách společnosti STMicroelectronics. Součástí této implementace je knihovna funkcí pro obsluhu ethernertové periferie. Tato však nepodporuje operace se subsystémem synchronizace PTP (snímání časových značek při příjmu a odeslání paketů, konfigurace PTP apod.). Pro zajištění funkce kódu projektu PTP deamon (viz. kapitolu 3) bylo potřeba implementovat ovladač pro práci s PTP na úrovni hardware a provést určité změny v implementaci lwIP ([4], [6]).

Nastavení taktování systémového času PTP

PTP subsytém je časován referenční frekvencí sběrnice AHB (HCLK) f_{HCLK} = 120 MHz. Soubor obsahuje inicializační parametry `ADJ_FREQ_BASE_ADDEND` a `ADJ_FREQ_BASE_INCREMENT` pro nastavení hodnoty registru ETH_PTPTSAR (time stamp addend register) a registru ETH_PTPSSIR (subsecond increment register). Postup jejich výpočtu je následující:

$$t_{increment} = 20ns \Rightarrow f_{increment} = 50MHz$$

$$ETH\_PTPSSIR = (2^{31} \times t_{increment})/10^9 = 42,9496 \doteq 43d = 2Bh$$

$$ETH\_PTPTSAR = 2^{63}/(f_{HCLK} \times ETH\_PTPSSIR) = 1787475200 = 6A8AB500h$$

Vypočené hodnoty jsou uloženy v souboru stm32f2x7_PTP_driver.h.

Funkce pro obsluhu PTP subsytému

Ovladač PTP tvoří soubor `stm32f2x7_PTP_driver.c`. Tento obsahuje funkce pro přístup k PTP registrům a jejich nastavení, funkce pro operace se systémovým časem a konfiguraci frekvence synchronizačního pulzu PPS.

PTP subsytém je inicializován voláním funkce `ETH_PTPStart(uint32_t UpdateMethod)`, kde parametr `UpdateMethod` může nabývat jedné z hodnot

- textbfETH_PTP_FineUpdate: Fine Update method
- textbfETH_PTP_CoarseUpdate: Coarse Update method

V popisované aplikaci je funkce volána při inicializaci lwIP stacku ve funkci `low_level_init(struct netif *netif)` a je použita časová korekce *Fine Update Method*, která umožňuje dynamickou korekci systémového času, při které nedochází k výrazným skokovým změnám v nastavení systémového času v průběhu synchronizace ([4]).

Ve funkci `low_level_init(struct netif *netif)` je provedena také inicializace RX/TX deskriptorů pro zpracování rámců ethernetovou periferií μC:

```
/* Initialize Tx Descriptors list: Chain Mode */
#if LWIP_PTP
ETH_DMAPTPTxDescChainInit(DMATxDscrTab,&Tx_Buff[0][0],ETH_TXBUFNB);
#else
ETH_DMATxDescChainInit(DMATxDscrTab,&Tx_Buff[0][0],ETH_TXBUFNB);
#endif
/* Initialize Rx Descriptors list: Chain Mode */
/* Rx Descriptors list initialization is the same for PTP
 and non-PTP apps */
ETH_DMARxDescChainInit(DMARxDscrTab,&Rx_Buff[0][0],ETH_RXBUFNB);
```

Definice funkce `ETH_DMAPTPTxDescChainInit()` je v souboru `stm32f2x7_PTP_driver.c`, funkce inicializuje řetězec Enhanced DMA TX deskriptorů s podporou PTP timestamps pro odesílané rámce. Definice konstanty `LWIP_PTP` byla přidána v konfiguračním souboru lwipopts.h.

Problematická funkce přerušení PTP

Při pokusu o využití funkce přerušení *Time Trigger Interrupt* se objevil problém. První vyvolání tohoto přerušení proběhlo podle očekávání. V přerušovací rutině ale program následně uvízl. Odhalení chyby bylo věnováno poměrně značné úsilí, řešení nakonec odhalil Ing. Jan Breuer z Katedry měření.

Problém je způsoben nepřesností v referenčním manuálu [4]. Zde je uvedeno, že příznak přerušení TSTS je automaticky nulován jeho čtením v registru ETH_MACSR. Tato operace k jeho nulování však nestačí, je třeba přečíst registr ETH_PTPTSSR.

7.2.4 Úpravy lwIP stacku a jeho inicializace

Hondoty systémového času zachycené při příjmu a odeslání rámců (timestamps) jsou zprostředkovány vyšším vrstvám pomocí přidaných parametrů struktury `pbuf`.

```
#if LWIP_PTP
/**
 * a field that contains the timestamp that this packet was received.
 */
s32_t time_sec;
s32_t time_nsec;
#endif
```

7.2.5 Funkce senzoru

Pro realizaci funkce senzoru je využit 12-bitový ADC převodník obsažený na čipu mikrořadiče. ADC může být využit k analýze a zpracování spojitých signálů.

Konfigurace ADC

Propojení vstupu převodníku s konkrétní vstupně-výstupní linkou μC (GPIO) je dáno pre-compile konfigurací (ADC_driver.h):

```
#define ADC_PORT            GPIOC
#define ADC_PIN             GPIO_Pin_0

#define ADC_ENGAGED         ADC1
#define ADC_PERIPH_ENGAGED  RCC_APB2Periph_ADC1
#define ADC_CHANNEL         ADC_Channel_10
#define ADC_GPIO_PORT       RCC_AHB1Periph_GPIOC
```

Vstup ADC je připojen ke špičce PC0 a je konfigurován jako 12-ti bitový v Single Conversion módu. Sběrnice APB2 je taktována na frekvenci $f_{PCLK2} = 60$ MHz, prescaler hodinového signálu ADCCLK (časování analogové části ADC) je nastaven na hodnotu 2. Frekvence časování ADCCLK je proto $f_{pclk2}/2 = 30$ MHz, což je její maximální hodnota, při které je doba konverze převodníku $t_c = 0.5\mu$s. Konfiguraci ADC zajišťuje kód v souboru ADC_driver.c.

Aplikační rozhraní senzoru

Aplikační vrstva softwaru (senzor) přistupuje k výstupu ADC pomocí volání funkce `scan_ADC()` (ADC_driver.h), které způsobí konverzi a vrátí hodnotu odměru.

```
uint16_t scan_ADC() {
        uint16_t ad_val;

        /* Perform AD conversion */
        ADC_SoftwareStartConv(ADC_ENGAGED);

        /* Wait until AD conversion is complete */
        while (ADC_GetFlagStatus(ADC_ENGAGED, ADC_FLAG_EOC) == RESET)
                ;

        /* Clear EOC flag */
        ADC_ClearFlag(ADC_ENGAGED, ADC_FLAG_EOC);

        /* Process converted value */
        ad_val = ADC_GetConversionValue(ADC_ENGAGED);
        return ad_val;
}
```

7.2.6 Výstup stavových informací

Software modulu odesílá stavové informace ve formě znakových řetězců po sériové lince (RS232) do počítače. Toho lze s výhodou využít zejména při ladění programu.

Komunikaci zajišťuje periferie USART1, vyvedená na špičkách PA9 (TX) a PA10 (RX), toto lze konfigurovat obdobným způsobem jako v 7.2.5.

Obslužný program využívá pro ukládání odesílaných zpráv kruhový buffer o velikosti 200 Bytů.

Parametry sériového rozhraní

Tabulka 7.2 uvádí nastavení sériové komunikace pro příjem stavových informací na sériovém portu.

Přenosová rychlost	115200 Bd
Počet datových bitů	8
Počet stop bitů	1
Parita	–

Tabulka 7.2: Nastavení parametrů UART pro komunikaci s moduly distribuovaného systému

7.2.7 Datová komunikace a zpracování událostí

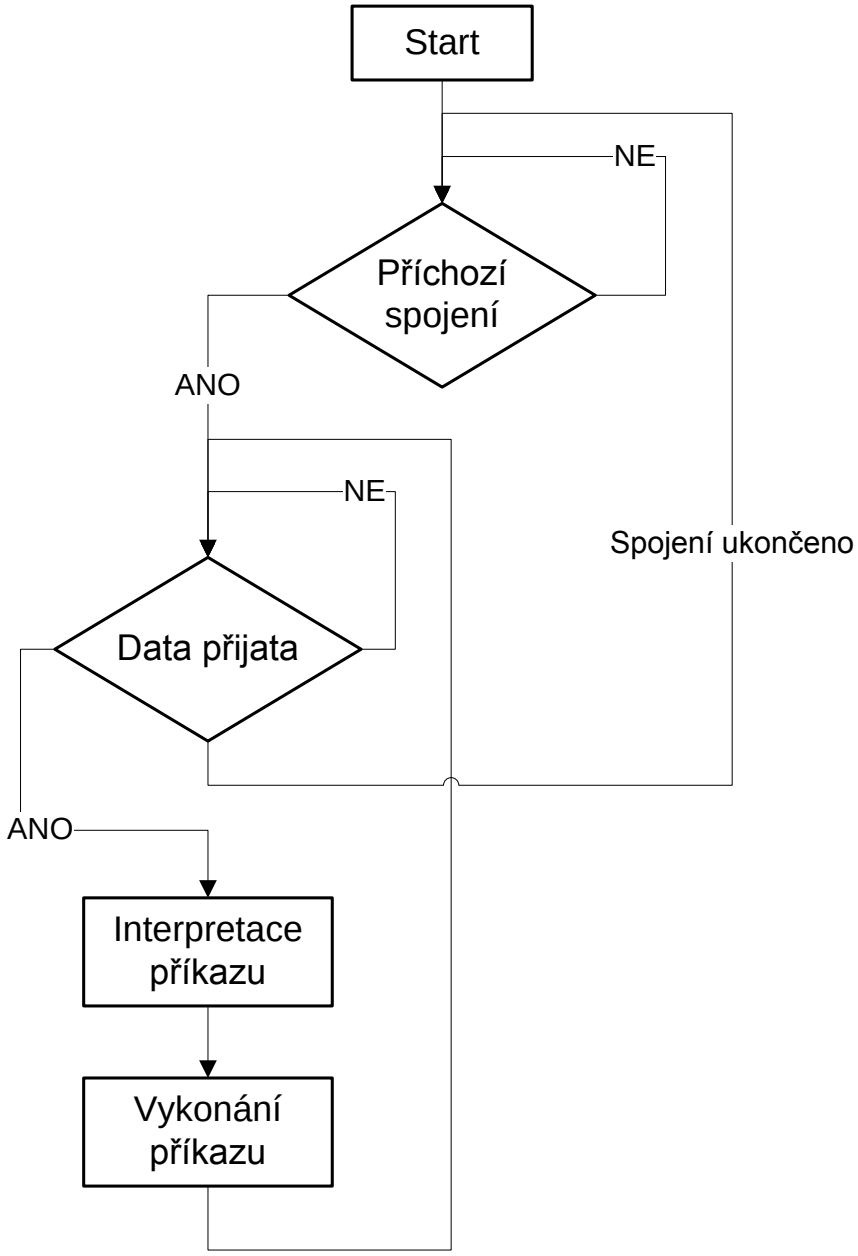
Popis funkce

Datová komunikace je realizována TCP spojením pomocí protokolu TELNET. Zařízení realizuje funkci serveru, ke kterému je možné připojit se pomocí aplikace běžící na vzdáleném stroji a řídit jeho funkci zasíláním požadavků (query). Tyto jsou zařízením interpretovány a následně jsou provedeny odpovídající akce. Princip zpracování příkazů znázorňuje vývojový diagram 7.3.

Datová komunikace, interpretace a vykonání přijatých požadavků jsou realizovány jako proces systému FreeRTOS. Síťové spojení a jeho správa bylo implementováno s použitím NETCONN API, které je součástí implementace lwIP stacku ([21]). NETCONN je abstrakce síťového spojení určená pro použití v prostředí operačního systému.

Sada podporovaných příkazů

- ***IDN?**: identifikace zařízení (inspirováno protokolem SCPI)
- **TIME?**: požadavek na systémový čas zařízení
- **PTP_STATE?**: požadavek na PTP status (Master/Slave)
- **GRAB**: požadavek na vyslání naměřených dat (pokud jsou k dispozici)
- **PPS <ex>**: Nastavení frekvence pulzu PPS, parametrem je číslo $ex \in < 0, 15 >$ a platí, že $f_{PPS} = 2^{ex}$.



Obrázek 7.3: Diagram obsluhy síťového připojení a zpracování přijatých požadavků

7.3 Uživatelské rozhraní systému

Nástrojem pro uživatelskou komunikaci se sítí modulů a její správu je grafická aplikace **Device Probe** vytvořená v jazyce Java.

7.3.1 Stručný popis funkce

Aplikace funguje jako TCP klient, který se připojuje ke vzdálenému datovému zařízení (modulu) prostřednictvím protokolu TELNET (port 23).

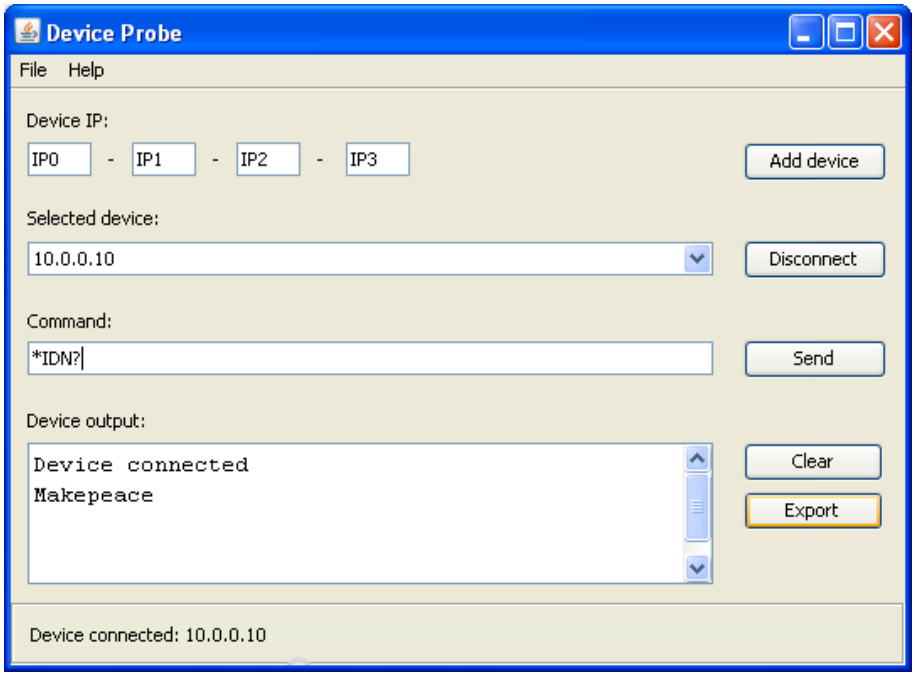
Při navázání spojení je instancován objekt třídy Socket, který představuje síťové spojení a objekty realizující jeho vstupy a výstupy (instance tříd BufferedReader a PrintWriter). Následně je vytvořeno vlákno pro příjem a zobrazení dat od vzdáleného zařízení, kterému je předána instance vstupního proudu.

7.3.2 Ovládání aplikace

Přidání nového zařízení

Přidat zařízení sítě lze provést zadáním IP adresy do čtveřice textových polí a stiskem tlačítka **Add device**. Je-li zadaná IP adresa korektní, je přidána položka v menu **Selected device**, v případě neplatného vstupu je uživatel vyzván k jeho opravě.

Připojení a zadávání příkazů



Obrázek 7.4: Okno uživatelského rozhraní systému (aplikace Device Probe)

Uživatelské rozhraní umožňuje připojení k libovolnému modulu sítě na základě jeho IP adresy vybrané v menu **Selected device** a stisku tlačítka **Connect**.

Připojený modul je ovládán z příkazové řádky **Command** (viz. 7.4). Výčet podporovaných příkazů je uveden v části 7.2.7.

Export dat

Výstup zařízení je možné exportovat do souboru stiskem tlačítka **Export**. Distribuované moduly jsou koncipovány tak, aby jejich datový výstup (výstupy měření) odpovídal formátu `csv` (Comma Separated Values), a mohl tedy být dále počítačově zpracován (například v prostředí MATLAB).

7.4 Ověření funkce systému

7.4.1 Funkce PPS

Pro měření kvality synchronizace jednotlivých uzlů sítě lze využít periodický synchronizační signál PPS (*Pulse Per Second*), který je součástí specifikace standardu IEEE-1588 (viz. [16]), a v případě μC STM32F207VGT6 v pouzdře LQFP100 je vyveden na pinu PB5.

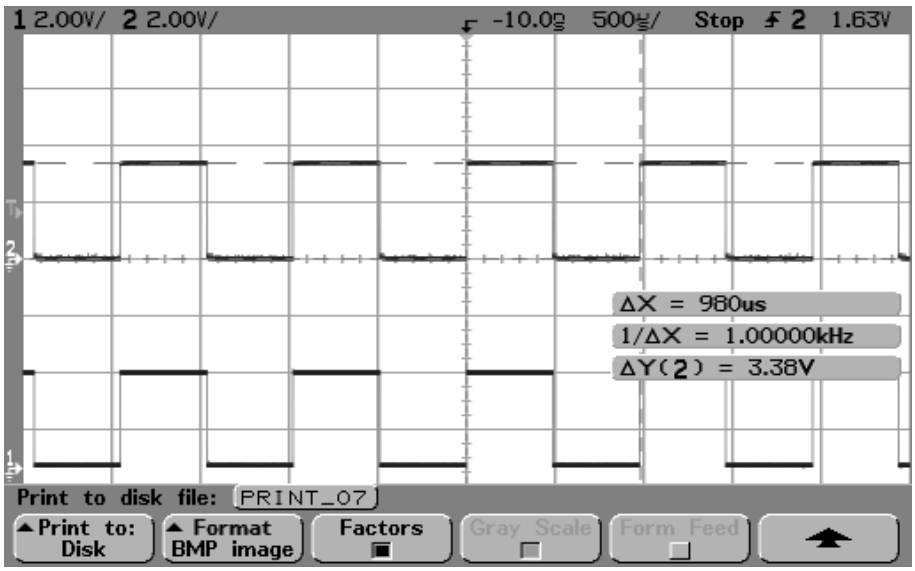
Frekvenci PPS lze programově měnit v nezáporných celočíselných mocninách dvou s nejvyšším přípustným exponentem 15. Šířku pulzu PPS při frekvenci 1 Hz lze konfigurovat na hodnotu 100 ms, resp. 125 ms. Při vyšších frekvencích má signál střídu $0,5$. Všechny potřebné informace týkající se konfigurace a využití PPS lze nalézt v [4].

Konfigurace měření

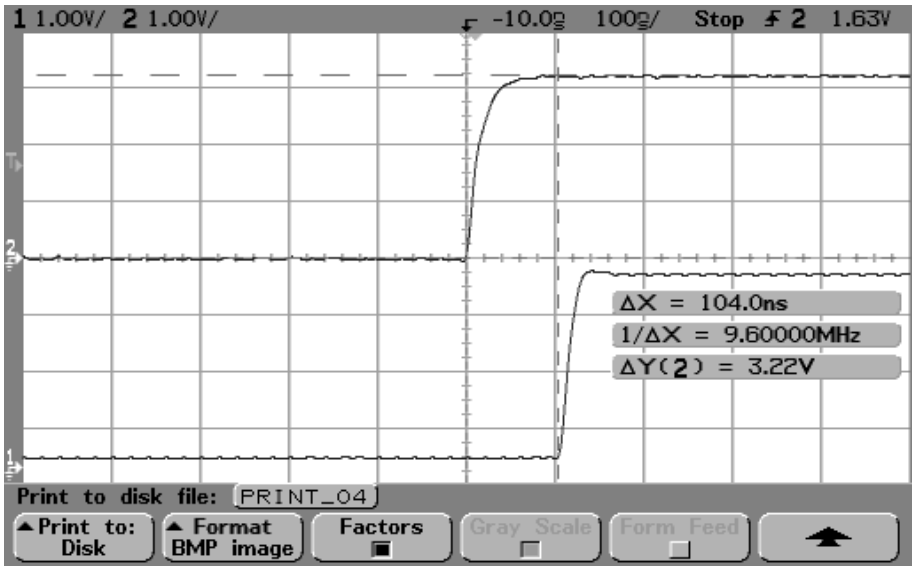
Průběh PPS generovaný dvojicí synchronizovaných modulů (Master a Slave zařízení), byl zaznamenán digitálním osciloskopem. Frekvence pulzu byla pro oba moduly shodně nastavena na 1 kHz. Moduly byly propojeny propojeny pomocí ethernetového switche **CQ point CQ-C005**.

Výstup měření

Dvojici průběhů ukazuje obrázek 7.5). Při dostatečném přiblížení lze měřením pozorovat velikost časového odstupu náběžných hran signálů. Situace je zachycena na obrázku 7.6). Během měření se offset pohyboval v řádu desítek až stovek ns.



Obrázek 7.5: Průběhy signálů PPS generované dvojicí synchronních modulů



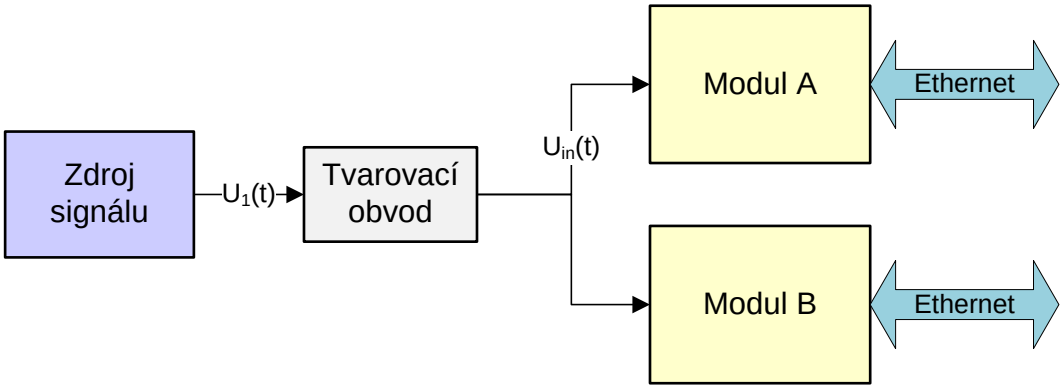
Obrázek 7.6: Časový odstup (jitter) náběžných hran dvojice PPS pulzů

7.4.2 Dvoukanálový sběr dat

Předmětem exeprimentu je simulace synchronního sběru dat distribuovaným systémem.

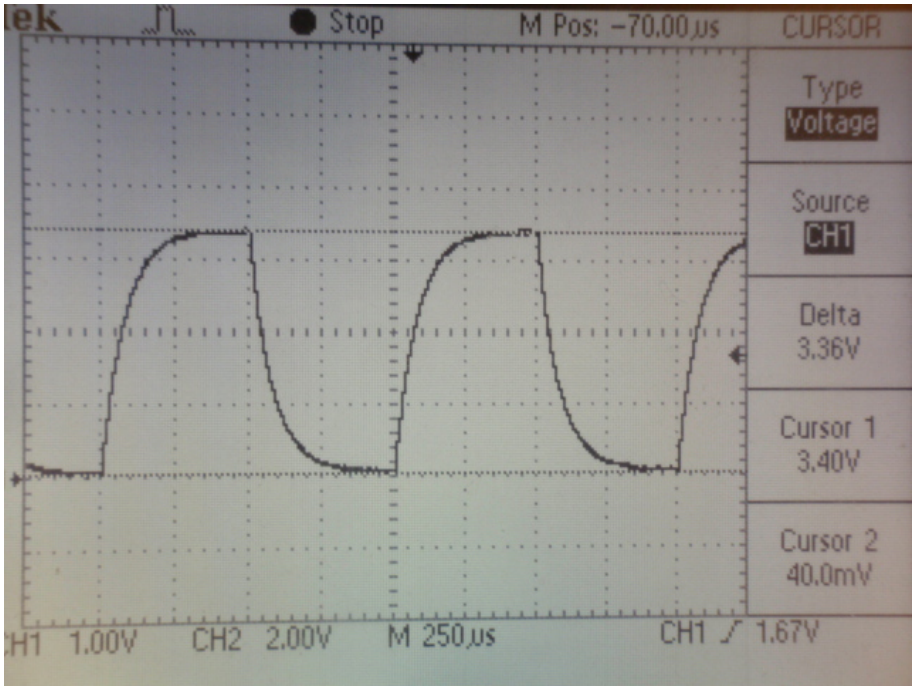
Konfigurace experimentu

Strukturu systému ilustruje schéma 7.7. Jako zdroj signálu je použit modul μLab (popsaný v kapitole 4). Modul generuje obdélníkový signál $U_1(t)$ o frekvenci $f = 1$ kHz se



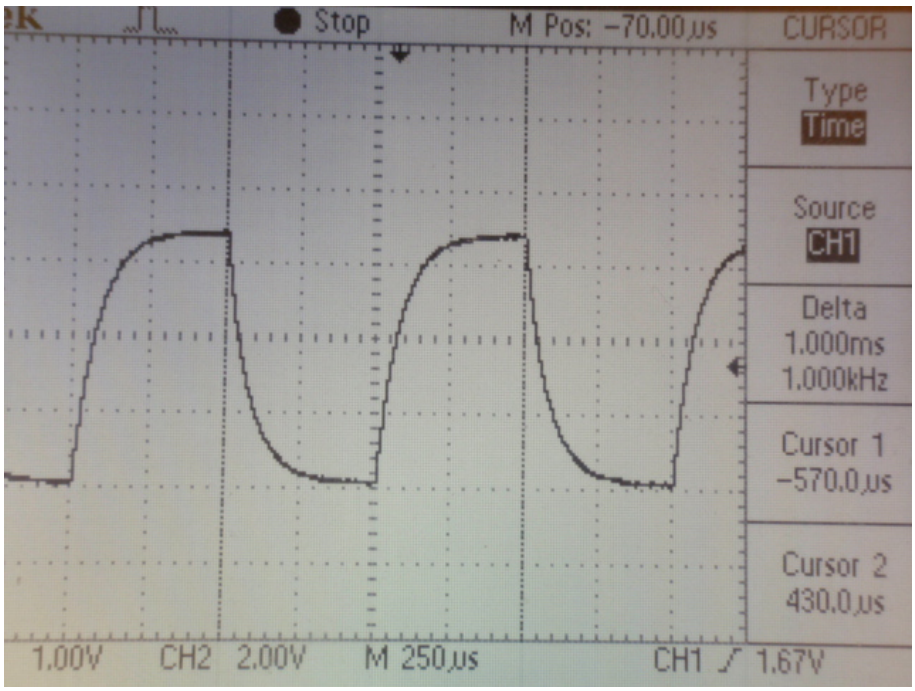
Obrázek 7.7: Blokové schéma distribuovaného systému pro dvoukanálový sběr dat

střídou 1/2. $U_1(t)$ je tvarován integračním článkem tvořeným sériovou dvojicí rezistoru



Obrázek 7.8: Amplituda signálu $U_{in}(t)$ (měřeno osciloskopem Tektronix TDS 210)

R = 1k0 a kapacitoru C = 100 nF. Výstupní signál tvarovače $U_{in}(t)$, který je vzorkován dvojicí modulů synchronizovaných pomocí PTP, je na obr. 7.8 a 7.9. Perioda vzorkování signálu je 10 μs.



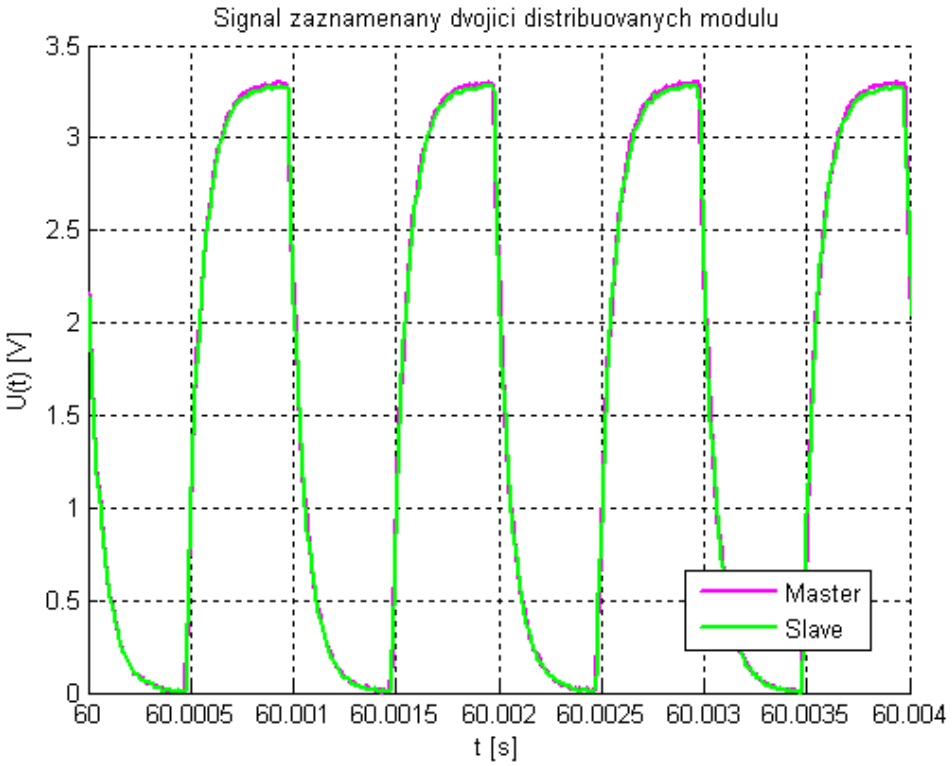
Obrázek 7.9: Frekvence signálu $U_{in}(t)$ (měřeno osciloskopem Tektronix TDS 210)

Výstup měření

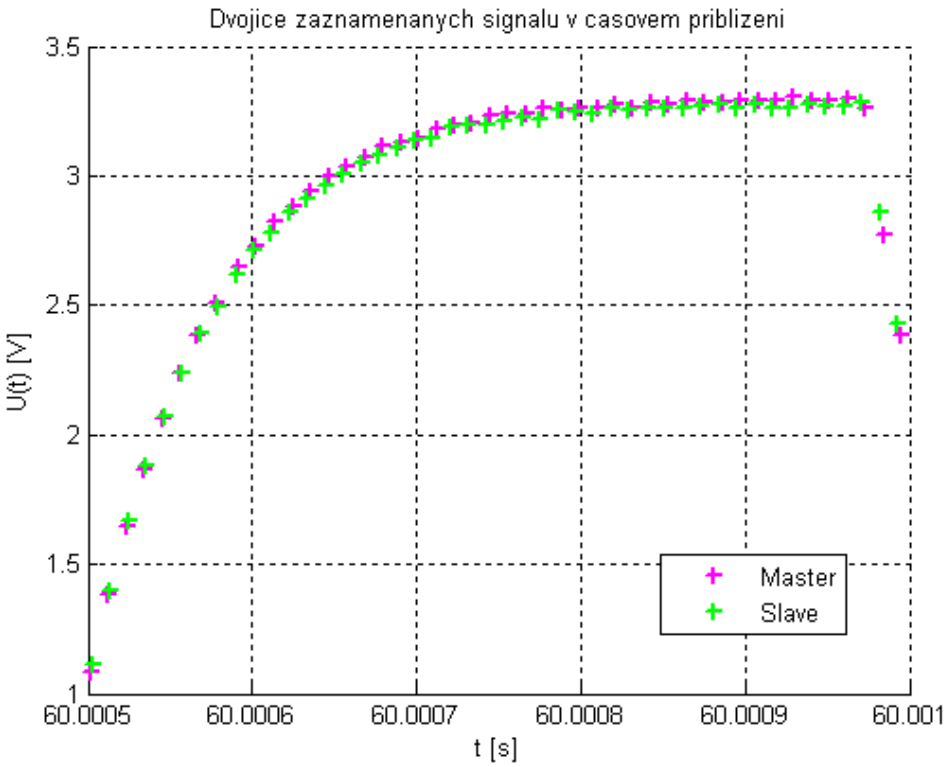
V grafu 7.10 byly průběhy signálu $U_{in}(t)$ změřené oběma moduly v čase 60 s položeny na společnou časovou osu. Naměřená data byla stažena do počítače pomocí aplikace **Device Probe** (7.3) a dále zpracována pro zobrazení pomocí skriptu v prostředí MATLAB. Kvalitu synchronizace, jak ukazuje graf 7.11 je možné snáze pozorovat při větším časovém přiblížení. V obrázku je časová osa rozdělena na dílky odpovídající časovým intervalům 100μs. Zejména na strmější části průběhu je dobře vidět, že odstup obou signálů (jitter) se pohybuje v řádu jednotek, nanejvýš desítek μs.

7.4.3 Analýza vývoje časového odstupu (jitter)

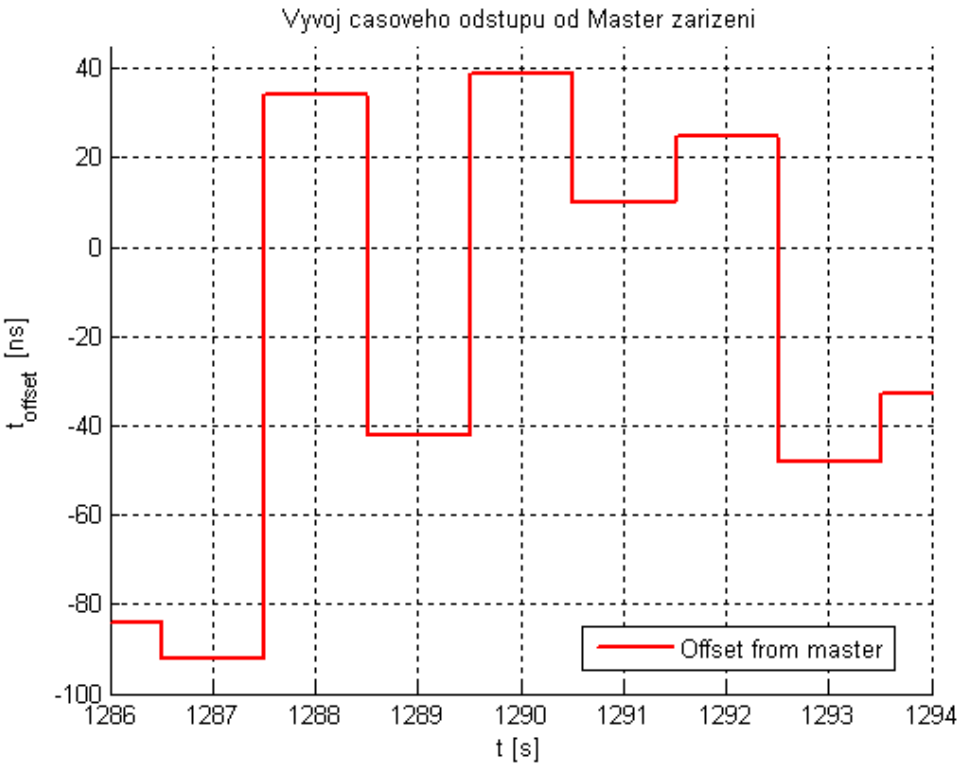
V časovém intervalu 10 s byl proveden záznam hodnot jitteru (*Offset from Master*). Hodnoty byly zaznamenávány s periodovou 10 ms, perioda záznamu ovšem v tomto případě není kritická, protože Master zařízení vysílá synchronizační zprávy s periodou 1 s, a zaznamenané hodnoty v rámci tohoto intervalu jsou proto konstantní. V měřeném intervalu se, jak ukazuje obrázek 7.12, hodnota jitteru pohybuje v řádu desítek ns. Tyto hodnoty odpovídají výstupu získaném při měření synchronizace PPS pulzů v části 7.4.1.



Obrázek 7.10: Průběh signálu $U_{in}(t)$ změřený dvojicí synchronizovaných modulů modulů



Obrázek 7.11: Dvojice synchronních signálů v časovém přiblížení



Obrázek 7.12: Vývoj časového odstupu dvojice Master a Slave zařízení

Kapitola 8

Závěr

8.1 Shrnutí provedené práce

Práce byla zahájena vývojem kompaktního laboratorního modulu μLab (kapitola 4). Modul byl navržen na dvoustranném plošném spoji a profesionálně vyroben v počtu přibližně dvou desítek kusů. Následně bylo provedeno ruční osazení modulu v počtu tří kusů.

Na funkčním hardwarovém modulu bylo možné začít experimentovat se systémem FreeRTOS a s tvorbou aplikace na něm postavené (její popis je v kapitole 6). K tomu bylo nejprve potřeba vytvořit knihovny funkcí pro správu užitečných hardwarových periferií modulu (obsluha klávesnice, LCD a uživatelské indikační LED, komunikace pomocí rozhraní USART).

Při tvorbě demonstrační aplikace pro FreeRTOS se ukázalo, že zvolená architektura programu skutečně je modulární a umožňuje velmi snadné přidání nové funkcionality a tím obohacení možností vyvíjené aplikace. Datová komunikace RS232 byla totiž implementována jako poslední, aniž by ji dosavadní návrh jakkoli uvažoval. Ze systémového pohledu je však její implementace analogická všem ostatním komponentám aplikační vrstvy.

S časem se projevily některé nedostatky v hardwarovém návrhu modulu a proto byla provedena revize návrhu plošného spoje μLab, v rámci níž byly nedostatky odstraněny. Nová verze plošného spoje byla profesionálně vyrobena v počtu 35-ti kusů, z časových důvodů však nebyla prakticky uplatněna v této práci.

Se znalostí FreeRTOS bylo možno postoupit k implementaci synchonního systému distribuovaných modulů (kapitola 7).

Hardware jednotlivých uzlů sítě je realizován již existujícím procesorovým modulem

s podporou rozhraní Ethernet, jehož plošný spoj byl ručně osazen v počtu dvou kusů. Pro zpřehlednění práce při programování modulů a snížení rizika zkratu, či mechanického poškození byla připravena společná platforma, ke které byly zkonstruované moduly (dvojice ethernetových modulů a modul μLab) připevněny.

Na zhotovené moduly byl následně instalován operační systém FreeRTOS, lwIP stack a software PTP deamon. Korektní funkce programu PTP deamon byla podmíněna implementací low-level ovadače pro práci s PTP subsytémem ethernetové periferie, zároveň byly vytvořeny uživatelské knihovny pro obsluhu užitečných periferií na úrovních operačního systému i hardware (obsluha ADC, USART, uživatelská LED) a dále byla implementována aplikační vrstva systému, která zahrnuje datovou komunikaci v síti Ethernet, zpracování událostí a řízení akcí modulu (viz. část 7.2). Výsledkem je dvojice univerzálních embedded modulů pro distribuovaný sběr dat, podporujících síťovou komunikaci pomocí sady protokolů TCP/IP a využívající synchronizační protokol IEEE-1588.

Poslední fází vývoje byla tvorba uživatelského rozhraní systému (část 7.3). Toto bylo navrženo v jazyce Java jako grafická síťová aplikace komunikující se sítí modulů prostřednictvím protokolu TELNET.

Funkce výsledného systému byla experimentálně ověřena (část 7.4) měřením synchronizace pulzů generovaných jednotlivými moduly systému, taktéž bylo provedeno dvoukanálové měření signálu systémem modulů a analýza dynamických parametrů synchronizace. Data získaná měřením byla pomocí uživatelského rozhraní stažena do počítače, kde byla zpracována pomocí softwarového nástroje MATLAB.

8.2 Zhodnocení dosažených cílů

Cílem této práce byl návrh a realizace synchronního systému pro sběr dat. Postup řešení zadané práce byl shrnut v předchozí části (8.1). Na základě toho lze usoudit, že zadání práce se podařilo splnit a cíle stanovené v části 1.1 byly dosaženy.

8.3 Praktické využití práce

Výsledky této práce lze prakticky uplatnit ve dvou rovinách.

První rovinu tvoří praktická aplikace výstupů práce. Procesorový modul μLab může být nasazen jako vestavěná řídící jednotka při realizaci praktických přístrojových aplikací. K usnadnění jejich implementace může být s výhodou použita vytvořená uživatelská

knihovna. Při návrhu modulu i jeho programového vybavení byl kladen důraz na možnost jeho dalšího využití v jiných projektech. Síťové moduly a jejich systémově orientovaný software mohou být široce využity při realizaci aplikací synchronního sběru dat nebo řízení připojeného systému.

Druhou rovinu možného využití této práce tvoří její uplatnění při vysokoškolské výuce předmětů specializovaných na aplikace embedded systémů. Modul μLab může být využit při realizaci semestrálních prací, jeho výhodou je kompaktní provedení a možnost jeho snadného programování pomocí rozhraní USB. Model distribuovaného systému může sloužit k demonstraci funkce synchronizačního protokolu PTP, ale i pohodlné datové komunikace v síti Ethernet.

8.4 Možnosti navazující práce

Navržený systém je zřejmě možné nadále rozvíjet, jak bylo částečně naznačeno už v předchozí části 8.3.

Přímočarým rozvinutím sytému je jeho možná aplikace při realizaci experimentu názorně demonstrujícího funkci synchronizace PTP. Sympatickým rozšířením systému by byla integrace jeho uživatelského rozhraní s webovým prohlížečem. Prostor pro aplikační rozšíření stávajícího software je také velký - například rozšíření sady podporovaných příkazů a její integrace s některým standardem, popřípadě implementace dalších funcí pro sběr dat.

8.5 Závěrečné slovo autora

Řešením této práce jsem získal množství zkušeností z různých oblastí technické praxe.

Součástí řešení byly rozmanité inženýrské činnosti, například: počítačový návrh hardware a příprava technologických dat pro jeho výrobu, ruční osazování desek plošných spojů se značným množstvím SMD komponent, implementace a ladění softwaru pro jednočipové mikrořadiče rodiny STM32 postavené na jádře ARM Cortex M3, práce s operačním systémem FreeRTOS a současnými síťovými technologiemi, i tvorba uživatelského rozhraní systému pro osobní počítač ve vyšším programovacím jazyce. Při zpracování datových výstupů systému bylo využito prostředí MATLAB.

Lze s potěšením konstatovat, že v průběhu řešení této práce nedošlo v souvislosti s ní k žádnému úrazu, poškození fyzického ani duševního zdraví jejího autora, ani jiné osoby.

Diplomová práce byla pro mě výzvou a jsem presvědčen, že zkušenosti a znalosti nabyté při jejím řešení i v průběhu celého předchozího studia uplatním ve své další elektrotechnické praxi.

Literatura

[1] The definitive Guide to the ARM Cortex- M3. Joseph Yiu Elsevier 2010

[2] STM32F105xx/STM32F107xx Datasheet, revision 2. ST Microelectronics

[3] STM32F205xx/STM32F207xx Datasheet, revision 3. ST Microelectronics

[4] STM32F205xx/STM32F207xx/STM32F215xx/STM32F217xx Reference manual, RM0033, revision 3. ST Microelectronics

[5] STM32F10xxx Cortex-M3 programming manual, PM0056, revision 3. ST Microelectronics

[6] IEEE 1588 precision time protocol demonstration for STM32F107 connectivity line microcontroller, AN3411, Application note, revision 1. ST Microelectronics

[7] STM32F microcontroller system memory boot mode, application note, AN2606, revision 10. ST Microelectronics

[8] USB DFU protocol used in the STM32^{TM} bootloader, AN3156, application note, revision 1. ST Microelectronics

[9] STM32F10xxx hardware development: getting started, AN2586, application note, revision 5. ST Microelectronics

[10] Oscillator designguide for ST microcontrollers, AN2867, application note, revision 5. ST Microelectronics

[11] HD44780U (LCD-II) - Dot Matrix Liquid Crystal Display Controller/Driver. HITACHI

[12] WM-C1602N 16x2 characters, datasheet.

[13] Analog Devices, +5 V Powered CMOS RS-232 Drivers/Receivers. ADL232 IC Datasheet.

[14] Stránky předmětu Aplikave Vestavných Systémů, http://measure.feld.cvut.cz/vyuka/predmety/A4M38avs.

[15] Moduly s rozhraním Ethernet pro systém sběru dat, Bakalářská práce. Adam Bařtipán ČVUT v Praze, 2011

[16] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems July 2008 IEEE Standard Association

[17] http://www.nist.gov/el/isd/ieee/ieee1588.cfm National Institute of Standards and Technology

[18] Precision Clock Synchronisation, The standard IEEE 1588. Hirschmann

[19] Hongkun Zhao, Xiaoli Wang Design and Implementation of Precision Time Synchronization System Based on IEEE1588. School of Mechanical, Electrical & Information Engineering Shandong University at Weihai

[20] http://ptpd.sourceforge.net PTP deamon

[21] Design and Implementation of the lwIP TCP/IP Stack. Adam Dunkels, Swedish Institute of Computer Science, 2001

[22] Průmyslový Ethernet I. až VIII. Seriál článků. František Zezulka, Ondřej Hynčica, AUTOMA, 2008

[23] Introduction to Real-Time Ethernet I. Paula Doyle, University of Limerick in Ireland, 2004

[24] Introduction to Real-Time Ethernet II. Paula Doyle, University of Limerick in Ireland, 2004

[25] Formalizing FreeRTOS: First Steps. David Déharbe, Stephenson Galvao, and Anamaria Martins Moreira

[26] Oficiální webové stránky projektu FreeRTOS. http://www.freertos.org

Seznam symbolů, veličin a zkratek

ARM	Advanced RISC Machine
API	Application Programming Interface
μ**C**	Microcontroller
HW	Hardware
SW	Software
GUI	Graphical User Interface
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
OTG	On-The-Go
UART	Universal Asynchronous Reciever/Transmitter
USART	Universal Synchronous/Asynchronous Reciever/Transmitter
DFU	Device Firmaware Upgrade
I2C	Inter-Integrated Circuit
I2S	Inter-Integrated Sound
CAN	Controller Area Network
ADC	Analogue/Digital Converter
DAC	Digital/Analog Converter
SWD	Serial Wire Debug

JTAG Joint Test Action Group

RTOS Real-Time Operating System

ISR Interrupt Service Routine

LAN LAN Local Area Network

CSMA/CD Collision Sense Multiple Access/Collision Detection

lwIP lightweight Internet Protocol

PTP PTP Precise Time Protocol

BMC BMC Best Master Clock

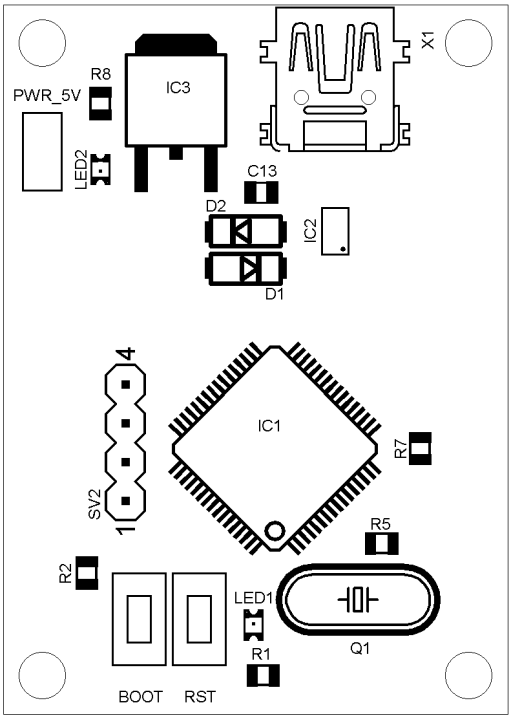
ESD ElectroStatic Discharge

DPS Deska Plošných Spojů

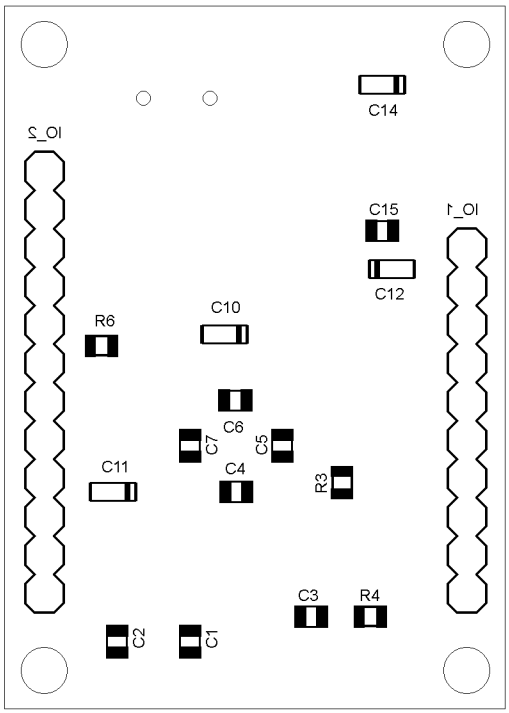
Příloha A

Výrobní podklady pro modul μLab

A.1 Osazovací plán

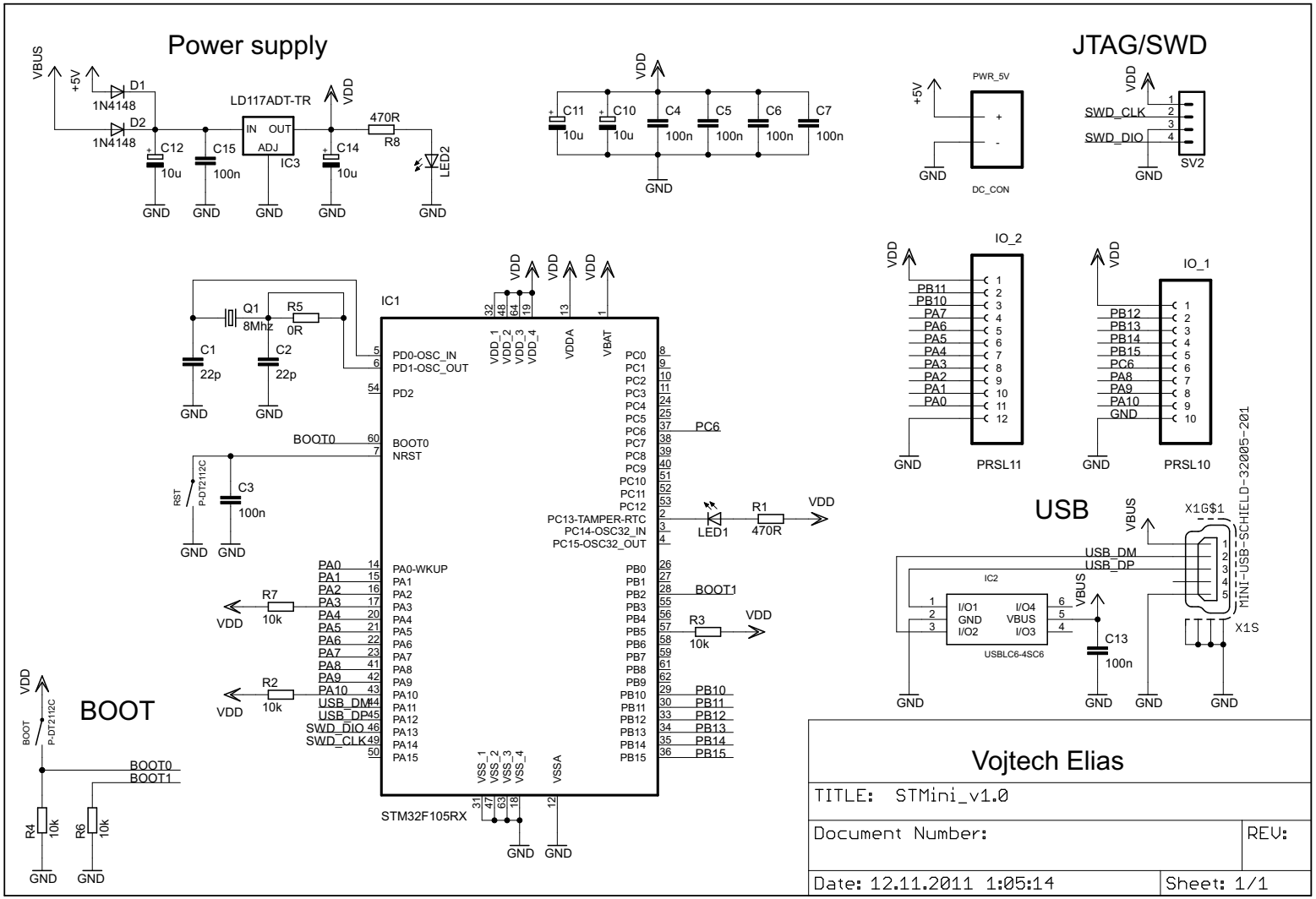


Obrázek A.1: Osazení součástek v horní vrstvě (TOP)



Obrázek A.2: Osazení součástek ve spodní vrstvě (BOTTOM)

A.2 Elektrické schéma



Obrázek A.3: Elektrické schéma

A.3 Seznam součástek

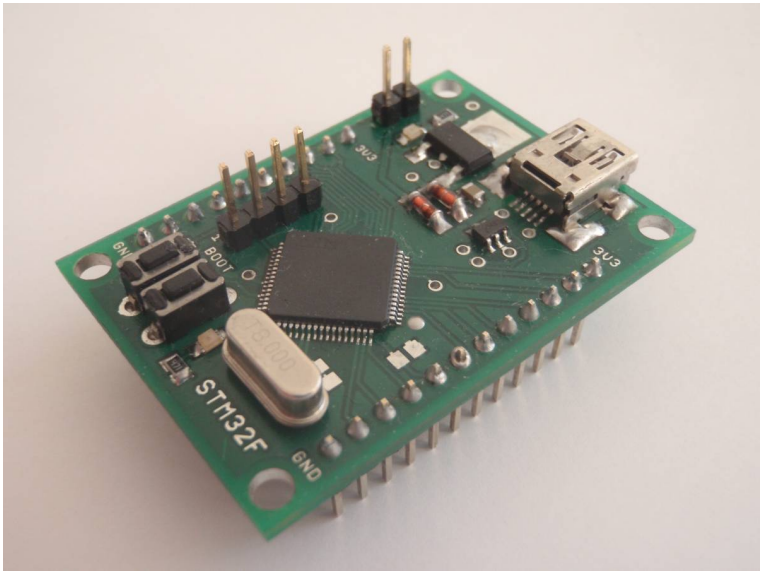
Počet	Označení	Hodnota	Popis
2	C1,C2	22p	keram. kapacitor
7	C3-C7,C13,C15	100n	keram. kapacitor
4	C10-C12,C14	10μ	elyt. kapacitor
2	R1,R8	470Ω	rezistor
5	R2-R4,R6,R7	10kΩ	rezistor
1	R5	0Ω	neosazen
2	D1,D2	1N4148	dioda
2	LED1,LED2	LED-0805	LED
2	BOOT,RST	P-DT2112C	tlačítko
1	Q1	8MHz	krystal HC49/S
1	IC1	STM32F10x	μC v pouzdře LQFP64
1	IC2	USBLC6-4SC6	ESD ochranný obvod
1	IC3	LD117ADT-TR	stab. (náhrada LD33)
1	IO_1	PRSL10	konektorová lišta (10 pinu)
1	IO_2	PRSL12	konektorová lišta (12 pinu)
1	PWR_5V	DC_CON	napájecí kon. (2 piny, rozteč 100mil)
1	SV2	PRSL4	konektorová lišta (4 piny)
1	X1	mini USB	mini USB konektor

Tabulka A.1: Seznam součástek pro konstrukci modulu μLab

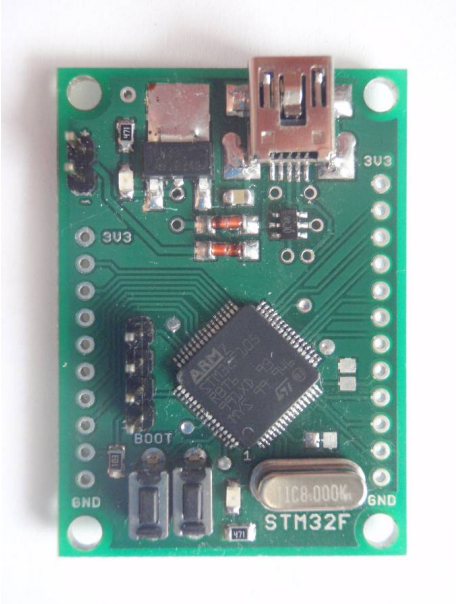
Příloha B

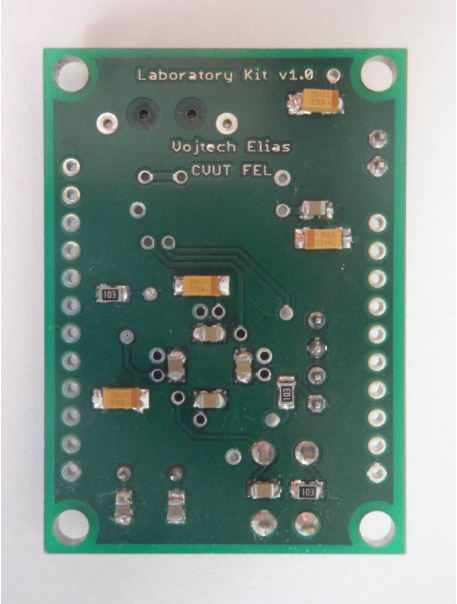
Fotodokumentace

B.1 μLab



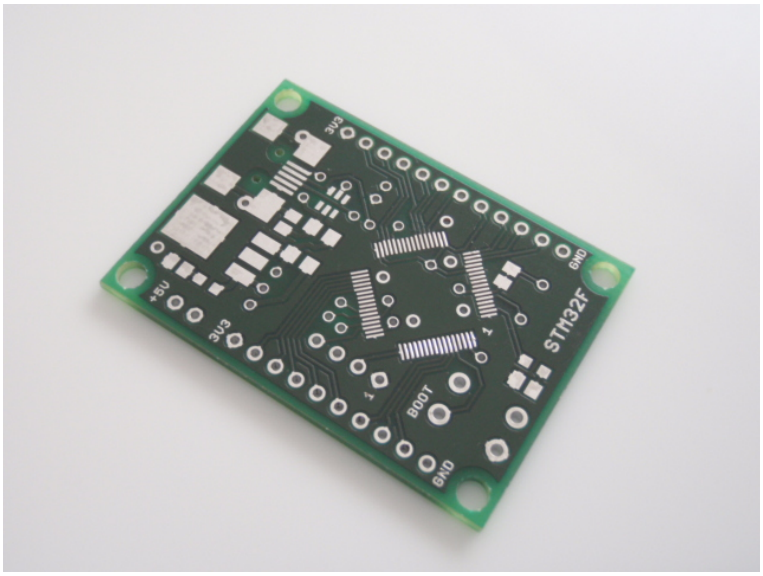
Obrázek B.1: Celkový pohled na vývojový modul μLab





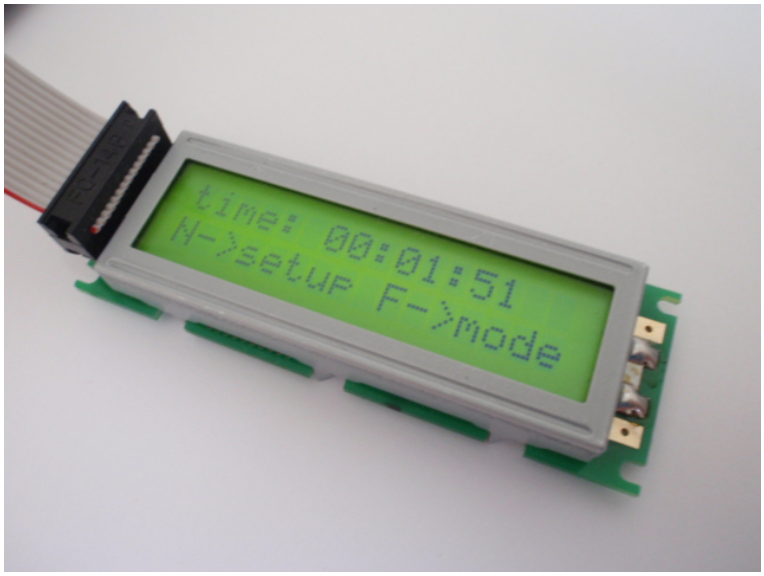
Obrázek B.2: Osazení plošného spoje modulu μLab (TOP)

Obrázek B.3: Osazení plošného spoje modulu μLab (BOTTOM)

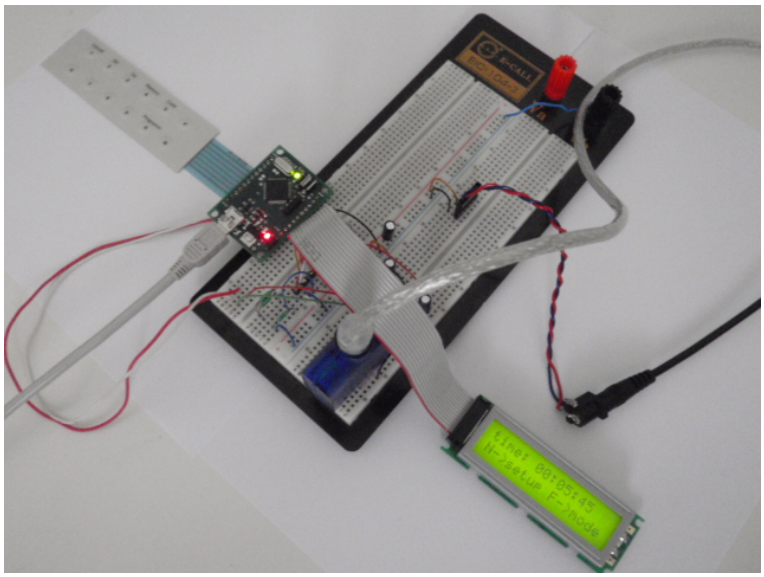


Obrázek B.4: Přepracovaný plošný spoj modulu μLab (verze 2.0)

B.2 Aplikace modulu μLab s podporou FreeRTOS



Obrázek B.5: Zobrazení údajů na displeji jednotky digitálních hodin

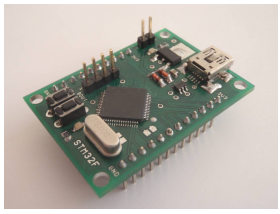


Obrázek B.6: Fyzická realizace jednotky digitálních hodin na kontaktním poli

FreeRTOS Based Embedded System

LABORATORY OF VIDEOMETRY

Department of Measurement
CTU FEE in Prague

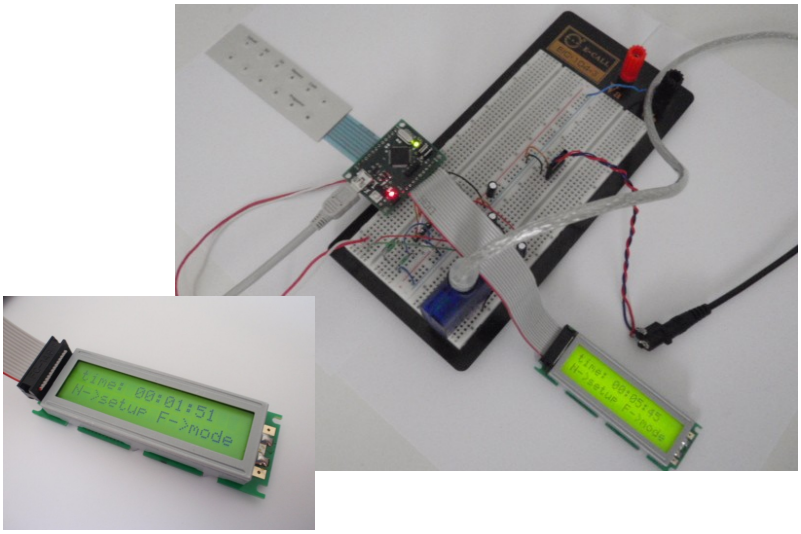


Application overview:

- Adjustable digital clock with 12/24 display modes
- Utilization of FreeRTOS (real time operating system kernel for embedded systems)
- Software architecture based on preemptive multitasking

Components:

- Lightweight laboratory modul equipped with **STM32F105RBT6** microcontroller
- LCD/keypad human interface
- RS232 data interface



Contacts:

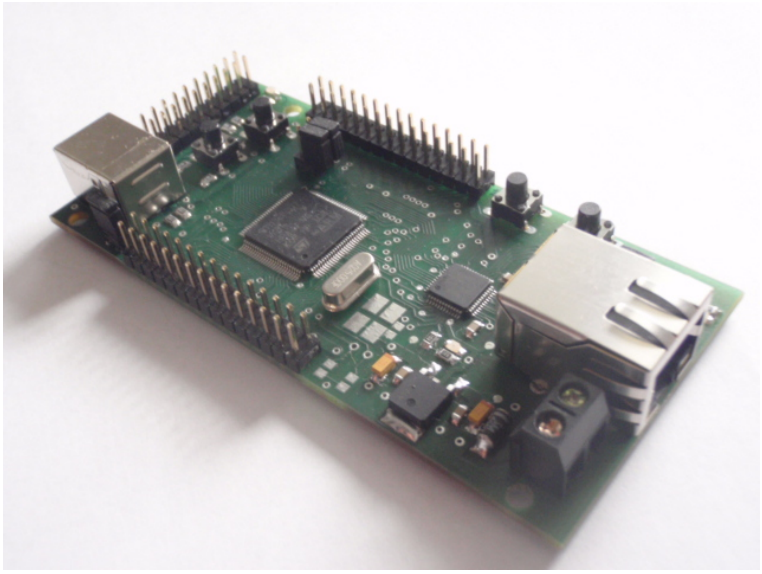
Creator: Vojtěch Eliáš (vojtech.elias@gmail.com)
Head of the project: doc. Ing. Jan Fischer, CSc. (fischer@fel.cvut.cz)

Cortex
Intelligent Processors by ARM

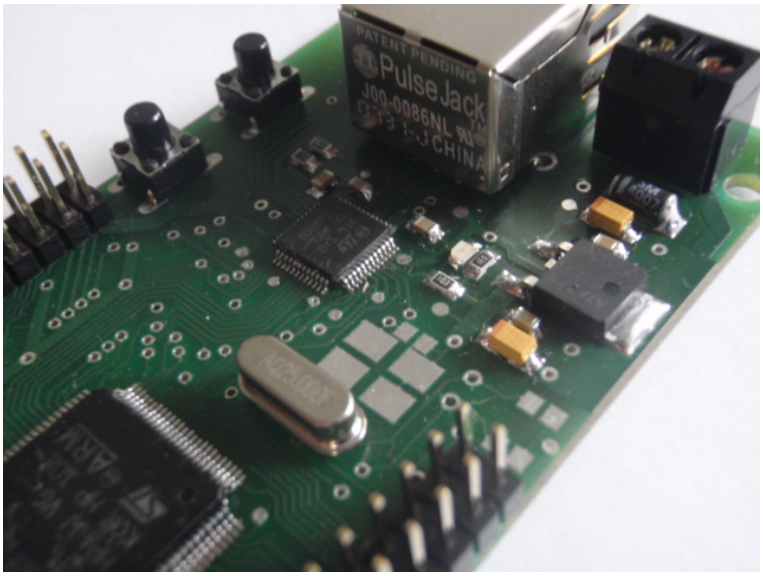
freeRTOS

Obrázek B.7: Prezentační prospekt aplikace

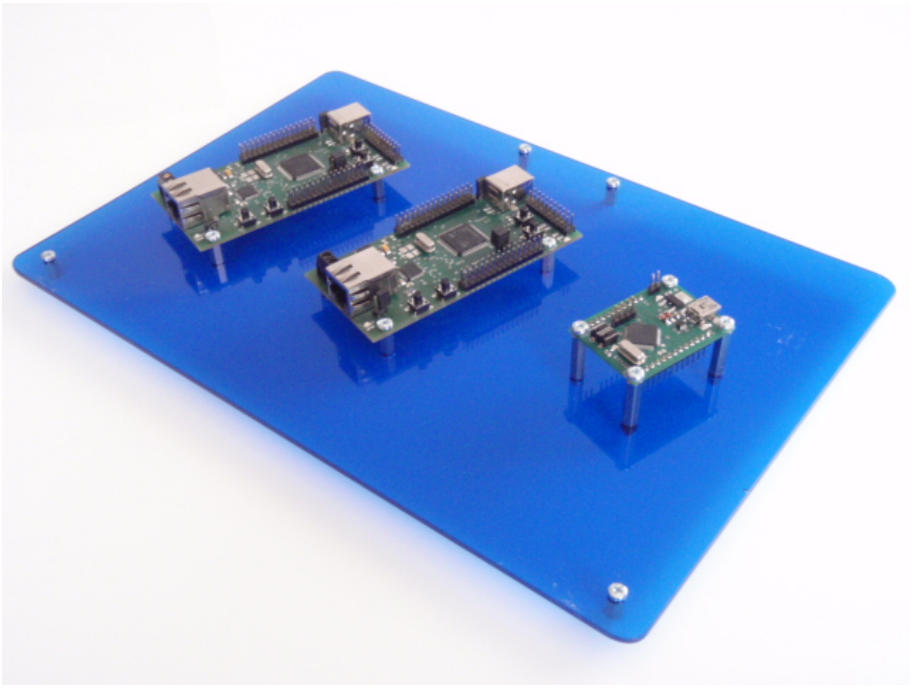
B.3 Realizace distribuovaného systému



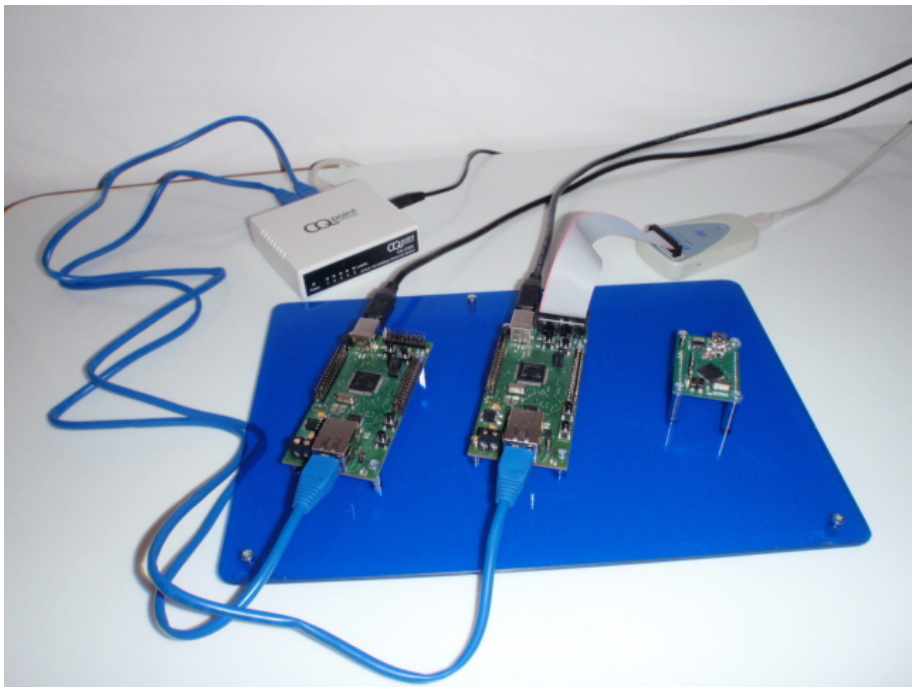
Obrázek B.8: Osazený modul pro distribuovaný sběr dat



Obrázek B.9: Detail osazení modulu pro distribuovaný sběr dat



Obrázek B.10: Instalace trojice modulů na společné experimentální platformě



Obrázek B.11: Připojení dvojice modulů k síti Ethernet