České vysoké učení technické v Praze
Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Záznamník dat pro RS-232

Praha, 2009 **Autor: Martin Novák**

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 22.1.2009

Martin Vo...

podpis

Poděkování

Zde bych rád poděkoval vedoucímu své diplomové práce Ing. Liborovi Waszniowskému, Ph.D. za cenné rady a podněty, Ing. Martinovi Kimrovi z firmy Kodys, s. r. o. za vstřícnost a svému kamarádovi Ing. Janu Smržovi za neocenitelnou podporu během celého studia.

Abstrakt

Tato práce popisuje kompletní návrh embedded zařízení pro záznam komunikace dvou prvků sběrnice RS-232. Autorem zadání je firma působící v oblasti systémů s využitím čárových kódů, jejíž motivací je detekce chyb v komunikaci při vývoji prvků systému. Tento záznamník dat ukládá komunikaci na paměťovou kartu Secure Digital, doplňuje záznam o časové značky a obsahuje také uživatelské rozhraní v podobě displeje a tlačítek.

Práce se zabývá výběrem mikrokontroléru, volbou vhodných vývojových prostředků a návrhem hardware a software.

Těžištěm práce je oživení mikrokontroléru Atmel AT91SAM7S256 s jádrem ARM a vývojových nástrojů GNU ARM a OpenOCD. Software je postaven na vlastní koncepci s využitím knihoven Newlib a EFSL.

Abstract

This thesis describes the development of a comprehensive embedded device for recording communication between two RS-232 bus elements. The assignment was set by a company engaged in the field of bar codes systems which also seeks means of error detection in communication during system elements development. This datalogger stores communication on a Secure Digital memory card and adds time stamps to the log and contains a user interface as well. The user interface provided for by means of a display and buttons.

This paper deals with the selection of the micro-controller as well as suitable development tools and the design of software and hardware.

The main focus of the thesis is bringing the Atmel AT91SAM7S256 micro-controller with ARM core into life in liaison with the GNU ARM and OpenOCD development tools. The software is based on the author's own conception using the Newlib and EFSL libraries.

České vysoké učení technické v Praze - Fakulta elektrotechnická

Katedra řídicí techniky *Školní rok:* 2006/2007

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Martin Novák

Obor: Technická kybernetika

Název tématu: Záznamník dat pro RS-232

Zásady pro vypracování:

1. Navrhněte záznamník dat pro monitorování dat na sériové lince v jednom směru. Záznamník bude data opatřovat časovou značkou a ukládat je na paměťovou kartu SD případně MMC ve formátu souboru čitelném v běžných čtečkách připojených k PC. Záznamník bude ovládán pomocí tlačítek a bude zobrazovat informace na displeji.

2. Sestavte prototyp přístroje a implementujte firmware.

3. Na jednoduchém příkladu demonstrujte jeho funkci.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí diplomové práce: Ing. Libor Waszniowski, Ph.D.

Termín zadání diplomové práce: zimní semestr 2006/2007

Termín odevzdání diplomové práce: leden 2008

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

prof. Ing. Zbyněk Škvor, CSc.
děkan

V Praze dne 21.02.2007

Obsah

Seznam obrázků

Seznam použitých zkratek

ALU	Arithmetical Logical Unit
API	Application Programming Interface
ARM	Advanced RISC Machines
CISC	Complex Instruction Set Computer
DMA	Direct Memory Access
DSP	Deska Plošných Spojů
EFSL	Embedded FileSystems Library
ELF	Executable and Linking Format
FIQ	Fast Interrupt reQuest
GCC	GNU Compiler Collection
GDB	GNU DeBugger
GNU	GNU's Not Unix
HAL	Hardware Abstraction Layer
I^2C	Inter-Integrated Circuit
IDE	Integrated Development Enviroment
IRQ	Interrupt ReQuest
JTAG	Join Test Action Group
LDO	Low DropOut regulator
LMA	Load Memory Adress
OpenOCD	Open On-Chip Debugger
RISC	Reduced Instruction Set Computer
RTC	Real-Time Clock
SD	Secure Digital
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmiter
VMA	Virtual Memory Adress

Kapitola 1

Úvod

1.1 Motivace

Práce popisuje vývoj zařízení, jehož specifikace vychází z požadavků firmy Kodys, s. r. o. Tato firma se zabývá vývojem identifikačních systémů, které využívají čárové kódy a sériový komunikační standard RS-232. Při vývoji prvků takového systému hraje významnou roli detekce chyb v jejich komunikaci.

Vznikla tedy potřeba zařízení, které bude monitorovat veškerou komunikaci mezi dvěma uzly sítě, umožní její rekonstrukci a tím identifikaci chyby i jejího původce. Pro vlastní využití takového zařízení v praxi jsou důležitými vlastnostmi: spolehlivost, snadnost ovládání, jednoduchý formát záznamu a v neposlední řadě také malé rozměry. Tyto požadavky vedou na *embedded zařízení*, které nazveme **záznamník dat pro RS-232 (datalogger)**.

Tento záznamník ukládá obsah komunikace do textového souboru na paměťovou kartu SD (Secure Digital) v běžně čitelném formátu. Pro přehledné a rychlé nastavování parametrů disponuje zařízení uživatelským rozhraním v podobě displeje a tlačítek. Zařízení je napájeno z externího zdroje nebo z baterie.

Záznam je doplňován o identifikátory směru komunikace, aby byla možná její úplná rekonstrukce. Součástí záznamníku je také obvod reálného času, aby mohl záznam obsahovat časové značky.

1.2 Použité technologie

Základem zařízení je jednočipový mikrokontrolér Atmel AT91SAM7S256 s jádrem ARM. Tento integrovaný obvod již obsahuje množství periferií usnadňujících vývoj embedded systémů, z nichž využijeme periferie UART, SPI a I^2C. Paměťová karta SD je k mikrokontroléru připojena přes SPI rozhraní a obvod reálného času komunikuje přes sběrnici I^2C.

Hlavním softwarovým nástrojem je v této úloze programovací jazyk C s vývojovými nástroji GNU ARM v distribuci YAGARTO. K In-System programování a debuggování je použit OpenOCD a GDB s integrací do Eclipse IDE. Výhodou všech těchto nástrojů je volná licence a kvalita do jisté míry srovnatelná s komerčními alternativami.

Použité technologie jsou popsány v kapitole 4.

1.3 Práce a výsledky

1.3.1 Návrh architektury

Návrh architektury se týká především volby mikrokontroléru s ohledem na požadavky úlohy. Mezi hlavní kritéria patří: vhodné periferie, dostatek paměti, postačující výkon, dostupnost vývojových prostředků, apod.

Zvolený obvod AT91SAM7S256 je postaven na 32-bitovém jádru ARM, které je dnes téměř nejrozšířenější v embedded aplikacích. Jeho předností je například RISC architektura s kvalitní instrukční sadou, velká rozšířenost tedy i dostupnost hardwaru i vývojových nástrojů a nízká spotřeba.

Další výhodou tohoto mikrokontroléru je periferní DMA (Direct Memory Access) kanál, který byl s výhodou použit v této aplikaci.

1.3.2 Návrh hardwaru

Pro potřeby úlohy byl vyvinut a kompletně oživen vlastní prototyp hardwaru. S použitím nástrojů OrCAD bylo vytvořeno obvodové schéma a rozvržení DSP. Výsledkem je plně funkční prototyp finálního produktu, jehož fotografie je na obrázku 1.1.

Obrázek 1.1: Výsledný prototyp záznamníku.

1.3.3 Implementace softwaru

Software je založen na vlastní modulární *bare-metal* koncepci (tedy bez využití operačního systému), jehož některé části jsou plně přenositelné na jiné platformy. Program je rozdělen do několika vrstev a částí.

Základem je tzv. *startup kód*, který nastavuje základní funkční parametry, definuje model přerušení a připravuje mikroprocesor na skok do hlavní programové smyčky. V hlavním programu se uplatňují ovladače periferií obsažených na čipu, na jejichž bázi jsou vytvořeny drivery okolních periferií, RTC, SD a LCD displeje. Ovladače i všechny prvky hlavní aplikace jsou postaveny na vlastní hardwarové abstrakci.

Nad ovladačem grafického LCD displeje byl implementován nástroj pro přehledné a pohodlné vytváření uživatelského rozhraní včetně kaskádového menu. Uživatelské rozhraní tak dovoluje nastavit funkci zařízení, např. parametry RS-232 sběrnice, datum a čas, apod.

Pro práci se souborovým systémem FAT na paměťovém médiu byla použita *open-source*knihovna EFSL (Embedded Filesystem Library), částečně rozšířena o potřebné funkce.

Kvalitním zdrojem standardních C funkcí byla *open-source* knihovna *Newlib* pro embedded aplikace.

Kapitola 2

Specifikace požadavků na zařízení

Požadavky na zařízení se odvíjejí z běžné praxe vývojářů firmy Kodys, s. r. o., se sídlem v Praze. Zařízení musí mít tyto vlastnosti:

- dva porty RS-232,
- přeposílání veškeré komunikace z jednoho portu na druhý při minimálním ovlivňování zbytku sítě (tj. s minimálním zpožděním),
- schopnost zaznamenávát tok dat v obou směrech, resp. přicházející z obou těchto portů,
- ukládání veškeré komunikace na paměťovou kartu typu *Secure Digital* ve formátu čitelném v běžných čtečkách,
- schopnost rozlišit, ze kterého portu daná data přicházejí,
- možnost doplňovat záznam o časové značky s měnitelnou periodou,
- obsahovat uživatelské rozhraní v podobě displeje a jednoduché klávesnice pro nastavení důležitých parametrů,
- formátovat záznam do textového souboru, tedy přímo čitelného uživatelem.

Požadované možnosti nastavení:

- parametry linky RS-232 – *baud rate, datové bity, stop bity, parita*,
- přesný čas (datum),
- perioda časových značek v řádu sekund,

- doplňování záznamu o identifikátor portu / surová data,
- vymazání obsahu paměťové karty.

Kapitola 3

Architektura zařízení

Výše zmíněnou funkčnost by nejspíše bylo možné zajistit například přenosným počítačem s externími sériovými porty (např. do USB) a softwarem zajišťujícím danou funkci.

Řešení je to ale poměrně nešikovné. Proti hovoří rozměry, přílišná robustnost, spolehlivost závislá na stavu a kvalitě operačního systému a při větším počtu kusů samozřejmě i cenové nároky.

Elegantnější řešení, navrhované i zadavatelem, je embedded zařízení, kde je hlavním článkem vhodný jednočipový mikrokontrolér.

3.1 Hardwarová část

3.1.1 Volba mikrokontroléru

Kritéria volby mikrokontroléru vychází z požadavků úlohy. Důležitými vlastnostmi jsou:

- vhodné periferie, dostatek paměti,
- postačující výkon, rozumná spotřeba,
- dostupnost a cena,
- dostupnost vývojových nástrojů a knihoven,
- případná rozšiřitelnost.

Řešení zadaného problému stojí na pomezí 8-bitových a 32-bitových aplikací. Potřebné periferie jsou v zásadě bytově orientované, což vede na 8-bitovou architekturu, nicméně

dobrý výkon, propustnost a dostatečný paměťový prostor najdeme spíše u 32-bitových mikrokontrolérů.

32-bitové mikrokontroléry se díky klesajícím cenám integrovaných obvodů dostávají na pole 8-bitových aplikací. Předností 32-bitové architektury je především větší propustnost a výkon při nižších pracovních frekvencích díky zpracování většího počtu dat v jedné instrukci.

Podle článku Pavla Píšy o moderních mikrokontrolérech [25] je nedostatkem dnešních 32-bitových procesorů (v návaznosti na RISC architekturu) poměrně malá hustota kódu, tedy velké paměťové nároky (pouzdra často obsahují větší paměti). Tyto nedostatky částečně řeší instrukční sady s kratší nebo proměnnou délkou, např. u Freescale ColdFire nebo ARM-Thumb jader.

Nejrozšířenějšími ve 32-bitových embedded aplikacích jsou mikrokontroléry s procesorovým jádrem ARM (zdroj: [2]).

ARM je rodina procesorů, které si získaly oblibu díky nízké spotřebě i při vysokém výkonu (jíž dosáhly především absencí mikrokódu), RISCovou architekturou a snahou udržet si co nejjednodušší design, tedy menší počet tranzistorů. Nabízí také velmi dobrý *power management* a přehlednou ochranu důležitých registrů díky různým pracovním režimům, což je dobře popsáno např. v knize Steva Furbera [36].

Nejúspěšnější třídou ARM jader je ARM7TDMI (kapitola 5.1.2). Jedná se sice o low-endové jádro, ale velmi dobře vybavené. Nabízí mj. tříúrovňový pipeline, dvě instrukční sady (32 a 16-bitovou), možnost hardwarového real-time debuggování nebo hardwarové breakpointy a watchpointy. Jeho spotřeba se při 18 μm technologii pohybuje okolo 0.21 mW/MHz a pracovní frekvence dosahuje až 115 MHz [5].

V této třídě najdeme i mikrokontroléry s pokročilými periferiemi, jako jsou USB, CAN nebo Ethernet. Lídry (zejména ve výběru obvodů a dostupnosti) jsou zde rodiny obvodů Philips LPC a konkurenční Atmel SAM7. Oba konkurenti jsou srovnatelné v dostupnosti vývojových prostředků.

Výhodou obvodů LPC může být např. lepší podpora pro CAN sběrnici, nebo rychlejší flash paměť bez „stavů čekání“ (frekvence čtení instrukcí z flash paměti u SAM7 je efektivně „pouze“ 30MHz – neplatí pro Thumb instrukce), viz manuály obvodů [23] [8].

Mikrokontroléry Atmel SAM7 [8] ovšem mají oproti svým konkurentům ojedinělou možnost využít DMA (Direct Memory Access) kanál pro vybrané periferie (mj. UART a SPI). Použitím DMA se periferie vyhýbá intervenci procesoru a odstraňuje jeho značný výdaj při obsluze přerušení, tím výrazně šetří hodinové cykly a energii. Tento model umožňuje relativní nezávistlost periferií na procesoru, důsledkem je větší propustnost

dat přes periferie a efektivní běh procesoru bez zbytečných přerušení.

V neposlední řadě nabízí Atmel co do přehlednosti a ucelenosti pro své konkurenty příkladnou dokumentaci.

Z výše uvedených důvodů byl pro naši úlohu zvolen mikrokontrolér z rodiny Atmel SAM7, konkrétně AT91SAM7S256 (více v kapitole 5.1).

3.1.2 Komunikační sběrnice, periferie

Secure Digital

Paměťová karta Secure Digital (SD) navazuje na zpětně kompatibilní standard MMC, se kterým sdílí tvar a částečně fyzickou vrstvu. Podle specifikace fyzické vrstvy [19] probíhá komunikace s kartou ve smyslu MASTER ↔ SLAVE, přičemž hostitel je vždy MASTER, karta vždy SLAVE. Specifikace mj. dovoluje připojení/odpojení za běhu (tzv. *Hot insertion*), přičemž hostitel musí zajistit tuto podporu především správným uspořádáním vodičů v konektoru, tzn. napájecí vodiče musí být delší, nežli datové.

Tato paměťová karta dovoluje tři typy ochrany dat:

- mechanický přepínač ochrany proti zápisu (zodpovědnost hostilete),
- vnitřní nastavení ochrany proti zápisu (zodpovědnost karty),
- ochrana části paměti pomocí hesla (zodpovědnost uživatele).

Zajímavostí je, že mechanický přepínač ochrany proti zápisu neovlivňuje vnitřní logiku, může a nemusí tedy být respektován hostitelem, viz manuál SD karet SanDisk [31].

Zmíněná specifikace popisuje dvě základní verze SD karet (jejich fyzických vrstev). Jsou to:

- Standard Capacity SD card (kapacita < 2GB),
- High Capacity SD card (kapacita > 2GB).

Obě verze podporují dva typy datových sběrnic, rychlejší paralelní a jednodušší sériovou. U obou typů sběrnic nesmí frekvence hodin přesáhnout 50MHz.

- **SD bus** je paralelní šestivodičová komunikační sběrnice, která se skládá z těchto vodičů:
  - CLK – hodinový signál MASTER → SLAVE,

  - CMD – command signál MASTER ↔ SLAVE,
  - D0-D4 – datové vodiče MASTER ↔ SLAVE.

- **SPI bus** je standardní sériová čtyřvodičová sběrnice. Skládá se z těchto vodičů:
  - SCLK – hodinový signál MASTER → SLAVE,
  - MOSI – Master Output Slave Input data, MASTER → SLAVE,
  - MISO – Master Input Slave Output data, SLAVE → MASTER,
  - CS – Chip Select.

Průměrná rychlost přenosu dat může být u *SD bus* až 50MB/s a u *SPI bus* až 6MB/s. Skutečná rychlost zápisu dat je ale vždy výrazně menší a je mj. závislá na kvalitě a kapacitě paměťové karty.

Pro naše účely postačuje standardní SPI sběrnice, která má dostatečnou propustnost a potřebuje menší počet vodičů.

Použitý mikrokontrolér navíc obsahuje její hardwarovou implementaci v podobě SPI periferie, takže není nutné implementovat ovladač dané sběrnice.

RTC

Záznamník dat je nutné doplnit o obvod reálného času (Real Time Clock) pro možnost vkládání časových značek do záznamu. K dispozici je velké množství takových obvodů, které se liší především v komunikační sběrnici.

Pro naše účely bude vhodný RTC obvod komunikující pomocí sběrnice I^2C, protože je tento standard podporován mikrokontrolérem AT91SAM7S (periferií *Two-wire interface*).

Konkrétně byl vybrán obvod Philips PCF8563, popsán v kapitole 5.2.

LCD

Při výběru vhodného LCD displeje budou rozhodujícími následující kritéria:

- dostatek prostoru pro požadovanou funkčnost, tzn. dostatečné rozlišení (pro grafický displej), nebo počet řádků a sloupců (pro alfanumerický displej),
- rozumné vnější rozměry displeje,
- co nejjednodušší komunikační rozhraní,

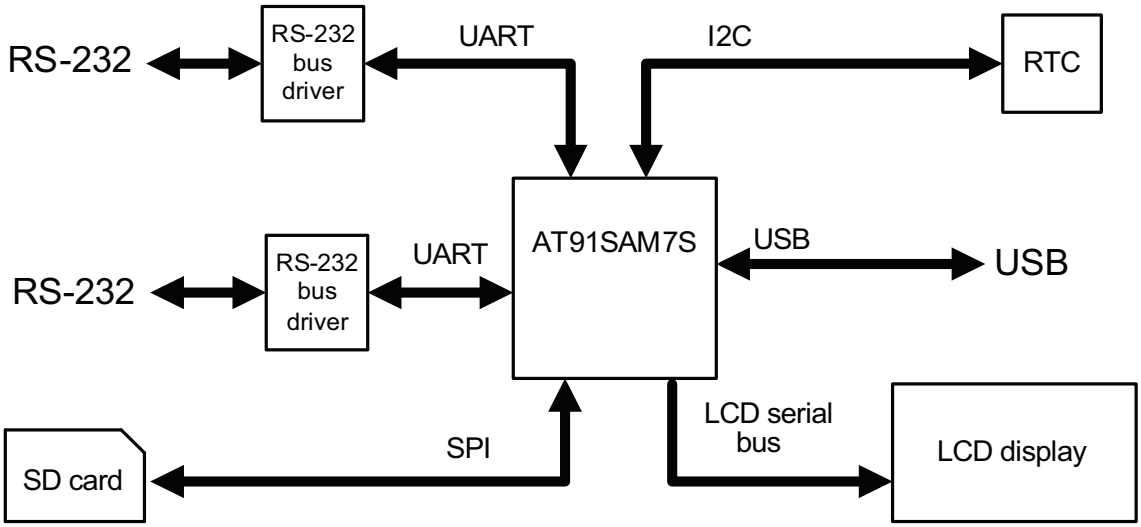
- vstupní napětí 3.3V,
- dostupnost a cena.

Kritéria splňuje vybraný grafický displej ATM12864D s rozlišením 128x64 bodů, který je řízen LCD řadičem KS0713. Tento řadič podporuje kromě běžné paralelní komunikační sběrnice také úspornější sériovou, celé řízení displeje tak probíhá pomocí pěti vodičů, viz kapitola 5.3.4.

3.1.3 Architektonické schéma zařízení

Mikrokontrolér AT91SAM7S256 obsahuje hardwarovou implementaci USB řadiče, proto je rozumné do zařízení umístit USB konektor pro napájení a případné pozdější rozšíření funkčnosti.

Blokové schéma navržené architektury zařízení je na obrázku 3.1.



Obrázek 3.1: Architektonické schéma zařízení.

3.2 Softwarová architektura

V této kapitole naznačíme hlavní rysy softwarového řešení úlohy.

3.2.1 Bare-metal

Pro výkonnější jednočipové mikroprocesory jsou k dispozici embedded real-time operační systémy, které umožňují multitasking, multithreading apod. Jde např. o komerční μC/OS-II nebo volné eCos, FreeRTOS či Embedded Linux.

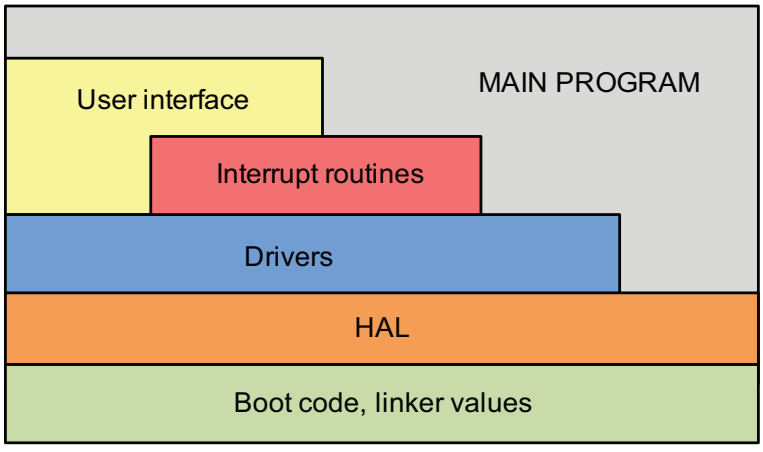
Naše úloha i její zpracování nevyžaduje příliš složitou softwarovou architekturu, resp. nevyžaduje zmíněné pokročilé možnosti. Z toho důvodu byl zvolen softwarový model bez operačního systému, tzv. *bare-metal*. Výhody jsou rychlejší odezva přerušení, jednoduchost, přehlednost, jednodušší psaní ovladačů a v neposlední řadě také studijní účely.

3.2.2 Logika aplikace

Návrh software je možné rozdělit do těchto částí:

1. startup kód (základní nastavení, stack, memory remap, příprava na skok do hlavní smyčky),
2. případná hardwarová abstrakce,
3. ovladače periferií,
4. uživatelské prvky a utlity,
5. hlavní aplikace.

Tyto části odpovídají vrstvám naznačeným na schématu 3.2. Podrobnosti najdeme v kapitole 6.



Obrázek 3.2: Schéma architektury softwaru.

3.3 Toolchain

Pro ARM platformu je k dispozici celá řada vývojových nástrojů, z nichž zde uvedeme ty nejdůležitější.

3.3.1 ARM® RealView® Suite

ARM RealView je komerční vývojový nástroj od společnosti ARM Ltd. [5]. Obsahuje vlastní C a C++ kompilátor, který patří k těm nejlepším, co se týče optimalizace a velikosti produkovaného kódu. Dále zahrnuje vlastní vývojové prostředí *ARM Workbench IDE*, které je postaveno na *open-source*prostředí Eclipse. To je obohaceno o další nástroje zjednodušující návrh systému, např. utility pro přehledné nastavení parametrů kompilátoru a linkeru, nástroje pro grafickou reprezentaci paměti namapované linkerem a další. Obsahuje také profilovací nástroj, který je schopen realtime on-chip analýzy softwaru.

3.3.2 IAR Embedded Workbench

Tento balík nástrojů od firmy IAR Systems [14] obsahuje vlastní C a C++ kompilátor, assembler, debugger nazývaný C-SPY a také vlastní IDE prostředí. IAR také nabízí vlastní vývojové desky k vybraným mikrokontrolérům včetně USB-JTAG hardwaru.

Zajímavostí je nástroj IAR visualSTATE, se kterým je možné vytvářet základ aplikace přímo z visuálního návrhu stavového automatu. Všechny tyto vývojové prostředky jsou komerční.

3.3.3 Keil™ Development Tools

Komerční vývojové nástroje německé firmy Keil Elektronik GmbH [16] spojují low-level nástroje (kompilátor, linker) RealView (kapitola 3.3.1) s vlastním IDE a debuggerem (μVision®), který má mnoho předností. Toto IDE např. zahrnuje velmi pokročilé simulátory jádra i důležitých periferií, dovede přehledně a rychle nastavovat parametry periferií, apod. Tato firma také nabízí vlastní vývojový hardware vysoké kvality.

3.3.4 GNU ARM Toolchain

V poslední době se stává konkurenceschopnou komerčním prostředkům sada vývojových *open-source* nástrojů projektu GNU (kapitola 4.1), kterou je možné najít v rozličných distribucích, jako je CodeSourcery (oficiální distribuce), Yagarto, WinARM a další.

Zajímavou je distribuce YAGARTO, která spojuje GNU ARM Toolchain, Eclipse IDE a debuggovací nástroj *OpenOCD* (*Open-source On-Chip Debugger*). Tato kombinace dovoluje pohodlné JTAG on-chip debuggování přímo v pokročilém prostředí Eclipse, čímž se stává mocným nástrojem.

Nevýhodou těchto nástrojů může být v porovnání s komerčními alternativami určitá „nedokonalost“, např. nutnost ručně upravovat komunikační a konfigurační skripty (linkerscript, debugger, OpenOCD, makefile), což může být zdlouhavé a náročné. K dispozici jsou pouze základní generické varianty těchto skriptů.

Velikou výhodou je ovšem fakt, že jde o software s volnou licencí. Více nalezneme v kapitole 4.1.

GNU ARM Toolchain v distribuci YAGARTO byl zvolen vývojovým nástrojem pro naši úlohu.

Kapitola 4

Použité technologie

V této kapitole přiblížíme použité technologie, vývojové prostředky, softwarové nástroje a knihovny.

4.1 GNU ARM Toolchain

Projekt s názvem *GNU* [12] běží od roku 1984 za účelem vzniku unixového operačního systému se svobodnou licencí, který však nebude obsahovat žádný kód z původního UNIXu (GNU je rekurzivní zkratka *GNU's Not Unix*). Dnes je projekt zaměřen obecně na svobodný software inspirovaný operačními systémy unixového typu. Software označený GNU se řídí licencí GPL (General Public License).

GNU ARM Toolchain [13] je sada *open-source* vývojových nástrojů pro platformu s jádrem ARM. Jeho výhodou je samozřejmě volná licence.

Je rozdělen do těchto částí:

- sada kompilátorů `gcc` (GNU Compiler Collection),
- debugger `gdb`,
- sada binárních utilit (linker, assembler, archiver),
- balík standardních knihoven `newlib`,
- grafická nadstavbu nad `gdb` s názvem *Insight*.

Kompilátor `gcc` je schopen kompilovat z jazyků C, C++, Objective-C, Objective-C++, Java, Fortran a Ada, viz manuál `gcc` kompilátoru [29].

4.1.1 ELF

Pro strukturu výstupních souborů používá GNU ARM Toolchain standard ELF (Executable and Linking Format), z čehož vychází i názvy binárních souborů toolchainu (např. `arm-elf-gcc.exe`). ELF postupně nahrazuje starší strukturu COFF (viz [39]).

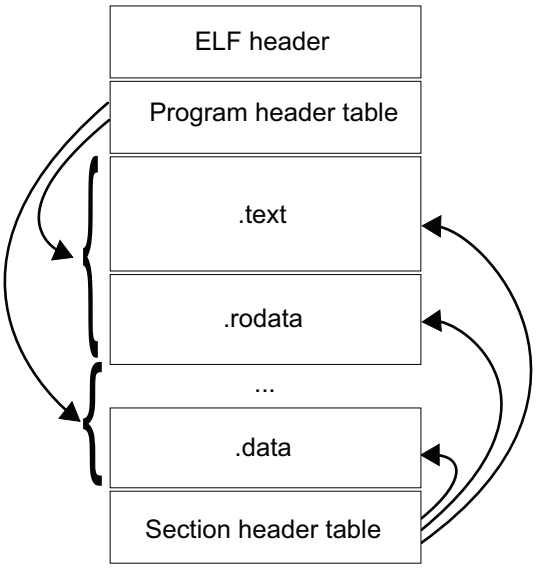
ELF struktura se týká souborů:

- `.o` – objektové soubory (výstupy kompilátoru a assembleru) – *object files*,
- `.out .elf` – spustitelné soubory (výstupy linkeru) – *executable files*,
- `.a .so` – archívy a sdílené knihovny – *archives, shared libraries*,
- `.dmp` – *core dumps*.

Vnitřní struktura souboru podle této normy je naznačena na obrázku 4.1.

ELF soubor obsahuje tyto části:

- *ELF header* – hlavička souboru,
- *Program header table* – popisuje segmenty, části dat použité při běhu programu,
- *Section header table* – popisuje jednotlivé sekce dat,
- datové sekce.



Obrázek 4.1: ELF layout (zdroj: [38]).

Každá sekce pojímá data stejného typu, např. v sekci `.text` se nachází spustitelná data, proměnné najdeme v sekci `.data`. V sekci `.bss` jsou proměnné, které v programu nemají počáteční hodnotu, je zde tedy (narozdíl od sekce `.data`) pouze uvedena velikost potřebné paměti.

Toto rozdělení do sekcí, namísto všech dat pohromadě (např. u spustitelných souborů MS-DOS), má své výhody v jednotném zpracování jednotlivých sekcí, čehož se navíc dobře využívá při linkování programu (kapitola 4.1.2).

4.1.2 GNU linker, linker script

Linker je využíván v jedné z finálních fází přípravy spustitelného souboru. Zpracovává všechny soubory `.o` a `.a` (viz 4.1.1), spojuje příslušné sekce, vybírá pouze použité symboly a funkce, spojuje veškeré reference mezi jednotlivými soubory apod.

GNU linker dokáže zpracovávat COFF i ELF soubory a pro jednotný přístup k souborům používá knihovny GNU BFD (Binary File Descriptor library).

Linker script

Běh linkeru je kontrolován tzv. *linker scriptem*, který mj. určuje, jak se jednotlivé sekce vstupních souborů mapují do výstupního souboru a také definuje rozvržení celého výstupu v paměti. Tyto úkoly se výrazně mění s platformou, proto jsou porozumění a úprava linker scriptu esenciálními úkony.

Z toho důvodu zde uvedeme alespoň základní operace a příklady jazyka linker scriptu (převzato z knihy o GNU linkeru Steva Chamberlaina [34]).

- skript může zakládat proměnné příkazem `symbol = expression;` včetně všech variant přiřazení známých z jazyka C, např.

  ```
  _stack_end = 0x20FFFC;
  ```

  Všechny proměnné založené linkerskriptem jsou přístupné z linkovaných objektů jako externí globální proměnné.

- Příkazem `MEMORY` definujeme názvy, adresy a velikosti pamětí dostupných na výstupní platformě. Příkaz má strukturu:

  ```
  MEMORY {    name [(attr)] : ORIGIN = origin, LENGTH = len     }
  ```

  například:

  ```
  MEMORY {    ROM : ORIGIN = 0x00100000, LENGTH = 256k
              RAM : ORIGIN = 0x00200000, LENGTH = 64k      }
  ```

- První instrukci programu, která se má spustit, tzv. *entry point*, můžeme nastavit pomocí příkazu

```
ENTRY(symbol)
```

`symbol` tedy představuje např. návěští v assembleru, nebo jméno funkce. Pokud tento příkaz nepoužijeme, za *entry point* bude považován symbol `start` (pokud je definovaný), nebo první instrukce sekce `.text` (pokud sekce existuje), nebo adresa 0.

- příkaz `SECTIONS` definuje mapování vstupních sekcí na výstupní sekce a jejich umístění v paměti. Formát příkazu je

```
SECTIONS {    sections-command
              ...                    }
```

příklad struktury příkazu `sections-command`:

```
.output_section [adress] [(type)] : [AT(LMA_adress)] {
                        *(.input_section)
                        ...
                        _end_output_section = .;
          } >mem_region [AT >LMA_mem_region]
```

Každá výstupní sekce má tzv. VMA (Virtual Memory Adress), což je adresa, kterou bude mít sekce při běhu programu, a LMA (Load Memory Adress), což je adresa, kam je sekce uložena. Typickým příkladem je sekce `.data` (proměnné), kterou chceme mít v paměti RAM při běhu programu. Paměť RAM je ale vždy smazána při odpojení systému od napájení, chceme tedy mít její inicializační obsah v paměti ROM (LMA) a při startu programu jej zkopírovat do paměti RAM (VMA).

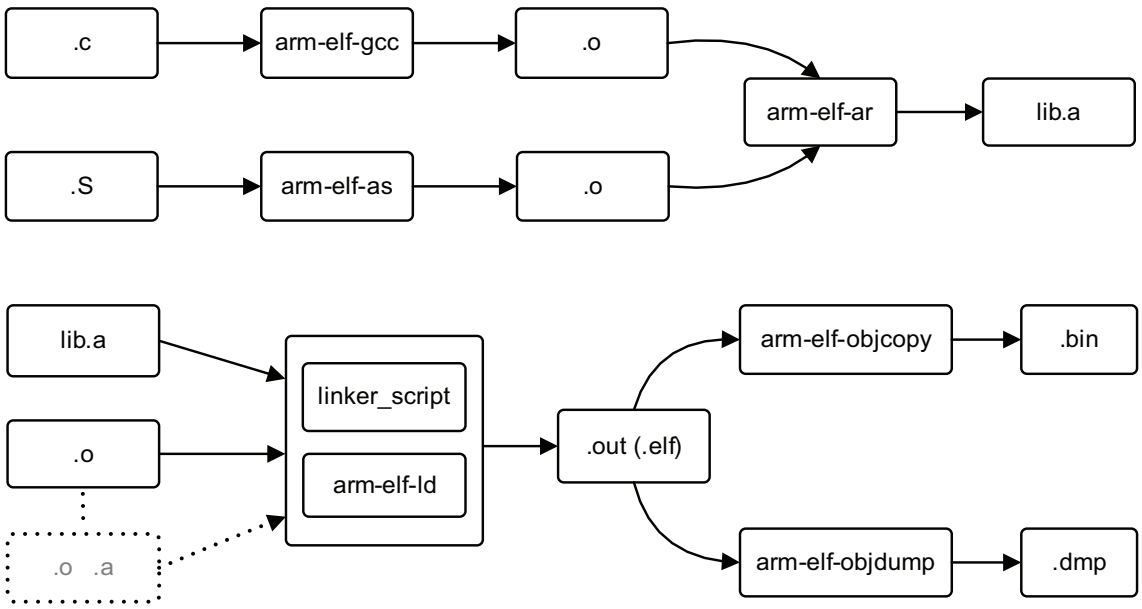
`Mem_region` je název paměti pro VMA a nepovinný parametr `adress` určuje její adresu. Jestliže není uveden, je pro adresu VMA použito lokální počítadlo daného oddílu paměti.

Nepovinné parametry `AT(LMA_adress)` resp. `[AT >LMA_mem_region]` určují adresu LMA resp. název paměti pro LMA.

Symbol „.“ je proměnná představující lokální počítadlo adresy v daném oddílu paměti. V naznačeném příkladu je do globální proměnné `_end_output_section` přiřazen obsah této proměnné, tzn. adresa v paměti, kde končí daná sekce (je to poslední příkaz).

4.1.3 Zpracování zdrojových souborů pomocí GNU ARM

Na obrázku 4.2 je vidět sekvence zpracování zdrojových souborů jednotlivými částmi vývojového nástroje a vznik spustitelného binárního souboru.



Obrázek 4.2: Zpracování zdrojových souborů pomocí GNU ARM.

Zdrojové soubory `.c` a `.S` (assembler) jsou zkompilovány do objektových ELF souborů `.o`, které je možné tzv. archivovat, tedy vytvořit z nich ELF knihovnu `.a`.

Objektové soubory spolu s knihovnami jsou pak slinkovány do výstupního „spustitelného“ ELF souboru `.out` (nebo `.elf`). Z tohoto výstupu je již vytvořen binární soubor `.bin` pro cílovou platformu.

Výhodným je také *dump* soubor `.dmp`, ve kterém najdeme přehledné informace o výstupu a průběhu zpracování, tedy informace o sekcích, adresy proměnných a funkcí, apod.

4.2 Newlib

Newlib je standardní C knihovna určená především pro embedded systémy. Nyní jí spravuje firma *Red Hat*. Dělí se na dvě části:

4.2.1 libc.a

Obsahuje standardní C funkce, jejichž definice najdeme ve standardních hlavičkách:

- `stdlib.h` – standardní funkce – `abs`, `atoi`, `malloc`, ...,
- `ctype.h` – makra a funkce znakových typů – `isascii`, `tolower`, ...,
- `stdio.h` – input, output – `read`, `write`, `printf`, ...,
- `string.h` – stringové a paměťové funkce – `strlen`, `memcpy`, ...,
- `wchar.h` – wide character string funkce – `wcslen`, `wmemcpy`, ...,
- `signal.h` – práce se signály – `raise`, `signal`, ...,
- `time.h` – časové funkce – `time`, `localtime`, ...,
- `locale.h` – lokální nastavení – `NULL`, `decimal_point`, ...,
- `iconv.h` – konverze řetězců – `iconv`, `iconv_open`, ...,

4.2.2 libm.a

Obsahuje nadstandardní matematické funkce, zejména pro výpočty s plovoucí řádovou čárkou, jako např. `log`, `sqrt`, `acosh`, apod.

Definice funkcí najdeme v hlavičce:

- `math.h`.

Předností knihovny Newlib je to, že může být zkompilována pro téměř jakoukoliv platformu. Pro správný běh celé knihovny je třeba implementovat pouze několik nízkoúrovňových systémových funkcí (viz kniha o nástrojích GNU pro embedded systémy od Rob Savoye [32]).

Tyto funkce se nazývají *System Calls*, mj. jde například o `read`, `write` pro možnost přesměrování výstupu `printf` a odvozených funkcí, nebo `sbrk` pro zvětšování haldy, čehož využívá funkce `malloc` (a jí příbuzné) pro dynamické alokování paměti (informace převzaty z manuálu knihovny Newlib Steva Chamberlaina [35]).

4.3 GNU debugger, JTAG a OpenOCD

4.3.1 GNU debugger

GNU debugger je program, který je dokáže sledovat, „co se děje uvnitř“ jiného programu. Je to především nástroj k hledání a odstraňování chyb v programu, který je schopen čtyř základních věcí:

- spustit program, ovlivňovat jeho chování,
- zastavit program při daných podmínkách,
- prozkoumat stav programu při zastavení,
- lokálně měnit program (obsah proměnných), tedy experimentovat s chybami.

Takto lze testovat program buď lokálně, nebo vzdáleně pomocí sériové linky či TCP/IP spojení, čehož se využívá zejména k debuggování embedded systémů.

GNU debugger používá pro vzdálené debuggování tzv. *GDB Remote Serial Protocol* (informace převzaty z GDB manuálu Richarda Stallmana [30]).

4.3.2 JTAG

JTAG (Joint Test Action Group) je vlastní název standardu IEEE 1149.1, vyvinutého pro tzv. *boundary scan*, tj. vzdálené testování vodivých spojů DSP a bloků uvnitř integrovaných obvodů (viz [38]).

JTAG je s výhodou možné použít také jako transportní mechanismus mezi programátorem a debuggovací jednotkou uvnitř integrovaného obvodu, tedy použít jej k debuggování embedded systémů.

JTAG využívá v zásadě těchto pět signálů:

- TDI (Test Data In),
- TDO (Test Data Out),
- TCK (Test Clock),
- TMS (Test Mode Select),
- TRST (Test Reset) – nepovinný.

4.3.3 OpenOCD

Open On-Chip Debugger (OpenOCD) je *open-source* software zaměřen na debuggování a In-System programování (programování integrovaného obvodu, který je již zakomponován v systému - na DPS).

Podporuje několik typů JTAG hardwaru, např. paralelní Wiggler, USB převodníky založené na FTDI FT2232, Amontec JTAG Accelerator a další (viz OpenOCD manuál [27]). Tento software se ale stále vyvíjí a roste počet typů podporovaného hardwaru.

OpenOCD je zatím zaměřeno na ARM platformy s jádry ARM7, ARM9, XScale a Cortex-M3.

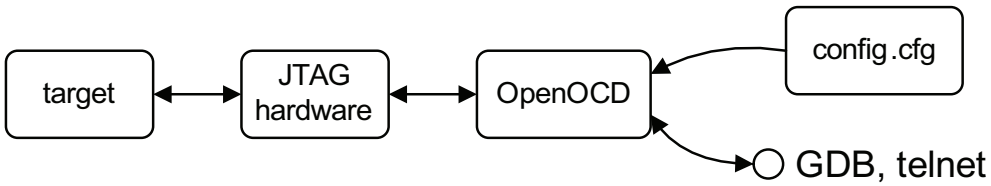
Program pracuje jako *daemon*, který akceptuje příchozí připojení přes telnet nebo GDB port, např. tedy překládá požadavky GDB debuggeru na JTAG signály.

Runtime konfigurace tohoto programu se provádí pomocí externího skriptu (příklad viz [27]).

Zde jsou popsány dva nejčastější pracovní modely OpenOCD:

Remote – GDB, telnet

Program je nastaven konfiguračním skriptem při spuštění programu. Pak je možné připojit se k němu pomocí TCP/IP nebo GDB portu. Tento mód je určen pro debuggování nebo „ruční práci“ (posílání příkazů přes telnet). Pracovní schéma v módu *remote* je na obrázku 4.3.

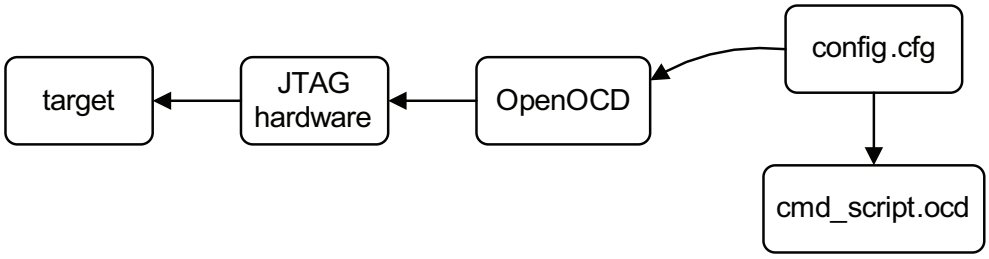


Obrázek 4.3: OpenOCD pracovní mód *remote*.

Lokální skript

V konfiguračním skriptu lze vnutit OpenOCD skript s příkazy, které má provést. To je vhodné především pro naprogramování flash paměti uvnitř integrovaného obvodu. V příkazovém skriptu stačí zadat sekvenci typu „načti soubor `firmware.bin`“, „naprogramuj paměť“ a „skonči“. Celý proces pak lze provést vhodným makefilem, `.bat` souborem, nebo jedním příkazem na příkazové řádce.

Pracovní schéma v módu *local script* je na obrázku 4.4.



Obrázek 4.4: OpenOCD pracovní mód *local script*.

4.4 YAGARTO

YAGARTO (Yet Another GNU Arm TOolchain) je neoficiální distribuce (kompilace) balíku GNU ARM Toolchain obsahující:

- GCC kompilátor, GNU Binutils, Newlib a Insight debugger,
- vlastní kompilaci OpenOCD,
- Eclipse IDE, Eclipse CDT a Zylin CDT plugin pro GDB embedded debuggování.

Distribuce je určena pro Windows a zkompilována pomocí MinGW (viz [11]), narozdíl od oficiální distribuce (CodeSourcery) zkompilované pomocí Cygwin tedy nepotřebuje další vrstvu knihoven, používá nativní Windows API.

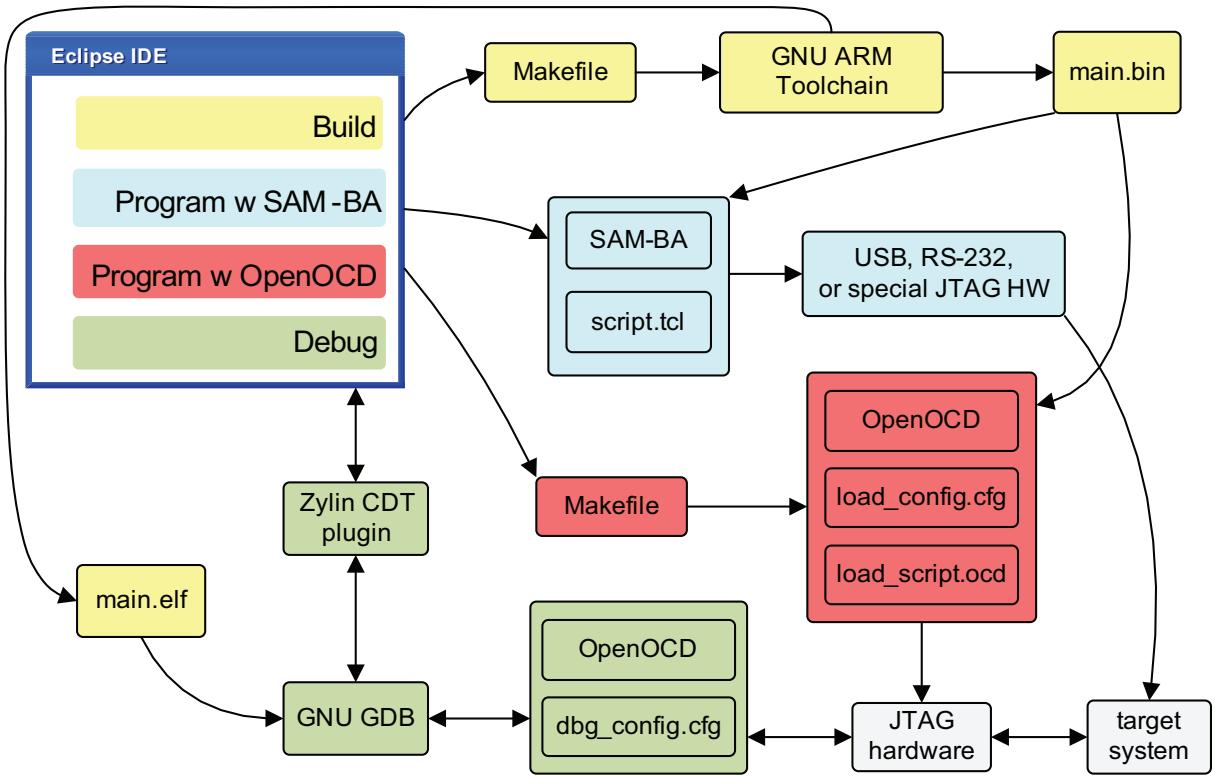
Její nevýhodou je, že zatím nepodporuje jádra Cortex-M3.

Výhodou je integrace OpenOCD a pokročilého prostředí Eclipse IDE s pluginem Zylin CDT, se kterým je možné velmi pohodlně debuggovat program přímo na čipu v prostředí Eclipse (samozřejmě s využitím GNU GDB).

Mikroprocesory Atmel AT91SAM dále nabízí možnost využít k přepsání flash paměti program SAM-BA (SAM Boot Assistant), který je schopen připojit se k implicitnímu bootloaderu na čipu a naprogramovat flash paměť pomocí USB, RS-232 nebo JTAGu (se speciálním hardwarem).

Tento proces lze také integrovat do Eclipse IDE. SAM-BA lze spustit z příkazové řádky s vhodnými parametry a TCL skriptem s příkazy pro naprogramování cíle.

Pracovní diagram všech nástrojů obsažených v distribuci je znázorněn na obrázku 4.5.



Obrázek 4.5: Pracovní diagram GNU ARM toolchainu v distribuci YAGARTO.

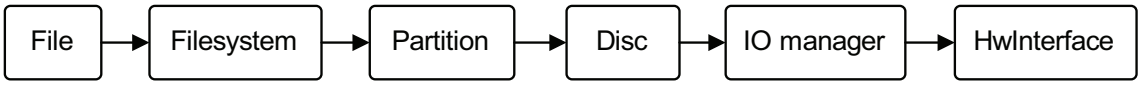
4.5 EFSL

Embedded FileSystems Library (EFSL) je *open-source* knihovna funkcí pro podporu souborových systémů na různých embedded zařízeních. Knihovna se neustále vyvíjí, zatím podporuje souborové systémy rodiny Microsoft FAT tedy FAT12, FAT16 a FAT32. Nutné a postačující minimum volné operační paměti pro běh knihovny je 1 kilobyte.

Předností knihovny je schopnost pracovat téměř s jakýmkoliv druhem média, ze kterého se dá číst/zapisovat. Takovým médiem může být disk, paměťová karta, část paměti, nebo třeba soubor (viz EFSL manuál [17]).

Knihovna udržuje cache sektorů (do uživatelem definované velikosti) a je schopna minimalizovat přístupy do FAT tabulky, tím šetří přímý přístup k médiu.

Vnitřní struktura (vrstvy, hierarchie) knihovny vychází z reálné hierarchie systému s pevným diskem, viz obrázek 4.6.



Obrázek 4.6: Vnitřní hierarchie knihovny EFSL.

Vrstvy *File*, *Filesystem*, *Partition* a *Disc* odráží své reálné jmenovce (parametry, způsob adresování), vrstva *IO manager* spravuje cache sektorů a v případě potřeby provádí zápis/čtení z hardwarové vrstvy. *HwInterface* zajišťuje fyzický přenos dat z/do média a její implementace je (v zásadě) zodpovědností uživatele.

Před použitím knihovny je nutné:

- implementovat několik read/write funkcí (nebo použít existující, např. obsažené v knihovně) umožňujících zápis/čtení sektoru (512 bytů) paměti ze zvoleného média,
- upravit konfigurační hlavičku, zde mj. definovat *endpoint* (tedy které read/write funkce se mají použít), nastavit velikost cache, nastavit, zda dokáže systém podporovat datum a čas, apod.
- v případě, že cílový systém podporuje přesné datum a čas (např. obsahuje obvod reálného času), implementovat 6 funkcí k jeho získání,
- knihovnu zkompilovat, nebo jí zahrnout do rozpracovaného projektu.

Pro naši aplikaci byla použita verze, která zahrnuje i ovladač Secure Digital karty přes SPI rozhraní. Tento ovladač byl upraven k využití HAL a použit v naší aplikaci (více v kapitole 6.4.3).

Dále byla pro účely naší úlohy (viz kapitola 2) do vrstvy *Filesystem* implementována chybějící funkce pro smazání celého obsahu adresáře včetně všech podadresářů, vhodná tedy například pro smazání obsahu celého média (viz dokumentace softwaru v příloze B).

Kapitola 5

Hardware

V této kapitole přiblížíme detaily hardwarového řešení, především vlastnosti použitého mikrokontroléru, ostatních integrovaných obvodů a aspekty návrhu hardwarové části zařízení.

5.1 Mikrokontrolér AT91SAM7S256

5.1.1 ARM

První procesor s názvem ARM vznikl r. 1985 ve společnosti Acorn Computers Ltd. ve Velké Británii. Vytvořila jej skupina postgraduálních studentů v čele se Stevem Furberem na základě Berkeley RISC designu.

ARM byla původně zkratka pro Acorn RISC Machines (nyní Advanced RISC Machines). Jednalo se o vůbec první RISC procesor určený pro komerční využití (informace převzaty z knihy o architektuře ARM od Steva Furbera [36]).

RISC

Veškeré tehdejší dostupné procesory měly architekturu CISC (Complex Instruction Set Computer), která s sebou nese mnoho nevýhod. Rysy RISC (Reduced Instruction Set Computer) architektury (narozdíl od CISC) jsou především:

- instrukční sada se sestává z menšího počtu jednoduchých instrukcí, které mají jednotnou šířku a formát,

- instrukce jsou vykonávány (většinou) v jednom hodinovém cyklu,
- k dispozici jsou identické registry, které je možné použít v jakémkoliv kontextu, což zjednodušuje práci kompilátoru,
- *load-store* architektura, kde instrukce zpracovávající data pracují pouze nad registry a jsou oddělené od instrukcí, které přistupují do paměti (*load*/*store*),
- menší počet hardwarových datových typů.

Z těchto vlastností plynou výhody, jako např. jednodušší architektura díky jednotnému dekódování a zpracování instrukcí, tedy menší počet tranzistorů, více místa pro registry, jednodušší a rychlejší návrh a také překvapivě větší výkon (ač s použitím jednoduchých instrukcí) díky schopnosti procesoru běžet na vyšší frekvenci.

Nevýhody RISC procesorů jsou především chabá hustota kódu kvůli jednotné šířce instrukcí a jejich přílišné jednoduchosti a také obecně neschopnost zpracovávat x86 kód – což je nevýhoda pouze ve světě *IBM compatible* počítačů (zdroj: článek Pavla Píšy o moderních mikrokontrolérech [25]).

Vlastnosti

ARM je nativně 32-bitové RISC jádro, používá Von Neumannovu architekturu, tedy jednu sběrnici a pole registrů pro adresy i data. Obsahuje ALU (Arithmetical Logical Unit) pro sčítací operace a MAC (Multiply-Accumulate Unit) pro násobení, nepoužívá mikrokód.

Jeho základní instrukční sada obsahuje 23 instrukcí šířky 32 bitů. Některé ARM architektury podporují i další instrukce, nebo jiné instrukční sady (viz [3]).

Módy

Procesor pracuje v tzv. módech. Ty se dělí na privilegované, jimiž jsou: *Abort*, *Fast Interrupt Request (FIQ)*, *Interrupt Request (IRQ)*, *Supervisor*, *Undefined* a neprivilegované *User* a *System* mód.

Privilegované módy umožňují jakýkoliv přístup do Status Registru, neprivilegované módy z něj umožňují pouze číst, tzn. nemohou přímo měnit mód.

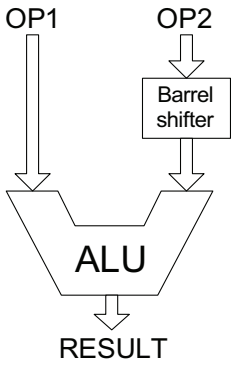
Stav procesoru včetně aktuálního módu je uložen v registru Processor Status Register.

Procesor obsahuje 16 registrů, přičemž každý mód má vlastní Stack Pointer a Link Register (viz kapitola 5.1.1). FIQ má více vlastních registrů (viz [2]).

Unikátní vlastnosti

- **Podmínečné vykonání každé instrukce** – první 4 bity instrukce obsahují podmínku jejího zpracování, což je většinou doménou skokových instrukcí. Tento model šetří skoky, které jsou pomalejší (musí se vyprázdnit pipeline), viz [26].

- **Barrel-shifter** – umožňuje předzpracování jednoho operandu ve smyslu shiftovacích operací (shift, rotation) a obecné zpracování v ALU v jednom kroku. To dovoluje menší počet komplexnějších instrukcí. Instrukční sada jinak zcela postrádá instrukce pro shiftovací operace (viz [26]).



- **Módy přerušení** – jednoduchý ale rychlý systém přerušení s dvěma úrovněmi priorit, vlastními módy a registry.

- **Link register** – speciální registr určen (například) pro návratovou adresu z funkce.

- **Thumb, Jazelle DBX** – některé vyšší řady ARM procesorů umí zpracovávat víc instrukčních sad. Především sadu s názvem *Thumb*, která má 16-bitovou šířku a dovoluje tak lepší hustotu kódu.

  Zajímavostí je *Jazelle DBX* (Direct Bytecode eXecution), což je hardwarová podpora Java Bytekódu, tedy jakýsi hardwarový Java Virtual Machine (viz [2]).

Přerušení a výjimky

V jádrech ARM může dojít k osmi typům přerušení, které se v literaturách [2] [36] označují jako výjimky. Když dojde k výjimce, procesor nastaví *program counter* na příslušnou pozici v tabulce vektorů přerušení (výjimek) umístěné na adrese 0x0. Tyto výjimky jsou stejné pro všechny druhy ARM procesorů.

Jedná se o:

1. *Reset* – při softwarovém nebo hardwarovém resetu (spouští se v módu Supervisor),
2. *Undefined* – při pokusu spustit neznámou instrukci (spouští se v módu Undefined),
3. *Software Interrupt* – softwarové přerušení (spouští se v módu Supervisor),
4. *Prefetch Abort, Data Abort* – odmítnutí při snaze vykonat instrukci na neznámé adrese, nebo získat data z neznámé adresy (spouští se v módu Abort),

5. *Reserved* – reservovaná výjimka pro pozdější designy,

6. *Interrupt Request* – přerušení různých typů, od periferií, externí přerušení (spouští se v módu IRQ),

7. *Fast Interrupt Request* – rychlé přerušení, určené pro rychlou odezvu na externí signál (spouští se v módu FIQ).

5.1.2 ARM7TDMI

Toto low-end low-cost jádro, které používají procesory AT91SAM7S, je momentálně jedno z nejrozšířenějších. Jeho design vychází z jádra ARM6.

Původ jeho názvu vychází z toho, že obsahuje:

- podporu pro 16-bitovou instrukční sadu **T**humb,
- podporu pro on-chip **D**ebuggování,
- vylepšenou násobičku (**M**ultiplier) s 64-bitovým výstupem,
- blok Embedded**I**CE podporující hardwarové breakpointy a watchpointy.

Procesor zpracovává instrukce s pomocí tříúrovňové pipeline (*Fetch*, *Decode*, *Execution*), má rozhraní pro coprocesor, JTAG rozhraní, aj. (viz [2]).

5.1.3 Vlastnosti AT91SAM7S256

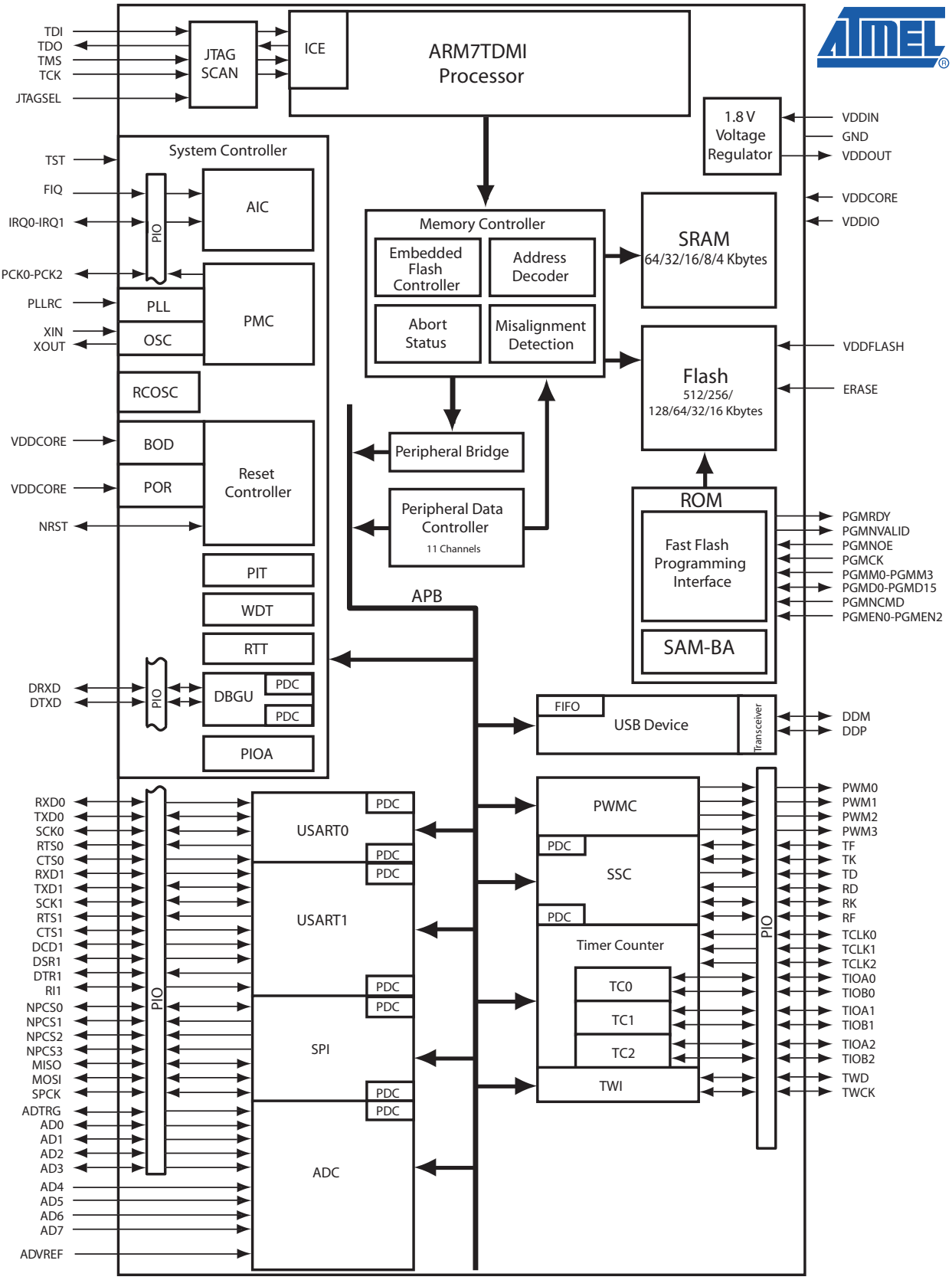
Jednočipový mikroprocesor Atmel AT91SAM7S256 mimo jiné obsahuje:

- 256kB flash paměti pro uložení programu, 64kB paměti SRAM,
- *Power Management Controller* – vypínání periferií, změny frekvence procesoru, idle mód,
- *Advanced Interrupt Controller* – osm úrovní priorit přerušení, vektorování obsluhy,
- *Debug Unit* – dvouvodičový UART, debuggovací výpis, může komunikovat s debuggovací jednotkou,
- *Periodic Interval Timer* – 20-bitový + 12-bitový čítač,

- *Watchdog* – 12 bitový čítač, reset, přerušení,
- *Real-Time Timer* – „free running“ 32-bitový čítač vteřin, interní RC oscilátor,
- *Parallel I/O Controller* – přerušení při změně stavu I/O, open-drain, pull-ups,
- *Peripheral DMA Channel* – DMA pro 11 periferií,
- USB 2.0 Full-speed, 12 Mb/s, 328 bytů FIFO,
- *Serial Synchronous Controller* – kompatibilní s I2S,
- 2× USART, baud rate generátor, IrDA mód, RS-485 mód, Hardwarové řízení toku, full-modem na jedné lince,
- *Serial Peripheral Interface* – master/slave, 8-16 bit šířka, 4x Chip Select,
- 3× Timer/Counter – 16 bitů, externí hodinové vstupy,
- 4× PWM Channel – 16 bitů,
- *Two Wire Interface* – master mód, kompatibilní s I2C,
- 8× ADC – 10 bitů,
- implicitní *boot program* uložený ve zvláštní paměti ROM, který je možné kdykoliv obnovit.

Tento mikroprocesor má vyvedených 32 I/O vodičů, které tolerují TTL úrovně. Vstupní napětí jádra je 1,8V, vstupní napětí paměti a periferií je 3,3V. Pouzdro obsahuje napěťový regulátor 3,3V → 1,8V.

Na obrázku 5.1 je blokové schema celého mikrokontroléru, procesor je zde znázorněn pouze jako blok.



Obrázek 5.1: Blokové schema mikrokontroléru řady AT91SAM7S64-512, převzato z [8].

Peripheral DMA Controller

Zajímavostí tohoto obvodu je DMA (Direct Memory Access) kontrolér, který může přenášet data přímo mezi periferiemi a pamětí.

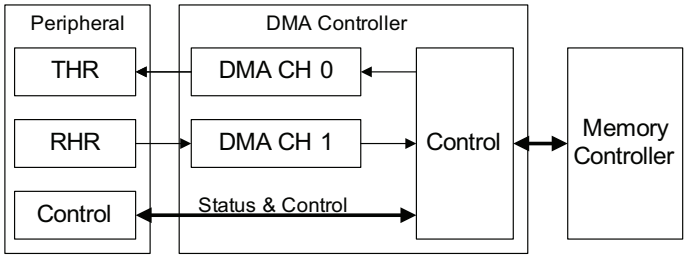
DMA kanál je pro dané periferie organizován v párech, tedy každému směru (vysílání/přijímání) je určen jeden kanál. Každý kanál obsahuje dvojici *pointer register – counter register* a dvojici *next pointer register – next counter register*.

Chceme-li zahájit přenos, stačí do *pointer registru* zapsat adresu v paměti odkud chceme vysílat data (resp. kam přijímat) a do *counter registru* zapsat počet bytů k přenosu (nebo jiných jednotek přirozených dané periferii). Přenos dat pak již probíhá zcela automaticky. Pokaždé se inkrementuje *pointer register* a dekrementuje *counter register*.

Na konci se buď vyvolá příslušné přerušení (pokud je povoleno), nebo nahrají nové hodnoty z *next pointer* a *next counter* registrů, tím se opět šetří intervence procesoru.

Při setkání DMA požadavků z více periferií v jeden okamžik je uplatněna určitá hierarchie priorit. Více informací je k dispozici v dokumentaci k danému mikrokontroléru [8].

Schema spojení mezi periferií a pamětí přes DMA kanál je naznačen na obrázku 5.2.



Obrázek 5.2: DMA přístup pro periferie.

5.2 PCF8563

Tento integrovaný obvod reálného času (RTC) udržuje přesný čas a datum (konkrétně rok, měsíc, den, hodinu, minutu a vteřinu).

Jeho hodinový signál je řízen externím krystalem s typickou frekvencí 32.768kHz.

Statické parametry obvodu jsou:

- Maximální vstupní napětí obvodu je $V_{DDmax} = 5.5\text{V}$, minimální je $V_{DDmin} = 1.8\text{V}$

při udržení komunikace na sběrnici, a $V_{DDmin} = 1$V při udržení čítače v chodu, kde V_{DD} je vstupní napětí obvodu.

- Vstupní proud I_D je ovlivněn vstupním napětím a frekvencí na sběrnici. Maximální hodnota je $I_D = 800\mu$A pro $f_{SCL} = 400$kHz a $V_{DD} = 5.5$V, kde I_D je vstupní proud, V_{DD} je vstupní napětí a f_{SCL} je hodinová frekvence I^{2}C sběrnice.

  Zdrojem těchto informací je dokumentace k danému obvodu [21].

5.2.1 I^{2}C

Tento obvod komunikuje s mikrokontrolérem po sériové synchronní dvouvodičové 8-bitově orientované sběrnici **I^{2}C** (Inter-Integrated Circuit). Ta je de facto standardem připojení obvodů jako jsou paměti EEPROM, RAM, LCD displeje, analogově digitální převodníky nebo RTC obvody.

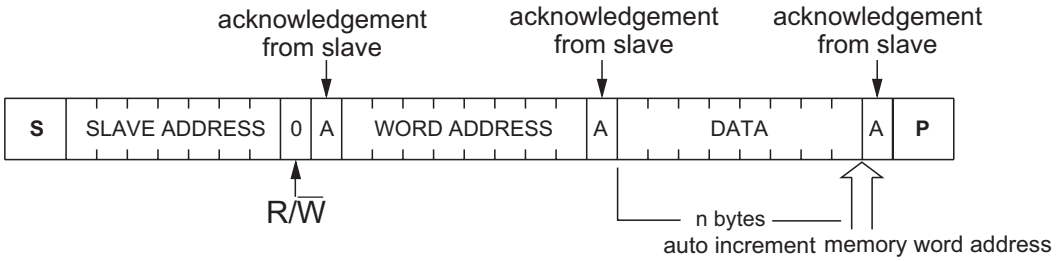
Sběrnice je určena pro obousměrnou komunikaci typu MASTER/SLAVE, skládá se pouze ze dvou vodičů:

- SDA - datový I/O vodič, adresa i data,
- SCL - hodinový signál.

Tyto vodiče je nutné připojit na napájecí napětí přes pull-up rezistory.

SLAVE zařízení je adresováno buďto 7-bitovou nebo 10-bitovou vnitřní adresou, což určuje počet zařízení, které je možné na sběrnici připojit (viz specifikace I^{2}C sběrnice [22]).

Obvod PCF8563 má pevnou 7-bitovou adresu (osmý bit vždy určuje zápis, nebo čtení), při komunikaci se chová jako standardní EEPROM. Jeho paměť je rozdělena na šestnáct 8-bitových registrů obsahujících stavové, kontrolní, časové a alarmové registry.



Obrázek 5.3: Komunikační schema na sběrnici I^{2}C, převzato z [21].

Komunikace je vždy navozena masterem startovací sekvencí, následuje nepovinná vnitřní adresa cílového zařízení a data. U každého bytu je potvrzován příjem.

Základní komunikační diagram je na obr. 5.3. Přenos vždy začíná adresou SLAVE zařízení, následuje adresa v jeho vnitřní paměti a data.

5.3 Návrh hardwaru

Návrh hardwarové části vychází částečně z vývojové desky SAM7S-P64 od firmy Olimex (viz [24]). Obvodové schema celého zařízení je v příloze A.

5.3.1 RS-232 periferie

Zařízení obsahuje dva porty RS-232 připojené k UART periferiím mikroprocesoru přes budiče sběrnice resp. převodníky napěťových úrovní (TTL ↔ EIA-232), konkrétně jde o dva obvody ADM3203, každý z nichž obsahuje dva dvousměrné kanály.

Tyto integrované obvody pracují se vstupním napětím 3.3V, z externích součástek je nutných pouze pět kondenzátorů s kapacitou 100nF, viz dokumentace k ADM3202 [1].

5.3.2 JTAG

Zařízení je doplněno standardním 20-pinovým konektorem typu JTAG ARM (viz [33]) pro programování vnitřní flash paměti mikroprocesoru a on-chip debuggování. Tento konektor obsahuje kromě základních JTAG signálů:

- TDI (input), TDO (output), TCK (input), TMS (input) – datové, hodinové komunikační signály,
- nTRST (input/output) – reset debuggovací jednotky,

také signály:

- nSRST (System Reset – input/output) – reset celého systému,
- RTCK (Return Test Clock – output) – synchronizace při velkých změnách pracovní frekvence mikroprocesoru,

- DBGRQ (Debug Request – input) a DBGACK (Debug Acknowledge – output) – požadavek a řízení debuggování.

Signály DBGRQ a DBGACK jsou v jádrech ARM externě vyvedeny pouze ve výjimečných případech, většinou jsou prováděny interně jako JTAG signály (viz [4]). Stejně tak je tomu u mikroprocesorů rodiny AT91. Tyto signály tedy z konektoru nejsou dále vyvedeny.

Zpětnovazební signál RTCK, který řídí frekvenci TCK v závislosti na frekvenci mikroprocesoru není používán v jádrech ARM7TDMI, namísto toho je třeba zvolit dostatečně nízkou frekvenci TCK, aby bylo možné komunikovat i při nižších frekvencích mikroprocesoru. Tento signál je v našem zařízení přímo spojen s TCK.

Mikroprocesory řady AT91 nemají zvášť vyveden pin pro reset debuggovací jednotky, ale pouze pin pro reset celého systému včetně všech periferií. Tento *open-drain* vstupně/výstupní pin (NRST) může být ovládán buďto vnitřně resetovací jednotkou jako reset všech periferií a signál pro externí debuggovací zařízení, nebo externě pro reset celého systému (viz dokumentace mikrokontroléru [8] a blokové schema 5.1).

JTAG signál nTRST je tedy bezúčelný, použitý je pouze signál nSRST, který je přímo připojen k pinu NRST (s vnitřním permanentním pull-up rezistorem).

Celé zařízení je také doplněno resetovacím tlačítkem zapojeným mezi zmíněný vodič a zem.

5.3.3 Vývojové a testovací piny

Na DPS jsou z mikrokontroléru také vyvedeny piny

- ERASE – pro smazání obsahu flash paměti,
- TST – pro obnovení boot programu SAM-BA v paměti flash,
- JTAGsel (BDS) – pro boundary scan,

všechny tyto piny obsahují permanentní vnitřní pull-down rezistor, viz dokumentace mikrokontroléru [8].

5.3.4 LCD řadič a konektor

Použitý LCD displej Displaytech 64128 je řízen řadičem KS0713 . Tento řadič již obsahuje obvody nutné pro buzení samotného displeje, stejně jako vlastní hodinový signál.

Potřebuje pouze jedno napájení pro buzení logiky i displeje, a to 3.3 V. Minimalizuje se tak počet externích součástek celého displeje (žádné nejsou nutné).

Řadič také obsahuje vnitřní paměť, která je přesným logickým otiskem zobrazeného pole, což zjednodušuje uvažování a návrh ovladače.

S mikrokontrolérem může komunikovat pomocí:

- paralelního rozhraní – 14 signálových vodičů, výběr ze dvou režimů (6800, 8080), zápis i čtení do vnitřní grafické paměti,
- sériového rozhraní – 5 signálových vodičů, jednodušší protokol, pouze zápis do vnitřní grafické paměti.

Kvůli jednoduchosti a menšímu počtu nutných vodičů bylo použito sériové rozhraní. To je založeno na pěti signálech:

- SID (Serial Data Input),
- SCLK (Serial Clock),
- CS (Chip Select),
- RESET,
- DI (Data/Instruction).

Informace jsou převzaty z dokumentace k displeji Displaytech 64128 [18] a k řadiči KS0713 [10].

Celé rozhraní displeje včetně napájení a vývodů podsvěcovací diody je umístěno do standardního 10-pinového konektoru, aby bylo možné umístit displej libovolně na konstrukční krabičce.

5.3.5 Ovládací tlačítka

Uživatelský vstup zařízení tvoří čtveřice ovládacích tlačítek, každé z nichž je připojeno mezi zem a I/O pin mikroprocesoru s vnitřním pull-up rezistorem. Z praktických důvodů jsou fyzicky umístěny na samostatné DPS zpod předního panelu konstrukční krabičky a k hlavní DPS připojeny přes standardní 6 pinový konektor obsahující také vývody pro možné podsvícení tlačítek.

5.3.6 Spotřeba zařízení

Celkovou teoretickou spotřebu zařízení můžeme odhadnout na základě statických parametrů použitých součástek. Majoritní podíl na celkové spotřebě bude mít:

- mikroprocesor – max $I_{CORE}\doteq$50mA, max $I_{FLASH}\doteq$10mA (viz [8]),
- budiče RS-232 – zkratový proud na výstupu vysílače je $I_{TxSC}\doteq$15mA (viz [1]),
- Secure Digital karta – proud při zápisu do paměti je $I_W\doteq$75mA (viz [31]),
- podsvěcovací led diody displeje a tlačítek – odpory omezují toto celkové maximum na přibližně $I_{LED}\doteq$100mA.

Celkový maximální vstupní proud do zařízení tedy může být teoreticky okolo $I_C\doteq$300mA (skutečná hodnota je o mnoho menší).

5.3.7 Napájení

Veškeré logické obvody byly vybrány pro vstupní napětí $V_{DD} = 3.3$V. Napájení obvodu je zajištěno pouze jedním lineárním regulátorem TPS79533 (tzv. Low-dropout linear voltage regulator, LDO). Výstupní napětí tohoto regulátoru je samozřejmě $V_{OUT} = 3.3$V, vstupní napětí V_{IN} může být v rozsahu od $V_{OUT} + V_{DO}$ až 6V, kde V_{DO} je tzv. *Drop-out Voltage* (minimální rozdíl výstupního a vstupního napětí), které u obvodu TPS79533 činí pouze $V_{DO} = 110$mV při plném zatížení (viz dokumentace obvodu [15]).

Napájení celého zařízení je možné buďto externím stejnosměrným zdrojem (5V–6V), z USB portu (5V), nebo ze tří baterií typu LR3 (celkem 4.5V).

Vodiče z těchto tří zdrojů jsou přes ochranné diody *D1*, *D2* a *D3* spojeny do jednoho uzlu a přes hlavní vypínač svedeny do LDO (viz schema obvodu v příloze A). Toto řešení jednak zaručuje, že proud nemůže protékat mezi zdroji, a zároveň je vždy vybrán zdroj s nejvyšším napětím (např. při připojení napájení z USB se automaticky odpojí baterie).

5.3.8 RTC

Napěťový vstup obvodu reálného času je spojen s katodami vstupních ochranných diod, což zaručuje napájení obvodu i při vypnutí zařízení (z baterie při odpojení vnějších zdrojů).

5.3.9 Velikost chladící plochy LDO

Lineární napěťový regulátor vydává tepelnou energii, jejíž velikost je úměrná celkové „promrhané“ energii, tedy rozdílu vstupního a výstupního napětí (viz dokumentace k TPS795xx [15]). Takto vydávanou průměrnou energii je možné vyjádřit jako

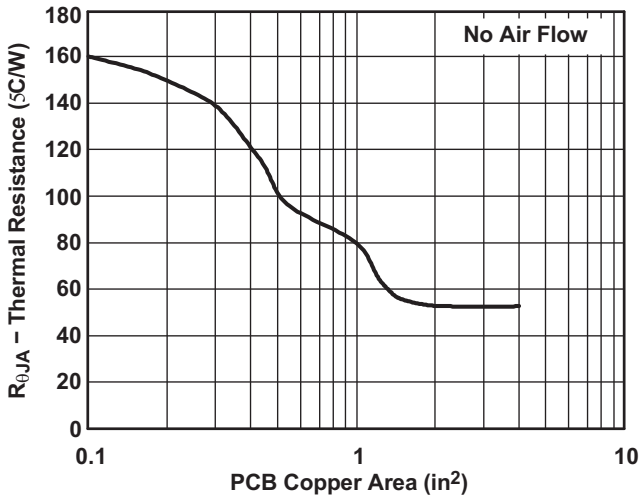
$$P_D max = (V_{IN} - V_{OUT}) \cdot I_{OUT} + V_{IN} \cdot I_Q, \tag{5.1}$$

kde $P_D max$ (Dissipated Power) je výkon promarněné energie, V_{IN} resp. V_{OUT} je vstupní resp. výstupní napětí regulátoru, I_{OUT} je výstupní proud a I_Q (Quiescent Current) je klidový proud.

Tuto energii lze odvádět buďto důkladným odvětráváním, přídavným chladičem, nebo dostatečně velkou měděnou plochou pod regulátorem, což je náš případ. Potřebnou velikost takové plochy lze odhadnout z grafu 5.4, kde $R_{\Theta JA}$ je tzv. tepelná rezistence mezi spojem a pouzdrem integrovaného obvodu, kterou lze odhadnout jako

$$R_{\Theta JA} = \frac{T_J - T_A}{P_D max}, \tag{5.2}$$

kde T_J je teplota chlazeného spoje a T_A je teplota okolí.



Obrázek 5.4: Závislost tepelné rezistence na velikosti chladící plochy LDO.

Vstupní napětí V_{IN} uvažujeme maximálně $V_{INmax} = 6$V, výstupní napětí je $V_{OUT} = 3.3$V. Maximální výstupní proud uvažujeme okolo $I_{OUT} = 300$mA (viz kapitola 5.3.6) a hodnota I_Q v rovnici 5.1 se pro obvod TPS79533 pohybuje okolo $I_Q \doteq 1\mu A$, můžeme ji tedy zanedbat.

Budeme-li uvažovat $T_J = 125°\text{C}$ a např. $T_A = 25°\text{C}$, teplotní rezistenci spočítáme jako

$$R_{\Theta JA} = \frac{T_J - T_A}{(V_{IN} - V_{OUT}) \cdot I_{OUT}} \doteq 120°\text{C/W} \tag{5.3}$$

a obsah potřebné chladící plochy bude přibližně

$$S_{hs} = 1.5\text{cm}^2. \tag{5.4}$$

Kapitola 6

Software

V této kapitole přiblížíme detaily softwarového řešení naší úlohy. Nejdříve probereme spouštěcí sekvenci procesoru a jeho nutná nastavení. Dále pak model přerušení, popis implementovaných ovladačů a utilit a na závěr koncepci hlavní aplikace.

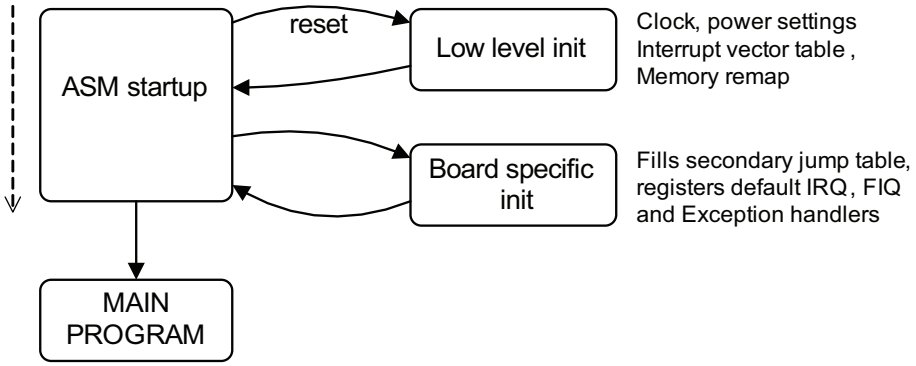
6.1 Architektura

Zvolili jsme tedy architekturu bez operačního systému, tzv. *bare-metal*, rozdělení programu do logických částí najdeme v kapitole 3.2. Nejdůležitější nízkoúrovňové části programu jsou založeny na sérii článků [20] Mirko Samka ze společnosti Quantum Leaps. Z těchto zdrojů je použit především upravený *startup (startovací) kód*, upravený model přerušení a struktura zásobníku včetně jeho umístění v paměti.

Velikosti, adresy pamětí (ROM, RAM), umístění jednotlivých ELF sekcí (kapitola 4.1.1) a odvozené důležité globální proměnné jsou definovány v linkerscriptu (kapitola 4.1.2), fyzické umístění zásobníků je definováno ve *startup kódu*.

6.2 Startup kód

Procesor při spuštění automaticky skočí do paměti na adresu 0x0 (v tu chvíli ROM, viz 6.2.2), kde je umístěn assemblerovský kód, generický pro ARM platformu. Na začátku je samozřejmě umístěna tabulka vektorů přerušení (dočasná), kde první položkou je skok do resetovací procedury.



Obrázek 6.1: Startup sekvence.

Resetovací sekvence je naznačena na obrázku 6.1 a popsána níže. Před skokem do hlavní aplikace se nejprve provede nízkoúrovňová inicializace, poté nastavení příslušné dané platformě a nastavení zásobníků a RAM sekcí.

6.2.1 Low level init

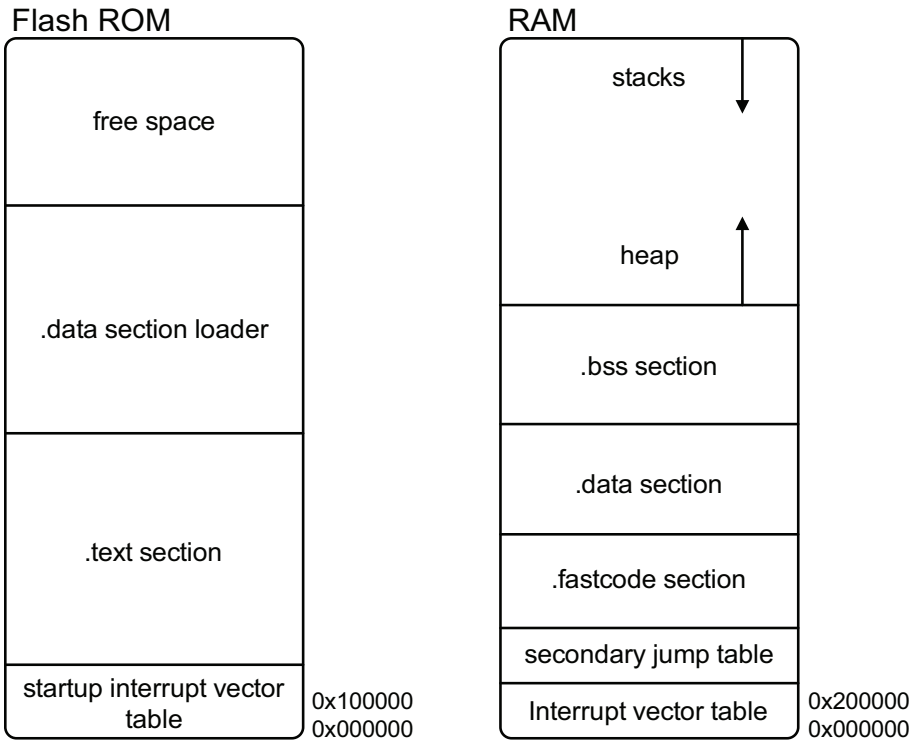
Resetovací procedura nejdříve provede skok do tzv. *low level inicializace*, která není generická, je závislá na specifické ARM platformě. Zde se provede důležité nastavení hodin, power managementu a naplní se v paměti RAM primární i sekundární tabulka vektorů přerušení (kapitola 6.2.3) dočasnými hodnotami. Nakonec se provede důležitý krok, tzv. přemapování paměti (*memory remap*).

6.2.2 Memory remap

Mikroprocesory rodiny AT91SAM7S obsahují dva typy pamětí, permanentní pomalejší flash ROM a rychlejší SRAM. Tyto paměti jsou v každém okamžiku přístupné na adresách 0x100000 (ROM) a 0x200000 (RAM). Obě tyto paměti lze pomocí periferie *memory controller* střídavě mapovat na adresu 0x0.

Toho se využívá k umístění vektorů přerušení do rychlé RAM, což je výhodou v možnosti přeprogramování vektoru přerušení, rychlejším odezvám a nižší spotřebě.[1] viz obrázek 6.2.

[1]Ne všechny ARM procesory podporují přemapování paměti, některé dovolují přemapovat pouze část paměti. Např. procesory rodiny Philips LPC dovolují na adresu 0x0 namapovat pouze 64 bytů paměti RAM, což je ale dostatečné pro primární i sekundární tabulku vektorů přerušení, viz [20].



Obrázek 6.2: Rozmístění částí kódu v ROM a RAM.

6.2.3 Primární a sekundární tabulka vektorů přerušení

Na adrese 0x0 se nachází tabulka vektorů přerušení s osmi pozicemi (viz kapitola 5.1.1). Na každé její pozici by se měla nacházet instrukce skoku do obslužné rutiny přerušení. Použitelná skoková instrukce (`B label`, viz [26]) ale dovoluje pouze relativní skok do ± 25-bitové vzdálenosti. Z toho důvodu je ihned za primární tabulkou umístěna tzv. *sekundární tabulka vektorů přerušení* obsahující adresy obslužných rutin.

V primární tabulce jsou pak instrukce `LDR pc, [pc, #0x18]`, které do registru `pc` (*program counter*) zkrátka zkopírují adresu umístěnou na příslušné pozici v sekundární tabulce. Tato adresa pak může být jakákoliv 32-bitová hodnota.

Výhodou sekundární tabulky je také možnost jednoduché změny obslužné rutiny pouhým přepisem adresy v sekundární tabulce.

6.2.4 Nastavení zásobníků a RAM sekcí

Resetovací sekvence pokračuje po low level inicializaci nastavením zásobníků pro všechny procesorové módy. Procesor je vždy přepnut do daného módu a do registru `SP` (*Stack*

Pointer), který je různý pro jednotlivé módy, je vložena žádaná hodnota.

Použitý model počítá se zásobníky umístěnými úplně na konci paměti RAM, rostoucími směrem dolů, přičemž zásobník pro *User* mód (uživatelský mód příslušný hlavní aplikaci) je umístěn nejníže a roste směrem do prostoru haldy, viz obrázek 6.2.

Velikosti zásobníků jsou definovány pomocí maker ve startup kódu.

ELF sekce, které mají být umístěny v paměti RAM (viz kapitola 4.1.1), jsou poté nastaveny pomocí proměnných z linkerscriptu. Sekce `.data` (proměnné) a `.fastcode` (kritické části kódu) jsou zkopírovány z ROM zavaděčů do RAM a pro sekci `.bss` (neinicializované proměnné) je v paměti RAM vyhrazen a smazán potřebný prostor, viz kapitola 4.1.2.

6.2.5 Model přerušení

Rodina procesorů ARM podporuje dva typy přerušení (IRQ a FIQ) a několik druhů výjimek (viz kapitola 5.1.1). Mikrokontroléry rodiny AT91SAM7S podporují několik zdrojů IRQ a jeden zdroj FIQ přerušení.

Obsahují také periferii *Advanced Interrupt Controller* (AIC), která podporuje tzv. *auto-vectoring* obslužných rutin přerušení. Ke každému typu přerušení je v této periferii přímo registrována obslužná funkce, která může být volána standardním jednoduchým způsobem z IRQ (FIQ) pozice primární tabulky vektorů přerušení, viz dokumentace mikrokontroléru [8].

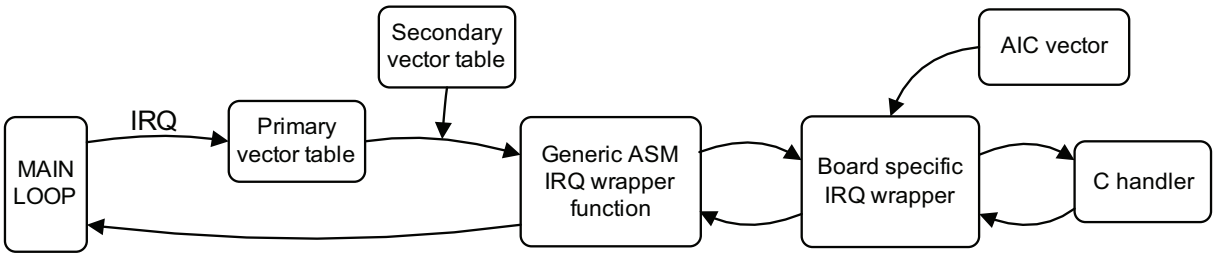
Tyto obslužné funkce ale nemohou být obyčejnými C funkcemi. Musí se deklarovat se speciálním atributem jako IRQ (FIQ) handler. Jejich kód je pak kompilátorem doplněn o vstupní a výstupní sekvenci, která uchová část registrů na zásobníku.

Tento model bohužel není vhodný pro vnořené přerušení, protože nejsou uloženy všechny registry potřebné pro vnoření dalšího přerušení, především není zálohován registr `SPSR` (*Stored Program Status Register*), viz [9] a [6].

Mirko Samek proto navrhuje v již zmíněném článku [20] model zahrnující výhody vnořených přerušení s využitím vektorů v AIC, kde obslužné rutiny mohou být obyčejné C funkce. Tento model navíc zachovává priority jednotlivých zdrojů IRQ přerušení definované v AIC.

Na obrázku 6.3 je znázorněn zmíněný proces obsluhy přerušení.

Při události přerušení skočí program do generické assemblerovské „obalové“ funkce, která uloží na zásobník všechny potřebné registry vč. `SPSR`, poté skočí do „obalové“ funkce specifické pro konkrétní aplikaci. Tato funkce povolí přerušení (tím umožní vnoření dalšího



Obrázek 6.3: Proces obsluhy přerušení využívající „obalové“ funkce.

přerušení) a zavolá obslužnou C funkci přes ukazatel, který je uložen v AIC pro příslušný zdroj přerušení.

Tento model je v našem programu používán pro všechny druhy přerušení a výjimek.

6.2.6 Inicializace specifická dané aplikaci

Proces vyznačený na obrázku 6.1 jako *board specific init* pak vyplní sekundární tabulku vektorů přerušení ukazateli na assemblerovské „wrapper“ funkce (viz kapitola 6.2.5) a nastaví implicitní obsluhy v AIC na externě definované uživatelské funkce, viz příloha B.

6.3 Hardware Abstraction Layer

Při programování ve vyšším jazyce (např. jazyk C) je výhodné použít určitou abstrakci přístupu k hardwaru. Výsledný kód je pak přehlednější, bezpečnější a je možné jej snadno a rychle modifikovat.

Hardwarovou abstrakci můžeme rozdělit do několika vrstev, kterými může být:

1. pojmenování periferních registrů, důležitých paměťových míst, a vytvoření jejich triviálních logických struktur,

2. základní funkce pro nízkoúrovňové nastavení periferií a základní periferní čtecí a zápisové funkce,

3. vyšší uživatelsky přívětivá správa periferií, jednoduché ovladače periferií obsažených na čipu.

Některé části (zejména první dvě) bývají realativně dostupné od výrobce. Třetí část je většinou záležitostí uživatele.

V rámci této práce byla rozpracována knihovna obsahující zmíněnou funkčnost pro vybrané periferie.

Jako základ byla použita knihovna maker a jednoduchých `inline` funkcí (*AT91lib*) od společnosti Atmel, vydaná v rámci *AT91SAM7S Evaluation Kit Software Package* [7]. Vybrané části byly setříděny, upraveny a doplněny.

V rámci výše zmíněného bodu 3 byla knihovna doplněna např. funkcemi, které zjednodušují správu časovačů a funkcemi pro přehlednou správu přerušení a jejich obslužných rutin, více v příloze B.

Úplnost ani detailní popis výsledné knihovny není účelem této práce.

6.4 Ovladače externích periferií

V této kapitole nastíníme řešení ovladačů vybraných externích periferií. Podrobnosti jsou v příloze B. Všechny ovladače jsou nadstavbou nad hardwarovou abstrakcí, popsanou v kapitole 6.3.

6.4.1 LCD

V našem zařízení je použit LCD displej s řadičem KS0713 (viz kapitola 3.1.2). Ten je s mikrokontrolérem spojen přes sériové rozhraní. V tomto režimu nelze z paměti displeje číst, lze jen zapisovat. Z toho důvodu obsahuje náš softwarový ovladač grafickou paměť, zrcadlovou k vnitřní paměti řadiče.

Displej má rozlišení 128×64 pixelů, nutná velikost grafické paměti je proto 1024 bytů.

Řadič KS0713 nepodporuje textový režim, proto je ovladač opatřen jeho emulací. Tento režim využívá externí font obsahující všechny základní ASCII znaky. Zobrazovat jednotlivé znaky pak lze např. jednoduše pomocí funkce `void lcd_putchar(char c)`.

Velikost řádkování a šířka sloupce je pevná a vychází z organizace vnitřní paměti řadiče a ze způsobu komunikace. Výhodou textového režimu je rychlost a jednoduchost, nevýhodou jsou pevné atributy zobrazovaného textu. Podporu pro tento režim lze zapnout při kompilování, více v softwarové dokumentaci v příloze B.

V grafickém režimu (používaném v naší aplikaci) není dostupný žádný implicitní font

ani funkce pro psaní znaků. Implementace této funkčnosti je pak již na uživateli.

Veškeré změny v grafické paměti ovladače je možné jednorázově aktualizovat na displej, přičemž lze obnovit celý displej, nebo jen některou jeho část, viz příloha B.

6.4.2 RTC

Ovladač RTC obvodu je kombinací obecného API a ovladače konkrétního obvodu tj. PCF8563, popsaného níže.

Toto API např. obsahuje datovou strukturu `rtc_t`, kterou je možné jednoduše naplnit aktuálními hodnotami data a času pomocí funkce `void rtc_read(rtc_t *rtc)`, viz příloha B. Tato funkce načte aktuální data z RTC obvodu, které překonvertuje z BCD kódu do decimálního, a naplní jimi danou strukturu.

API také obsahuje funkci, která navrátí zformátovaný výstup do podoby časové značky použité v záznamu viz kapitola 6.6.4.

PCF8563

Obvod PCF8563 komunikuje s mikrokontrolérem po sběrnici I^2C, připojené k periferii TWI (*Two-Wire Interface*). API ovladače podporuje pouze základní čtecí a zápisové funkce.

Vnitřní ukazatel do paměti obvodu lze nastavit, není tedy nutné pokaždé číst jeho celou vnitřní paměť.

6.4.3 Secure Digital

Pro komunikaci s paměťovou kartou byl použit upravený ovladač obsažený v knihovně EFSL (kapitola 4.5). Tento ovladač využívá SPI periferii.

Jeho úpravy zahrnují:

- použití hardwarové abstrakce (kapitola 6.3),
- doplnění o detekci vložení karty (*Card Detect* pin) a detekci ochrany proti zápisu (*Write Protected* pin),
- možnost použít DMA kanál pro zápis/čtení celého sektoru (nastavení při kompilaci).

Tento ovladač implementuje fyzickou vrstvu SPI SD protokolu (dle specifikace [19]) pro SD karty typu *Standard Capacity*, tj. do kapacity 2 GB. Karty s větší kapacitou nejsou tímto ovladačem rozpoznány a jsou odmítnuty.

6.5 Ostatní utility

6.5.1 Window

Utilita *window* slouží jako pomůcka, jestliže informací na „stránce“ je více, než je možné zobrazit na displeji.

Okno (window) obsahuje vlastní paměť pro zobrazovaná data. Velikost okna může být větší i menší, než je rozlišení displeje. Každému oknu přísluší pouze definovaná podoblast displeje.

V programu je možné definovat více oken, jejichž zobrazované oblasti se mohou libovolně překrývat. Uživatel tedy může mít např. vyhrazeny nezávislé části displeje pro různé účely, nebo může mít několik oken (a jejich oblasti) přes sebe a libovolně mezi nimi přepínat.

6.5.2 Textbox

Textbox je nadstavbou nad utilitou *window* a umožňuje zobrazit text na libovolné pozici v její paměti. Tato utilita využívá externí font, uchovává pouze samotný text, nikoliv grafická data.

Parametry textu (řádkování, mezery mezi znaky) i font je možné libovolně nastavit, k dispozici je také např. funkce pro inverzi celého *textboxu*, vhodná pro zvýraznění.

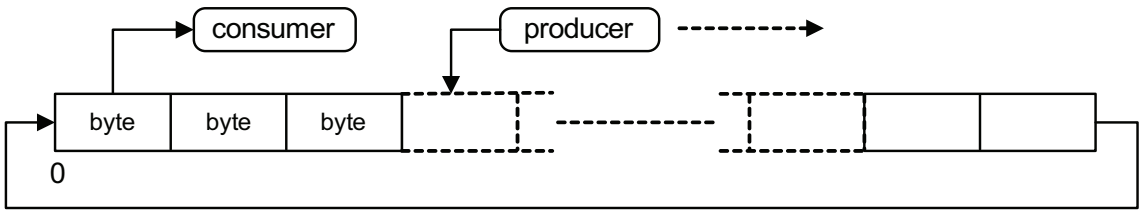
6.5.3 Menu

Tato utilita slouží k vytvoření kaskádového uživatelského menu. Každá položka menu používá k zobrazení vlastní textbox, obsahuje také čtveřici ukazatelů na nastavitelné callback funkce (nahoru, dolů, doleva, doprava).

6.5.4 Cyklický buffer

Cyklický buffer je speciálním typem vyrovnávací paměti typu FIFO (First In, First Out). Využívá datové pole a dva ukazatele (producent a konzument).

Jestliže kterýkoliv ukazatel narazí na konec datového pole, nastaví se opět na začátek, viz obrázek 6.4.



Obrázek 6.4: Pracovní schema cyklického bufferu.

Použitá implementace dovoluje velikost bufferu pouze:

$$N_{bf} = 2^x, x \in 1, 2, ... \tag{6.1}$$

kde N_{bf} je velikost bufferu v bytech a x je libovolné nenulové kladné číslo.

Výhoda tohoto omezení (rovnice 6.1) je v rychlosti vnitřních přepočtů. Např. rozdíl mezi adresou producenta a konzumenta Δ_{PC} lze pak jednoduše spočítat pomocí logického ANDu jako:

$$\Delta_{PC} = (A_P - A_C + N_{bf}) \ \& \ M, \tag{6.2}$$

kde A_P resp. A_C je adresa producenta resp. konzumenta a maska

$$M = N_{bf} - 1. \tag{6.3}$$

Daná implementace také detekuje přetečení bufferu.

6.6 Hlavní aplikace

V hlavní aplikaci se před skokem do nekonečné smyčky nejdříve:

1. inicializují všechny potřebné periferie a utility,
2. naváže spojení s externími periferiemi (SD, RTC, LCD),
3. vygeneruje celý strom uživatelského menu a zobrazí hlavní podmenu,
4. nastaví obsluhy a povolí přerušení.

6.6.1 Softwarové aspekty řešení problému

Zpracování dat v této úloze má následující rysy:

- data ze sériových linek přichází po jednotlivých bytech. Tyto byty je nutné co nejdříve přeposlat na protější port a zaznamenat.

- Zápis do souboru (byť bez přímého uložení na médium) knihovnou EFSL stojí určité fixní časové náklady (viz [17]). Je tedy vhodné ukládat data po větších celcích.

Z těchto důvodů je výsledný záznam připravován do vyrovnávací paměti, ze které je po blocích ukládán do cílového souboru.

6.6.2 Stavový diagram

Na obrázku 6.5 je vidět základní stavový diagram celé aplikace.

V hlavní nekonečné smyčce programu se v případě potřeby ukládá obsah vyrovnávací paměti do připraveného souboru (po blocích definované velikosti).

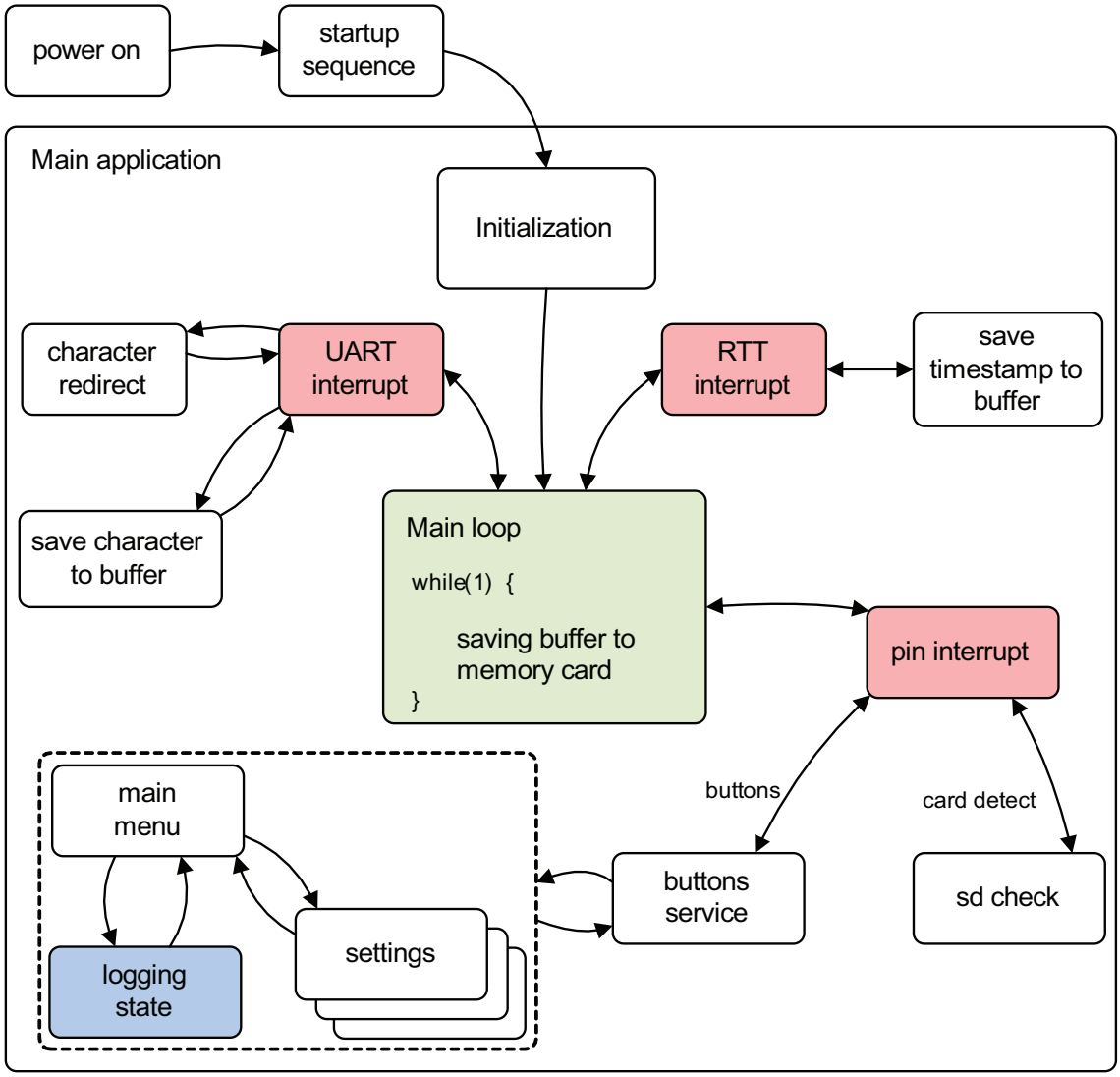
Jako vyrovnávací paměť byla zvolena FIFO paměť ve formě cyklického bufferu (kapitola 6.5.4).

Přerušení hlavní smyčky může způsobit:

- především UART periferie příchozím znakem,

- RTT (Real-Time Timer) periferie, která měří zvolený interval vkládání časových značek,

- změna logické hodnoty na vybraných externích I/O pinech, což jsou signály uživatelských tlačítek a pin detekce vložení (nebo vyndání) paměťové karty, *Card Detect*.



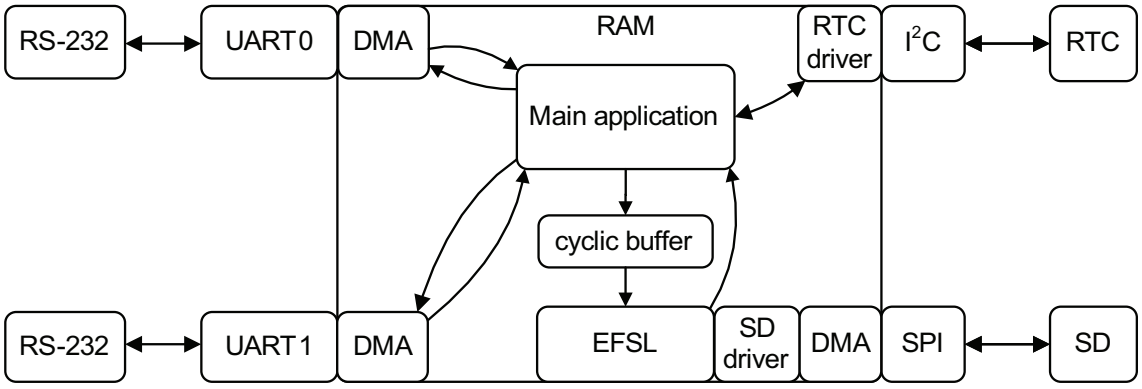
Obrázek 6.5: Stavový diagram aplikace.

6.6.3 Tok dat

Tok dat je zřetelný na obrázcích 6.6 a 6.5.

Při přerušení periferie RTT se připraví časová značka z aktuálních dat v RTC. Příchozí byte z UART periferie se přes DMA uloží na určené místo a vyvolá se přerušení. Tento byte je ihned přesměrován na DMA kanál protější UART periferie a je uložen (v případě potřeby s metadaty – časová značka, směr toku) do vyrovnávací paměti.

Z vyrovnávací paměti se pak ukládají veškerá finální data přes knihovnu EFSL, SD ovladač, DMA kanál a SPI periferii na paměťovou kartu.



Obrázek 6.6: Tok důležitých dat v rámci aplikace a celého zařízení.

6.6.4 Popis výsledného záznamu

Změna směru toku dat je uvozena sekvencí příslušnou danému směru, tedy znak *line feed* a `DCE>`/`DTE>` (zkratky odpovídají typům zařízení podle normy EIA-232D viz standardy TIA asociace [37]). Časové značky (hodiny, minuty, vteřiny) jsou vkládány ve formátu `<hh:mm:ss>`.

Pro úplnost zde uvádíme stručný příklad výsledného záznamu v souboru na paměťové kartě:

```
DCE>message 1 from device 1
DTE>message 1 from device 2
mess<10:35:26>age 2 from device 2
DCE>message 2 from device 1<10:35:27>
...
```

Kapitola 7

Testování

V této kapitole naznačíme provedené testy zařízení a jejich výsledky. Nastíníme také, jak se výsledky testů promítly do finální koncepce.

7.1 Zkoušky v zadavatelské firmě

V zadavatelské firmě (viz kapitola 2) bylo zařízení otestováno na modelu reálného prostředí. Zaznamenávala se komunikace mezi počítačem a snímačem čárových kódů. Komunikace se týkala konfigurace snímače a sběru dat.

Parametry RS-232 linky byly 9600baud, sudá parita, sedm datových bitů a dva stop-bity.

Zkoušky prokázaly tyto nedostatky:

1. občasné zpoždění komunikace,
2. občasné vynechání znaku v komunikaci,
3. zpoždění komunikace při I/O přerušení,
4. zpožďování a ztráty komunikace při záznamu s časovými značkami.

7.1.1 Řešení problémů

Software

Zpozorované problémy byly odstraněny úpravami koncepce hlavního programu.

1. Občasné zpoždění celé komunikace mohlo být způsobeno prioritami DMA kanálů. Z dokumentace mikrokontroléru AT91SAM7S [8] plyne, že veškeré přijímací DMA kanály mají větší prioritu, než vysílací. Může se tedy např. stát, že při silném čtení (např. FAT tabulky) z paměťové karty je upřednostněna SPI před UART periferií. Tento problém byl vyřešen tím, že se použil SPI DMA kanál pouze pro zápis.

2. Občasné vynechání znaku v komunikaci bylo zřejmě způsobeno mj. zamykáním přerušení při zpracování příchozího znaku, což při kolizi znaků z obou směrů může vést až ke ztrátě jednoho z nich. Využilo se tedy modelu vnořených přerušení bez jejich zamykání a tím se odstranil tento důležitý problém.

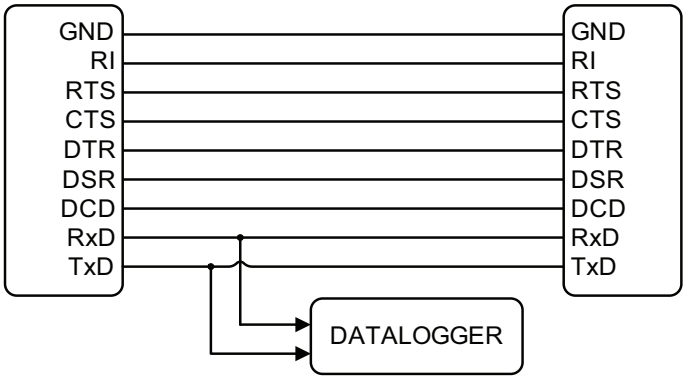
3. Příčinou zpoždění komunikace při I/O přerušení zřejmě byla špatně nastavená priorita přerušení. Řešením tedy bylo, dát nejvyšší prioritu obsluze příchozího znaku.

4. Problémy s použitím časových značek byly opět důsledkem jen nevhodně nastavených priorit přerušení.

Hardware

Potíže vedly vzhledem k jejich důležitosti také k návrhu hardwarového řešení některých problémů. Celou koncepci zařízení je možné změnit na „pasivní záznamník“, který bude pouze číst příchozí data, porty však budou hardwarově propojeny. Z principu se tak odstraní i teoretický vliv záznamníku na komunikaci.[1]

Tuto úpravu lze provést vhodným napojením stávajícího záznamníku do sítě, viz obrázek 7.1.



Obrázek 7.1: Možná úprava zapojení záznamníku do sítě.

[1]Koncepce „aktivního záznamníku“ byla zvolena s ohledem na dřívější požadavky na zařízení.

7.2 Závěrečné testy

Testy finální podoby zařízení (pouze se softwarovými korekcemi, viz kapitola 7.1) probíhaly s pomocí počítače se dvěma sériovými porty. S výhodou byl použit terminálový program *Realterm* [28] pro správu a komunikaci přes sériové porty a také vlastní program pro testování přenesených dat a záznamu, jehož zdrojový kód je v příloze B.

Program *Realterm* se umí jednak chovat jako „terminál“ (tak je možné testovat základní funkčnost např. zadáváním znaků z klávesnice) a také umožňuje přes sériový port posílat soubory a ukládat příchozí data do souboru (čímž je možné testovaný systém dostatečně zatížit). Soubory je možné posílat buď kontinuálně, nebo po řádcích s definovaným časovým odstupem.

Funkčnost zařízení tedy vyzkoušíme tak, že proti sobě vyšleme dva soubory. Správnost přenesených dat i záznamu poté ověříme ve zmíněném programu. Takový přenos je pro záznamník náročný především proto, že se neustále střídá směr toku dat. Téměř každý byte tedy musí být doplněn o identifikátor portu (viz kapitola 6.6.4) a tím se dosáhne maximálního možného toku ukládaných dat.

Opakované testy již neprokázaly, že by zařízení ovlivňovalo průběh komunikace, přenosy při všech rychlostech a nastaveních byly korektní.

Záznam dat je při tomto testu korektní až do rychlosti 38400baud. Při vyšších rychlostech může dojít k přetečení vyrovnávací paměti a záznam je pak neplatný (detekce přetečení se provádí v každém případě a uživatel má tuto informaci k dispozici, viz dokumentace zařízení v příloze B).

Přetečení vyrovnávací paměti je však při všech rychlostech v reálném provozu téměř vyloučeno, jelikož běžná smysluplná komunikace je „střídavá“.

Chyby se tedy podařilo odstranit a zařízení je možné označit za funkční i bez hardwarové úpravy v kapitole 7.1.

Kapitola 8

Závěr

Tato práce se zabývá návrhem embedded zařízení pro záznam komunikace po sériové lince RS-232 na paměťovou kartu typu Secure Digital dle specifikace firmy z průmyslové praxe. Podařilo se vyvinout funkční prototyp zařízení, což bylo hlavním cílem práce.

Hardwarová část se opírá o mikrokontrolér Atmel AT91SAM7S s jádrem ARM, na jehož bázi byl navržen a zkonstruován hardware vlastní koncepce. Zařízení disponuje uživatelským rozhraním a je zcela autonomní.

Softwarová architektura nevyužívá operační systém, místo toho byla vyvinuta vlastní struktura software typu *bare-metal* s využitím knihoven Newlib a EFSL. Architektura využívá generické části kódu pro ARM platformu.

Jedním z nejdůležitějších kroků byla úspěšná aplikace volně šiřitelných vývojových nástrojů GNU ARM a OpenOCD. Výsledkem byl snadný vývoj software včetně možnosti *on-chip* debuggování přes JTAG rozhraní a GDB přímo z prostředí Eclipse IDE.

Na základě vlastní hardwarové abstrakce byly vytvořeny nebo upraveny potřebné ovladače periferií a vyvinuty utility usnadňující návrh uživatelského rozhraní a celého programu.

Klíčovou vlastností pro bezchybnou funkčnost programu bylo uplatnění DMA kanálu pro periferie a model přerušení bez zamykání.

Praktické zkoušky byly zdrojem softwarových úprav a návrhu hardwarového zjednodušení, viz kapitola 7. Výsledné testy zařízení jsou dobré.

Námětem na zlepšení může být zmíněná hardwarová úprava a využití USB portu např. pro živé sledování komunikace, přístup k záznamu, nebo pro vzdálenou správu zařízení.

Literatura

[1] Analog Devices Inc. *ADM3202/ADM3222/ADM1385 - Product Datasheet*. Analog Devices Inc., 2001.

[2] Chris Wright Andrew N. Sloss, Dominic Symes. *ARM System Developers Guide*. Elsevier, 2004.

[3] ARM Ltd. *ARM Architecture Reference Manual*. ARM Limited, 2000.

[4] ARM Ltd. ARM JTAG signals, 2008. http://infocenter.arm.com/help/.

[5] ARM Ltd. The ARM Ltd. official web site, 2008. http://www.arm.com/.

[6] ARM Ltd. Arm: Writing interrupt handlers, 2008. http://www.arm.com/support/faqdev/1456.html.

[7] Atmel Corporation. At91 software package, 2008. http://atmel.com/dyn/products/tools_card.asp?tool_id=4343.

[8] Atmel Corporation. *AT91SAM7S ARM Thumb-based Microcontrollers*. Atmel Corp., 2008.

[9] Atmel Corporation. *Interrupt Management: Auto-vectoring and Prioritization*. Atmel Corporation, 1998.

[10] Samsung Electronics. *KS0713 LCD driver datasheet*. Samsung Electronics, 1998.

[11] Michael Fischer. YAGARTO GNU ARM distribution official web site, 2008. http://yagarto.de/.

[12] Free Software Foundation. Free software foundation (GNU) official web site, 2008. http://www.gnu.org/.

[13] GNU ARM Project. GNU ARM Toolchain project official web site, 2008. http://www.gnuarm.com/.

[14] IAR Systems. The IAR System official web site, 2008. http://www.iar.com/.

[15] Texas Instruments. *TPS795xx family LDO specification*. Texas Instruments, 2006.

[16] Keil Elektronik GmbH. Keil elektronik official web site, 2008. http://www.keil.com/.

[17] Michael De Nil Lennart Ysboodt. *Embedded Filesystem Library manual*. EFSL, 2005.

[18] Displaytech Ltd. *Displaytech 64128 LCD Product Specifications*. Displaytech Ltd., 1998.

[19] Toshiba Corporation Matsushita Electric Industrial Co. Ltd., SanDisk Corporation. *Secure Digital Physical Layer Specification*. Secure Digital Association, 2006.

[20] Mirko Samek. Building Bare-Metal ARM Systems with GNU. *Embedded.com*, 2007.

[21] NXP B. V. *PCF8563 Product Specification*. Philips NXP B. V., 1999.

[22] NXP B. V. *I2C - bus specification and user manual*. Philips NXP B. V., 2007.

[23] NXP B. V. *LPC2131/32/34/36/38 Product Datasheet*. Philips NXP B. V., 2007.

[24] Olimex Ltd. *AT91SAM7-P64 Get Started Guide*. Olimex Ltd., 2005.

[25] Pavel Píša. *Moderní mikrokontroléry a využití vývojového prostředí GNU*. České Vysoké Učení Technické v Praze, 2001.

[26] Stephen Welsh Peter Knaggs. *ARM Assembly Language Programming*. School of Design, Engeneerign and Computing, Bournemouth University, Poole, UK, 2004.

[27] Dominic Rath. *Open On-Chip Debugger*. Spen, 2008.

[28] Realterm. Realterm Terminal Program official web site, 2008. http://realterm.sourceforge.net/.

[29] Richard M. Stallman and the GCC Developer Community. *Using the GNU Compiler Collection*. GNU Press, a division of the Free Software Foundation, 2005.

[30] Stan Shebs Richard M. Stallman, Roland Pesch. *Debugging with gdb*. GNU Press, a division of the Free Software Foundation, 2004.

[31] SanDisk Corporation. *SanDisk SD Card Product Manual*. SanDisk Corporation, 2004.

[32] Rob Savoye. *Embed With GNU*. Cygnus Support, 1995.

[33] SECONS s. r. o. Jtag pinouts list, 2008. http://www.jtagtest.com/pinouts/.

[34] Ian Lance Taylor Steve Chamberlain. *Using the GNU Linker*. GNU Press, a division of the Free Software Foundation, 2003.

[35] Jeff Johnson Steve Chamberlain, Roland Pesch. *The Red Hat Newlib C Library*. Red Hat Inc., 2004.

[36] Steve Furber. *ARM System-On-Chip Architecture*. Addison Wesley, Pearson Education Ltd., 2000.

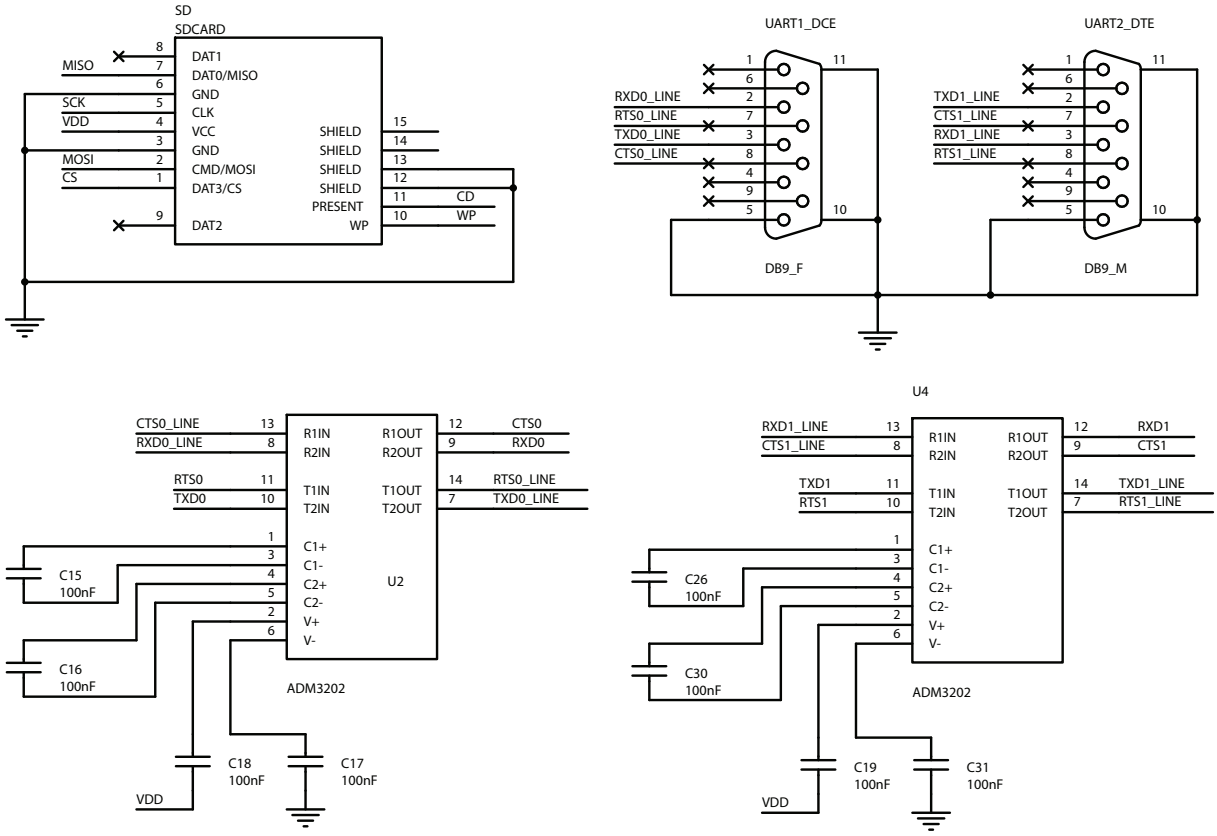
[37] TIA. Telecomunications industry association standards, 2008. http://www.tiaonline.org/standards/.

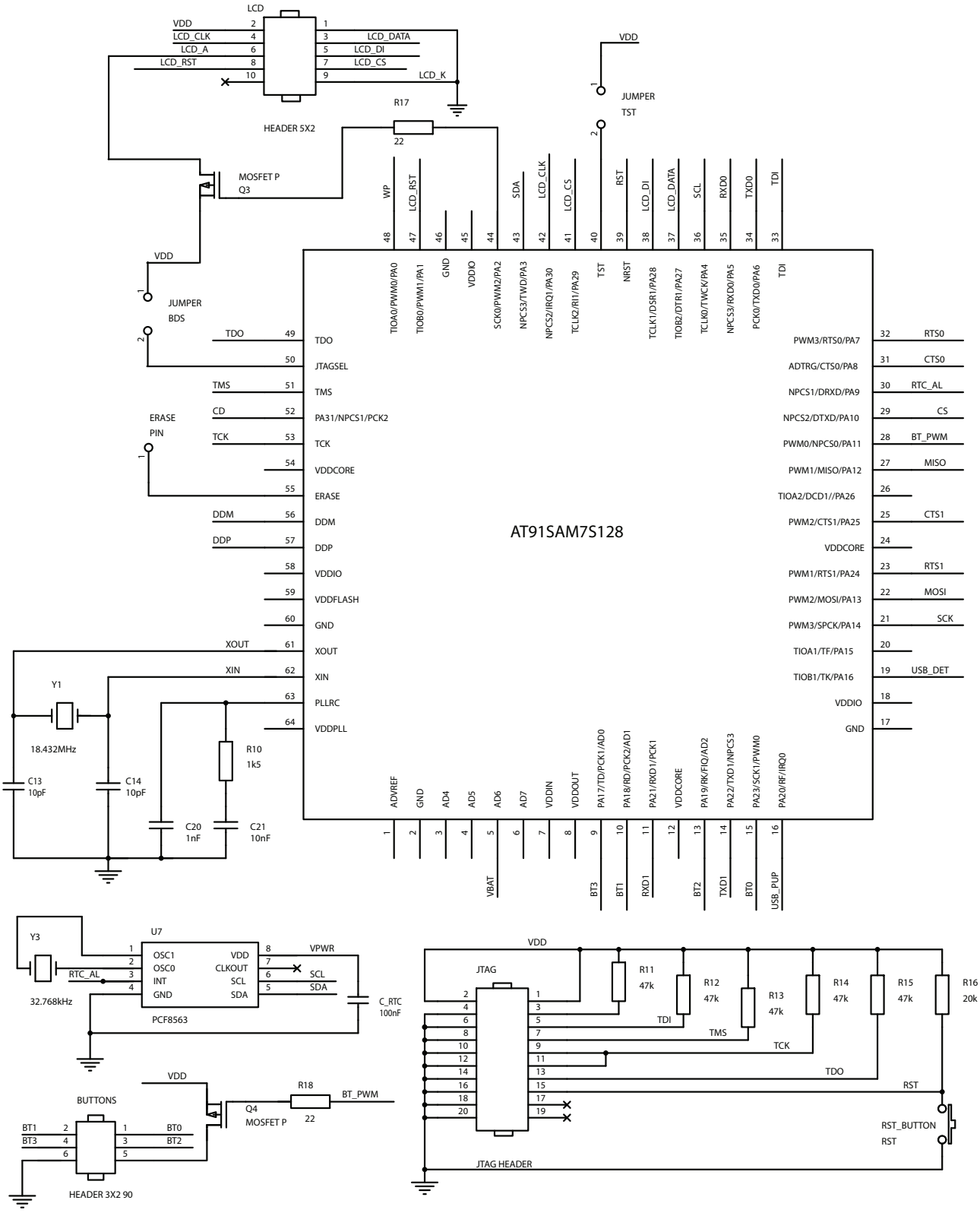
[38] wikipedia.org. Encyclopedia wikipedia, 2008. http://en.wikipedia.org/.

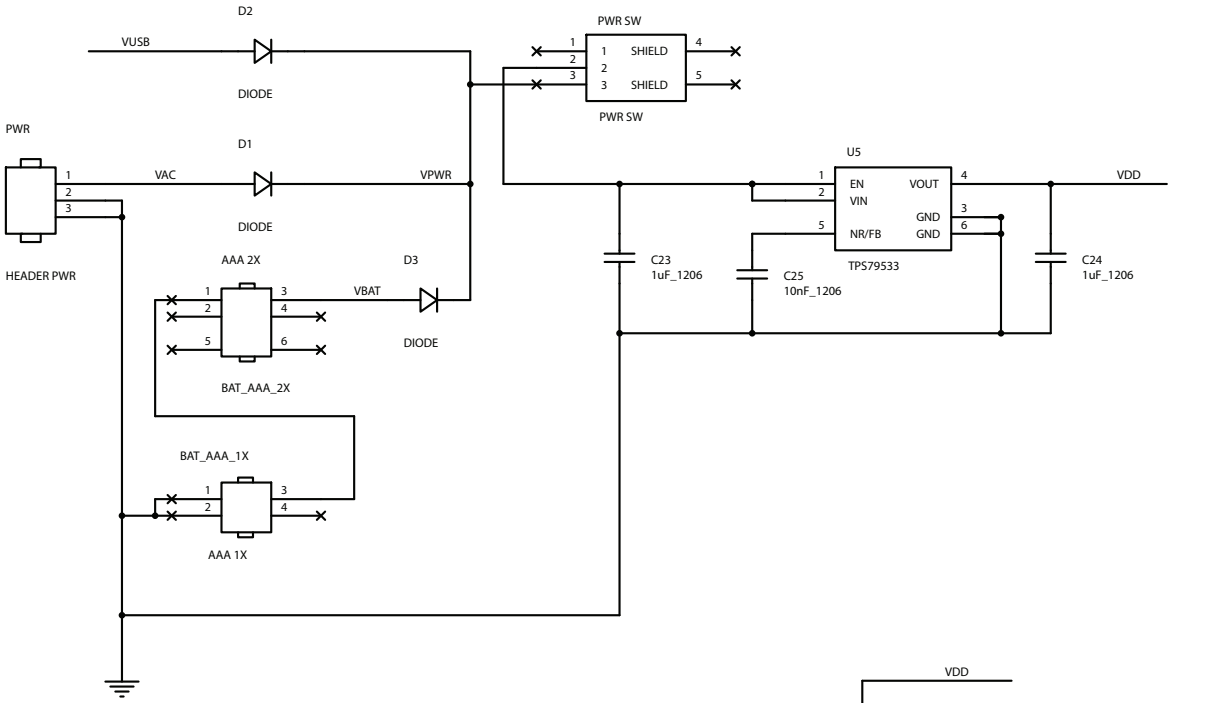
[39] Eric Youngdale. The elf object file format: Introduction. *The Linux Journal*, 1995.

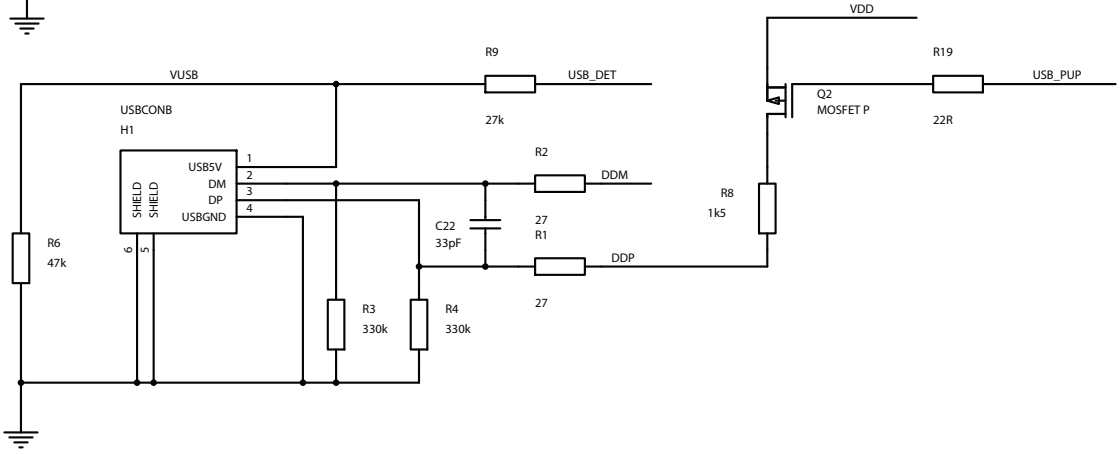
Příloha A

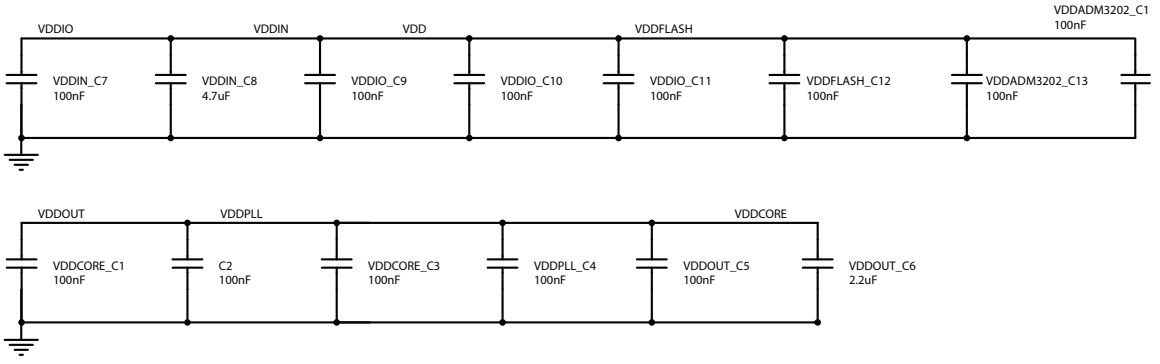
Obvodové schéma zařízení




LCD
VDD
LCD_CLK
LCD_A
LCD_RST
LCD_DATA
LCD_DI
LCD_CS
LCD_K
HEADER 5X2
R17
22
MOSFET P
Q3
JUMPER
TST
JUMPER
BDS
ERASE
PIN
AT91SAM7S128
TDO
JTAGSEL
TMS
PA31/NPCS1/PCK2
TCK
VDDCORE
ERASE
DDM
DDP
VDDIO
VDDFLASH
GND
XOUT
XIN
PLLRC
VDDPLL
ADVREF
AD4
AD5
AD6
AD7
VDDIN
VDDOUT
VBAT
PA17/TD/PCK1/AD0
PA18/RD/PCK2/AD1
PA21/RXD1/PCK1
VDDCORE
PA19/RK/FIQ/AD2
PA22/TXD1/NPCS3
PA23/SCK1/PWM0
PA20/RF/IRQ0
BT3
BT1
RXD1
BT2
TXD1
BT0
USB_PUP
PWM3/RTS0/PA7
ADTRG/CTS0/PA8
NPCS1/DRXD/PA9
NPCS2/DTXD/PA10
PWM0/NPCS0/PA11
PWM1/MISO/PA12
TIOA2/DCD1//PA26
PWM2/CTS1/PA25
PWM1/RTS1/PA24
PWM2/MOSI/PA13
PWM3/SPCK/PA14
TIOA1/TF/PA15
TIOB1/TK/PA16
RTS0
CTS0
RTC_AL
CS
BT_PWM
MISO
CTS1
RTS1
MOSI
SCK
USB_DET
Y1
18.432MHz
C13
10pF
C14
10pF
R10
1k5
C20
1nF
C21
10nF
Y3
32.768kHz
U7
PCF8563
C_RTC
100nF
VPWR
BUTTONS
HEADER 3X2 90
Q4
MOSFET P
R18
22
BT_PWM
JTAG
JTAG HEADER
R11
47k
R12
47k
R13
47k
R14
47k
R15
47k
R16
20k
RST_BUTTON


D2
VUSB
DIODE
PWR SW
SHIELD
SHIELD
PWR SW
PWR
D1
VAC
VPWR
DIODE
HEADER PWR
U5
EN
VIN
NR/FB
VOUT
GND
GND
TPS79533
VDD
C23
1uF_1206
C25
10nF_1206
C24
1uF_1206
AAA 2X
D3
VBAT
DIODE
BAT_AAA_2X
BAT_AAA_1X
AAA 1X


VDD
R9
VUSB
USB_DET
R19
USB_PUP
Q2
MOSFET P
27k
22R
USBCONB
H1
USB5V
DM
DP
USBGND
SHIELD
SHIELD
R2
DDM
R8
1k5
C22
33pF
27
R1
DDP
27
R6
47k
R3
330k
R4
330k


VDDIO
VDDIN
VDD
VDDFLASH
VDDADM3202_C1
100nF
VDDIN_C7
100nF
VDDIN_C8
4.7uF
VDDIO_C9
100nF
VDDIO_C10
100nF
VDDIO_C11
100nF
VDDFLASH_C12
100nF
VDDADM3202_C13
100nF

VDDOUT
VDDPLL
VDDCORE
VDDCORE_C1
100nF
C2
100nF
VDDCORE_C3
100nF
VDDPLL_C4
100nF
VDDOUT_C5
100nF
VDDOUT_C6
2.2uF

Příloha B

Obsah přiloženého CD

- Documentation: dokumentace celého zařízení,
- Firmware: zdrojový kód firmwaru,
- Firmware Documentation: dokumentace k firmwaru,
- Libraries: použité knihovny,
- Materials: vybrané literární zdroje,
- PCB: soubory pro OrCAD s návrhem DPS,
- Test Program: program pro testování správnosti záznamu,
- Text: zdrojový kód tohoto textu,
- Toolchain: použité vývojové nástroje,
- Utilities and Software: ostatní nástroje a software.