VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

DEPARTMENT OF COMPUTER SYSTEMS

OVLÁDÁNÍ VESTAVĚNÉHO SYSTÉMU PŘES INTERNET

CONTROL OF EMBEDDED SYSTEM THROUGH INTERNET

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ DVOŘÁK**

AUTHOR

VEDOUCÍ PRÁCE **Ing. ROLAND DOBAI, Ph.D.**

SUPERVISOR

BRNO 2017

Zadání

Zadání bakalářské práce/19045/2016/xznebe00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Dvořák Tomáš**
Obor: Informační technologie
Téma: **Ovládání vestavěného systému přes Internet**
Control of Embedded System Through Internet
Kategorie: Vestavěné systémy

Pokyny:

1. Seznamte se s problematikou ovládaní vestavěných systémů přes Internet.
2. Navrhněte a implementujte systém pro ovládání vestavěného systému na bázi Xilinx Zynq přes Internet.
3. Ověřte funkčnost implementovaného systému pro jednoduché úlohy ovládání vybraných prvků, které jsou dostupné na vývojové desce.
4. Zhodnoťte dosažené výsledky a diskutujte možnosti dalšího vývoje projektu.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodu 1 zadání, demonstrace rozpracovanosti bodu 2 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Dobai Roland, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá návrhem jednotlivých částí systému pro ovládání vestavěného systému na bázi Xilinx Zynq přes internet. Možností návrhů takových systémů je přitom celá řada. U každé části poskytuje přehled alternativních řešení a dále rozvíjí řešení vybrané v implementaci. Práce poukazuje na modularitu, jednoduchost a rozšiřitelnost výsledné implementace. Pro demonstraci zdánlivé nezávislosti výsledné serverové a webové aplikace na hardwarové platformě byly navrženy a implementovány 2 varianty systému pro ovládání vybraných prvků na dané vývojové desce. První varianta systému je schopna ovládat LED a přepínače na desce, zatímco druhá i vestavěný displej. Práce rozebírá jednotlivé etapy řešení od návrhu hardwarové platformy, přes operační systém, aplikaci serveru až po webovou aplikaci. Závěr práce je věnován testování a ověřování funkčnosti obou systémů.

Abstract

This bachelor‘s thesis deals with the process of designining individual parts of system for control of embedded system based on Xilinx Zynq over the Internet. Alternative solutions are provided in each section. The thesis points out to the modularity, simplicity and extensibility of the final implementation. The thesis describes 2 implementations of such systems to demonstrate the independence of the final server and web application on the hardware platform. The first system is capable of controling the LEDs and switches on the board, while the other systém can also control the embedded display. The thesis analyzes the individual stages of the solution from the design of the hardware platform, through the operating system, the server application to the web application. The conclusion of the thesis is devoted to testing and verification of the functionality of both implemented systems.

Klíčová slova

ovládání přes internet, vestavěný systém, Xilinx, Zynq, Zedboard, FPGA, HTML, Javascript, Ajax, uživatelské rozhraní, Linux, webový server

Keywords

control through internet, embedded systém, Xilinx, Zynq, Zedboard, FPGA, HTML, Javascript, Ajax, user interface, Linux, web server

Citace

DVOŘÁK, Tomáš. *Ovládání vestavěného systému přes Internet*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Dobai Roland.

Ovládání vestavěného systému přes Internet

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Rolanda Dobaie Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Dvořák
17. května 2017

Poděkování

Chtěl bych poděkovat svému vedoucímu bakalářské práce panu Ing. Rolandu Dobaiovi, Ph.D. za veškerou pomoc při tvorbě této práce a cenné rady.

Obsah

Slovník

- **ANSI** – American National Standards Institute – americká standardizační organizace
- **ASCII** – American Standard Code for Information Interchange – kódová tabulka znaků
- **CSS** – Cascading Style Sheets – definice stylů v HTML
- **DDR** – Double data rate – typ operační paměti
- **DEVFS** - Virtuální souborový systém
- **EMIO** – Extended Multiplexed Input/Output – multiplexovaná sada vstupů/výstupů
- **FPGA** – Field-programmable gate-array – programovatelné hradlové pole
- **GPIO** – General Purpose Input/Output – vstupy/výstupy obecného určení
- **GPL** – GNU General Public License – všeobecná veřejná licence GNU
- **HTML** – HyperText Markup Language – značkovací jazyk
- **HTTP** – Hypertext Transfer Protocol – aplikační protokol rozšířený na internetu
- **IP core** – Intellectual Property core – blok tvořící určitou funkcionalitu v FPGA
- **LED** – Light-Emitting Diode – světlo vyzařující dioda
- **OLED** – Organic Light-Emitting Diode - technologie organických elektroluminiscenčních diod
- **Pin** – Zakončení bitového spoje
- **PL** – Programmable Logic – programovatelná logika vestavěného systému
- **PS** – Processing System – procesorový systém vestavěného systému
- **SoC** – System On Chip – integrovaný obvod zahrnující elektronický systému do jediného čipu
- **URI** – Uniform Resource Identifier – identifikátor prostředků na serveru
- **Websocket** – prostředek pro komunikaci mezi kódem jazyka Javascript a aplikačním serverem

Kapitola 1

Úvod

Vestavěné systémy představují většinou kompaktní jednoúčelová zařízení [5]. Taková zařízení obvykle nedisponují prvky umožňující jejich pokročilé ovládání. To může být rovněž nepřístupné z konstrukčních důvodů, jakým je třeba vestavění do jiné, větší součásti, případně stroje atd. Zodpovědností takových systémů mohou být úkony například pro zajišťování plynulosti dopravy na dálnicích pomocí informačních tabulí, semaforů a podobně. Výhodou takových systémů je jejich výrobní cena, která je oproti komplexnějším zařízením zpravidla daleko nižší [7].

Práce je primárně zaměřena na ovládání vybraných prvků jednoho určitého vestavěného systému. Vedlejším, daleko zajímavějším zaměřením práce je poskytnutí prostředků pro rozšíření cílové aplikace na úrovni softwaru. Cílem práce je tedy vytvoření autonomního systému pro ovládání vybraných prvků na desce a poskytnutí jednoduchého rozhraní pro rozšíření systému. Systém bude složen z hardwarové platformy, operačního systému, webového serveru a webové aplikace.

Pro vytvoření webového serveru běžícího v operačním systému je nezbytné znát základy platformy daného vestavěného systému, včetně jejich možných využití. Uvedenou problematiku stručně charakterizuje druhá kapitola.

Další, neméně důležitou částí práce je prozkoumání možností komunikace systémů přes internet, s čímž souvisí výběr komunikačního protokolu a možnosti jeho využití. Zvolený způsob využití je do větších podrobností rozebrán ve třetí kapitole. Ta se dále zabývá možnostmi tvorby uživatelského rozhraní a návrhy jejich napojení na aplikační server. Kapitola se rovněž věnuje operačního systému Linux [6] a jeho souvislostí s vestavěnými systémy.

Čtvrtá kapitola je věnována implementaci systémů pro vybrané úlohy. Jsou zde řešeny 2 varianty implementace výsledného projektu lišící se mimo jiné vybranými ovládanými prvky, jejich uživatelská rozhraní a podrobnější informace týkající se rozšíření navrženého systému.

Pátá kapitola je zaměřena na ověřování funkčnosti obou výše zmíněných systémů pro vybrané úlohy.

Poslední kapitola zhodnocuje výsledky práce a uvádí praktická využití navrženého a implementovaného systému.

Kapitola 2

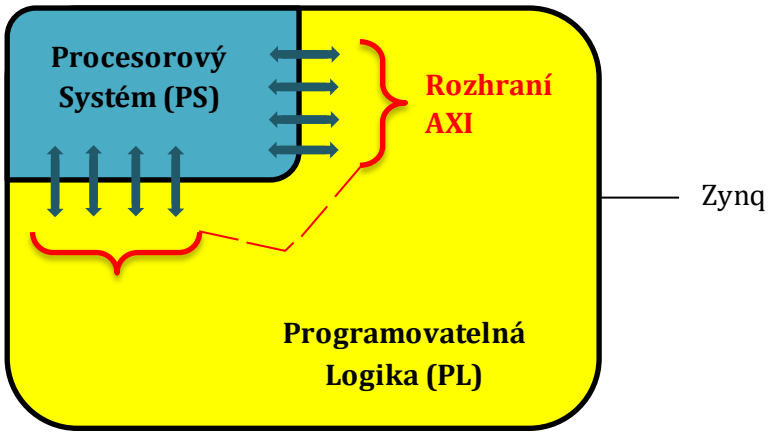
Vestavěný systém na bázi Xilinx Zynq

Vestavěný systém je počítačový systém, který má svou řídicí jednotku zabudovanou do zařízení, které řídí [5]. Typické vestavěné systémy jsou většinou složeny pouze z částí, nezbytných pro splnění účelu, pro které byly navrženy. Systémy na bázi Xilinx Zynq cílí na vývoj prototypových řešení pomocí vývojových desek a dalších zařízení [4].

Tato zpráva je zaměřena na vestavěné systémy z rodiny Zynq ® 7000, založené na architektuře Xilinx programmable system-on-chip (AP SoC)[1]. K jejich ovládání je potřebná znalost architektury Xilinx Zynq, která je v této kapitole podrobněji rozebrána.

2.1 Architektura vestavěného systému Xilinx Zynq-7000

Základním rysem architektury vestavěných systémů na bázi Xilinx Zynq-7000 je kombinace dvou navzájem propojených částí, a to procesorového systému (*Processing system* - PS) a programovatelné logiky (*Programming logic* - PL) [4]. Na obrázku 2.1 je znázorněn zjednodušený model architektury a propojení jednotlivých částí, které jsou více popsány v následujících podkapitolách.



Obrázek 2.1: Zjednodušený model architektury Xilinx

Architektura byla navržena i pro silnou podporu zabezpečení, čehož však žádná dílčí část práce ani zprávy nevyužívá, proto nebude dále rozebírána.

[1] https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

2.1.1 Programovatelná logika

Programovatelná logika tvoří stěžejní část architektury, a to hned z několika důvodů. Mezi ty hlavní patří nezávislost na PS (algoritmy mohou běžet čistě na PL) a vysoká výpočetní rychlost vzhledem k PS [4]. Jádrem programovatelné logiky je programovatelné hradlové pole (FPGA), na kterém je možno mimo jiné realizovat paměťové bloky, aritmetické a logické operace a hlavně vytvářet jednoduchá propojení jednotlivých komponent PL. Pro vytváření programů pro programovatelnou logiku se využívá mnoho technologií, které spojuje typ výstupu, kterým je soubor obsahující data pro FPGA.

2.1.2 Procesorový systém

Procesorový systém je část architektury, kterou řídí procesor. Jádrem systému je dvoj jádrový procesor ARM Cortex-A9 [10]. Je tudíž možno zde provozovat operační systém, a s tím související uživatelské programy [7]. Právě tahle část bude stěžejní při návrhu a implementaci systému pro ovládání vybraných prvků, neboť bude hostovat nejrozsáhlejší část navrhovaného systému.

2.2 Prostředí pro běh uživatelských programů

Při návrhu programů pro vestavěné systémy na bázi Xilinx Zynq je zapotřebí vhodným způsobem rozvrhnout části implementace mezi hardware (PL) a software (PS). Na hardware je vhodné implementovat výpočetně náročné operace a triviální obslužné operace. V softwarové části systému se, oproti hardwaru, proces implementace příliš neliší od vývoje aplikací cílených na známější počítačové platformy, jako jsou například Wintel (složení Intel x86 a operačního systému Windows®), Lintel (Intel x86 a Linux) a další [5].

Uživatelské programy, jimiž bude i část vyvíjeného systému pro ovládání, bude běžet v uživatelském prostoru operačního systému. Možnou alternativou je implementace aplikace běžící čistě na daném procesoru ARM bez asistence a všeobecné přítomnosti operačního systému. Takové aplikace (tzv. *bare metal* aplikace, viz [9]) často vyžadují dodatečné úkony pro inicializace hardware platformu, na které běží.

2.3 Zavádění systému

Pro úspěšné spuštění uživatelských programů v systému je nutností jej správně zavést [4]. Zavádění systému probíhá v několika po sobě jdoucích krocích.

- *Stage-0 Boot (BootROM)* – Při resetu systému, softwaru, nebo zapnutí systému. Implementace je pevně zakódována ve speciální paměti ROM (*boot ROM*) a spouští ji primární procesor.
- *First Stage Bootloader (FSBL)* – Obvykle obsahuje instrukce procesoru k pokročilejší konfiguraci PS a PL. Je plně v režii vývojáře systému.
- *Second Stage Bootloader (volitelné)* – Může být uživatelský program, zavaděč operačního systému, je plně v režii vývojáře.

V prvním kroku je spuštěn *Stage-0 Boot*, jehož jediným cílem je určit umístění následujícího programu (*FSBL*), který se podílí na zavádění, a předat mu řízení. Možností konkrétního umístění FSBL je více. V architektuře Xilinx Zynq patří mezi nejpoužívanější následující stručně charakterizované možnosti [11]:

- NAND – 8 bitové nebo 16 bitová NAND flash zařízení, další část zavaděče se musí nacházet v adresovém prostoru zařízení v prvních 128 MB dané paměti.
- NOR – 8 asynchronních flash zařízení o velikostech až 256 Mb. Podporuje možnost spouštět kód z vlastního umístění, bez přesunu do operační nebo jiné paměti.
- SD Karta – Standardní SD nebo SDHC karty. Souborový systém musí být buď FAT 16, nebo FAT 32, obojí o maximální velikosti 32 GB.
- Quad-SPI – Další část zavaděče se hledá v paměti SPI flash.
- JTAG – Nepodporuje žádné prvky zabezpečení. Uživatel systému je zodpovědný za nahrání další části zavaděče od operační paměti a následný start procesoru. Tahle možnost má velký význam pro ladění.

V implementaci je využita varianta s SD Kartou, na které bude uložen výsledný systém.

2.4 Přehled existujících řešení

Problematikou ovládání vestavěných a jiných systémů se zabývá například firma *Netop* svým produktem *Remote Control*[1], který zvládá ovládání vestavěných systémů vybavených operačním systémem Linux, nebo třeba i Windows® a mnoha dalších.

S problematikou ovládání vestavěného systému přes internet úzce souvisí stále více se rozmáhající fenomén *The Internet Of Things* (*IoT*)[2]. Hlavní myšlenkou IoT je připojování různých vestavěných a jiných zařízení do internetu, a jejich následné ovládání. Oblastí IoT se zabírá velké množství firem, jako třeba Cisco[3], AT&T[4] a spousta jiných.

[1] http://www.netop.com/remotesupport.htm
[2] https://tools.ietf.org/html/rfc7452
[3] http://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html
[4] https://iotplatform.att.com/

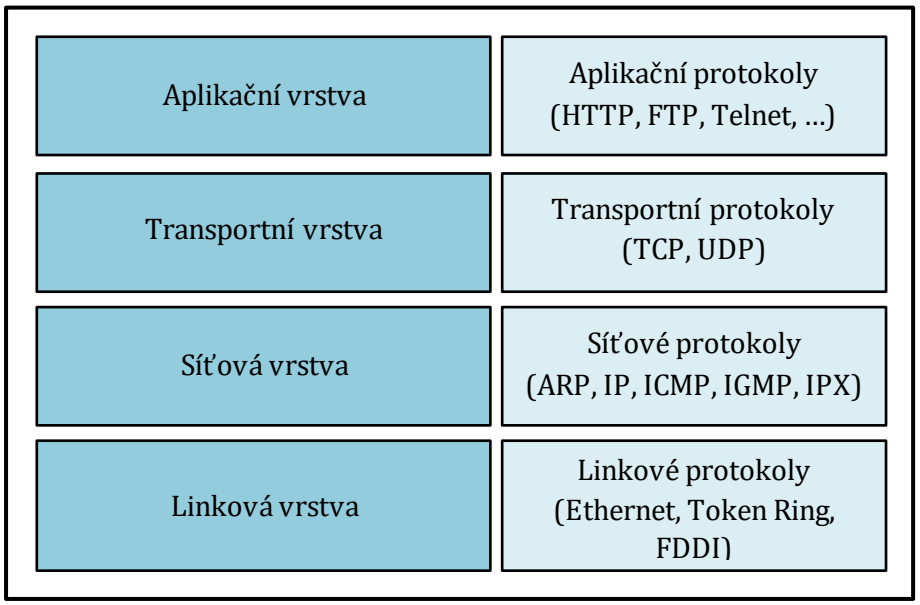
Kapitola 3

Návrh systému

Při návrhu systému musíme zvážit mnoho aspektů, které zásadním způsobem ovlivní následný vývoj. Zadání vyžaduje ovládání vestavěného systému přes internet. Musíme tedy prozkoumat možnosti ovládání vestavěných systému a způsoby komunikace přes internet a mezi systémy obecně.

3.1 Komunikace v síti

Komunikací v síti rozumíme dorozumívání se dvou a více systémů připojených do sítě. Taková komunikace, oproti například mezilidské, musí být deterministicky strojově zpracovatelná [1]. V běžných situacích probíhá dorozumívání pouze mezi dvěma zařízeními, která jsou spolu spojena napřímo (tj. bez žádného mezi prvku) například kabelem, nebo bezdrátovým spojením. Při komunikaci zařízení, která spolu nejsou takovým zapojením spojena, se využívá směrovacích prvků [12]. Dorozumívání systémů v síti probíhá pomocí zasílání datových bloků, nazývaných *packety*. Každý packet má svého odesílatele, a nějakého příjemce. Z důvodu usnadnění implementace komunikačních principů byl v síti internet zaveden model komunikace ve vrstvách TCP/IP [12]. Jeho strukturu znázorňuje obrázek 3.1.



Obrázek 3.1: TCP/IP model

Na obrázku 3.1 jsou zobrazeny všechny 4 vrstvy modelu TCP/IP, které využívá síť internet, spolu s protokoly pracující na dané vrstvě. Daná architektura je navržena tak, aby každá vyšší vrstva nebyla obeznámena s vrstvou nižší, kterou obaluje [12]. Takový princip zajišťuje nezávislost jednotlivých vrstev například na přenosovém médiu, nebo operačním systému. Nejnižší vrstva (*Linková vrstva*) obsahuje údaje o packetu na jeho nejnižší úrovni (například Ethernetová adresa odesílatele a příjemce). Protokol je obvykle závislý na přenosovém médiu. Vrstva Síťová zajišťuje hlavně adresaci zařízení v rámci sítě, předávání a směrování jednotlivých packetů. Transportní vrstva je úzce spjatá s cílovou aplikací a operačním systémem běžícím v zařízení. Jsou v ní implementovány protokoly pro spolehlivý přenos dat (TCP) a nespolehlivý, za to rychlejší, přenos dat (UDP). V nejvyšší úrovni je vrstva Aplikační. Data v aplikační vrstvě jsou plně v režii vývojáře cílové aplikace.

3.2 Možnosti komunikace přes internet

Komunikací přes internet rozumíme dorozumívání dvou a více stran připojených libovolným způsobem do sítě internet na aplikační úrovni. Veškerá taková komunikace v síti internet probíhá pomocí aplikačních protokolů s předem jasně definovanou strukturou a využitím [1]. Důležitým faktorem pro výběr vhodného aplikačního protokolu pro komunikaci jsou požadavky na komunikující strany, složitost zpracování protokolu a s tím související rychlost a důraz na důslednost implementace. Je rovněž rozumné vzít v potaz zátěž sítě při komunikaci. Protokol by neměl přenášet příliš mnoho zbytečných a redundantních dat.

Z výše uvedených důvodů by ideální aplikační protokol měl být navržen přímo na míru daného problému. Implementace takového protokolu by však vyžadovala klientskou aplikaci, která by sloužila přímo pro připojení k vestavěnému systému a jeho ovládání. Jestliže zvážíme nutnost přenositelnosti a nezávislosti výsledné klientské aplikace na klientské platformě jako důležité kritérium pro výběr aplikačního protokolu, tak se jejich počet značným způsobem zredukuje. V době vzniku téhle práce je jedním z nejvíce rozšířených protokolů splňující výše zmíněná kritéria protokol HTTP[1].

3.3 HTTP protokol

HTTP Protokol[1] je aplikační protokol založený na textové a člověku srozumitelné podobě [1]. V současné době je jedním z nejvíce využívaných protokolů pro komunikaci v rámci sítě internet. Komunikace probíhá mezi klientem a serverem. Klient zasílá serveru požadavky v předem dohodnutém formátu, a na každý z nich obdrží odpovídající odpověď serveru.

Jako klientská aplikace zde figuruje program zpracovávající protokol HTTP (tzv. *prohlížeč* [1]). Prohlížečem bývá obvykle v základu vybaven každý operační systém, určený pro stolní počítač [3]. Prohlížeč mimo zpracování protokolu rovněž interpretuje jazyk *HTML* v grafické podobě. Součástí většiny prohlížečů bývá i interpret

[1] https://tools.ietf.org/html/rfc2616 a https://tools.ietf.org/html/rfc7230

jazyka *Javascript*. Oba jazyky (*HTML* i *Javascript*) jsou popsány v následujících podkapitolách.

3.3.1 Struktura protokolu

V době prvotních návrhů protokolu probíhala komunikace na internetu obvykle v textové podobě, a proto je i struktura HTTP protokolu textová. Je nutné rozlišovat strukturu požadavku klienta na server a odpovědi serveru klientovi, i když mají některé prvky společné. Základním společným rysem obou struktur je kompozice hlavičky a těla [1]. Hlavička je část struktury, která je svým obsahem vymezena specifikací[1]. Slouží jako řídicí prvek protokolu, a je umístěna před tělem. Podrobněji je rozebírána v následujících podkapitolách, které se věnují oběma verzím hlavičky. Tělo, které následuje bezprostředně za hlavičkou, obsahuje vlastní přenášená data a může být v krajním případě i prázdné (neobsahovat žádná data). Následující podkapitoly se věnují struktuře požadavku i odpovědi, čehož bude využívat podkapitola 3.3.5.

Struktura hlavičky

Formát hlavičky je definovaný jako nenulová posloupnost řádků, oddělených posloupností znaků `CR LF` (ASCII kódy `0x0D` a `0x0A`). Obsah prvního řádku je určen typem hlavičky (požadavku nebo odpovědi). Zbylé řádky obsahují definici dat ve formátu `NÁZEV: [HODNOTA] CR LF`, kde `NÁZEV` je libovolná neprázdná posloupnost ASCII znaků, mimo oddělovačů. V notaci Rozšířené Backus-Naurově formy (ABNF[2]) jde o následující definici:

```
        NÁZEV = 1 * <any ZNAK or ODDĚLOVAČ >
         ZNAK = < any character ( octets 32 – 126 ) >
    ODDĚLOVAČ = "(" | ")" | "<" | ">" | "@" | "," | ";" | ":" |
"\" | <"> | "/" | "[" | "]" | "?" | "=" | "{" | "}" | MEZERA |
HTAB
       MEZERA = %x20
         HTAB = < horizontal tab >
```

`[HODNOTA]` poté představuje libovolnou (i prázdnou) posloupnost ASCII znaků, mimo oddělovačů. V ABNF se jedná o následující předpis:

```
      HODNOTA = *( OBSAH / MEZERA )
        OBSAH = ZNAK [ 1*( SP / HTAB ) ZNAK ]
         ZNAK = VZNAK / TEXT
         ZNAK = < any visible character >
         TEXT = %x80-FF
    ODDĚLOVAČ = CRLF 1*( MEZERA / HTAB )
       MEZERA = %x20
         HTAB = < horizontal tab >
```

[1] https://tools.ietf.org/html/rfc2616 a https://tools.ietf.org/html/rfc7230
[2] https://tools.ietf.org/html/std68

Hlavičku ukončují 2 po sobě jdoucí posloupnosti znaků `CR LF`. V případě výskytu nepovoleného znaku v poli `HODNOTA` neb `NÁZEV` se takové znaky nahradí odpovídající kombinací znaků řídicích.

Struktura požadavku

Specifikace protokolu předepisuje minimální obsah hlavičky požadavku, který se považuje za platný. Je nezbytné poskytnout název metody, domény, URI prostředku na serveru a verzi protokolu. Obecná forma prvního řádku požadavku v notaci Backus-Naurově formy vypadá následovně:

```
    POŽADAVEK = METODA MEZERA CESTA MEZERA VERZE-PROTOKOLU CRLF
      METODA = "GET" | "POST" | "HEAD" | "PUT" | "DELETE" |
"CONNECT" | "OPTION" | "TRACE"

      MEZERA = %x20
```

kde `CESTA` je URI prostředku na serveru a `VERZE-PROTOKOLU` je verzí protokolu v předem daném formátu. Prostředkem na serveru může být HTML dokument, obrázek nebo libovolný platný koncový bod [1].

Požadavky lze rozdělit do několika typů, podle oblasti jejich využití. Základním kritériem dělení může být kupříkladu umístění uživatelských dat v požadavku. Obecně rozlišujeme následující typy[1]:

- GET – Uživatelská data se nacházejí na prvním řádku hlavičky požadavku v poli `CESTA`. Server na tento typ požadavku odpoví obvyklou odpovědí. Nachází své uplatnění tam, kde je dovolena uživateli prohlížeče jednoduchou úpravou URI, a opětovným odesláním požadavku, změna odesílaných dat. Platí zde ale omezení na délku takto odeslaných formulářových dat, dané specifikací protokolu.
- POST – Uživatelská data se odesílají v těle požadavku. Nejsou tedy běžnému uživateli prohlížeče jednoduše přístupná. Používá se při odesílání například velkého objemu dat, obrázků, souborů a podobně.
- HEAD – Obdoba typu GET s tím rozdílem, že odpověď serveru nesmí obsahovat tělo odpovědi. Používá se například pro zpracování metainformací serveru bez nutnosti přenášet celou odpověď.
- PUT – Používá se jako žádost o změnu stavu určitých prostředků na serveru. Z pohledu běžného uživatele webového klienta však nenalézá oproti předchozím typům tak zásadní uplatnění.
- DELETE – Žádá o změnu asociace mezi jistým prostředkem na serveru a jeho funkcionalitou. Navzdory svému názvu se nejedná o mazání prostředků na serveru.
- CONNECT – Příjemce žádosti je žádán o vytvoření spojení se serverem specifikovaným v poli `CESTA` na prvním řádku hlavičky požadavku a následným obousměrným přeposíláním dat mezi žadatelem a nově vytvořeným spojením. Své uplatnění nachází při směrování komunikace mezi více servery.

[1] Detaily viz https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

- OPTIONS – Žádá o dodatečné informace ohledně možností komunikace, které jsou dostupné pro daný prostředek serveru.
- TRACE – Požadavek je zasílán serveru, který na něj patřičně odpovídá. Z praktického hlediska nalézá největší uplatnění při ladění.

Struktura odpovědi

Odpověď se generuje vždy jako reakce na nějaký požadavek. Struktura odpovědi je skoro identická struktuře požadavku. Jedinou odlišností je první řádek hlavičky. V notaci ABNF má následující podobu:

```
ODPOVĚĎ = VERZE-PROTOKOLU MEZERA STATUS MEZERA POPIS CR LF
 MEZERA = %x20
```

Kde `STATUS` je číselné vyjádření stavu, s jakým dopadlo generování výsledné odpovědi. `POPIS` pak obsahuje textové vyjádření daného stavu. Tělem odpovědi může být HTML stránka, obrázek nebo libovolné vyjádření nějakého prostředku na serveru [1].

3.3.2 HTML

HTML je značkovací jazyk využívaný pro tvorbu webových stránek [3]. Umožňuje vzájemné provázání jednotlivých stránek a poskytuje prostředky pro vizualizaci dat, jako jsou například tlačítka, tabulky, formuláře a mnoho dalších. Struktura HTML souborů včetně více detailů viz [3].

3.3.3 Javascript

Javascript je programovací jazyk, který se v kontextu HTTP protokolu využívá spolu s HTML pro vytváření a manipulaci webových stránek [2]. Jádro interpretu, jemuž se předává zdrojový text, má přístup k HTML stránce přes tzv. *Document Object Model* (zkráceně *DOM*, viz [8]). DOM je hierarchická struktura popisující celý HTML dokument. Umožňuje manipulaci jednotlivých prvků HTML stránky. Zdrojové texty Javascriptu mají typicky příponu `js` a jsou vkládány na webovou stránku buď přes DOM, nebo speciální párovou značku `<script>` přímo v souboru HTML.

3.3.4 Kaskádové styly

Kaskádové styly (CSS) jsou definicí vizuálních stylů jednotlivých HTML elementů [3]. Základní myšlenkou je definice a aplikace stylů pro množinu HTML prvků, odpovídající jisté množině pravidel. Mezi pravidla se řadí například název značky, identifikátor značky (parametr `id`), nebo třeba příslušnost do stylové třídy (atribut `class`) [3].

3.3.5 Možnosti využití protokolu

Protokol disponuje hned několika možnostmi přenosu dat. V době návrhu protokolu se jeho zaměření ubíralo spíše informativním směrem. HTTP servery poskytovaly informace ve formě statických HTML stránek navzájem provázaných hypertextovými odkazy [3]. V dnešní době protokol disponuje hned několika možnostmi obousměrného přenosu dat, které jsou v následujících podkapitolách dále rozbírány.

Formuláře GET a POST

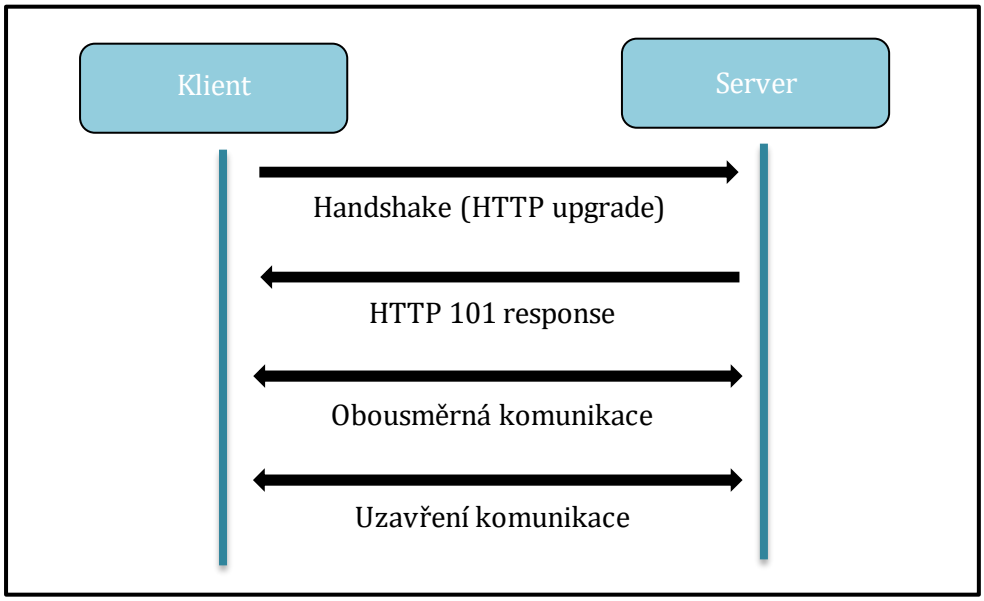
Formuláře slouží už podle názvu k odesílání formulářových dat. Obecně zamýšlený princip výměny dat je následující [3]:

1. Klient zašle serveru požadavek na HTML stránku
2. Server odpoví HTML stránkou obsahující úplnou definici formuláře včetně cíle odeslání formuláře
3. Klient vyplní formulář a odešle jej na formulářem definovaný cíl
4. Server vyhodnotí formulářová data a vygeneruje odpovídající stránku, která se zobrazí klientovi

Konkrétní typ formuláře určuje s konečnou platností hlavička požadavku klienta v bodě 3. Omezení plynou z definice typů hlaviček GET a POST rozebraných v předcházejících podkapitolách.

Websocket

Oproti formulářům poskytuje technologie webových socketů (označovaných jako *Websockety*, viz [8]) větší volnost využití. Princip komunikace je naznačen na obrázku 3.2.



Obrázek 3.2: Princip komunikace pomocí Weboscketů

Na obrázku figurují 2 komunikující strany (*Klient* a *Server*). Je zde vyjádřena posloupnost jednotlivých úkonů, nezbytných pro úspěšné navázání komunikace [8]. Tu zahajuje klient odesláním požadavku na připojení k serveru (tzv. *Handshake*). Takovému požadavku může server vyhovět, a odeslat odpověď se stavovým kódem `101`, nebo zamítnout z libovolného důvodu. Následná komunikace probíhá pomocí datových rámců, obvykle v nějakém dalším protokolu [8]. Komunikaci ukončuje libovolná ze stran.

Důležitým aspektem tohoto řešení je nutnost aplikačního serveru, se kterým webový klient komunikuje. Z toho důvodu je oproti ostatním zde uvedeným řešením

náročnější na prostředky hostitelského stroje. Na rozdíl od nich však z definice umožňuje komunikaci klienta a serveru v reálném čase. Je tedy vhodné pro implementaci aplikací jako například chat, vzdálené ovládání systémů v reálném čase, systémů současné obsluhy více klientů, terminálů apod.

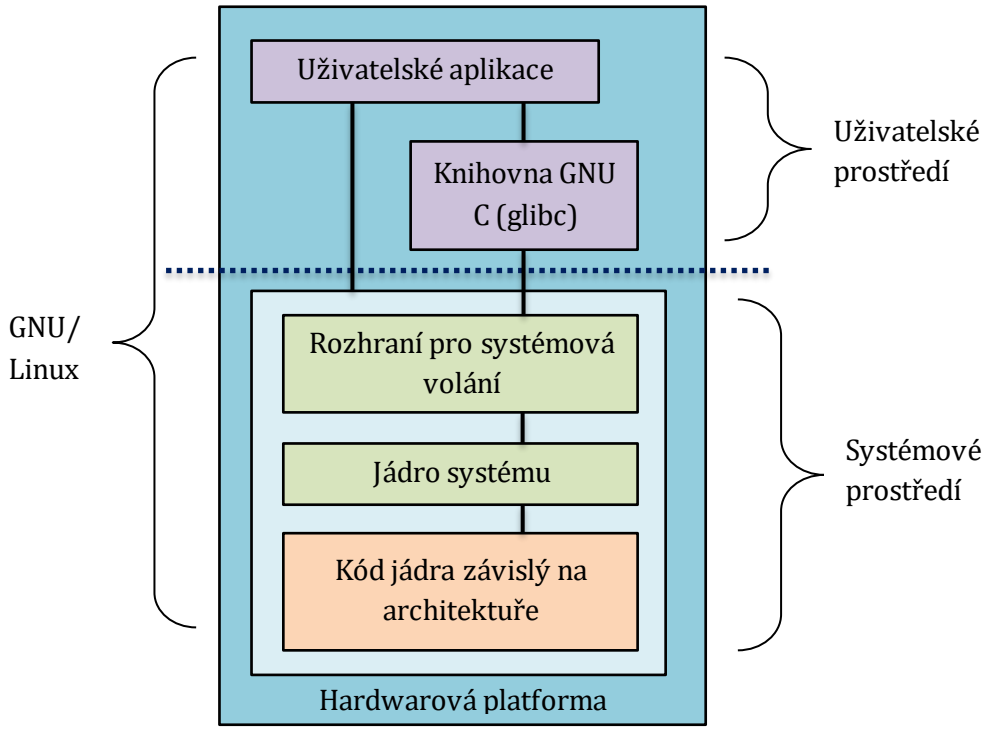
Ajax

Technologie označovaná jako Ajax [8] je úzce spjatá s programovacím jazykem Javascript, probíraným v předchozích podkapitolách. Princip spočívá v odesílání požadavků z Javascriptu na webový server, který na ně odpovídá, jako kdyby je obdržel přímo od webového klienta. V mnoha případech není možné na straně serveru rozpoznat požadavky Ajaxu od požadavků webového klienta.

Vzhledem k časové prodlevě mezi odesláním požadavku a přijetím odpovědi rozlišujeme *synchronní* a *asynchronní* požadavky. Synchronní požadavky zastaví provádění dalšího kódu až do přijetí odpovědi. Zastavení většinou způsobí dočasného nereagování webové stránky. Oproti tomu asynchronní požadavky provádí odesílání požadavku a příjem odpovědi na pozadí, a po dokončení obou akcí je volána funkce, která může pokračovat ve zpracování odpovědi (detaily viz [8]).

3.4 Operační systém Linux

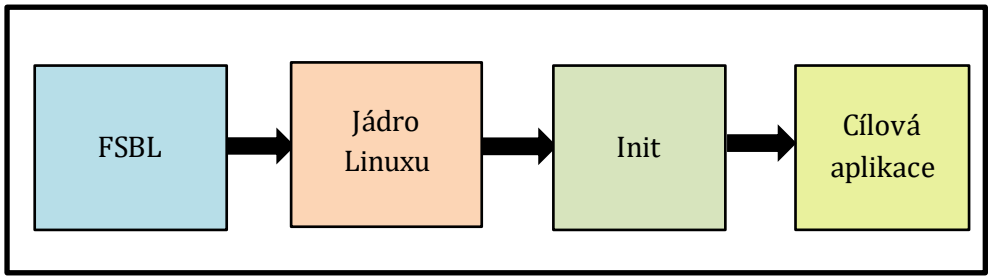
Linux je operační systém založený na myšlence reprezentace všech zařízení jako soubory v souborovém systému [6]. Systém je distribuován ve formě otevřených zdrojových textů, což je jedním z důvodů jeho využívání v oblastní vestavěných systémů. Jednoduchou strukturu systému znázorňuje obrázek 3.3.



Obrázek 3.3: Architektura systému GNU/Linux

Pro úspěšné spuštění systému je zapotřebí zajistit kód jádra závislý na architektuře. Mezi něj se řadí i Strom zařízení, probíraný v následující podkapitole.

Při spouštění systému je ze zavaděče (viz kapitola 2.3) voláno jádro systému, které postupně spouští další procesy. Stručný popis spouštění systému je znázorněn na obrázku 3.4.



Obrázek 3.4: Diagram spouštění operačního systému

3.4.1 Strom zařízení

Při zavádění systému Linux potřebuje jádro operačního systému znát hardware, na kterém bude běžet. Jednou z možností je použití tzv. Stromu Zařízení (*Device Tree*, viz [6]), což je forma databáze popisující kompletní hierarchii hardwaru daného zařízení. Strom zařízení je jednou ze zásadních informací, kterou předává zavaděč systému jádru operačního systému [4]. Je proto nezbytnou součástí vývoje vestavěných systémů.

3.4.2 Adresář /dev

Adresář `/dev` v systému Linux představuje stěžejní součást jádra systému. Jedná se o přípojný bod speciálního virtuálního souborového systému *DEVFS* [6]. Jsou v něm obsaženy speciální soubory a soubory reprezentující jednotlivá zařízení v systému. Tyto soubory mohou být organizovány buď přímo v adresáři samostatně, nebo v podadresářích podle zaměření a funkčnosti daného zařízení. Takové podadresáře jsou označovány jako *třídy* a vytváří hierarchickou strukturu tohoto souborového systému.

3.4.3 Přímý přístup do registrů komponent

Pro většinu periferních zařízení vestavěného systému je zapotřebí nějakým způsobem zpracovávat jejich požadavky na přerušení a přistupovat do vnitřní paměti komponent [10]. Obvykle se jedná o jednoduché registry, které jsou přístupné pomocí přímého mapování paměti. V operačním systému Linux se v adresáři `/dev` vyskytuje soubor zařízení `mem` (obecně znám jako `/dev/mem`). Tento soubor je obrazem hlavní paměti systému a může být využit například pro získávání jinak nepřístupných dat z jádra systému, nebo jejich úpravu [6]. Adresování v tomto souboru (adresa jako počet bajtů od začátku souboru) odpovídá fyzickému adresování paměti systému. Neodborným přístupem k souboru, jako je například zápis na náhodná místa uvnitř souboru, může dojít k pádu programu, nebo celého operačního systému.

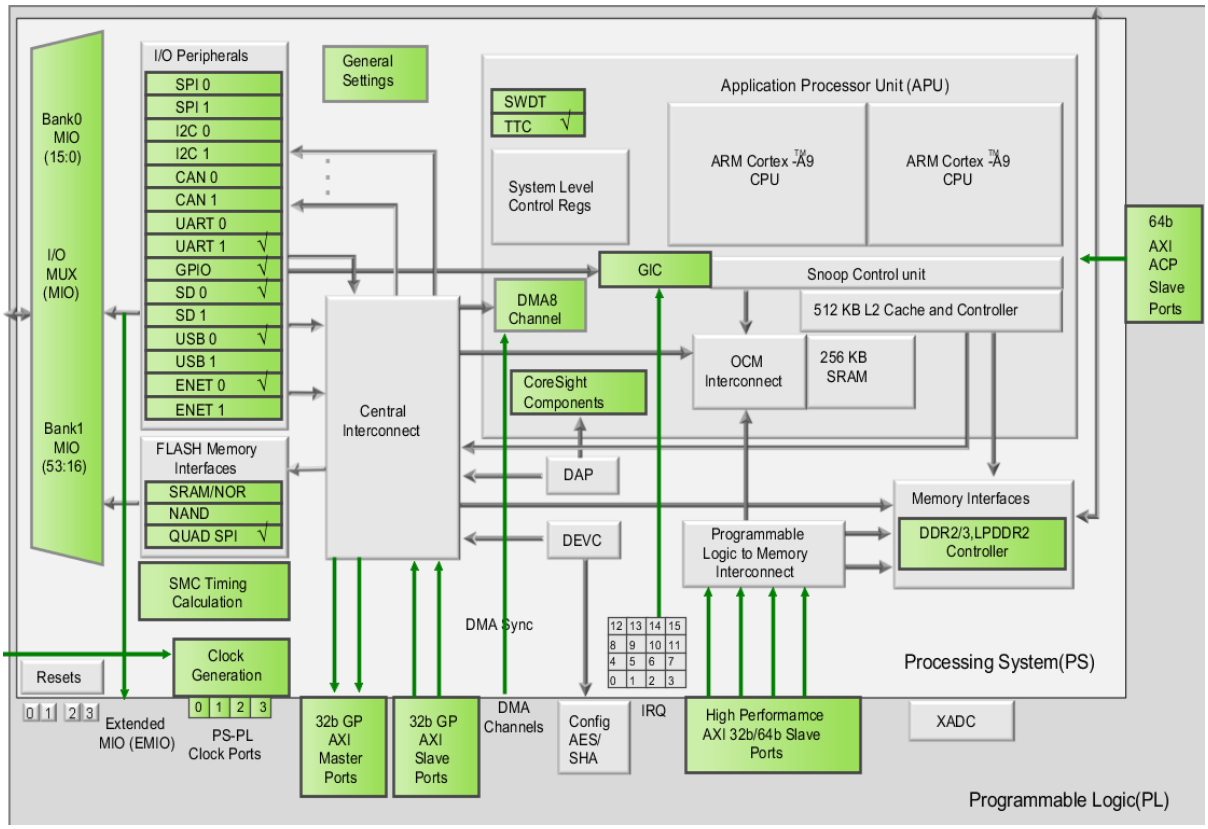
Nepopiratelnou výhodou tohoto přístupu je fakt, že není třeba psát systémové ovladače pro triviální zařízení. Zásadní nevýhoda však vyplívá v nutnosti znát fyzickou adresu daného zařízení, která může být v různých systémech jiná (obvykle však zjistitelná ze Stromu Zařízení, viz podkapitola 3.4.1). Z tohoto důvodu tenhle přístup obecně považuje za nedoporučovaný a v téhle zprávě je uveden pouze jako jednou z možností pro případná budoucí rozšíření.

3.5 Cílené zařízení

Jako cílené zařízení byla vedoucím práce doporučena vývojová deska ZedBoard Zynq-7000 ARM/FPGA SoC Development Board (dále jen vývojová deska) [4]. Vývojová deska je užitečným prostředkem k rychlému a levnému prototypování vestavěných systémů, neboť obsahuje na své rozměry velký počet periferií a také různé typy sběrnic.

3.5.1 Struktura zařízení

Vnitřní struktura zařízení je znázorněna na obrázku 3.5.



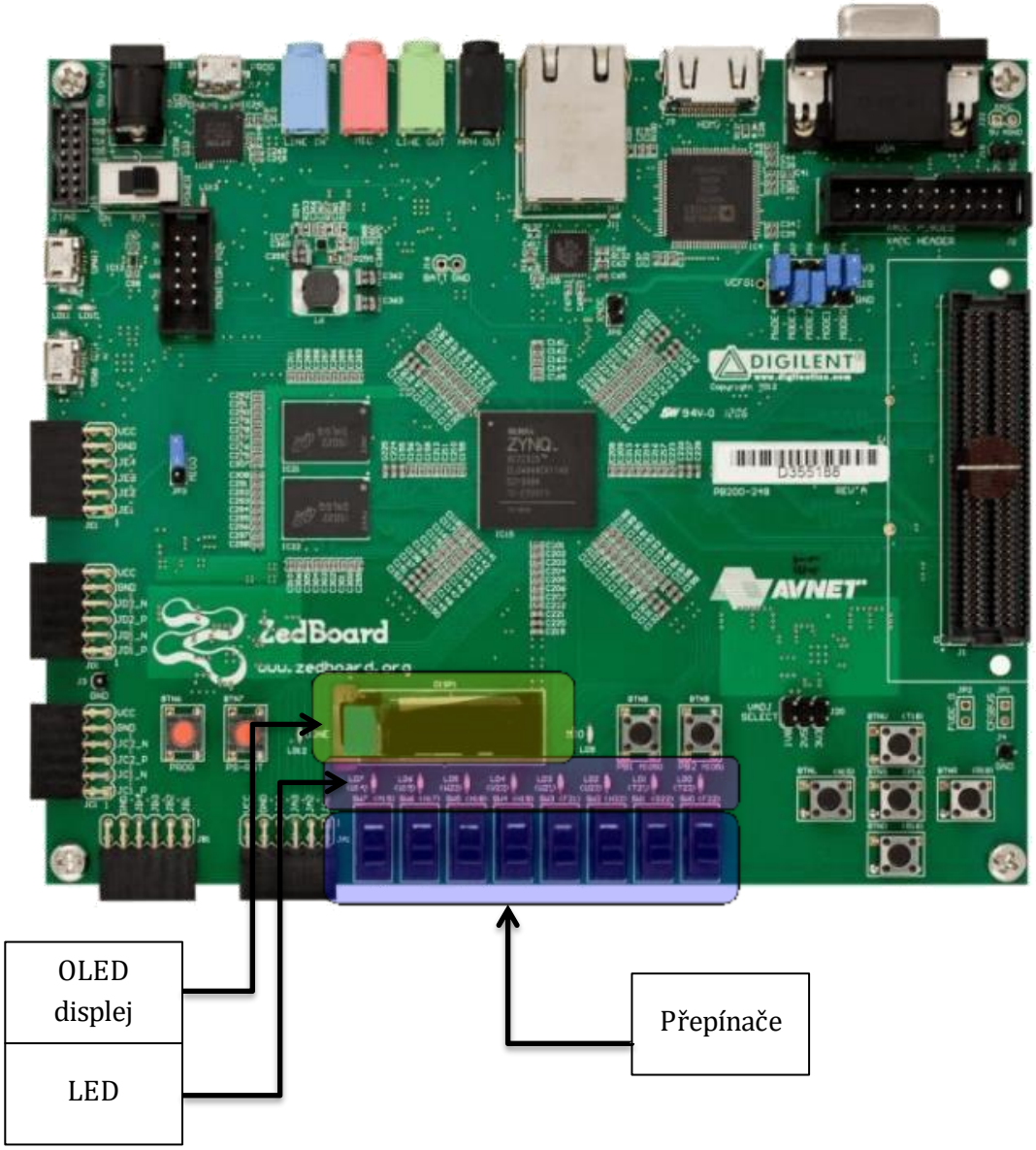
Obrázek 3.5: Schéma vývojové desky ZedBoard Zynq-7000 ARM/FPGA SoC Development Board

Na obrázku je patrné rozložení PS a PL popsané v předchozí kapitole. V levé dolní části PS je znázorněno propojení k PL označené jako *Extended MIO (EMIO)*. Jedná se o sadu pinů

adresovatelných z PS [11]. EMIO je zajímavé tím, že lze velmi snadno namapovat na mnoho periferií tohoto systému, jak toho využívá jedna z částí implementace.

3.5.2 Dostupné komponenty

Na vývojové desce Zedboard se nachází nezvykle velký počet komponent, jak možno pozorovat na obrázku 3.6.



Obrázek 3.6: Cílené zařízení

Pro úlohy ovládání přes internet byly vybrány prvky na desce *OLED displej*, *LED* a *přepínače*. OLED displej umožňuje zobrazování dat pomocí podsvícené matice bodů. Možnost ovládání 8 LED spočívá pouze v jejich rozsvěcování a zhasínání. Ovládání 8 přepínačů je manuální (stav přepínačů se mění ručně přímo na vývojové desce), a vybraná úloha pro jejich ovládání systémem je zjišťování jejich stavu (sepnut a nesepnut). Následující kapitola rozebírá proces implementace.

Kapitola 4

Implementace systému

Implementace systému je realizována na několika úrovních. Na nejvyšší úrovni je cílová uživatelská webová aplikace, pomocí které je vestavěný systém ovladatelný. Na nižších úrovních jsou implementovány nezbytné operace pro přenos dat, ovládání webového serveru a jednotlivých komponent na desce. Ve vrstvě nejnižší je implementován design hardwaru na FPGA. Pro implementaci části webového serveru byl zvolen jazyk C z důvodu co nejvyšší míry úspory prostředků výsledného systému [13]. Pro urychlení implementace byla využita existující knihovna pro práci s HTTP protokolem *Mongoose*[1].

Projekt byl vyhotoven ve dvou variantách, které se od sebe liší na úrovni návrhu operačního systému, designu hardwaru a seznamem komponent, které jsou daným systémem ovladatelné. V následujících podkapitolách jsou rozebrány jednotlivé etapy řešení.

4.1 Webový server

Webový server je mezivrstvou spojující webovou uživatelskou aplikaci a operační systém. Pro implementaci webového serveru byla zvolena existující knihovna Mongoose[1], která je dále rozebírána v následující podkapitole. Úkolem webového serveru je mimo zpracovávání a reagování na požadavky klientů rovněž udržování vnitřních stavů jednotlivých prvků v paměti po celou dobu běhu programu. Dlouhodobé udržování stavů je spolu s úsporou prostředků výsledného systému jedním z hlavních důvodů, proč nebyl pro implementaci vybrán žádný skriptovací ani jinak interpretovaný jazyk.

4.1.1 Knihovna Mongoose

Knihovna Mongoose[1] je knihovna napsaná v jazyce C pro vytváření programů postavených na HTTP protokolu, jako jsou například servery webových stránek, Websocketové aplikační servery, a jiné. Pomocí ní lze jednoduše implementovat HTTP klienty i servery, čehož výsledný systém využívá. Výhoda knihovny spočívá v jednoduchosti jejího použití a taky v tom, že je distribuována jen ve dvou souborech, hlavičkovém souboru a souboru s kódem. Veškerá práce s knihovnou je realizována v jediném souboru s kódem, pro dosažení co nejvyšší abstrakce nad danou knihovnou. Knihovna je licencována pod licencí `GPLv2`[2].

[1] https://docs.cesanta.com/mongoose/master/

[2] https://www.gnu.org/licenses/gpl-2.0.html

4.1.2 Webové rozhraní

Webové rozhraní realizuje statická HTML stránka, která komunikuje s webovým serverem pomocí technologie *Ajax*. Ačkoliv je stránka statická, její obsah je generován pomocí Javascriptu při jejím nahrání. Uživatelské rozhraní pro ovládání vybraných prvků generují soubory Javascriptu, které jsou do stránky vloženy pomocí funkcí Javascriptu rozebraných v předchozích kapitolách této zprávy.

Dynamika webového rozhraní vychází z předpokladu, že ne každý systém pro ovládání bude schopen ovládat všechny vybrané prvky na vývojové desce. Proto je výběr jednotlivých uživatelských rozhraní pro ovládání jednotlivých prvků pevně určen uvnitř volání funkce `insertComponents` v souboru `main.js` ve složce `js` kořenového adresáře webového serveru. Předpokládá se nezávislé vytváření a samostatné generování uživatelského rozhraní uvnitř každého takového souboru. Společným prvkem, nahrávaným přednostně, je v tomto případě technologie *Ajax*, jejíž funkce jsou obsaženy v souboru `ajax.js` ve stejné složce, jako výše zmíněný soubor. Webové rozhraní dále popisují podkapitoly implementace jednotlivých systémů.

4.1.3 Mapování adres na komponenty

Základním principem odesílání ovládacích příkazů na server je odeslání HTTP požadavku na danou stránku technologií Ajax probíranou v kapitole 3. Webový server implementuje několik stránek se speciálním významem, označovaných jako *koncové body*. Jejich přehled je v tabulce 4.1.

Cesta		Popis
/leds	/set/*n*	Rozsvítí diodu *n*
	/clear/*n*	Zhasne diodu *n*
	/	Vrátí aktuální stav všech diod
/switches	/	Vrátí aktuální stav přepínačů
/oled	/clearLine/*n*	Vymaže řádek *n* a přesune kurzor na začátek daného řádku
	/clearAllLines	Vymaže všechny řádky a nastaví kurzor všech řádků na jejich začátky
	/writeLine/*n*/*text*	Připíše *text* na řádek *n* a odpovídajícím způsobem přesune kurzor
	/writeLineClean/*n*/*text*	Vymaže řádek *n*, přesune kurzor na začátek daného řádku, připíše *text* na daný řádek a upraví kurzor
	/append/*text*	Připíše text na místo globálního kurzoru a ten následně upraví
	/appendClean/*text*	Vymaže všechny řádky, přesune globální kurzor na začátek prvního řádku, připíše *text* a upraví globální kurzor
	/	Vrátí aktuální data na displeji

Tabulka 4.1: Přehled speciálních stránek webového serveru

4.1.4 Přenositelnost serveru

Kód serveru je psaný způsobem umožňujícím jednoduchou změnu knihovny pro zpracování HTTP protokolu. Jediná závislost na aktuálně použité knihovně *Mongoose* je v hlavičkovém souboru `server_wrapper.h` a souboru s kódem `server_wrapper.c`. Konkrétně se tedy jedná o strukturu `_mongoose_server`, která je definována ve výše uvedeném hlavičkovém souboru, a dále o všechny členy struktur definovaných v daném souboru, které začínají znakem podtržítka. V uvedeném souboru s kódem jsou závislost na dané knihovně pouze ve funkcích začínajících předponou `wserver`. Při změně knihovny je tedy nutné patřičným způsobem změnit deklarace těchto struktur a definice odpovídajících funkcí.

4.1.5 Možnosti rozšíření serveru

Architektura implementace serveru je navržena způsobem umožňujícím jednoduché přidávání dalších ovládacích prvků. Pro přidání nového ovládacího prvku je zapotřebí do projektu přidat soubor s funkcemi pro ovládání daného prvku a soubor obsahující sadu funkcí, které jsou nezbytné pro inicializaci, ukončení a mapování ovládacích koncových bodů serveru (tzv. souboru *wrapperu*).

Doporučeným způsobem přidávání nových ovládacích prvků je zakomponování hlavičkového souboru wrapperu do souboru `server.h` pomocí direktivy `#include` [13]. Inicializaci takto vloženého prvku je vhodné provádět vhodným voláním funkce ve funkci `server_start`. Pro ukončování naopak voláním z funkce `server_stop`. Pro mapování koncových bodů je příhodně poskytnuta funkce `handle_message`, kde se rovněž vyskytuje několik předdefinovaných volání do wrapperů existujících prvků.

Po přidání a definici koncových bodů serveru v souboru wrapperu do webového serveru je rovněž nutno vytvořit uživatelské rozhraní, které s daným wrapperem bude komunikovat. Pro jednoduchost stačí pouze vytvořit soubor v jazyce Javascript, umístit jej do složky `js` kořenového adresáře serveru a přidat ho do seznamu nahrávaných uživatelských rozhraní, jak je uvedeno v podkapitole 4.1.2 této zprávy.

4.2 Implementace vestavěného systému

Při implementaci vestavěného systému je nutno vytvořit popis designu systému, který bude odpovídat finálnímu propojení komponent programovatelné logiky. Ten se poté vloží na vstup procesu syntézy a implementace vhodně zvoleného nástroje. Výstupem procesu je soubor tzv. *bitstreamu*, který se ve vhodný okamžik nahraje do programovatelné logiky. Programování programovatelné logiky probíhá pouhým zapsáním obsahu souboru bitstreamu do souboru `/dev/xdevcfg` v operačním systému na vývojové desce. K výše popsaným činnostem vývoje designu systému byl vytvořen balík nástrojů *Vivado Design Suite*[1].

Dalším krokem je výběr operačního systému. Pro účely projektu byl vybrán systém Linux. Výstupem implementace je kolekce souborů na SD kartě pro spuštění na vývojové desce. Proces přípravy SD karty a popis spouštění implementace je uveden v příloze C.

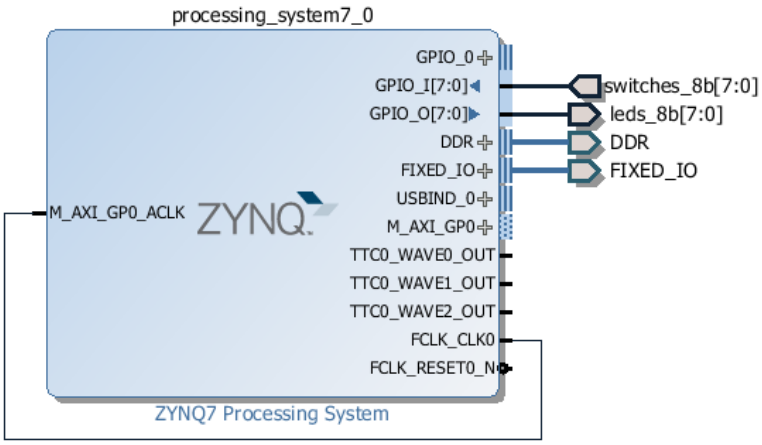
[1] https://www.xilinx.com/products/design-tools/vivado.html

4.2.1 Systém pro ovládání LED a přepínačů na desce

Jako vybrané úlohy implementované tímto systémem byly zvoleny ovládání LED přes webovou aplikaci a monitorování stavu přepínačů, rovněž ve stejné webové aplikaci. Úvodním krokem implementace tohoto systému je rozložení řešeni mezi hardware (PL) a software (PS). Pro co nejjednodušší ovládání vybraných periferií má hardware na starosti vytvoření propojení mezi PS a daných periferních zařízení. K tomuto účelu bylo využito rozhraní GPIO EMIO, které je probíráno v předchozí kapitole.

Hardware

Pro návrh hardwaru byl využit nástroj *Vivado* verze 2016.2 z balíku nástrojů *Vivado Design Suite 2016*. Na obrázku 4.1 je ukázka blokového schéma zapojení jednotlivých částí.



Obrázek 4.1: Blokové schéma hardwaru systému pro ovládání LED a přepínačů

Na obrázku 4.1 je část prostředí jednoho z nástrojů balíku *Vivado Design Suite*. Je zde zobrazen IP blok *ZYNQ7 Processing System*, který v architektuře Xilinx Zynq představuje procesorový systém. Blok slouží jako rozhraní pro Zynq-7000 Processing System, který je podrobněji probírán v předcházejících kapitolách.

Alternativou zvoleného přístupu by bylo přidání IP Bloku *AXI GPIO* a procesem automatizace propojení, který by vytvořil další bloky v návrhu designu. Vývoj by se poté vydal cestou psaní hardware ovladače pro dané zařízení, který by propojoval vybrané komponenty přes rozhraní *AXI* a zpřístupnil jejich ovládání z uživatelského prostředí operačního systému. Důvodem zvolení vybraného přístupu je jednoduchost jeho využití.

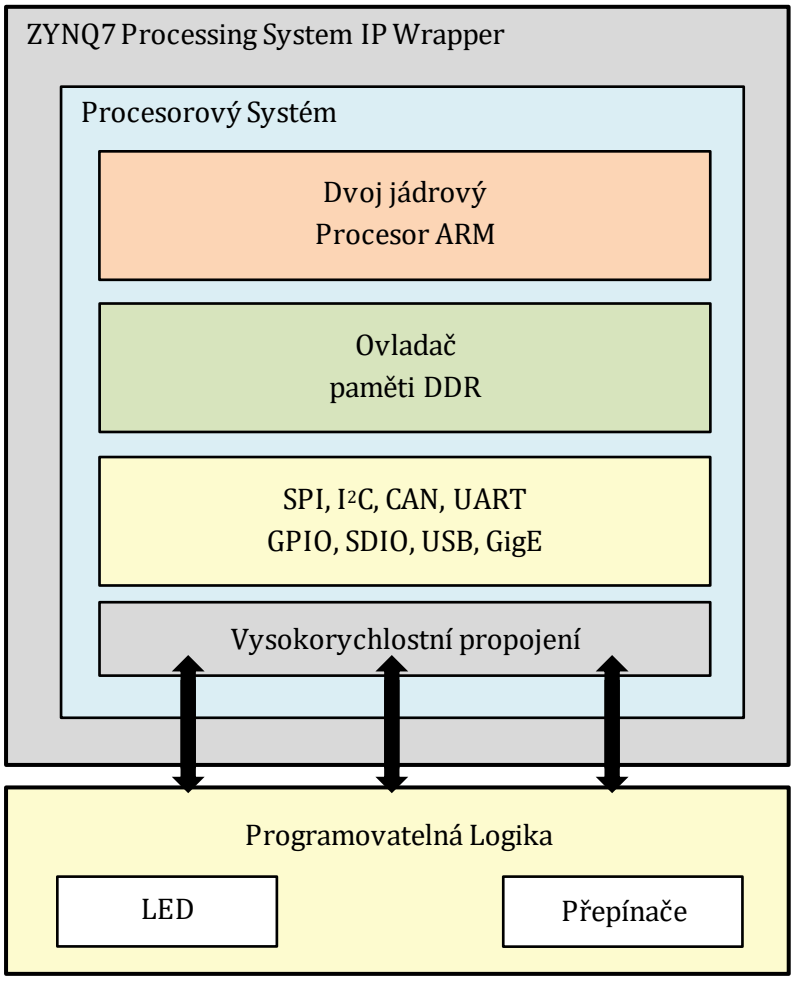
Z obrázku 4.1 je rovněž patrné využití externích portů. Tyto porty (*switches_8b* a *leds_8b*) jsou připojeny k procesorovému systému na GPIO (umístění GPIO v návrhu hardware viz obrázek 4.2). Označení obou portů napovídá oblasti jejich využití. *Switches_8b* je zapojen na vstup GPIO na prvních 8 bitů, zatímco *leds_8b* je zapojen na výstup prvních 8 bitů.

Dále je nezbytné určit napájení a cílové zapojení portů v programovatelné logice, a to konkrétně k LED a přepínačům na vývojové desce. K tomuto účelu slouží definiční soubory zvané *Constraints*. Propojení se realizuje následující definicí:

```
set_property PACKAGE_PIN U14 [ get_ports { leds_8b[7]}]
set_property PACKAGE_PIN U19 [ get_ports { leds_8b[6]}]
set_property PACKAGE_PIN W22 [ get_ports { leds_8b[5]}]
set_property PACKAGE_PIN V22 [ get_ports { leds_8b[4]}]
set_property PACKAGE_PIN U21 [ get_ports { leds_8b[3]}]
set_property PACKAGE_PIN U22 [ get_ports { leds_8b[2]}]
set_property PACKAGE_PIN T21 [ get_ports { leds_8b[1]}]
set_property PACKAGE_PIN T22 [ get_ports { leds_8b[0]}]
set_property PACKAGE_PIN M15 [ get_ports { switches_8b[7]}]
set_property PACKAGE_PIN H17 [ get_ports { switches_8b[6]}]
set_property PACKAGE_PIN H18 [ get_ports { switches_8b[5]}]
set_property PACKAGE_PIN H19 [ get_ports { switches_8b[4]}]
set_property PACKAGE_PIN F21 [ get_ports { switches_8b[3]}]
set_property PACKAGE_PIN H22 [ get_ports { switches_8b[2]}]
set_property PACKAGE_PIN G22 [ get_ports { switches_8b[1]}]
set_property PACKAGE_PIN F22 [ get_ports { switches_8b[0]}]
```

Kde `set_property` má význam příkazu přiřazení a `PACKAGE_PIN` definice typu vlastnosti následované 2 koncovými body. Ve všech výše uvedených případech je prvním koncovým bodem pin na desce (viz [10]) a druhým pin daného externího portu. Všechny piny desce navíc musí být připojeny k nějakému zdroji napájení, což se zajistí také definicí v daném definičním souboru.

Výsledný design je poté nástrojem Vivado syntetizován a implementován. Výstupem implementace je soubor *bitstream* pro pozdější nahrání do programovatelné logiky [4]. Na obrázku 4.2 je pro větší názornost zobrazena zjednodušená struktura IP bloku *ZYNQ7 Processing System*.



4.2: Schéma *Processing System IP* bloku

Na obrázku jsou zobrazeny sběrnice (*SPI*, *I²C*, …) které se mohou podílet na propojení PL a PS [11]. Implementace této části projektu však využívá jen GPIO umístěné v PS za asistence propojení jednotlivých pinů v PS ke komponentám LED a přepínačů. Část systému, která je napsaná v jazyce C, bude vykonávána na procesoru. Při jejím spuštění operační systém příslušný program nahraje do operační paměti (DDR), kde budou uloženy i stavy jednotlivých komponent v systému (které však nemusí za všech okolností odpovídat stavům komponent na desce).

Software

Na softwarovou část lze nahlížet z 2 pohledů. V prvním řešíme operační systém a tvorbu funkcí pro práci s hardwarem, v tom druhém poté distribuci daných funkcí pro webový server. První části této podkapitoly se tedy zaměřuje na operační systém a propojení s vytvořeným návrhem hardwaru.

Jako operační systém byl zvolen systém Linux. Jádro systému i souborový systém samotný byl převzat z předpřipravených použitelných verzí (viz příloha C). Podle přiloženého návodu připravíme SD kartu (překopírujeme soubory jádra

systému, souborového systému a všech zavaděčů, viz zmíněná příloha). V souborovém systému však musíme provést několik změn (viz zmíněná příloha).

Samotné ovládání vybraných komponent z prostředí operačního systému je zajištěno jednoduchou prací s rozhraním *GPIO* a na něj připojeném *EMIO* (viz [11]). V operačním systému jej najdeme v adresáří `/sys/class/gpio`. Práce s rozhraním funguje na principu výběru ovládaného pinu a čtením, případně modifikací, jeho hodnoty. Pro ovládání pinu rozhraní *GPIO* je nutno jej identifikovat. Identifikace je číselná, přičemž v dodaném systému je první adresou rozhraní *GPIO* adresa 906, a první *EMIO* pin na adrese vyšší o 54, tedy 960 (viz [11]). V adresáři se nachází 2 speciální soubory systému pro práci s rozhraním, jak je vidět na obrázku 4.3, znázorňujícím část výstupu příkazu
`ls -l` (příkaz výpisu obsahu adresáře).

```
root@zedboard-zynq7:/sys/class/gpio# ls -l
--w-------    1 root     root          4096 Jan  1  1970 export
--w-------    1 root     root          4096 Jan  1  1970 unexport
```

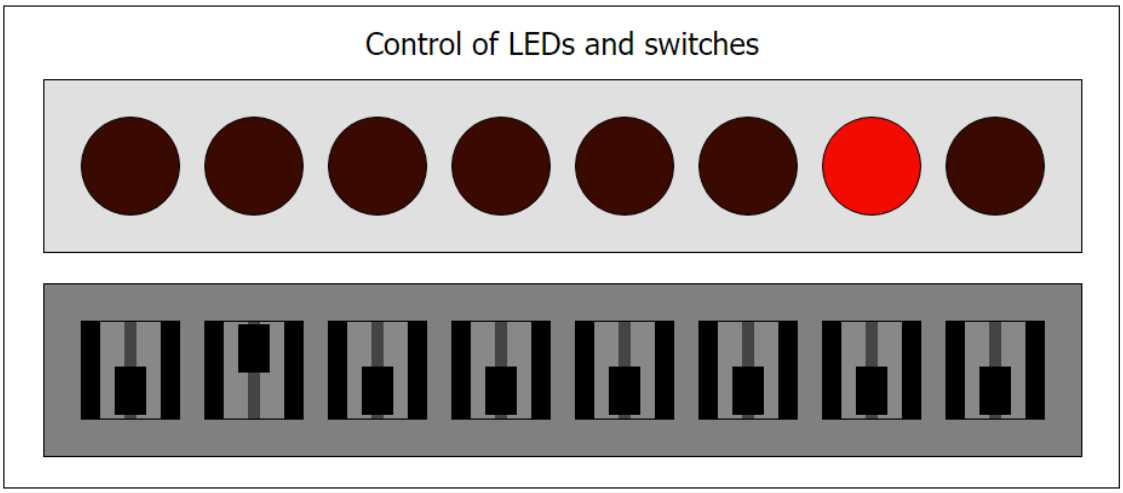
Obrázek 4.3: Část výpisu seznamu souborů v adresáři `/sys/class/gpio`

Pro zahájení práce s vybraným pinem ho musíme vybrat. Výběr provádíme zápisem adresy pinu do souboru `export`, který otevíráme pouze pro zápis, a po zápisu ho zase zavřeme. Po vybrání pinu v adresáři přibude podadresář reprezentující daný pin. V něm nalezneme několik souborů, přičemž projektem jsou využity pouze soubory `direction` a `value`. Soubor `direction` zajišťuje přepínání mezi čtením hodnoty a zápisem hodnoty na daný pin. Pro práci s LED budeme na piny zapisovat a pro práci s přepínači budeme z pinů číst. Pro nastavení pinu pro čtení zapíšeme do souboru `direction` text "`out`" stejným způsobem, jako se zapisuje do souboru `export`. Naopak pro nastavení pinu pro výstup do souboru zapíšeme text "`in`". Na pinu se z definice může nacházet pouze hodnota 0 nebo 1 [10]. Pro LED má hodnota 1 význam rozsvícené diody, pro přepínač má hodnota 1 význam sepnutého přepínače. Hodnota 0 indikuje nesvítící diodu, nebo nepřepnutý spínač. Změna hodnoty se na zařízení projevuje okamžikem ukončení zápisu souboru `value`. Pro ukončení práce zapíšeme adresu vybraného pinu do souboru `unexport` stejným způsobem, jsme ji zapsali do souboru `export`. Vybraný může být vždy jediný pin.

Ačkoliv je v implementaci přirozenější reprezentovat stav přepínačů a diod jako pole 8 bitů, které odpovídá v jazyce C datovému typu `Char` [13], v implementaci bylo využito pole ASCII znaků (8 bitů na 1 užitečný bit). V uvedeném prvním případě by změna stavu znamenala sérii logických operací s následným přiřazením zpět do proměnné. V případě druhém se jedná o prosté přiřazení znaku do pole znaků.

Webová aplikace

V implementaci webové části systému byla využita technologie Ajax pro komunikace s operačním systémem a v něm běžícím webovým serverem. Webová aplikace je navržena tak, aby její ovládání bylo co možná nejjednodušší. Figurují v ní z uživatelského hlediska pouze grafické znázornění LED a přepínačů, jak vyobrazuje obrázek 4.4.



Obrázek 4.4: Snímek webové aplikace systému pro ovládání LED a přepínačů

V horní části obrázku je znázorněno 8 diod, přičemž předposlední je rozsvícena, ostatní jsou zhasnuty. V dolní části jsou ve stejném počtu přepínače, přičemž druhý zleva je v poloze sepnut, ostatní jsou poloze nesepnuty.

Při kliknutí myši na grafickou reprezentaci LED se odešle příkaz na koncový bod webového serveru, který změní stav LED na desce a odešle zpětně nový stav všech diod. V aplikaci se následně synchronizují stavy všech diod se stavy přijatými ze serveru. V případě přepínačů je situace složitější, protože přepínače na desce nedisponují prostředky pro jejich pohyb. Jediná možnost ovládání je tedy manuální manipulace s přepínači na vývojové desce a monitorování aktuálního stav přepínačů. Aplikace odesílá každou vteřinu požadavek na stav přepínačů, jejichž grafickou reprezentaci poté upravuje. Změna stavu obou vyobrazených prvků v uživatelském prostředí znamená pouze přidání, a/nebo odebrání CSS třídy z daného elementu HTML.

4.2.2 Systém pro ovládání vestavěného displeje

Úloha implementovaná tímto systémem je zobrazování znaků a nastavování pixelů na vestavěném displeji. Pro ovládání vestavěného displeje byl využit existující ovladač pro Linux *pmoled* [viz příloha C]. Ovladač poskytuje možnost ovládání displeje přímo z uživatelského prostředí operačního systému přes speciální soubor s názvem `zed_oled` v adresáři `/dev`. Přímým zápisem do uvedeného souboru se mění stav displeje na vývojové desce.

Strukturu vestavěného displeje je možno charakterizovat jako dvourozměrnou bitovou matici o rozměrech 128 sloupců a 32 řádků. Uspořádání bitů formátu, očekávaným ovladačem, je však odlišný od intuitivního formátu vnitřně využívaným v paměti. Pro jednoduchost implementace systém pro ovládání vestavěného displeje pracuje nad zdánlivě dvourozměrným polem indexovaným intuitivně po řádcích zleva doprava směrem dolů. Jednotlivý bit je v tomto poli reprezentován libovolnou nulovou (nepodsvícený bit), či nenulovou hodnotou (podsvícený bit). Díky tomu bylo možno implementovat jednoduché vykreslování znaků. Pole bitů je možno vnímat jako kreslící plátno složené z pixelů, přičemž každý pixel může mít jen 2 barvy.

Vykreslování znaků

Při vykreslování znaku na plátno je nutno řešit hned několik problémů. Na následný vývoj má největší vliv rozhodnutí, zda-li budou všechny znaky zabírat stejně velké místo. V projektu je použito právě takové řešení, ve kterém všechny znaky zabírají právě 64 pixelů, čili 8 na výšku a 8 na šířku. Bylo rovněž nutné vhodně předdefinovat znakovou sadu. Definice takové sady spočívá ve vytvoření bitmapové grafiky (bitové mapy) jednotlivých písmen, a její převod do jednoduše zpracovatelného tvaru. Pro základní použití stačí vytvořit prvních 128 znaků ANSI[1] a uložit je v poli. Uspořádání pole je v implementaci takové, aby ASCII hodnota znaku odpovídala indexu v poli, na kterém jsou uložena odpovídající data znaku. Pro znaky s ASCII hodnotou mimo rozsah velikosti pole je uvažován první znak v daném poli, tedy znak netisknutelný.

Například pro znak `R` je index do pole znaků 82 (ASCII hodnota znaku). Na uvedeném indexu se v poli nachází pole 8 bajtů (v jazyku C se jedná o datový typ `Char`) popisujících bitovou mapu znaku. Konkrétně se jedná o následující data: `0x3F, 0x66, 0x66, 0x3E, 0x36, 0x66, 0x67, 0x00`. Jednotlivé bajty popisují vždy 1 řádek ve výsledné bitové mapě, přičemž první bajt popisuje první řádek, a poslední bajt zase poslední řádek. Uspořádání bitů v bajtu je z důvodu přehlednější implementace takové, že nejméně významný bit zaujímá místo na začátku daného řádku. Po převedení uvedených čísel do binární podoby, za využití výše uvedených pravidel, a následným nahrazení všech výskytů číslice 0 prázdným znakem, dostaneme bitovou mapu, jaká je znázorněna na obrázku 4.5. Pro důraznější naznačení významu jsou všechny výskyty číslice 1 zvýrazněny.

	0	1	2	3	4	5	6	7
0	1	1	1	1	1	1		
1		1	1			1	1	
2		1	1			1	1	
3		1	1	1	1	1		
4		1	1		1	1		
5		1	1			1	1	
6	1	1	1			1	1	
7								

Obrázek 4.5: Bitová mapa písmene *R*

Obrázek lze rovněž chápat jako grafickou reprezentaci levého horního úseku plátna. Podobným způsobem je možno realizovat libovolný znak, případně malý obrázek. Při vykreslování znaku do plátna se ve výsledném poli přepisují všechny hodnoty v rozsahu znaku. Není tedy možné vykreslovat znaky „přes sebe“, jako je tomu například u tiskařských strojů.

[1] Viz prvních 128 znaků dokumentu na adrese https://msdn.microsoft.com/en-us/library/aa245259(v=vs.60).aspx

Jak již bylo zmíněno výše, plátno je tvořeno celkem 32 pixely na výšku a 128 pixely na šířku. Při použití velikosti znaků 8 pixelů se jednoduchý výpočtem odvodí celkový počet řádků znaků, tedy 4, a celkový počet sloupců znaků na řádku, v tomto případě 16. Projekt implementuje vykreslování znaků do 4 řádků samostatně a nezávisle na ostatních řádcích (lokální zápis). Důsledkem toho se ke každému takovému řádku vztahuje i informace o indexu naposledy vykresleného znaku (tzv. *kurzor řádku*). Samotný kód rovněž disponuje prostředkem pro zápis souvislého textu bez ohledu na nezávislost jednotlivých řádků (globální zápis), tedy pozici naposledy zapsaného globálního znaku (tzv. *globálního kurzoru*). Při překročení rozsahu řádku, jak globálního, tak lokálních, se pozice patřičného kurzoru přesune na začátek řádku. V případě globálního kurzoru se kurzor přesouvá na začátek prvního řádku.

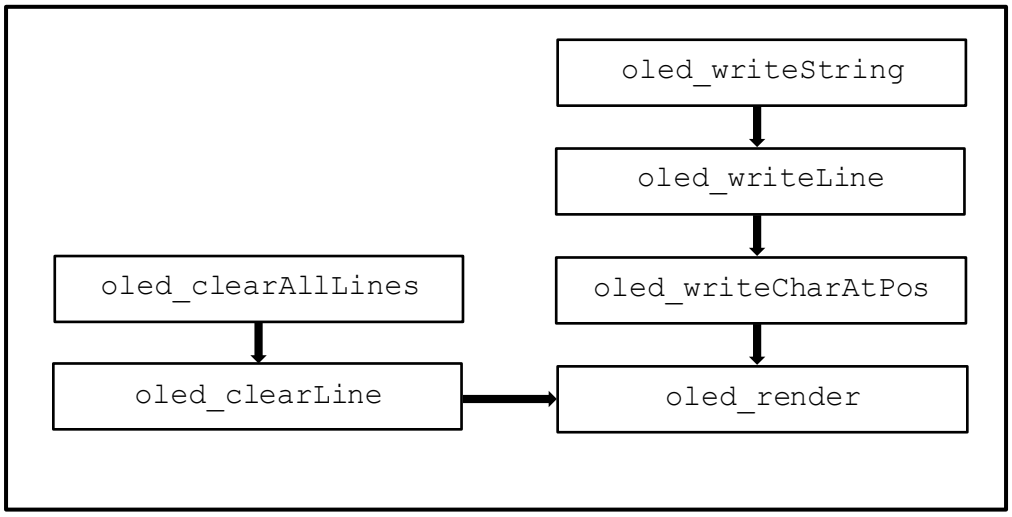
Implementace rovněž dovoluje přímý přístup k jednotlivým pixelům plátna pomocí následující funkce definované v souboru `oled.c`:

```
void oled_setPixel(char* canvas, int pixelX, int pixelY, char enabled);
```

Parametry mají popořadě význam dat plátna, souřadnice pixelů a hodnotu výsledného pixelu. Tahle funkce, ačkoliv vypadá jednoduše, pouze poskytuje rozhraní ostatním částem projektu a případným rozšířením pro vykreslování jiných objektů, než znaků. Vykreslování znaků je realizováno níže uvedenou funkcí definovanou v témže souboru, jako funkce předchozí:

```
void oled_render(char* canvas, int code, int xPos, int yPos);
```

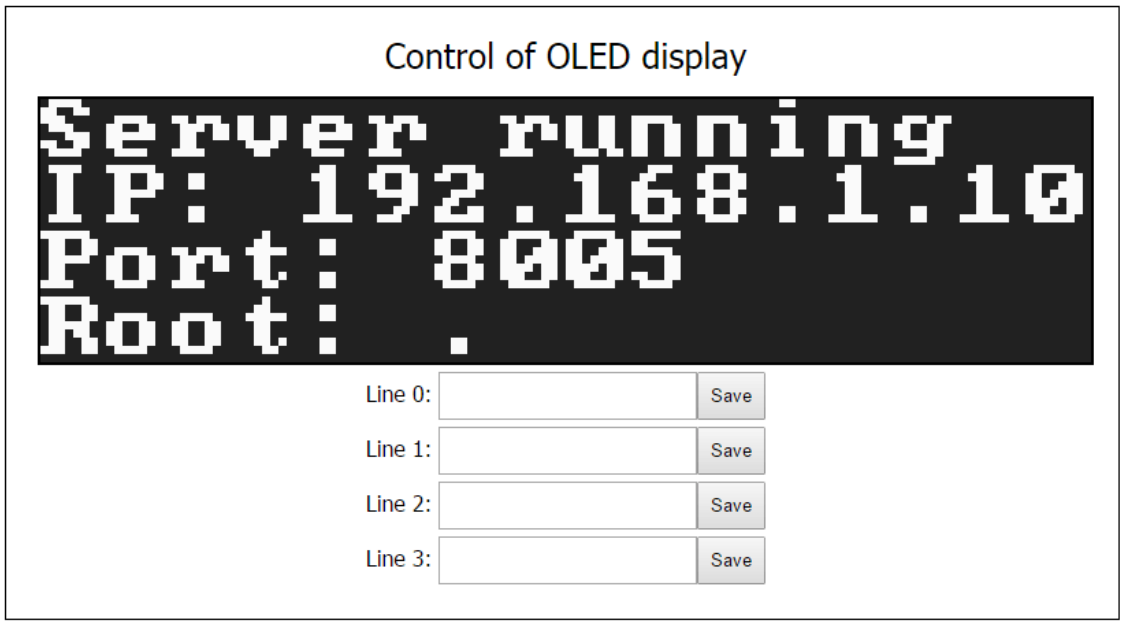
Parametry mají zleva význam dat plátna, ASCII kódu znaku (v jazyce C mu odpovídá i přímo hodnota daného znak, viz [13]) a souřadnice řádku a sloupce na plátně. Nástavbou funkce `oled_render` jsou funkce pro vykreslování znaků do jednotlivých řádků oled_`writeCharAtPos`, `oled_writeLine` a `oled_writeString`. První 2 uvedené funkce pracují s konkrétními řádky, funkce poslední pracuje s globálním kurzorem. Diagram volání funkcí vyjadřuje obrázek 4.6. Nutno podotknout, že funkce `oled_render` nevykresluje na displej, pouze na plátno reprezentované polem v paměti.



Obrázek 4.6: Diagram volání funkcí pro manipulaci s plátnem vestavěného displeje

Webová aplikace

Návrh aplikace byl zaměřen především na zobrazení obsahu displeje na webové stránce, a změnu textu na displeji. Odpovídá tomu i navržené uživatelské rozhraní, které je vyobrazeno na obrázku 4.7.



Obrázek 4.7: Snímek webové aplikace systému pro ovládání vestavěného displeje

V horní části obrázku je zobrazen displej v takovém stavu, jenž odpovídá reálnému stavu displeje na vývojové desce. Níže jsou umístěna 4 textová pole, umožňující změnu textu na jednotlivých řádcích. Komunikace probíhá, stejně jako v předchozí variantě systému, pomocí technologie Ajax. Za zmínku stojí fakt, že server odesílá data displeje v textovém formátu, tedy 4096 znaků (128 na šířku a 32 na výšku). Webová aplikace data rovněž

odesílá v textové podobě, a v odpovědi na každý takový požadavek obdrží úplný stav displeje, který webový server zapíše i na reálný displej na vývojové desce.

4.2.3 Operační systém

Pro první variantu implementace byl zvolen přednastavený systém Linux (viz příloha C). Pro variantu druhou to byl systém *Zedboard Out Of Box*[1] (viz také příloha C), což je výrobcem cílené vývojové desky dodávaná implementace hned několika referenčních řešení. Oba použité operační systémy jsou složeny z jádra, obrazu souborového systému, zavaděče a případně dalších souborů.

Pro dosažení větší flexibility manipulace s výsledným webovým serverem byl do souborového systému přidán skript pro spuštění dané aplikace po zavedení operačního systému. Jádro skriptu tvoří 2 soubory, přičemž jeden z nich je umístěn přímo v obrazu souborového systému. Účelem tohoto souboru je pouze připojit souborový systém SD karty, jejíž soubor zařízení je uložen v adresáři `/dev` (viz kapitola 3). V souborovém systému SD karty je mimo souboru webového serveru uložen i druhý soubor skriptu, kterému první soubor předává řízení. Tento soubor nastaví patřičným způsobem internetové připojení, případně může sestavovat seznam podporovaných ovládaných prvků webové aplikace. Avšak hlavním účelem tohoto souboru je nastavit webový soubor pro spuštění a spustit ho. Postup při úpravě souborového systému obsahuje příloha C.

4.2.4 Spouštění a provoz systému

Spuštěním systému se rozumí správná konfigurace hardware. Zejména nastavení napájení, připojením napájecího kabelu, vložením odpovídající SD karty se systémem a zapojením vývojové desky do sítě pomocí UTP kabelu.

Oba systémy jsou navrženy jako autonomní, proto není z hlediska uživatele nutná jakákoliv další konfigurace. V řadě případů je však ještě nezbytné nastavit vnitřní paměť vývojové desky před jejím prvním použitím s první variantou systému. Její předchozí provozovatel totiž mohl zanechat svá zaváděcí a jiná data v nevolatilní paměti *SPI Flash*, což by mohlo způsobit chybu v zavádění operačního systému. Postup viz příloha B.

4.3 Způsoby ladění

Důležitou součástí vývoje je ladění. V oblasti vývoje vestavěných systémů, oproti například vývoji softwaru na stolní počítače je náročnost ladění výrazně vyšší. Ve fázi vývoje představuje ladění například využívání prostředků pro zobrazování obsahu proměnných v jednotlivých částech kódu, ale také krokování kódu. Za určitou formou ladění lze považovat i ladící výpisy. Při návrhu hardwaru se pro účely ladění obvykle volí počítačové simulace ve speciálních programech, které v řadě případů poskytují velké množství stavových informací různých typů.

Při návrhu softwarové části projektu se využívají ladící programy, jako je například *GDB*. *GDB* je terminálový ladící nástroj využívaný v systémech UNIX. Při vývoji webového serveru byla částečně využita varianta *Remote GDB*, která umožňuje ladění

[1] http://zedboard.org/content/zedboard-out-box-sd-card-image-and-source

programu, který běží v jiném systému, než ladící program. Z větší části byl však server laděn jako lokání program v systému, ve kterém byl vyvíjen. Umožňoval to fakt, že zdrojové texty neobsahují žádný kus kódu, který by byl platformě závislý. Jinak řečeno byly využity pouze standardní knihovny jazyka C.

Způsoby ladění webové aplikace spočívají ve využití ladících prostředků prohlížeče. K tomuto účelu byl využit nástroj *Chrome DevTools*[1].

4.4 Využití pro jiné vestavěné systémy

Ačkoliv je výsledný webový systém cílen na vestavěný systém na bázi Xilinx Zynq, je možné jej adaptovat na libovolný jiný vestavěný systém, který je vybavený operačním systémem a připojením k síti. Celá webová aplikace je navíc přenositelná i mezi jednotlivými variantami systému pro ovládání (s patřičnou úpravou souboru `main.js`). Adaptací se rozumí patřičná úprava bitstreamu a souborů pro práci s jednotlivými zařízeními.

4.5 Výhody, nevýhody a omezení

Tahle kapitola shrnuje výše zmíněné výhody a nevýhody implementace systému v obou variantách.

Výhoda implementovaného systému spočívá v přenositelnosti jeho jednotlivých částí. Výhodou zvoleného jazyka je rozšířená správa paměti, její úspora a velmi rychlé provádění jednotlivých příkazů. Další výhodu představuje zvolená cílová platforma webu. Webovým prohlížečem je v dnešní době v základu vybaven skoro každý operační systém, proto je možno využívat webovou aplikaci z libovolného počítače vybaveného prohlížečem a připojením k internetu.

Zdánlivá nevýhoda vyplívá rovněž ze zvoleného implementačního jazyka, a tou je nutnost vlastní správy paměti. Téměř každé systémové volání, tudíž i alokace paměti, je kontrolováno na chybu. Další nevýhoda spočívá v návrhu hardware první varianty. Při zjišťování stavu přepínačů je výstup daného přepínače připojen na vstup odpovídající diody. To způsobí rozsvícení diody při zjišťování stavu sepnutého spínače. V implementaci je tohle vyřešeno následným obnovením předchozího stavu diody po ukončení čtení stavu přepínače. Způsobuje to ovšem krátké probliknutí diod.

Omezení není mnoho. Jedním z nich je však nemožnost restartovat operační systém druhé varianty implementace. Důvodem je zanedbaná chyba v dodaném operačním systému. Dalším omezením je současný přístup více uživatelů přes webovou aplikaci. Při takových případech se změny provedené v jedné z instancí neprojeví v instancích ostatních. Tahle situace by se vyřešila buď použitím jiné webové technologie, jako například *Websocket*, nebo periodickým dotazováním na stav všech ovládaných prvků.

[1] https://developer.chrome.com/devtools

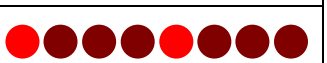
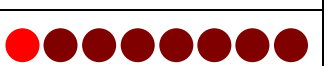
Kapitola 5

Ověření funkčnosti navržených systémů pro vybrané úlohy

Při ověřování funkčností jednotlivých částí systému lze postupovat více způsoby. Tahle kapitola se zabývá ověřováním na úrovni webového serveru, běhového prostředí a vizuální podoby výsledného řešení. Postup ověřování funkčnosti byl zvolen s ohledem na jeho provedení a dokázání správné funkcionality daného systému. Z uvedených důvodů probíhá ve dvou částech. V části první zkoumáme správnost odeslaných příkazů na server pomocí nástroje Chrome DevTools. V části druhé sledujeme přenos a zpracování příkazů, jejich vyhodnocení a pozorování reakce vybraných prvků na vývojové desce.

5.1 Ověření systému pro ovládání LED na desce

Pro ověření funkčnosti ovládání LED budeme z webového prohlížeče sledovat volání, která provádí webová aplikace. Na vývojové desce sledujeme přímo stav diod. Laicky řečeno, sledujeme, jestli se dioda na vývojové desce rozsvítí po kliknutí na nesvítící diodu v uživatelském rozhraní. Výsledky shrnuje tabulka 5.1.

Odeslaný příkaz	Očekávané chování	Chování serveru	Odpozorované chování
`/leds/set/3`	Rozsvícení 4. diody zprava	Očekávané	
`/leds/set/7`	Rozsvícení 1. diody zleva	Očekávané	
`/leds/clear/4`	Žádná reakce	Očekávané	
`/set/clear/3`	Zhasnutí 4. diody zprava	Očekávané	

Tabulka 5.1: Výsledky ověřování systému pro ovládání LED na desce

5.2 Ověření systému pro zobrazování stavů přepínačů

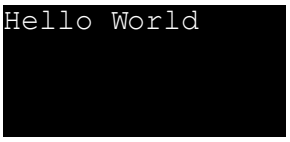
Při ověřování této části systému postupujeme obdobným způsobem, jako v předešlé podkapitole. Změníme stav přepínačů, odešleme požadavek na server a porovnáme výsledná data s očekávaným výsledkem. Jako testovací případy byly vybrány 2 stavy, lišící se na všech pozicích. Výsledky shrnuje tabulka 5.2.

Ručně nastavený stav přepínačů na desce	Číselné vyjádření stavu	Odpověď serveru
	194	194
	61	61

Tabulka 5.2: Výsledky ověřování systému pro zobrazování stavu přepínačů

5.3 Ověření systému pro práci s OLED displejem

Ověřování tohoto systému je o něco náročnější, neboť je nutno ověřit chování velkého množství možností ovládání displeje. Jak je ale patrné na obrázku 4.6, ovládání spočívá ve volání stěžejní funkce `oled_render`. Namísto testování všech možností lze tedy testovat pouze funkce, které výše uvedenou funkci využívají. V tomto případě se jedná o funkce `oled_writeCharAtPos` a `oled_clearLine`. Jedním z koncových bodů serveru pro manipulaci s displejem je `/writeLineClean/`*`n`*`/`*`text`*, jehož implementace obou funkcí využívá. Lze tedy testovat pro reprezentativní výsledky jen tenhle koncový bod. Před testováním byly vykonány úkony pro smazání obsahu displeje (ten je tedy na začátku ověřování prázdný). Výsledky shrnuje tabulka 5.3.

Odeslaný příkaz	Očekávaný stav displeje po vykonání příkazu	Stav displeje navrácený serverem
`/writeLineClean/0/`		Očekávaný
`/writeLineClean/0/Hello%20World`		Očekávaný
`/writeLineClean/0/Test`		Očekávaný
`/writeLineClean/3/%20Line 4.`		Očekávaný

Tabulka 5.3: Výsledky ověřování funkčnosti systému pro ovládání OLED displeje

Kapitola 6

Závěr

Cílem práce bylo vytvořit systém pro ovládání vestavěného systému na bázi Xilinx Zynq. Funkčnost navrženého a implementovaného systému byla ověřena na jednoduchých úlohách ovládání vybraných prvků dostupných na vývojové desce. Na základě výsledků testování jej tedy lze prohlásit za funkční.

Výsledný webový server byl implementován v jazyce C. Celý systém je navíc modulární, a je proto možné do něj libovolným způsobem vkládat nové ovládací moduly. Při kompilaci se jedinou direktivou v jediném zdrojovém souboru dá nastavit seznam podporovaných modulů, bez nutnosti jakékoliv další manipulace s jejich soubory. Tento princip se ukázal jako stěžejní pro dokázání nezávislosti kódu serveru (s výjimkou výše uvedeného souboru definující danou variantu) na výběru varianty implementace vestavěného systému, kterou ovládá. Jinak řečeno obě varianty využívají stejný zdrojový kód, s výjimkou souboru definující konstanty dané varianty.

Uživatelské rozhraní bylo vytvořeno v jazyce HTML generovaného pomocí jazyka Javascript, což zaručuje modularitu výsledného rozhraní vzhledem oběma variantám implementace. Jinak řečeno obě varianty využívají stejný kód pro uživatelské rozhraní, s výjimkou zdrojového souboru definující ovládané prvky. Tento princip umožňuje například omezovat ovládání vybraných prvků některým uživatelům.

V implementaci byla využita technologie Ajax způsobem neumožňujícím komunikaci se serverem v reálném čase. Důsledkem toho systém sice umožňuje současný přístup více uživatelů, ale změny stavu provedené jedním uživatelem se neprojeví v uživatelském rozhraní uživatelů jiných. Další vývoj projektu by se tedy ubíral k technologii Websocketů.

Literatura

[1] SKLENÁK, Vilém. *Data, informace, znalosti a Internet*. V Praze: C.H. Beck, 2001, xvii, 507 s. : il. ; 24 cm. ISBN 80-7179-409-0.

[2] Zakas, N.Z.: *JavaScript pro webové vývojáře - Programujeme profesionálně*, Computer Press, 2009

[3] CASTRO, Elizabeth a Bruce HYSLOP. *HTML5 a CSS3: názorný průvodce tvorbou WWW stránek*. Brno: Computer Press, 2012. ISBN 9788025137338.

[4] CROCKETT, Louise H., Ross A. ELLIOT, Martin A. ENDERWITZ a Robert W. STEWART. *The Zynq book: embedded processing with the ARM® Cortex®-A9 on the Xilinx® Zynq®-7000 all programmable SoC*. Strathclyde Academic Media: Glasgow, 2014. ISBN 978-0-9929787-0-9.

[5] Kamal, R.: *Embedded Systems: Architecture, Programming and Design*. McGraw-Hill Education, březen 2009, ISBN 0070151253, 978-0070151253.

[6] Daniel P. Bovet, M. C.: *Understanding the Linux Kernel*. O'Reilly Media, Inc., třetí vydání, únor 2005, ISBN 0-596-00565-2.

[7] Viktorin, J.: *HW/SW Co-design for the Xilinx Zynq Platform*. diplomová práce, FIT VUT v Brně, Brno, 2013.

[8] GOODMAN, Danny. *JavaScript bible*. 7th ed. Indianapolis, IN: Wiley, c2010. ISBN 0470526912.

[9] Xilinx, J. M.: *Simple AMP: Bare-Metal System Running on Both Cortex-A9 Processors, XAPP1079*, v 1.0.1. leden 2014.

[10] Xilinx*: Zedboard hardware user's guide*, v 1.1. srpen 2012.

[11] Xilinx: *Zynq-7000 All Programmable SoC - Technical Reference Manual, UG585*, v 1.10. únor 2015.

[12] DOSTÁLEK, Libor a Alena KABELOVÁ. Velký průvodce protokoly TCP/IP a systémem DNS. 3. aktualiz. a rozš. vyd. Praha: Computer Press, 2002, xiv, 542 s. ISBN 80-7226-675-6.

[13] KERNIGHAN, Brian W a Dennis M. RITCHIE. *Programovací jazyk C*. Brno: Computer Press, 2006, 286 s. ISBN 80-251-0897-X.

Přílohy

Seznam příloh

Příloha A

Obsah CD

CD obsahuje:

- Pracovní prostředí Xilinx SDK verze 2016.2 se zdrojovými soubory webového serveru
- Vytvořený projekt nástroje Xilinx Vivado 2016.2 první varianty implementace
- Referenční operační systémy obou variant implementace a obecné informace jejich použití a nastavení
- Návod k použití jednotlivých částí

Příloha B

Nastavení vývojové desky před prvním použitím

Postup je velmi snadný. Nastavíme konfiguraci podle [1], přičemž přerušíme libovolnou klávesou zaváděcí program U-Boot v momentě, kdy se v terminálu objeví hláška "`Hit any key to stop autoboot: `" následovaná odpočtem. Dále provedeme následující příkazy:

```
env default -a -f

saveenv
```

První příkaz obnoví prostředí do jeho výchozího stavu. Příkaz druhý poté nastavení uloží. Celý proces, včetně výstupů, shrnuje obrázek B.1.

```
U-Boot 2016.01 (May 03 2016 - 21:27:08 +0530)

Model: Zynq Zed Development Board
Board: Xilinx Zynq
DRAM:  ECC disabled 512 MiB
MMC:   sdhci@e0100000: 0
SF: Detected S25FL256S_64K with page size 256 Bytes, erase size 64 KiB, total 32
 MiB
In:    serial@e0001000
Out:   serial@e0001000
Err:   serial@e0001000
Model: Zynq Zed Development Board
Board: Xilinx Zynq
Net:   ZYNQ GEM: e000b000, phyaddr 0, interface rgmii-id
eth0: ethernet@e000b000
Hit any key to stop autoboot:  0
Zynq> env default -a -f
## Resetting to default environment
Zynq> saveenv
Saving Environment to SPI Flash...
SF: Detected S25FL256S_64K with page size 256 Bytes, erase size 64 KiB, total 32
 MiB
Erasing SPI flash...Writing to SPI flash...done
Zynq>
```

Obrázek B.1: Proces resetu nevolatilní paměti SPI Flash

Po provedení příkazů je možno vývojovou desku libovolným způsobem resetovat, což způsobí úplné spuštění cílového systému.

Příloha C

Přeložení a spuštění implementace v cílovém zařízení

Kompilace webového serveru

Kompilace zdrojových textů probíhá v prostředí vývojového nástroje *Xilinx SDK* verze 2016.2. Po spuštění programu je nutno vybrat pracovní prostředí. Prostředí je ve složce `src/workspace_sdk` kořenového adresáře CD. Vzhledem k nemožnosti úprav CD je nutno složku s prostředím překopírovat na jiné médium a dále pracovat s kopií složky. Po vybrání pracovního prostředí je dále vhodné ověření, případné nastavení kompilátoru, pokud je nástroj *Xilinx SDK* nově nainstalován. V hlavním menu programu `Project` zvolíme volbu `properties`. V levé části nového dialogového okna rozbalíme menu `C/C++ Build` a v něm vybereme položku *`Tool Chain Editor`*. V pravé části dialogového okna nastavíme možnost `Current toolchain` na volbu `Xilinx ARM GNU/Linux Toolchain`. Následně upravíme hodnotu možnosti `Current builder` na `GNU Make`. Nastavení uložíme sticknutím tlačítka `OK`. Pro přeložení projektu volíme položku `Project` v hlavním menu a v něm možnost `Rebuild Project`. Výsledný program se sestaví v adresáři `debug` (případně `release`) ve složce projektu pracovního prostředí.

První varianta implementace zahrnuje i design systému vytvořený v programu Xilinx Vivado verze 2016.2. Zdrojové texty se nachází ve složce `/src/workspace_vivado` kořenového adresáře CD.

Úprava souborového systému

Úprava souborového systému spočívá v extrahování jeho obrazu, přímou úpravu extrahovaných souborů a následnou kompresí zpět do obrazu systému. Postupujeme podle návodu vytvořeného dodavatelem obrazu daného systému[1].

Spuštění implementace na vývojové desce

Pro spuštění implementace je zapotřebí zvolit její variantu. Na přiloženém CD jsou ve složce `sd` podadresáře `variant_1` a `variant_2`, které obsahují všechny náležitosti pro spuštění dané varianty ve vybraném vestavěném systému. Stačí jednoduše překopírovat obsah jednoho z adresářů na čistě zformátovanou SD kartu se souborovým systémem FAT16 nebo FAT32. Webový server je uložen v obou uvedených adresářích ve složce `server`. V obou adresářích je navíc soubor `init.sh`, který zajišťuje dodatečnou

[1] http://www.wiki.xilinx.com/Build+and+Modify+a+Rootfs

inicializaci operačního systému. Ve variantě 2 je v tomto souboru možno změnit předdefinovanou IP adresu serveru, která je 192.168.1.10. První varianta však v operačním serveru obsahuje DHCP server, tudíž je její IP adresa převzata z DHCP serveru, pokud takový existuje. Je rovněž možno nastavit pevnou konfiguraci IP adresy, případně další změny systému v souboru `init.sh`. Samotný soubor serveru je ve složce `server` v adresářích obou variant.