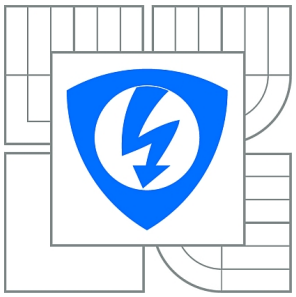
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

POLOHOVATELNÝ STOJAN PRO PŘEHLEDOVOU KAMERU

POSITIONABLE STAND FOR A SURVEILLANCE CAMERA

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. ZDENĚK MATERNA

AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVEL KUČERA, Ph.D.

SUPERVISOR

BRNO 2011

ZDE VLOŽIT LIST ZADÁNÍ

Z důvodu správného číslování stránek

ABSTRAKT

Práce řeší návrh ovládání polohovatelného stojanu přehledové kamery přes Ethernet. Popisuje zvolené hardwarové řešení využívající vývojový kit s mikroprocesorem ARM a připojenou deskou s přídavnou elektronikou. Dále je diskutováno softwarové řešení využívající operační systém Linux vytvořený pomocí balíku Buildroot a upravený pro řízení v reálném čase.

KLÍČOVÁ SLOVA

kamera polohovatelný stojan arm linux

ABSTRACT

Thesis deals with design of control for surveillance camera positionable stand over Ethernet. It describes the selected hardware solution based on development kit with ARM microprocessor and connected board with additional electronics. Thesis also discusses software solution using the Linux operating system, created using Buildroot package and modified for real-time control.

KEYWORDS

camera positionable stand arm linux

MATERNA, Zdeněk *Polohovatelný stojan pro přehledovou kameru*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizační a měřící techniky, 2011. 87 s. Vedoucí práce byl Ing. Pavel Kučera, PhD.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Polohovatelný stojan pro přehledovou kameru“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

[1] General Purpose Input/Output

[2] Timer Counter

ÚVOD

Cílem práce je navrhnout ovládání polohovatelného stojanu pro přehledovou kameru přes Ethernet. Polohovatelný stojan má dva stupně volnosti a umožňuje tak natáčet kamerou v horizontální i vertikální rovině. Pohony stojanu tvoří malé stejnosměrné motorky s převodovkou a snímáním polohy potenciometrem. Pro ovládání stojanu bylo třeba navrhnout a realizovat jak elektronickou, tak softwarovou část.

Kapitola 1 je úvodem do problematiky bezpečnostních kamer a možností jejich polohování. V kapitole 2 je popsán návrh dvou variant elektroniky pro řízení stojanu a návrh možnosti připojení kamery s kompozitním výstupem. Dále se práce věnuje popisu použitého OS Linux a jeho RT rozšíření (kapitola 3) nutného pro získání časových odezev vhodných pro řízení. Pozornost je věnována užitečnému balíku Buildroot, který slouží k nakonfigurování a vytvoření linuxového systému. Stručně popsáno je také API POSIX pro psaní RT aplikací a několik zásad pro jejich programování. Kapitola 4 se věnuje identifikaci pohonů stojanu a návrhu vhodného regulátoru. Poslední kapitola práce (5) popisuje vlastní softwarové řešení problému a v závěru (6) jsou shrnuty dosažené výsledky a navrženy možnosti dalšího vývoje.

1 BEZPEČNOSTNÍ KAMERY

Přehledové, jinak také bezpečnostní kamery se dnes používají na řadě míst z důvodu ochrany osob a majetku, v průmyslu například pro vzdálený dohled u nebezpečných procesů.

Přehledové kamery je možné dělit podle umístění na vnitřní a vnější. Na kamery určené do interiéru v podstatě nejsou kladeny žádné speciální požadavky, protože se předpokládá stálá teplota i vlhkost prostředí. Jediným rizikem je tedy úmyslné či náhodné poškození. Složitější situace je u vnějších kamer, kde je nutné zajistit jednak odolnost proti povětrnostním vlivům, korozi, prachu, ale také může být nezbytné vytápění (odstranění námrazy), případně odmlžování sklíčka krytu kamery. Zcela speciální kategorií jsou kamery pro průmyslová prostředí, kde je požadována maximální spolehlivost, odolnost proti vlivům agresivního prostředí, vibrací, silného elektromagnetického pole atd.

Co se týče způsobu přenášení obrazu, je možné kamery rozdělit na analogové a digitální. V nabídce firem dnes jasně dominují digitální bezpečnostní kamery nabízející vyšší rozlišení[1] a možnosti pokročilého zpracování obrazu. Digitální kamery bychom mohli rozlišovat podle média pro přenos obrazu. Standardem dnes začíná být Ethernet, případně jeho bezdrátová podoba WiFi, kdy odpadá nutnost instalace metalického vedení a zároveň i možnost jeho poškození. Výhodou metalického vedení je však možnost napájet kameru pomocí POE[2]. Stále používané jsou i kamery s analogovým výstupem (obvykle v normě PAL[3]), jejichž nezanedbatelnou výhodou je nízká cena a snadné připojení k analogové televizi, případně videorekordéru.

Kamera může být umístěna buď staticky, pro snímání jednoho místa, nebo na polohovatelném stojanu, díky čemuž je možné s jednou kamerou pokrýt širší prostor. Polohu kamery řídí buď operátor pomocí ovládacího panelu, nebo software. Moderní bezpečnostní kamery vybavené pokročilým softwarem mohou samy detekovat pohyb, zaostřit jej a sledovat, případně upozornit operátora. U kamer s pokročilým softwarem je častou funkcí také ukládání určitých nastavení polohy pro jejich rychlé a přesné změny. Obvyklou funkcí bezpečnostních kamer je pořizování záznamu, přičemž některé kamery umožňují záznam spustit pouze při detekovaném pohybu, čímž může být významně ušetřen prostor pro archivaci záznamů. Polohovatelné kamery mohou být řešeny jako jeden konstrukční celek nebo kombinace polohovatelného stojanu a libovolné kamery. Výrobci jsou obvykle polohovatelné kamery označovány zkratkou PTZ[4].

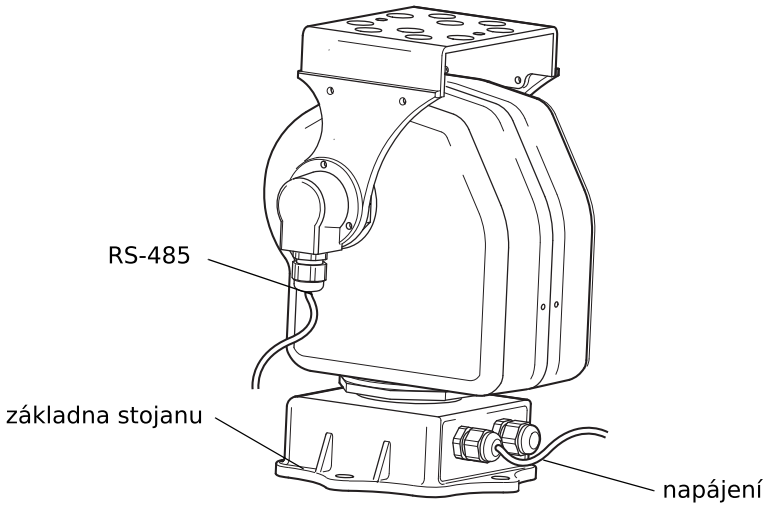
[1] Běžně 2 Mpx s 1200 řádky, oproti max. cca 600 u normy PAL.

[2] Power Over Ethernet - napájení přes Ethernet (48 V, 400 mA)

[3] Phase Alternating Line

[4] Pan Tilt Zoom - vyklápění, natáčení, zoom

Profesionální řešení polohovatelných stojanů kamer se vyznačují robustní konstrukcí a typicky ovládáním přes rozhraní RS-485 protokolem Pelco-P, nebo Pelco-D. Příkladem spadajícím do této kategorie může být stojan YP3040 od výrobce Axis umožňující naklápění a otáčení, řízení zmíněným protokolem Pelco (ve variantě D) a uložení 32 předvoleb polohy. Stojan disponuje krytím IP66, takže může být použit jak v interiéru, tak ve vnějším prostředí.



Obr. 1.1: Polohovatelný stojan Axis YP3040 (převzato a počeštěno z materiálů výrobce)

Protokol Pelco je používán výrobci (Pelco, Axis, Bosch atd.) k přenosu informací mezi ovládacím pultem, PC, nebo jiným zdrojem a PTZ kamerou. Fyzickou vrstvu protokolu tvoří sériové rozhraní RS-485. Výhodou tohoto řešení je jednoduchost a odolnost proti rušení. Zpráva protokolu je velmi jednoduchá, dlouhá 7 bytů. Tvořená je synchronizačním bytem, adresou cílového zařízení, dvěma příkazy, dvěma byty dat a kontrolním součtem, jak ukazuje tabulka 1.1.

byte	1	2	3	4	5	6	7
obsah	synch.	adresa	příkaz 1	příkaz 2	data 1	data 2	kont. součet

Tab. 1.1: Zpráva protokolu Pelco D

Hodnota jednotlivých bitů příkazů 1 a 2 potom určuje požadovanou akci (např. pohyb vzhůru, otevření clony a zaostření na blízko). Byte data 1 určuje rychlost otáčení (pan), data 2 rychlost naklápění (tilt). Kamera nebo polohovatelný stojan příjem dat nijak nepotvrzuje, pouze může upozornit na výskyt 0-8 alarmů.

Další možností v případě potřeby polohovatelné bezpečnostní kamery je využití malé IP kamery, které nabízí mnoho výrobců. Jsou poměrně rozšířené, a proto je i jejich cena relativně nízká, velmi však závisí na požadovaných schopnostech. Většinou jsou v provedení indoor (výjimkou je např. typ Ovislink OD-600HD nebo JVC VN-V686WPBU). Sledování obrazu i ovládání polohy probíhá přes specializované webové rozhraní. Rozlišení takovýchto kamer se obvykle pohybuje mezi 640x480 a 1280x1024 pixely. Polohování bývá možné naklápěním, nebo naklápěním i otáčením. Běžně podporovanými formáty přenosu videa jsou MPEG-4, MJPEG a některé dražší kamery podporují také H.264. V závislosti na ceně mohou mít tyto kamery také WiFi, integrovaný mikrofon, možnost připojení reproduktoru, slot na paměťovou kartu, detekci pohybu v obraze, infračervené přisvícení, programovatelné GPIO, optický zoom, POE.

Obr. 1.2: Indoor IP kamera CISCO WVC210 (převzato z amazon.com)

Zajímavou, byť statickou kameru s pokročilými funkcemi nabízí český výrobce Jablotron. Typ EYE-02 komunikuje bezdrátově s využitím sítě GSM nebo 3G pro přenos videa na mobilní telefon. EYE-02 kromě samotné kamery obsahuje také infračervené přisvícení, PIR čidlo, mikrofon, náklonový a vibrační senzor detekující neoprávněnou manipulaci. Výhodou může být zálohovací baterie pro provoz při výpadku napájení. Pro tuto práci činí kameru EYE-02 zajímavou použití mikroprocesoru Atmel AT91SAM9260 a možné využití operačního systému Linux [15][5]. Citovaná

[5]V práci [15] jsou popisovány úpravy OS Linux pro tuto kameru. Jestli je v kameře tento systém skutečně použit není známo.

práce se věnuje zejména vývoji ovladače rozhraní ISI[6] použitého mikroprocesoru, ke kterému je v kameře připojen CMOS snímač MT9V011.

[6]Image Sensor Interface - rozhraní pro připojení obrazového snímače

2 HARDWARE

Pro potřeby řízení polohy stojanu po síti Ethernet byl jako vhodný mikroprocesor vybrán typ Atmel AT91SAM9260 blíže popsaný v 2.1. Ten je pro podobné aplikace výhodný díky příznivé ceně, podpoře standardních OS[1], rozhraní Ethernet, USB a vysokému výkonu. Nezanedbatelnou výhodou je také dostupnost vývodového pouzdra umožňujícího ruční pájení. Díky výkonu až 210 MIPS[2] by bylo do budoucna možné nejen streamovat obraz z připojené kamery, ale i provádět jeho složitější zpracování.

V rámci práce byly navrženy dvě možné varianty elektroniky. První varianta (popsaná v 2.2) počítala s vlastní navrženou deskou pro mikroprocesor. Kvůli technologickým obtížím a chybám v návrhu desky však tato varianta nebyla realizována. Použitá druhá varianta (2.3) spočívá ve využití levného vývojového kitu Olimex SAM9-L9260 založeného na zvoleném typu mikroprocesoru. Ke kitu je pomocí rozšiřovacích konektorů připojena deska s budiči motorů a další potřebnou elektronikou. V případě realizace první varianty by bylo její výhodou možnost připojení kamery s digitálním výstupem pomocí rozhraní ISI.

2.1 AT91SAM9260

AT91SAM9260 je 32 bitový mikroprocesor od firmy Atmel založený na jádře ARM9. Díky vysokému výkonu až 210 MIPS a jednotce pro správu paměti MMU[3] umožňuje běh standardních OS (Linux, WinCE apod.) Jeho použití pro tuto práci je výhodné také proto, že má rozhraní ISI pro připojení kamery a snadný přenos obrazu z ní do paměti. Kameru (nebo jiné zařízení) je však možné připojit i pomocí USB host portu. V následujících kapitolách budou popsány periferie využité v práci a vyžadující v OS Linux konfiguraci z uživatelského programu nebo ovladače. Popis rozhraní (USB, Ethernet) jejichž funkčnost zajišťuje OS je možné najít v [1].

2.1.1 Bootloader

Paměť ROM integrovaná v mikroprocesoru obsahuje bootloader (obdobou je např. BIOS u PC). Bootloader je program starající se o prvotní konfiguraci mikroprocesoru (DBGU - sériový port pro ladění, USB device port, inicializace SRAM) a spuštění uživatelského programu. Pořadí ve kterém se hledá spustitelný program v jednotlivých pamětech je následující: DataFlash připojená na SPI, NAND Flash. Pokud je v některé

[1] operační systém

[2] Million instructions per second

[3] Memory Management Unit

z pamětí nalezen spustitelný kód (obvykle jde o bootloader druhé úrovně, například U-Boot), proběhne jeho přesunutí do SRAM a spuštění. Pokud je hledání neúspěšné, dojde k spuštění SAM-BA monitoru a na rozhraní DBGU a USB se čeká na spojení s PC, odkud je následně možné provést download programu do paměti.

2.1.2 Vstupně výstupní piny

Mikroprocesor AT91SAM9260 disponuje třemi 32 bitovými porty - celkem je tedy k dispozici 96 programovatelných vstupně-výstupních pinů (GPIO). Správu GPIO má na starosti jednotka PIO[4], která nastavuje jednotlivé piny jako vstup, výstup, nebo jim přiřadí speciální funkci zajišťovanou některou z periférií (každý pin může mít až dvě speciální funkce).

U pinů nastavených jako vstup je možné aktivovat integrovaný pull-up rezistor, nastavit generování přerušení při změně stavu, zapnout filtrování krátkých impulzů. Výstup může být nastaven do režimu otevřený kolektor. Jednotka PIO umožňuje synchronní změnu až 32 GPIO pomocí zápisu do jednoho 32 bitového registru, kde každý bit odpovídá jednomu pinu. Je tedy možné naráz nastavit či vynulovat celý jeden port. Pokud by bylo požadováno v jednom okamžiku některé piny jednoho portu nastavit a jiné vynulovat, je možné pro zvolené piny povolit zápis do registru PIO_ODSR (viz. [1]). Pokud je požadován pouze výstup, může být jednotka PIO deaktivována, čímž se sníží spotřeba. Pro využití přerušení a možnost číst stav pinů je naopak nutné pomocí PMC[5] tuto jednotku povolit. Přerušení od změny stavu pinů nejsou v práci využita a proto nebudou dál popisována.

Přehled vybraných registrů pro práci s GPIO uvádí tabulka 2.1. Pro přístup k registrům je nutné znát bázovou adresu daného portu (viz. B.13) a k ní přičíst offset požadovaného registru (první sloupec tabulky 2.1).

2.1.3 Čítače časovače

Mikroprocesor AT91SAM9260 obsahuje dva bloky 16 bitových tříkanálových čítačů časovačů (TC). Ty je možné nastavit pro měření frekvence, generování periodického přerušení nebo PWM[6]. V práci jsou použity pro generování PWM signálu s proměnou střídou pro řízení motorků stojanu. Každý kanál má dva výstupy (TIOAn, TIOBn) na kterých může být generována PWM jedné frekvence a různé střídy. Celkově je tedy možné generovat až 12 různých PWM, protože ale jeden z bloků čítačů časovačů (který je možné nastavit v konfiguraci jádra) využívá také OS Linux jako

[4]Parallel Input/Output Controller

[5]Power Management Controller

[6]pulse width modulation

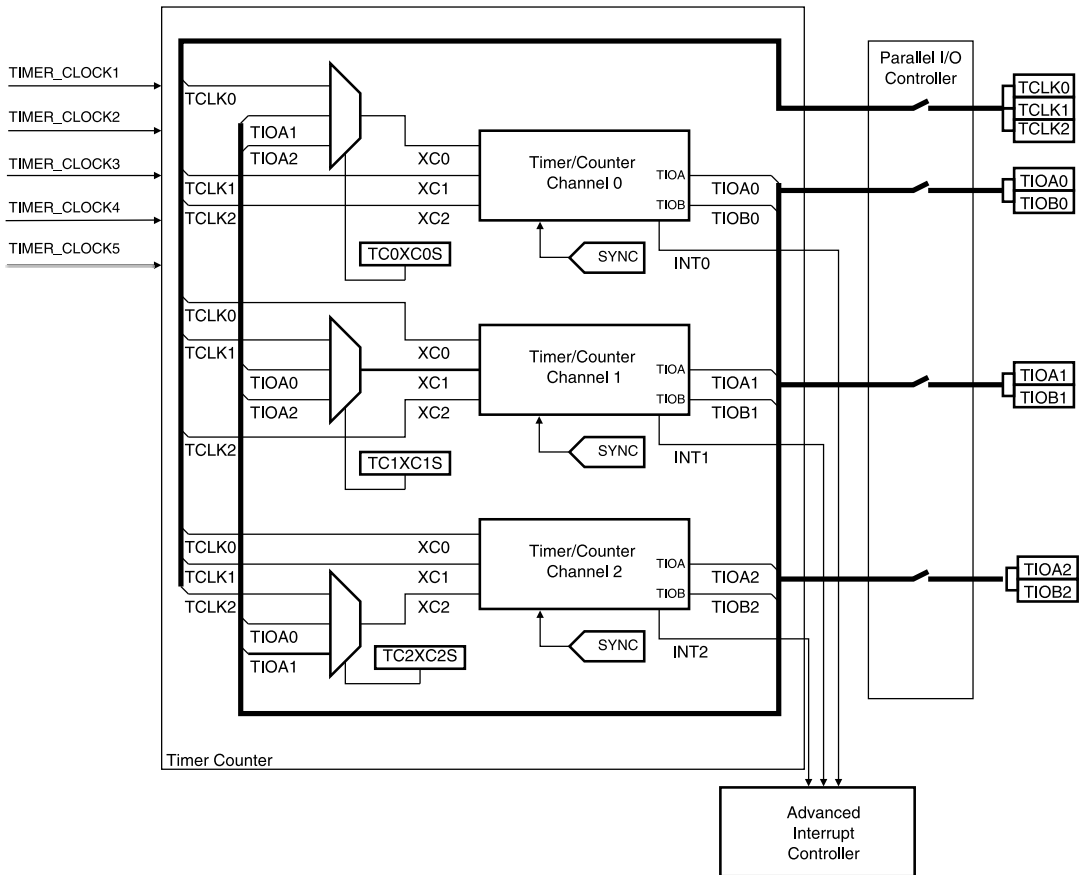
offset	registr	jméno	popis
0x0000	PIO_PER	PIO Enable	W1: pin je řízen PIO
0x0004	PIO_PDR	PIO Disable	W1: pin je řízen periférií
0x0008	PIO_PSR	PIO Status	R0: pin je řízen periferií R1: pin je řízen PIO
0x0010	PIO_OER	Output Enable	W1: pin nastaven jako výstup
0x0014	PIO_ODR	Output Disable	W1: pin nastaven jako vstup
0x0018	PIO_OSR	Output Enable	R0: nastaven jako vstup R1: nastaven jako výstup
0x0020	PIO_IFER	Input Filter Enable	W1: filtr zapnut
0x0024	PIO_IFDR	Input Filter Disable	W1: filtr vypnut
0x0028	PIO_IFSR	Input Filter Status	R0: filtr vypnut R1: filtr zapnut
0x0030	PIO_SODR	Set Output Data	W1: výstup do log. 1
0x0034	PIO_CODR	Clear Output Data	W1: výstup do log. 0
0x0038	PIO_ODSR	Output Data Status	R0: výstup v log. 0 R1: výstup v log. 1
0x003C	PIO_PDSR	Pin Data Status	stav pinu nezávisle na konfiguraci (vstup, výstup, periferie)
0x0060	PIO_PUDR	Pull-up Disable	W1: pull-up rezistor vypnut
0x0064	PIO_PUER	Pull-up Enable	W1: pull-up rezistor zapnut
0x0068	PIO_PUSR	Pad Pull-up Status	R0: pull-up rezistor zapnut R1: pull-up rezistor vypnut
0x0070	PIO_ASR	A Select	W1: periferie A
0x0074	PIO_BSR	B Select	W1: periferie B
0x0078	PIO_ABSR	AB Status	R0: periferie A R1: periferie B

Tab. 2.1: Adresy a popis vybraných registrů pro práci s GPIO (Wx: zápis hodnoty x, Rx: čtení x)

zdroj hodinového signálu pro plánovač procesů, je možné při použití tohoto OS využívat pouze jeden z bloků. Blok TC má dva globální registry. V registru BCR[7] je možné zápisem do jediného přístupného bitu (SYNC) vyvolat softwarový trigger (vynulování čítače) pro všechny kanály současně. Pomocí registru BMR[8] je možné

[7]Block Control Register

[8]Block Mode Register



Obr. 2.1: Blokové schéma čítače časovače [1]

zvolit jaké signály budou připojeny na X0-X2 (viz. 2.1). Dále jen velmi stručný popis registrů ovlivňujících funkci jednotlivých kanálů. V registru CCR[9] je možné povolit/zakázat hodiny pro odpovídající kanál. Registr CMR[10] slouží k zvolení zdroje hodin (bity TCCLKS) a nastavení módu (bit WAVE a bity WAVSEL). Dále je možné nastavovat externí triggery, přerušení apod., ale protože tyto možnosti nejsou v práci využity, nejsou zde ani popsány. Pro zápis do jednotlivých registrů je nutné znát jejich adresu v paměti (viz. 2.2). Ta se pro registry společné pro všechny kanály jednoho bloku TC (BCR, BMR) skládá z bázové adresy (viz. B.13) odpovídajícího TC a offsetu daného registru. K adresám registrů jednotlivých kanálů se přičítá ještě offset tvořený vynásobením hodnoty 0x40 číslem daného kanálu (0-2). Každý z kanálů (jak je vidět v blokovém schématu 2.1) má tři externí hodinové vstupy (X0-2), pět interních hodinových vstupů (viz. 2.3) a dva vstupně/výstupní signály, jejichž funkci je možné nastavit (TIOAn, TIOBn). Kanály je možné zřetězit tak, že výstup jednoho je hodinovým vstupem jiného. Kanál je možné nastavit do jednoho

[9] Channel Control Register

[10] Channel Mode Register

offset	registr	jméno	popis
ch * 0x40 + 0x00	TC_CCR	Channel Control Register	
ch * 0x40 + 0x04	TC_CMR	Channel Mode Register	
ch * 0x40 + 0x10	TC_CV	Counter Value	obsah čítače
ch * 0x40 + 0x14	TC_RA	Register A	registr A
ch * 0x40 + 0x18	TC_RB	Register B	registr B
ch * 0x40 + 0x1C	TC_RC	Register C	registr C
ch * 0x40 + 0x20	TC_SR	Status Register	příznaky
0xC0	TC_BCR	Block Control Register	
0xC4	TC_BMR	Block Mode Register	

Tab. 2.2: Adresy a popis vybraných registrů pro práci s TC, kde ch je číslo kanálu (0-2)

ze dvou módů. V Capture Mode jsou piny TIOAx/TIOBx nastaveny jako vstupy a slouží pro měření frekvence, nebo počítání událostí. Druhou možností je Waveform Mode sloužící k generování frekvence, nebo PWM. V tomto režimu je TIOA nastaven vždy jako výstup a funkci TIOB je možné zvolit. Buď může sloužit jako druhý výstup, nebo takzvaný externí trigger - v tomto případě je TIOB nastaven jako vstup a náběžná hrana na něm zresetuje odpovídající čítač. Capture Mode není v práci využit a proto bude podrobněji popsán pouze Waveform Mode, který se aktivuje nastavením bitu WAVE z registru CMR na hodnotu jedna. V tomto módu jsou kanálem generovány jeden nebo dva signály se stejnou frekvencí a nezávisle nastavitelnou střídou. Rozlišení a střídu je možné nastavit pomocí komparačních registrů RA, RB a RC. Význam jednotlivých registrů určuje kombinace bitů WAVSEL. Shoda obsahu 16 bitového čítače a registru RA/RB znamená nastavení odpovídajícího výstupu. Význam registru RC se liší podle konfigurace - buď nemá žádnou funkci, nebo se pomocí něho nastavuje vrchol čítače. Čítač může v závislosti na nastavení počítat pouze nahoru, nebo nahoru a dolu. Pro správnou funkci je nutné přiřadit příslušné I/O piny k čítači časovači (Pehipheral Function) a povolit pomocí PMC hodiny pro odpovídající blok TC. Podrobný popis registrů a jejich funkce je možné najít v [1].

2.2 První varianta

Nerealizovaná varianta hardwaru spočívala v návrhu jedné desky obsahující mikroprocesor i všechny potřebné periferní obvody.

int. zdroj hodin	význam	pro MCK=90 MHz
TIMER_CLOCK1	MCK/2	45 MHz
TIMER_CLOCK2	MCK/8	11,25 MHz
TIMER_CLOCK3	MCK/32	2,8125 MHz
TIMER_CLOCK4	MCK/128	0,703125 MHz
TIMER_CLOCK5	SLCK	32,768 kHz (nezávislé na MCK)

Tab. 2.3: Interní zdroje hodinového signálu pro blok TC

2.2.1 Deska plošných spojů

Deska plošných spojů pro nerealizovanou variantu zařízení byla navržena jako čtyřvrstvá, o rozměru 140x140 mm, který je záměrně zvolen tak, aby deska mohla být umístěna do krabičky pod podstavu stojanu. Návrh DPS[11] je v rastru 0,3175 mm, nejtenčí spoje jsou šířky 0,25 mm a nejmenší průměr vrtání činí 0,4 mm. Pro usnadnění osazování či výměny součástek byl vytvořen servisní potisk.

Na horní straně desky (vrstva TOP) je umístěna většina součástek a jsou zde vedeny kritické signály (s nejvyšší frekvencí) - datová a adresová sběrnice pro připojení SRAM[12] k mikroprocesoru, sběrnice RMII[13] pro připojení obvodu KSZ8041 (Ethernet PHY[14]). Na horní straně (B.7) jsou taktéž umístěny všechny konektory - jako napájecí je použit konektor SPC21364 („power jack“). Pro připojení externích snímačů slouží tři konektory CANON 9. Na desce je dále MLW20 pro rozhraní JTAG[15], MLW20 pro ISI, slot pro SD kartu, konektor RJ-45, USB typ A, USB typ B. V horní i spodní (BOTTOM) vrstvě je rozlitá měď, spojená se zemí pro omezení EMI[16]. Vnitřní vrstvy slouží pro napájení a je v nich vedené jen nutné minimum signálových spojů. Jedna vnitřní vrstva je spojená se zemí a druhá s napětím 3,3 V. Na desce je možné připojením ampérmetru na piny z tabulky (2.4) měřit proud odebíraný jednotlivými napěťovými větvemi. Pro indikaci přítomnosti jednotlivých napájecích napětí slouží tři diody: LED1 pro 3,3 V, LED2 pro 5 V a LED3 pro 1,8 V. Diody LED4 a 5 jsou připojeny na pin procesoru. Propojka JP5 nastavuje, z jaké paměti bude procesor bootovat. Pokud je propojka rozpojená (na pinu BMS je log 1), procesor spustí bootloader v interní paměti ROM. Ten se pokusí najít v některé z externích pamětí spustitelný program (například bootloader druhé úrovně), zkopírovat ho do paměti

[11] deska plošných spojů

[12] Static random access memory

[13] Reduced Media Independent Interface - propojení Ethernet MAC a PHY

[14] PHYsical Interface - obvod realizující fyzickou vrstvu Ethernetu

[15] Joint Test Action Group - rozhraní pro testování a programování integrovaných obvodů

[16] elektromagnetické interference

měřící piny	proud
JP1	z napájecího zdroje
JP2	odebíraný větví 3,3 V
JP3	odebíraný větví 5 V
JP4	odebíraný větví 1,8 V

Tab. 2.4: Popis pinů pro měření proudu

SRAM a spustit. Při zkratované propojce (BMS=0) se pouze spustí program z externí paměti a ten musí procesor kompletně nakonfigurovat sám [1]. Pájecí propojky SJ1-3 slouží k přizpůsobení funkce rozhraní JTAG. Pomocí SJ5 je možné odpojit vodič CS od datové paměti dataflash (IC20). To se může hodit v případě, že v paměti je nefunkční program a potřebujeme dosáhnout spuštění SAM-BA monitoru pro nahrání nového obsahu paměti. Stejně tak je možné pomocí SJ4 odpojit paměťovou kartu SD (IC17). Propojka JP7 při zkratování zamče obsah paměti flash (IC20) proti přepsání. Na měřících bodech TP1-4 je možné měřit napětí úměrné proudu odebíranému přes jednotlivé H-můstky.

2.2.2 Schéma zapojení

Schéma zapojení vychází z výrobcem doporučovaného katalogového zapojení jednotlivých součástek. Zapojení mikroprocesoru a pamětí vychází z kitu AT91SAM9260-EK od firmy Atmel. Schéma je pro vyšší přehlednost rozděleno do šesti částí, které jsou dále popsány.

V první části schématu (B.1) je řešeno napájení desky. Napájecí napětí 12 V z externího stabilizovaného zdroje (konektor J1) je přes vratné pojistky vedeno k motorům a stabilizátorům napětí. Všechny použité stabilizátory jsou spínané, díky čemuž nedochází ke vzniku nadměrného množství tepla a nejsou tak třeba rozměrné chladiče. Obvod LM2673S (IC22) snižuje vstupní napětí na 3,3 V pro potřeby mikroprocesoru a dalších IO. Z tohoto napětí se pomocí TPS60502DGS (IC3) tvoří 1,8 V pro napájení jádra mikroprocesoru. Pro potřeby napájení periferií připojených k USB snižuje stabilizátor LT1767 (IC2) vstupní napětí 12 V na 5 V. Druhá část schématu (B.2) obsahuje mikroprocesor Atmel AT91SAM9260 v pouzdře PQFP208. Kromě procesoru jsou ve schématu také všechny potřebné blokovací kondenzátory různých kapacit, konektor CON1 sloužící k připojení JTAG adaptéru a dva krystaly.

K ovládání motorů stojanu slouží dva obvody L6206 (IC8, IC12), viz. schéma B.3. L6206 je dvojitý integrovaný H-můstek, blíže popsaný v 2.4. Proud odebíraný

motorky stojanu je měřen jako úbytek napětí na měřících rezistorech. Pro co nejmenší výkonovou ztrátu je vhodné použít rezistory o co nejmenším odporu a úbytek napětí na nich zesílit operačním zesilovačem. Použitý operační zesilovač TS912 je typu rail-to-rail a může být napájen unipolárním napětím. Napětí na rezistorech je odděleno sledovačem, filtrováno RC článkem a zesíleno neinvertujícím zesilovačem. Zesílení operačního zesilovače TS912 je nastaveno pomocí 1% rezistorů 1K a 68R na 15,71. RC-filtr má pomocí rezistoru 3K3 a kondenzátoru 100 nF nastavený mezní kmitočet 483 Hz. SMD rezistory velikosti 1206 použité jako bočníky mají omezenou mezní výkonovou ztrátu (obvykle 0.25 W) a proto je vždy pro měření proudu jedním motorkem použita paralelně zapojená dvojice, čímž dojde k omezení namáhání rezistorů a prodloužení jejich životnosti. Bočník je tedy tvořen dvojicí rezistorů 0R2, což při zesílení 15,71 a napájecím napětí 3,3 V operačního zesilovače umožňuje měřit proud až cca 2 A. Zesílená napětí ze snímacích rezistorů jsou přiváděna na vstupy 12 bitového AD převodníku ADC128S052 (IC10), který je pomocí rozhraní SPI[17] připojený k mikroprocesoru. Na další vstupy je přivedeno napětí z potenciometrů snímajících polohu stojanu a pomocí odporového děliče také vstupní napětí 12 V. Zdrojem přesného referenčního napětí 3,3 V pro AD převodník a napájení potenciometrů je ADR366 (IC6). Napěťová reference je napájena ze zdroje 5 V přes LC filtr pro omezení rušení.

Na schématu (B.4) jsou konektory pro USB. X4 je typu host a slouží k připojení flash disku, USB kamery apod. Napájení zařízení připojených na tento port je zajišťováno zdrojem 5 V a jištěno pomocí vratné pojistky. X5 slouží pro připojení desky k PC. Napětí na napájecím pinu tohoto konektoru je přes odporový dělič přivedeno na GPIO pin procesoru a slouží k detekci připojení k PC. Datové vodiče obou USB jsou obvodem PRTR5V0U2X chráněny proti ESD. Schéma dále obsahuje konektor pro připojení kamery s paralelním 8 bitovým rozhraním na port ISI. Na konektor je kromě ISI přivedeno také rozhraní I^2C (například pro nastavování parametrů kamery) a napětí 3,3 V. Dále je zde pro případné budoucí využití sériový port (pouze piny TXD, RXD) sestávající z konektoru CANON 9 a převodníku úrovní MAX3221 (IC15) a dvě signalizační LED diody. K procesoru jsou připojeny dvě SRAM paměti MT48LC64M4A2 (IC18, IC19) s kapacitou 2x64 Mb, tedy o celkové kapacitě 16 MB (schéma B.5). Protože se jedná o obvody pracující s vysokou frekvencí, je jejich napájení blokováno množstvím keramických kondenzátorů s hodnotami 1 nF, 10 nF a 100 nF. Připojení pamětí na datovou a adresovou sběrnici je převzato ze zapojení referenčního kitu. Přes SPI je k procesoru připojena flash paměť AT45DB642D (IC20) o kapacitě 64 Mb (8 MB) a slot pro SD kartu (IC17). Flash paměť sloužící k uložení operačního systému a programů je možné pomocí přerušení pájecí propojky

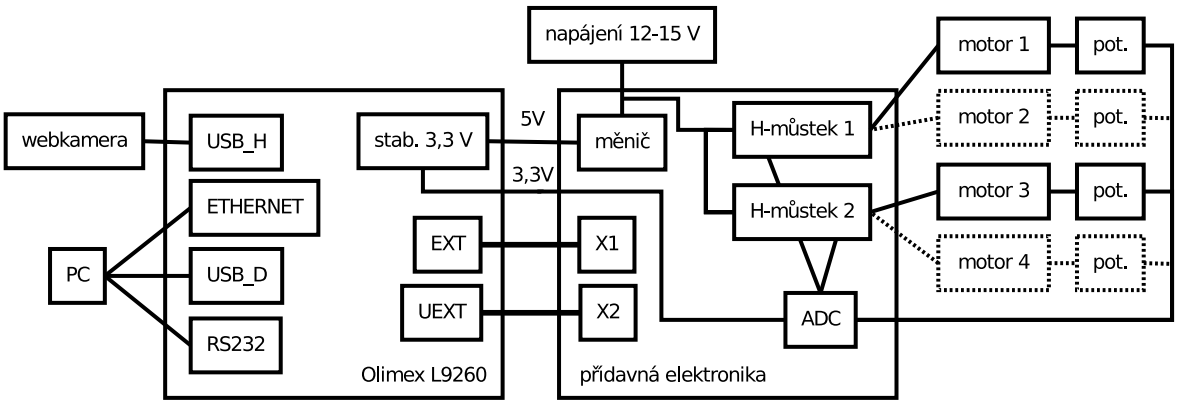
[17]Serial Peripheral Interface

v případě potřeby vyřadit z činnosti. Stejně tak SD kartu. U flash paměti je navíc možné spojením jumperu JP6 zapnout ochranu proti přepisu.

Mikroprocesor AT91SAM9260 má zabudovanou podporu pro Ethernet, ale potřebuje externí obvod zajišťující komunikaci na fyzické vrstvě Ethernetu, tzv. PHY. Tuto funkci na desce vykonává KSZ8041 (IC21) - viz. schéma (schéma B.6). K mikroprocesoru je připojený pomocí rozhraní RMII. Obvod je napájen napětí 3,3 V a 1,8 V a dále pomocí napětí získaných z těchto dvou pomocí filtrů, zapojených dle katalogového listu. Diferenciální páry RX+, RX- a TX+, TX- jsou vedeny do konektoru SI-60062-F, což je RJ-45 konektor s integrovaným vysokofrekvenčním oddělovacích transformátorem. 50 MHz krystalový oscilátor XO91050UITA (U1) generuje přesné referenční hodiny pro rozhraní RMII.

2.3 Druhá varianta

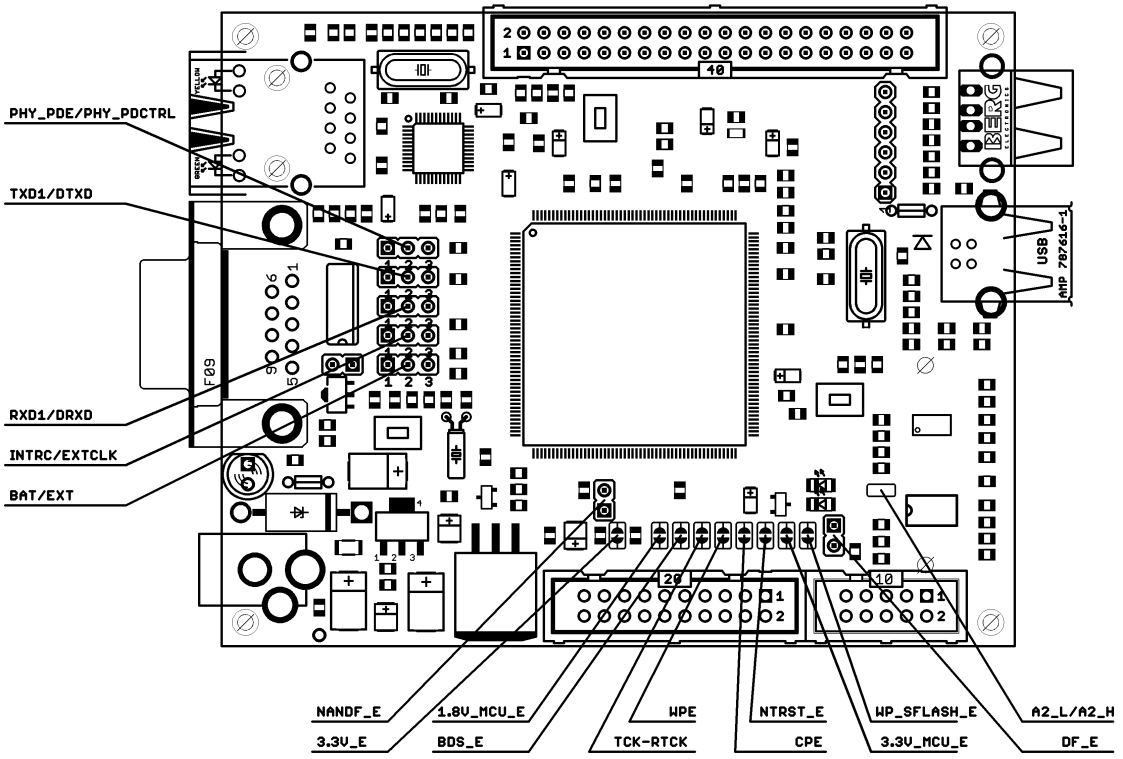
Realizovaná varianta hardwaru spočívá ve využití vývojového kitu Olimex SAM9-L9260 a k němu připojené desky s výkonovými budiči a další nezbytnou elektronikou. Blokové schéma této varianty ukazuje obrázek 2.2, kde je znázorněno, že k desce s přídavnou elektronikou mohou být připojeny až čtyři stejnosměrné motory a u každého je možné regulovat polohu, měřit proud a snímat koncové spínače (to v schématu z důvodu zachování přehlednosti není). Další možností je připojení dvou krokových motorů, které také mohou být vhodné pro použití u polohovatelného stojanu. Vývojový kit je k PC možné připojit přes USB port (v práci využito pouze k programování pamětí), RS-232 port (např. sledování postupu bootování), přes Ethernet (přihlášení přes SSH atd) a v případě nutnosti přes JTAG (což není běžně potřeba a proto není tato možnost ani v schématu).



Obr. 2.2: Blokové schéma druhé verze elektroniky

2.3.1 Olimex SAM9-L9260

Kit je osazen mikroprocesorem Atmel AT91SAM9260, 64MB SDRAM, 2 MB dataflash, 512MB NAND Flash a konektorem pro paměťové karty typu SD/MMC. Deska dále obsahuje Ethernet PHY (Micrel KSZ8721), dva USB porty (host, device), 40 pinový rozšiřovací konektor s vyvedenými nevyužitými piny procesoru (označený jako EXT), 10 pinový konektor s rozhraním SPI a I^2C (UEXT) a standardní 20 pinový JTAG pro ladění a programování. Paměť dataflash připojená přes SPI slouží k uložení bootloaderu a linuxového jádra. Do paměti NAND se potom ukládá souborový systém, uživatelské aplikace a data. Na vývojovém kitu je od výrobce předinstalován OS Linux vycházející z distribuce Debian 5.0 [2]. Předinstalovaný OS však díky své komplexnosti velmi dlouho startuje a proto byl v rámci práce nahrazen vlastním jednodušším systémem vytvořeným pomocí balíku Buildroot. K desce výrobce dodává sadu skriptů pro programování paměti, které byly využity i pro naprogramování vlastního systému. Využity byly taktéž patche pro jádro Linuxu, které do něj doplňují podporu pro tuto desku. Postup pro přepsání dataflash a NAND



Obr. 2.3: Konfigurační propojky vývojového kitu [2]

flash je popsán v [2], kapitola „Restoring the default bootloader and kernel.“ Na obrázku 2.3 s vývojovým kitem jsou označeny konfigurační propojky - pro aktivaci SAM-BA monitoru je nutné před připojením desky k napájecímu napětí rozpojit

propojky NANDF_E a DF_E, čímž dojde k odpojení výběrových (CS) vodičů a tyto paměti se stanou pro mikroprocesor „neviditelné“.

2.3.2 Deska pro řízení motorků

Deska o rozměrech 110x70 mm je navržena jako oboustranná s prokovy (schéma viz. B.9 a B.10). K propojení s kitem slouží dva ploché kabely (40 a 10 žil). Napájecí napětí 12-15 V se připojuje na konektor J1. Toto napětí je přes polovodičovou vratnou pojistku připojeno k budičům motorků. Druhá větev odvozená z napájecího napětí, opět jištěná polovodičovou pojistkou, slouží po snížení napětí spínaným měničem LM2596 na 5 V k napájení vývojového kitu. Protože spínání motorků může vyvolávat krátkodobé poklesy napájecího napětí, je větev se snižujícím stabilizátorem napájející kit oddělena schottkyho diodou s nízkým úbytkem napětí, která zabrání poklesu napětí na filtračních kondenzátorech. Ty jsou na vstupu měniče zařazeny dva paralelně, pro snížení ESR[18] a zvýšení proudové zatížitelnosti. Před přepětím, případně krátkodobými špičkami chrání měnič TVS[19] dioda D3. Přítomnost napětí 5 V na výstupu stabilizátoru signalizuje svítivá dioda LED1.

Způsob ovládání motorků a měření jejich proudu byl převzat z první varianty řešení a je podrobněji popsán v příslušné části práce (2.2.2), proto bude zapojení popsáno jen stručně. Napětí 3,3 V pro napájení AD převodníku ADC128S052 (IC5) je odebíráno z vývojového kitu. Napěťová reference ADR366 (IC2) je přes LC filtr tvořený součástkami L3, C13 a C14 napájena napětím 5 V. Její výstup je použit jako referenční napětí AD převodníku, k napájení operačních zesilovačů a potenciometrů snímajících polohu. Analogová a digitální zem je oddělena feritovou perličkou L2. Pro připojení motorků jsou použity tří pinové svorky ARK500/3, kde je na třetí svorku vyvedena zem, takže je možné použít stíněný napájecí vodič. Změna oproti první variantě spočívá v použití bočníků s větší hodnotou odporu, pro lepší využití dynamického rozsahu AD převodníku. Použita je paralelně zapojená dvojice rezistorů s hodnotou odporu 1R, čili výsledný bočník má hodnotu 0R5. Použitý bočník umožňuje měřit proud maximálně cca 400 mA, což se však ukázalo jako dostatečné. Další změnou je zvětšení velikosti kapacity v RC článku na 1 μF tak, aby měl RC článek mezní frekvenci cca 50 Hz. Změnou provedenou až dodatečně, na hotové desce, bylo přemostění prvního operačního zesilovače (ve funkci sledovače), kdy se ukázalo, že jeho minimální výstupní napětí cca 30 mV (navíc dále zesílené druhým OZ) je příliš velké. Dále se jako nevhodné ukázalo připojení hardwarově generované PWM na vstupy IN použitých H-můstku. Byl proto vyroben plochý kabel a provedeny úpravy na desce měnící propojení pinů konektorů na kitu a pinů konektorů na desce

[18]Equivalent series resistance

[19]Transient voltage suppressor

pro řízení motorků. Změny oproti schématu popisuje tabulka B.2. Dále byly na desce přemostěny rezistory na vstupech AD převodníku měřících výstupní napětí potenciometrů. Mezi vstupy a zem byl přidán keramický kondenzátor o hodnotě 100 nF. Ukázalo se totiž, že díky velkému odporu potenciometrů neměří AD převodník přesně. Lepších výsledků by mohlo být dosaženo doplněním napěťového sledovače.

2.4 Obvody použité v obou variantách

V obou variantách hardwaru je pro měření proudu a polohy použit AD převodník ADC128S052. Jedná se o osmikanálový 12bitový převodník s postupnou aproximací, komunikující po sběrnici SPI. Pro snížení vlivu rušení z digitálních obvodů má oddělené napájení (2,7-5,25 V) pro analogovou a digitální část. Převodník nepotřebuje krystal, protože převod je řízen hodinami rozhraní SPI. Tím je také dáno to, že hodiny nemohou mít libovolnou frekvenci. Výrobce pro optimální výkon doporučuje rozsah 3,2-8 MHz, hraniční hodnoty jsou však 0,8-16 MHz. Klidová logická úroveň na vodiči SCLK je vysoká, data jsou na výstup přiváděna při sestupné hraně, na vstupu jsou vzorkována při náběžné hraně. Toto chování se obvykle označuje jako SPI MODE 3. Komunikace s převodníkem je velmi jednoduchá. Přenos je vždy obousměrný, kde 16bitové slovo vyslané převodníku představuje číslo kanálu pro příští převod posunuté doleva o jedenáct bitů a 16 bitové přijaté slovo je výsledek převodu pro aktuálně zvolený kanál.

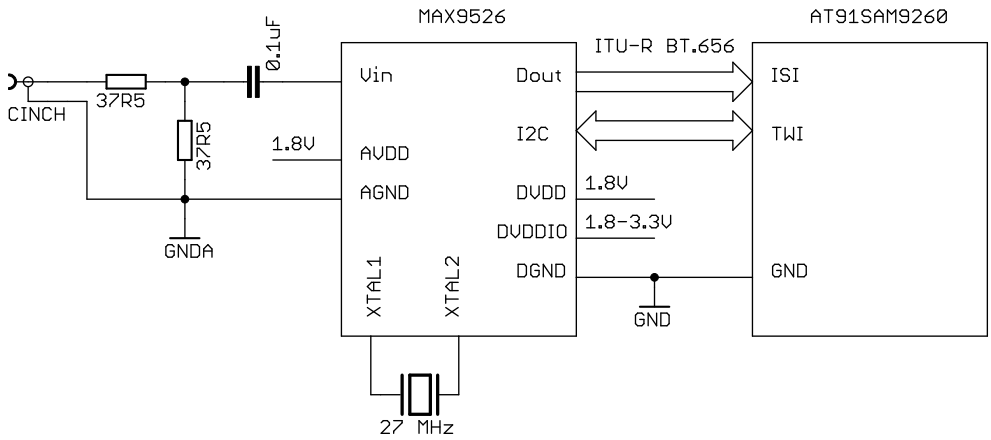
Jako budič motorků je použit integrovaný obvod L6206, což je dvojitý H-most tvořený tranzistory DMOS, doplněný o ochranné obvody (vypnutí při přehřátí, ochrana proti zkratu, diagnostický výstup). Jeden obvod může sloužit jako budič dvou stejnosměrných motorků, nebo je možné propojit odpovídající vstupy a výstupy obou kanálů pro proudové posílení. Podle datasheetu je odpor v sepnutém stavu 0,3 Ω, díky čemuž se obvod obejde bez chladiče. Proud, který může být trvale dodáván jedním kanálem je 2,8 A. Špičkově je možné odebrat až 5,6 A. Frekvence PWM může dosahovat 100 kHz. Výhodou obvodu je, že obsahuje integrované ochranné diody, které tak nezabírají místo na plošném spoji. Proud připojenými motory je možné měřit nepřímo jako úbytek napětí na rezistorech připojených mezi výstup SENSE a zem. Obvodu lze snadno pomocí externích rezistorů omezit maximální proud, podle potřeby aplikace. Diagnostické výstupy OCD typu otevřený kolektor jsou uzemněny pokud dojde k přehřátí obvodu nebo výskytu zkratu.

2.5 Připojení kamery s kompozitním výstupem

Na trhu je velké množství levných kamer s kompozitním výstupem v normě PAL, proto by bylo výhodné, kdyby takovou kameru bylo možné k zařízení připojit. Kompozitní signál (anglická zkratka CVBS[20] [7]) je co se týče kvality nejhorší z běžně používaných analogových způsobů přenosu obrazu [8]. Jeho výhodou je však přenos barevného obrazu po jednom signálovém vodiči. Jde vlastně o televizní signál, před přidáním zvukové stopy a VF modulací. V signálu jsou zakomponovány tři složky barevného modelu YUV. Složka Y představuje jas, může být zobrazena samostatně jako černobílý obraz a zároveň obsahuje synchronizační pulzy. Barvy jsou reprezentovány složkami U (odstín) a V (sytost), které jsou pro přenos namodulovány na dvě ortogonální fáze barvonosného signálu o kmitočtu 4,43361875 MHz daném normou PAL [6] a dohromady tvoří tzv. chrominanci. Kombinací chrominance a jasového signálu vznikne barevný kompozitní signál.

Pro převod analogového kompozitního video signálu na digitální signál standardu ITU- R BT.656, který je kompatibilní s rozhraním ISI mikroprocesoru AT91SAM9260, může být využit například integrovaný obvod MAX9526 od výrobce Maxim. Jedná se o videodekodér umožňující převést obraz v normě PAL/NTSC na digitální 8, nebo 10 bitový komponentní obraz, přičemž detekce použité normy probíhá automaticky. Obvod se spotřebou 200 mW disponuje dvěma vstupy, obsahuje interní 10 bitový AD převodník s vzorkovací frekvencí 54 MHz (pro čtyřnásobné převzorkování), provádí automatické nastavení zesílení tak, aby byl maximálně využit rozsah převodníku, a také obsahuje integrovaný analogový antialiasingový filtr. Pro konfiguraci obvodu slouží rozhraní I^2C a 18 registrů [8].

[20] Composite Video, Blanking, and Sync



Obr. 2.4: Blokové schéma zapojení obvodu MAX9526

3 LINUX A EMBEDDED ZAŘÍZENÍ

Linux je jádro operačního systému, jehož první verzi naprogramoval v roce 1991 Fin Linus Torvalds. Díky uvolnění zdrojového kódu pod licencí GPL jej mohli začít vylepšovat další programátoři. Jádro je skoro celé napsáno v jazyce C, jen části jsou v jazyce symbolických adres. Původně byl Linux napsán pro architekturu i386, ale časem byl portován na mnoho dalších architektur, jako například ARM, AVR32, Alpha, PowerPC, SPARC a mnohé další. Dnes se běžně jako Linux označují operační systémy tvořené jádrem, knihovnami, pomocnými nástroji a aplikačním softwarem, zatímco dříve se pojmem Linux označovalo pouze samotné jádro. Linuxové jádro je koncipováno jako monolitické (s možností načítání modulů), podporou preemptivního multitaskingu, síťové komunikace, správy paměti, virtuální paměti atd. Zdrojový kód jádra verze 2.6.33.7 použité v této práci sestává z 34314 souborů zabírajících 515 MB. Kód vytvářejí dobrovolníci z celého světa, ale velkou měrou přispívají také světově významné IT firmy [9].

Kromě stále většího využití u desktopových počítačů a serverů se Linux stává i často používaným řešením pro embedded zařízení. Díky své licenci je pro výrobce takových zařízení velmi výhodný - je zdarma, není nutné platit žádné licenční poplatky, je portován na mnoho architektur, obsahuje ovladače pro nejrůznější zařízení a vývojové nástroje a utility jsou také zdarma.

Hardware embedded zařízení se často velmi liší od hardwaru běžně používaných PC. Architektura bývá odlišná od x86, paměť je často typu flash a její velikost z cenových důvodů omezená, výpočetní výkon bývá zpravidla menší. Na druhou stranu embedded zařízení oproti PC disponují řadou různých sběrnic a specifických periferií, které je třeba obsluhovat. U embedded zařízení je také častý požadavek na RT[1] odezvu operačního systému pro účely řízení nebo komunikace.

3.1 U-Boot

Pro zavedení Linuxu (nebo jiného OS), případně stažení obrazu do prázdné paměti flash se velmi často využívá bootloader druhé úrovně, například U-Boot (licence GPL). Základní inicializaci provede bootloader v ROM mikroprocesoru a následně předá řízení bootloaderu druhé úrovně, U-Bootu. Ten provede inicializaci složitějších periferií mikroprocesoru a díky tomu je například možné stáhnout do paměti program pomocí síťového rozhraní, nebo přímo nabootovat ze sítě. U-Boot je do mikroprocesoru možné nahrát například přes SAM-BA, ve vývojovém kitu L9260 je nainstalován a nastaven (použité typy pamětí, jejich velikost, připojení, MAC adresa síťového rozhraní atd.)

[1] real-time

od výrobce. Komunikace s uživatelem probíhá obvykle po sériovém rozhraní, u kitu L9260 je to port DBGU s rychlostí 115200, 8 bity a jedním stop bitem. Při spouštění systému vypíše U-Boot informace o své konfiguraci a nabídne možnost vstoupit do interaktivního režimu. V něm je možné měnit konfiguraci a lze do něj vstoupit několik vteřin (prodleva je konfigurovatelná) po zapnutí kitu posláním libovolného znaku. Příkazem *help* je možné získat přehled o všech dostupných příkazech, *help xyz* vypíše podrobnější nápovědu k příkazu *xyz*. U-Boot podporuje bootování ze sítě, takže je možné příkazem *tftpboot* načíst linuxové jádro z TFTP serveru. V práci tato možnost však nebyla použita, protože změny jádra nejsou příliš časté, a jeho načítání probíhalo z paměti dataflash. Ukázka výpisu při spuštění U-Bootu:

```
U-Boot 2009.06-00374-g3427faf (Jul 16 2009 - 16:56:00)

DRAM:  64 MB
NAND:  512 MiB
DataFlash:AT45DB161
Nb pages:   4096
Page Size:    528
Size= 2162688 bytes
Logical address: 0xD0000000
Area 0:\0x09D0000000 to D00041FF (RO) Bootstrap
Area 1:\0x09D0004200 to D00083FF      Environment
Area 2:\0x09D0008400 to D0041FFF (RO) U-Boot
Area 3:\0x09D0042000 to D0251FFF      Kernel
Area 4:\0x09D0252000 to D020FFFF      FS
In:    serial
Out:   serial
Err:   serial
Net:   macb0
macb0: Starting autonegotiation...
macb0: Autonegotiation complete
macb0: link up, 100Mbps full-duplex (lpa: 0x45e1)
Hit any key to stop autoboot: 3 2 1 0
```

Důležitým příkazem je *setenv*, který umožňuje nastavit různé proměnné, jejichž přehled vypíše *printenv*:

```
ethaddr=3a:1f:34:08:54:54
bootdelay=3
baudrate=115200
```

```
#
bootcmd=cp.b 0xD0040000 0x22200000 0x0017BF74; bootm 0x22200000
ethact=macb0
bootargs=...
stdin=serial
stdout=serial
stderr=serial

Environment size: 582/16892 bytes
```

Z těchto proměnných, které jsou předdefinovány výrobcem kitu je nejčastěji měněna proměnná `bootargs`. Její hodnota je při spuštění jádra předána jako tzv. kernel command line, kde musí být specifikována velikost paměti RAM, standardní výstup a umístění kořenového filesystému (rootfs). Nastavení se provede příkazem *setenv bootargs='xyz'*. Při vývoji byl používán rootfs připojený přes síť pomocí NFS[2] s následující hodnotou `bootargs`:

```
mem=64M console=ttyS0,115200 rootfstype=nfs root=/dev/nfs rw
nfsroot=192.168.1.37:/home/zdenal/nfsroot,v3,proto=tcp ip=dhcp
init=/linuxrc printk.time=1 lpj=447488 reboot=soft uvcvideo.quirks=128
```

V proměnné je specifikována IP adresa serveru, umístění sdíleného adresáře, verze protokolu (v3) a umístění rodiče všech procesů *init*. Parametr `printk.time=1` zapíná u jednotlivých řádků vypisovaných při bootu Linuxu časovou značku, parametr `lpj=447488` nastavuje na pevnou hodnotu LPJ[3], která se normálně zjišťuje při startu, což zabere cca 250 ms. Parametr `reboot=soft` urychluje reboot systému a `uvcvideo.quirks=128` je aktivace „špinavého“ režimu ovladače webkamery.

Kořenový filesystém je možné připojit také z flash disku, potom by proměnná `bootargs` vypadalo takto:

```
mem=64M console=ttyS0,115200 root=/dev/sda1 rootdelay=10
```

V neposlední řadě je možné připojit rootfs také z NAND paměti osazené na kitu:

```
mem=64M console=ttyS0,115200 root=/dev/mtdblock1 rw
rootfstype=jffs2 init=/linuxrc
```

Nastavení provedené příkazem *setenv* se použije pouze pro následující start systému. Pokud požadujeme trvalé uložení v dataflash, je nutné použít příkaz *saveenv*. Po provedení všech požadovaných nastavení se zavoláním příkazu *boot* vyvolá bootování systému.

[2]Network File System - protokol pro sdílení souborů přes síť

[3]loops per jiffy

3.2 Buildroot

Builroot je balíček skriptů zásadně usnadňující vývoj embedded systému. Slouží k automatizovanému vytvoření souborového systému, konfiguraci a kompilaci jádra, vývojových nástrojů, bootloaderu a knihoven (například úsporná knihovna jazyka C[4] uClibc - „the microcontroller C library“) - viz. dále. Buildroot také vytvoří prostředí pro křížovou kompilaci programů (cross-compilation toolchain) pro cílovou platformu na PC, což může velmi usnadnit a urychlit vývoj díky vyšší rychlosti a pohodlí práce. Podporována je řada běžných vývojových desek (kitů) a je možné snadno doplnit podporu pro vlastní desku. Pro konfiguraci slouží textové menu (vyvolané příkazem *make menuconfig*), kde je možné nastavit vlastnosti jádra, uClibc, BusyBoxu (viz. dále), aplikovat potřebné patche atd. Buildroot obsahuje kromě základních utilit nutných pro běh systému velké množství programů nejrůznějšího účelu (od přehrávání multimédií po různé knihovny, textový editor, nebo hru šachy), které mohou být snadno zkompilovány pro cílovou platformu, případně mohou být jednoduchým způsobem přidány další. Výsledkem činnosti Buildrootu jsou obrazy (image) paměti s jádrem a rootfs pro cílovou platformu [3].

Knihovna uClibc je úspornější verzí Glibc běžně používané na linuxových systémech. Oproti ní je méně univerzální, cílí na embedded platformy a může být používána i na mikroprocesorech bez podpory správy paměti (bez jednotky MMU, tedy např. mikroprocesory z rodiny ARM7TDMI) . Výhodou je zejména významně menší velikost, která může být v embedded zařízeních kritickou veličinou. Velikost a „schopnosti“ uClibc je možné detailně ovlivnit konfigurací a obvykle se pohybuje v řádu stovek kB, přičemž velikost Glibc bývá v řádu desítek MB. Nevýhodou oproti Glibc je, že některé funkce nejsou implementovány [5], nebo pracují jinak.

Důležitou součástí systému vytvořeného pomocí Buildrootu je Busybox. Linuxový systém pro svůj běh potřebuje řadu utilit (GNU Core Utilities), které jsou tradičně samostatnými binárními soubory. Busybox supluje funkčnost až dvou stovek utilit a je tvořen jediným binárním souborem s cílem uspořit paměť. Eliminací režie mnoha samostatných souborů, pokročilých funkcí některých utilit a možností sdílení kódu bez vytváření knihoven dosahuje významné úspory místa. V systému používajícím Busybox jsou pro zachování kompatibility vytvořeny symbolické odkazy pro jednotlivé příkazy (ls, cp, mv atd.) vedoucí na `/bin/busybox`. Podle [12] je však Busybox oproti tradičním GNU Core Utilities výrazně pomalejší [6].

[4]Knihoven jazyka C vytvořených speciálně pro embedded zařízení existuje více. Přehledné srovnání jejich vlastností a výkonu lze najít na www.etalabs.net/compare_libcs.html

[5]Např. funkce `clock_nanosleep()` byla v uClibc implementována až v průběhu vytváření této práce.

[6]Test však byl prováděn na verzích 0.50.3 a 0.60.4, přičemž aktuální verze je 1.18.4.

V konfiguračním menu Buildroot je možné (mimo jiné) nastavit:

- pro jakou architekturu bude systém vytvořen (ARM, AVR32, X86, MIPS atd.),
- varianta této architektury (v případě této práce ARM926T),
- použité ABI[7], [8]
- možnosti kompilace: úroveň optimalizací, (ne)použití debuggovacích symbolů atd.,
- zdroj toolchainu: zkompilovat vlastní, externí (dodaný např. výrobce vývojového kitu), nebo toolchain vytvořený v rámci projektu Crosstool-NG,
- verzi linuxového jádra a hlavičkových souborů, uClibc, kompilátoru GCC, Binutils (nástroje pro manipulaci s objektovým kódem),
- požadované vlastnosti toolchainu (podpora C++, Fortranu, IPV6...),
- které aplikace se mají zkompilovat (stovky různých programů),
- jaký souborový systém se má použít pro generovaný obraz systému,
- používaný bootloader (Barebox, U-Boot, AT91 Bootstrap, AT91 DataFlashBoot),
- jestli se má zkompilovat také jádro a jaké patche se mají použít.

Kromě popsaných možností konfigurace dostupných příkazem *make menuconfig* je možné detailně nastavit možnosti uClibc (příkaz *make uclibc-menuconfig*), Busyboxu (*make busybox-menuconfig*) a samozřejmě také jádra (*make linux26-menuconfig*). Všechny konfigurační volby se ukládají do souborů .config, které je možné snadno zálohovat.

Po provedení požadovaných nastavení Buildrootu se příkazem *make* vyvolá kompilace všech vybraných komponent, což zahrnuje:

- stažení zdrojových kódů,
- vytvoření prostředí pro křížovou kompilaci,
- kompilaci vybraných balíčků,
- vytvoření image jádra,
- vytvoření image bootloaderu,
- vytvoření image rootfs ve zvolených formátech.

Po kompilaci je veškerý výstup uložen v adresáři `output`, který obsahuje několik podadresářů. V `images` jsou uloženy obrazy jednotlivých paměti (jádro, bootloader,

[7] Application binary interface - rozhraní pro komunikaci mezi aplikačním programem a OS

[8] Původní systém, předinstalovaný na vývojovém kitu, používal starší variantu ABI (OABI), která je oproti nyní použitému EABI výrazně pomalejší při výpočtech s plovoucí řádovou čárkou [4]. Mikroprocesor AT91SAM9260 nemá jednotku pro výpočty v plovoucí řádové čárce a proto je nutné její instrukce emulovat. Při použití OABI dojde k vyvolání výjimky neexistující instrukce a patřičná instrukce je emulována v jádře systému (tzv. hardfloat), což je pomalé, protože musí dojít k přepnutí kontextu, což sebou nese režii. S EABI je možné pro výpočty s plovoucí řádovou čárkou použít knihovny z uživatelského prostoru (tzv. softfloat). Krom toho EABI nabízí několik dalších vylepšení [5]

rootfs). Obsahem `build` jsou všechny zkompilované balíčky, každý v odpovídajícím podadresáři. Pokud je třeba nějaký balíček překompilovat, je nutné smazat jeho adresář. V adresáři `build/xyz` jsou také ukládány konfigurační soubory jádra, Busyboxu a uClibc. Protože při použití *make clean* (smazání všech výstupů) je vše v `build` smazáno, je užitečné si příslušné konfigurační soubory zálohovat. Adresář `build/staging` obsahuje téměř kompletní filesystém. Jsou zde všechny binární soubory, knihovny a hlavičkové soubory (`/usr/share/include`), které mohou být includovány při křížové kompilaci. Obdobou `staging` je adresář `target`, který také obsahuje kompletní filesystém, tentokrát bez vývojových souborů. Nemůže však být přímo použit jako rootfs, protože neobsahuje důležitý adresář se soubory zařízení `/dev` (problém s právy při vytváření). Pokud je tedy potřeba mít kompletní rootfs (například pro připojení přes NFS, je nutné nastavit Builroot tak, aby vytvořil zkomprimovaný balík `roofs.tar`, který obsahuje i adresář `/dev` a je uložen v `images`, odkud ho lze s právy uživatele root rozbalit.

Dalším důležitým podadresářem `output` je `host`, kde je vytvořeno prostředí pro křížovou kompilaci. V `host/usr/bin` najdeme kompilátor (*arm-linux-gcc*), linker atd. Pro křížovou kompilaci je nutné provést následující nastavení prostředí spuštěním příkazů:

```
export ARCH=arm
export CROSS_COMPILE=arm-linux-
export PATH=$PATH:~/buildroot/output/host/usr/bin
```

Poté je možné na hostitelském PC kompilovat zdrojové kódy pro cílovou platformu následujícím způsobem:

```
arm-linux-gcc -O3 -Wall -I~/buildroot/output/staging/usr/include test.c
```

Pokud jsou požadovány změny v obrazu rootfs (změny konfiguračních souborů, přidání vlastních dat nebo programů), je nejjednodušší cestou jak toto zajistit, provést je v adresáři `output/target`. Možné jsou však i další způsoby detailně popsané v [3].

3.3 Real Time

U operačních systémů reálného času, které jsou používány například pro řízení, jsou kladeny především nároky na ukončení výpočtu v časovém limitu - deadline. Překročení tohoto limitu se bere jako chyba. Standardní Linux není navržen jako operační systém reálného času, protože se snaží dosáhnout maximální propustnosti v multi-uživatelském prostředí. Problémy s latencí způsobuje nedeterministické

chování některých služeb jádra (např. alokace paměti), spinlocky (aktivní čekání), vykonávání obsluh přerušení, inverze priorit atd.

Existují však různé možnosti, jak dosáhnout nízkých latencí potřebných pro řízení v reálném čase. Na trhu je několik firem nabízejících své modifikace Linuxu včetně podpory pro vývojáře. Z volně dostupných řešení je to například hard RT OS Xenomai. V něm běží Linux jako jedno z vláken, které je spuštěno, pokud o CPU nežádá žádné z vláken s RT prioritou. Nevýhodou je, že RT procesy nejsou normálními linuxovými procesy, je nutné používat speciální API.

Jádro standardního Linuxu je postupně vylepšováno tak, aby se zlepšila jeho odezva. Od verze 2.6 jsou dostupná některá významná vylepšení: O(1) plánovač (naplánování běhu procesů vyžaduje konstantní čas nezávisle na jejich počtu), volitelně může být vypnuta podpora virtuální paměti (adresní prostor je sdílen všemi procesy, urychlí se přepnutí kontextu) a bylo plně implementováno POSIX RT API, což je standardní Unixové API pro psaní RT aplikací.

Dále přibyla možnost zapnout preemptivitu jaderného kódu a to dokonce hned ve dvou variantách. Vypnutá preemptivita jaderného kódu (volba CONFIG_PREEMPT_NONE) znamená, že běh jaderného kódu nebude nikdy přerušen. Tato možnost přináší vysoký průměrný výkon a je vhodná zejména pro intenzivní výpočty. Volba CONFIG_PREEMPT_VOLUNTARY umožňuje přerušení běhu jaderného kódu v určitých definovaných bodech, je vhodná zejména pro desktopy (rychlejší reakce na vstup od uživatele) a nepřináší významné zhoršení celkové propustnosti. CONFIG_PREEMPT umožňuje přerušit běh většiny jaderného kódu, kromě výjimek jako jsou kritické sekce implementované pomocí spinlocků. Tato volba je vhodná pro embedded systémy a desktopy s požadavky na maximální latenci v řádu milisekund.

3.3.1 Path CONFIG_PREEMPT_RT

Jedná se o path vytvořený malou skupinou jaderných vývojářů vedených Ingo Molnarem. Path dostupný pro nejběžnější embedded platformy mění jádro Linuxu tak, že většina jeho kódu může být přerušena plánovačem. Díky tomu se z Linuxu stává RTOS. Výhodou je, že není potřeba žádné speciální API ani druhé RT jádro. Změny se dotýkají pouze jádra, uživatelské aplikace není třeba nijak měnit, ani znovu kompilovat. Pro spuštění RT aplikace nejsou potřeba superuživatelská práva. Pokud ale má aplikace využívat zamykání paměti a RT priority, může být nezbytné upravit soubor */etc /security/limits.conf*. RT vlastností je podle [13] dosaženo následujícími úpravami:

- aktivní čekání v kódu jádra (spinlocky) jsou nahrazeny rtmutexy,
- implementace „priority inheritance“ v jádře, [9]

[9] Týká se však pouze jádra. Implementace priority inheritance v uživatelském prostoru záleží

- obsluha přerušení ve vláknech (která mohou být přerušena),
- přidání časovačů s vysokým rozlišením (ns).

Na stránce [13] je také vyjmenován seznam nejdůležitějších bodů, které je třeba dodržet při vytváření RT aplikace:

- co nejdříve ve funkci *main* zavolat *mlockall*, která zajistí, že virtuální paměť volajícího procesu bude uzamčena v RAM,
- vytvořit všechna vlákna hned po spuštění aplikace,
- vyhnout se volání funkcí generujících výpadky stránek (*fopen* apod.),
- vyhnout se dynamické alokaci paměti.

3.3.2 Fast context switch extension

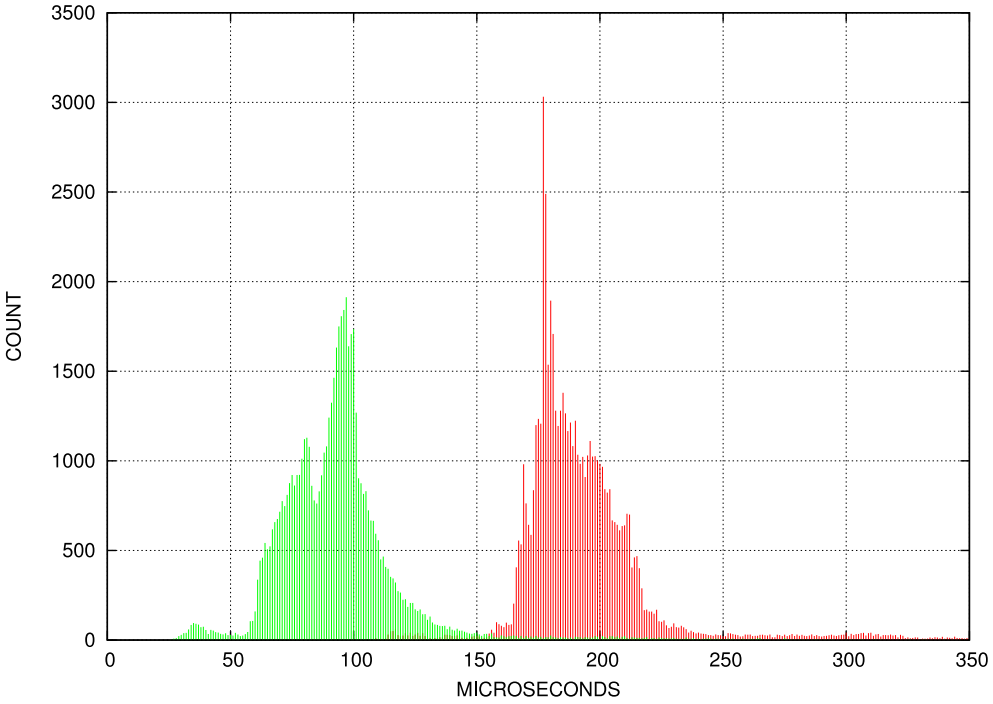
Další možností jak dosáhnout lepší RT odezvy na platformě ARM a Linuxu je využití rozšíření FCSE[10], které nabízí některé mikroprocesory. V standardním Linuxu není podpora pro tuto periférii dosud obsažena, takže je nutné použít patch, jehož funkce je popsána v [14].

Mikroprocesory ARM obsahují dva druhy cache paměti - jedna pro instrukce, druhá pro data. Když chce CPU provést instrukci `load` nebo `store`, požádá jednotku správy paměti (MMU) o danou virtuální adresu. Pokud jsou data uložená v cache, jednotka vrátí CPU přímo požadovaná data. V multitaskingovém OS s ochranou paměti má každý proces svůj prostor virtuálních adres. Z toho důvodu musí být před přepnutím kontextu obsah cache pamětí zneplatněn, aby se zajistilo, že proces nebude moct přistupovat k datům náležejícím jinému procesu. Pokaždé když je proces naplánován k běhu, musí být načtena jeho data, což je (oproti čtení z cache) časově náročné. Podle [14] může být cena této operace v závislosti na konkrétních podmínkách až 200 μs. Jednou z možností jak snížit cenu přepnutí kontextu je použít tzv. „flat memory model“ (využívá například uClinux[11]), kde jádro i všechny procesy sdílejí stejný adresní prostor. Nevýhodou je, že chyba v jednom procesu může „shodit“ celý systém. Další možností je využít FCSE dostupné v některých mikroprocesorech ARM, které umožňuje obejít nutnost zneplatnit obsah cache pamětí. Přesný popis mechanismu funkce tohoto rozšíření a popis jeho ovladače je v [14]. Zlepšení latence při použití rozšíření FCSE ukazuje histogram 3.1.

na použité knihovně jazyka C (podpora mutexů typu `PTHREAD_PRIO_INHERIT`). Knihovna uClibc použitá v současné verzi toto bohužel nepodporuje.

[10]Fast Context Switch Extension

[11]Varianta Linuxu určená pro mikroprocesory bez jednotky ochrany paměti - například AT91SAM7S256 s jádrem ARM7TDMI, viz. www.uclinux.org



Obr. 3.1: Histogram zobrazující četnost jednotlivých hodnot latence s použitím FCSE (vlevo) a bez něj (vpravo). Zdroj [14].

3.3.3 Nástroje pro testování

Nástrojem pro otestování real-time schopností platformy a operačního systému je například často citovaný *cyclictest*, naprogramovaný Thomasem Gleixnerem. Program volaný s následujícím parametry

```
cyclictest -m -a -t -n -p99
```

spustí pro každé jádro procesoru jedno testovací vlákno. První má prioritu specifikovanou parametrem p, každé další o jedna menší. Do konzole se vypisuje minimální, průměrná a maximální latence v us. Tabulka 3.1 uvádí výsledky získané na platformě x86 a ARM9. Ve všech případech byla perioda vláken 1 ms a zátěž byla generována pomocí „záplavového“ pingu. První řádek tabulky ukazuje výsledky testu na x86 s běžným desktopovým jádrem, kdy byl testovací program spušten bez práv superuživatele a tudíž neměl rt prioritu. Latence dosáhla maximální hodnoty 23 ms pro vlákno s větší prioritou (hodnota pro druhé vlákno je uvedena v závorce). Při spuštění s RT prioritou byla maximální latence „jen“ cca 6 ms. Na procesoru ARM byl test spušten s RT jádrem, s různým nastavením: plánování procesů jednou za 1 ms (první řádek), s plánováním podle potřeby (druhá řádek, volba jádra `CONFIG_NO_HZ`).

platforma	jádro	min.	avg.	max.
x86_64 (non-rt)	2.6.38-8-generic-ck	8 (9)	597 (544)	22799 (26324)
x86_64 (rt)	2.6.38-8-generic-ck	4 (5)	17 (25)	5681 (5465)
ARM9 (1000 Hz)	2.6.33.7.2-rt30	59	103	251
ARM9 (dyn.)	2.6.33.7.2-rt30	70	116	291

Tab. 3.1: Výstup cyclictestu pro různé platformy a jádra

3.4 POSIX API

POSIX je standard (IEEE 1003, ISO/IEC 9945) určující, jak má vypadat API (rozhraní) operačních systémů. Většina Unixových systémů je POSIX kompatibilní.

Mimo jiné standard POSIX také obsahuje funkce pro vytváření real-timových programů. K dispozici je až 100 priorit pro běh procesů a skupině procesů je možné nastavit druh plánování pomocí tříd. Ve třídě `SCHED_FIFO` proces běží dokud není dokončen, nebo není k běhu připraven proces s vyšší prioritou, `SCHED_RR` - procesy jsou plánovány pomocí Round Robin algoritmu, `SCHED_OTHER` - určeno pro ostatní procesy, s ne-RT prioritou. Pro spouštění RT procesů slouží utilita `chrt` (je součástí balíku BusyBox), ale program může deklarovat i sám sebe jako RT. Dále budou popsány některé důležité funkce použité v práci.

Funkce pro manipulaci s vlákny jsou v knihovně `pthread`. K vytvoření nového vlákna slouží funkce *pthread_create*, jejíž hlavička je:

```
int  pthread_create(pthread_t  *  thread,
                    pthread_attr_t * attr,
                    void *(*start_routine)(void *),
                    void * arg)
```

Parametr *thread* je ukazatel na strukturu `pthread_t`, která bude touto funkcí inicializována, *attr* je nepovinný ukazatel na strukturu `pthread_attr_t`, díky které je možné ovlivnit některá nastavení vlákna. *Start_routine* je funkce, která bude vláknem vykonávána a *arg* je ukazatel na data, která mohou být této funkci předána.

Funkce *pthread_join* s hlavičkou

```
int pthread_join(pthread_t thread, void **retval);
```

slouží k pozastavení běhu volajícího vlákna do té doby, než skončí běh jiného vlákna, určeného parametrem *thread*. Pomocí parametru *retval* je možné získat návratovou hodnotu ukončeného vlákna. Běžící vlákno je možné ukončit také „násilně“, voláním *pthread_cancel* s hlavičkou:

```
int pthread_cancel(pthread_t thread);
```

Funkce *pthread_cancel* pošle požadavek na ukončení vláknu *thread*. Pokud je vlákno ve zrušitelném stavu (nastavení funkcí *pthread_setcancelstate*), bude zrušeno okamžitě. Pokud ne, bude požadavek uložen ve frontě a proveden, jakmile dojde ke změně stavu vlákna. Nastavováním zrušitelnosti vlákna je tedy možné definovat úseky kódu, kdy vlákno nemůže být ukončeno. Možnost ukončení běhu vlákna je dále ovlivněna nastavením typu zrušitelnosti (*pthread_setcanceltype*) na asynchronní kdy vlákno může být ukončeno kdykoliv, nebo odloženou kde zrušení je možné pouze při volání funkcí představujících bod zrušení.

Synchronizační prostředky jsou důležité pro každý operační systém a jeho správnou funkci. U systémů na které se kladou RT požadavky toto platí dvojnásob. Jedním z běžně používaných synchronizačních prostředků je mutex, kterých Linux nabízí hned čtyři druhy. Prvním typem je „fast“ mutex, u nějž je preferována rychlost nad korektností. Při pokusu o zamčení nejsou prováděny téměř žádné kontroly, například není kontrolován vlastník, takže mutex může být odemčen jiným vláknem než tím, které ho zamčelo. Dále může nastat deadlock, pokud se vlákno pokusí zamčít mutex, který zamčelo už dříve. Mutexy jsou deklarovány typem **pthread_mutex_t** a jejich inicializace může být buď statická:

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

nebo run-time za běhu programu:

```
pthread_mutex_t mutex;
...
pthread_mutex_init (&mutex, NULL);
```

Druhý parametr funkce *pthread_mutex_init* je odkaz na strukturu typu **pthread_mutexattr_t**. Hodnota *NULL* znamená výchozí, tedy rychlý-fast. Dalším typem mutexu je „error checking“, který je pomalejší, ale bezpečnější, proto může být výhodné používat tento typ při vývoji programu. Deklarovat se může opět staticky:

```
pthread_mutex_t mutex = PTHREAD_ERRORCHECK_MUTEX_INITIALIZER;
```

nebo za běhu programu, tentokrát již s definicí struktury **pthread_mutexattr_t**:

```
pthread_mutex_t mutex;
pthread_mutexattr_t attr;
pthread_mutexattr_init (&attr);
pthread_mutexattr_settype (&attr, PTHREAD_MUTEX_ERRORCHECK);
pthread_mutex_init (&mutex, &attr);
```

Tento typ vrátí při pokusu o znovu-zamčení hodnotu *EDEADLK* a při pokusu o odemčení mutexu jiného vlastníka *EPERM*. Třetím typem je rekurzivní mutex

(*PTHREAD_MUTEX_RECURSIVE*). Ten může být zamčen vícekrát, počet zamčení je uchováván v čítači a musí být tolikrát i odemčen. Posledním typem je adaptivní mutex. Ten je určen pro vícejádrové mikroprocesory. Kombinuje spinlock (aktivní čekání) s tradičním mutexem. Při pokusu o zamčení mutexu je nejdříve po krátkou dobu použito aktivní čekání a teprve nepodaří-li se během něj uzamknout mutex, je běh vlákna zablokován. Na jednojádrových systémech nemá aktivní čekání smysl, takže není používáno a tento typ je ekvivalentní rychlému mutexu.

O získání mutexu je možné se pokusit více způsoby. Volání funkce *pthread_mutex_lock* zablokuje běh vlákna do doby, než dojde k získání mutexu. Funkce *pthrad_mutex_trylock* se o zamčení pouze pokusí. Pro použití v RT systémech může být výhodná funkce *pthread_mutex_timedlock*, které se jako parametr předává absolutní hodnota času, kdy má dojít k přerušení čekání a vrácení hodnoty *ETIMEDOUT*. Odemčení mutexu zajišťuje funkce *pthread_mutex_unlock*. Při rozhodování, které vlákno čekající na uvolnění mutexu bude naplánováno k běhu, mají přednost RT vlákna a mezi nimi je určující priorita. U běžných vláken hraje roli délka čekání.

Pokud má být mutexem chráněna například sdílená paměť, je potřeba, aby byl mutex přístupný z více procesů. To se zajistí následující inicializací:

```
pthread_mutexattr_setpshared (&attr, PTHREAD_PROCESS_SHARED);
```

U RT procesů je požadováno, aby nedošlo k inverzi priorit, kdy vlákno s vyšší prioritou čeká na mutex držený vláknem s nižší prioritou. Tomu je možné zabránit použitím „priority inheritance“ mutexů. Jejichž nastavení vypadá následovně:

```
#define __USE_UNIX98
#include <pthread.h>

pthread_mutex_t mutex;
pthread_mutexattr_t attr;

pthread_mutexattr_init (&attr);
pthread_mutexattr_setprotocol (&attr, PTHREAD_PRIO_INHERIT);
pthread_mutex_init (&mutex, &attr);
```

Tento atribut může být nastavena u všech čtyřech typů zmíněných výše. Ale z důvodu velké režie nemá příliš smysl používat jej u rychlých a adaptivních mutexů. Dále může být mutexu nastaven atribut *PTHREAD_MUTEX_ROBUST_NP* (před includováním `pthread.h` je nutné definovat *__USE_GNU*). Robustní mutex má tu výhodu, že pokud z nějakého důvodu ukončí činnost proces nebo vlákno vlastnící mutex, je tato skutečnost vláknům čekajícím na jeho uvolnění signalizována návratovou hodnotou

EOWNERDEAD. Protože je v případě využití této vlastnosti vyžadována další režie, není tento typ příliš rychlý.

Pro nastavování RTe priority vlákna slouží funkce *pthread_setschedparam* definovaná v hlavičkovém souboru `sched.h`, z knihovny `pthread`, jejíž hlavička je:

```
pthread_setschedparam(pthread_t thread, int policy,
                      const struct sched_param *param);
```

Parametr *thread* je vlákno pro které se bude nastavení provádět, *policy* je třída ve které bude vlákno plánováno a *param* je ukazatel na strukturu definující prioritu. Samotné nastavení priority pak vypadá následovně:

```
struct sched_param my_param;
my_param.sched_priority = 99;
pthread_setschedparam(pthread_self(), SCHED_FIFO,&my_param);
```

3.5 Ovladače v Linuxu

V Linuxu je několik možností jak vytvářet ovladače. První cestou je vytvořit modul, který může být za běhu systému vložen do jádra a bude zprostředkovávat pomocí svého rozhraní procesu v uživatelském prostoru například přístup k hardware. Veškerá obsluha zařízení potom probíhá v jaderném kódu, je možné přistupovat kamkoliv v paměti, využívat přerušení. Další poměrně moderní možností je UIO framework pro ovladače v uživatelském prostoru. V jádře je taktéž zaveden modul, ale v tomto případě se stará jen o nejzákladnější operace (např. obsluha přerušení) a těžiště práce ovladače je přesunuto do uživatelského prostoru. Poslední možnost představuje ovladač běžící kompletně v uživatelském prostoru, který mapuje fyzickou paměť zařízení do virtuální. Výhodou tohoto řešení je jednoduchost a rychlost, nevýhodou nemožnost využívat přerušení a určitá bezpečnostní rizika.

3.5.1 Jaderné moduly

Ovladače zařízení jsou tradičně součástí jádra. Mohou být přímo zakompilovány do jeho kódu, nebo mohou tvořit takzvané moduly, které je možné do jádra vložit za běhu systému. Díky této vlastnosti je někdy linuxové jádro, běžně označované jako monolitické, považováno za hybridní.

Zařízení se v Linuxu obvykle dělí do tří tříd: znaková, bloková a síťová. Protože Linux vychází z filosofie Unixu, podle které je vše soubor, je možné k zařízením prvních dvou zmíněných tříd přistupovat jako k souborům. Jedná se o speciální soubory umístěné v adresáři `/dev`.

Znaková zařízení se vyznačují tím, že jde v podstatě o proudu bytů-znaků. Ty je do nich možné zapisovat, nebo je z nich číst. Obvykle není možné se v proudu dat přesouvat, je tedy možné zapisovat pouze na konec proudu, ale není to pravidlem. Typickým zástupcem znakových zařízení je sériový port (`/dev/ttyS0` apod.). Ovladač znakového zařízení obvykle implementuje systémová volání *open*, *close*, *read* a *write*. Komunikace s blokovými zařízeními probíhá pomocí bloků dat o pevné velikosti, které je možné adresovat. Typickým zástupcem této třídy zařízení je například pevný disk. Třída síťových zařízení je speciální, protože neposkytuje rozhraní v podobě souboru v adresáři `/dev` a nebude zde popisována.

Ukázka kódu nejjednoduššího možného modulu převzatá z [11]:

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
static int hello_init(void)
{
  printk(KERN_ALERT "Hello, world\n");
  return 0;
}
static void hello_exit(void)
{
  printk(KERN_ALERT "Goodbye, cruel world\n");
}
module_init(hello_init);
module_exit(hello_exit);
```

V modulu je definována použitá licence a dále dvě funkce. Funkce `hello_init` je zavolána při inicializaci modulu a `hello_exit` při odstranění modulu z jádra. Pro výpis je použita funkce `printk`, což je obdoba běžné `printf`, kterou však není možné používat, protože v jádře nejsou dostupné funkce ze standardní C knihovny.

3.5.2 UIO framework

UIO framework byl začleněn do jádra verze 2.6.22. Jeho účelem je nabídnout vývojářům možnost programovat ovladače zařízení v uživatelském prostoru. Výhodou ovladače v uživatelském prostoru je to, že kód „tradičního“ ovladače běžící v prostoru jádra může napáchat při chybě velké škody a dokonce shodit celý systém, naproti tomu kód v uživatelském prostoru zdaleka není tak nebezpečný. Toto může představovat značnou výhodou zejména při nedostupnosti zdrojového kódu ovladače, kdy není možné ověřit jeho kvalitu. Podle [10] je ovladač tohoto typu vhodný pro zařízení které:

- mají mapovatelnou paměť a je možné je řídit zápisem do této paměti,
- generují přerušení, která je potřeba obsluhovat,
- se nehodí pro obsluhu žádným z jaderných subsystémů.

Jako vhodný kandidát pro tento typ ovladače jsou uváděny průmyslové I/O karty. Potřeba jaderného modulu není odstraněna úplně, ale rozsah jeho kódu je minimalizován. Modul se v podstatě stará jen o zpřístupnění paměti zařízení (která může být mapována procesem v uživatelském prostoru) a obsluhu přerušení.

Každé zařízení využívající UIO framework je reprezentováno speciálním znakovým souborem `/dev/uioX` (kde x je číslo zařízení). Tento soubor zpřístupňuje paměť zařízení, která může být namapována do virtuálního adresního prostoru procesu-ovladače. Přerušení jsou obsluhována čtením souboru zařízení. Blokující čtení souboru `/dev/uioX` se vrátí jakmile dojde k vyvolání přerušení. Funkce *read* vrací počet přerušení vygenerovaných od minulého čtení. Samozřejmě je možné použít také neblokující čtení.

3.5.3 Mapování paměti

Nejjednodušší cestou, jak programu v uživatelském prostoru zpřístupnit fyzickou paměť, tedy například konfigurační registry periferií mikroprocesoru, je pomocí mapování fyzické paměti do virtuální paměti procesu. Do fyzické paměti je potom možné zapisovat stejně snadno, jako do jakékoliv proměnné programu. Nevýhodou tohoto přístupu je nutnost synchronizovat přístup více vláken (nebo procesů) ke stejným adresám a dále zejména nemožnost využívat přerušení. Pokud ale k danému rozsahu adres bude přistupovat pouze jedno vlákno a přerušení nejsou vyžadována, jedná se o velmi výhodný způsob jak pracovat s hardwarem. V Linuxu je fyzická paměť reprezentována speciálním souborem `/dev/mem`. Potenciálním bezpečnostním rizikem může být, že proces, který chce mapovat paměť musí běžet pod uživatelem root (což je ale v embedded systémech běžné). Další možností je zpřístupnit speciální soubor `/dev/mem` pro čtení a zápis i jiným uživatelům, ale tento přístup také není bez rizika.

Mapování paměti probíhá pomocí funkce *mmap* definované v hlavičkovém souboru `sys/mman.h`. Její hlavička je:

```
void  *mmap(void *addr,
            size_t length,
            int prot,
            int flags,
            int fd,
            off_t offset);
```

Parametr *addr* specifikuje počáteční adresu virtuální paměti kam se bude mapovat. Pokud je *addr* rovno *NULL*, rozhodne o virtuální adrese jádro, což se také často používá. Virtuální adresa je v případě úspěchu funkcí vrácena jako návratový parametr. Druhý parametr *length* určuje počet mapovaných bytů ze souboru, jehož identifikátor je funkci *mmap* předán jako *fd*. Mapování ze souboru se provádí počínaje adresou *offset*, která musí být násobkem velikosti stránky dané architektury (např. 4096 bytů). Argument *prot* umožňuje kombinací hodnot *PROT_EXEC*, *PROT_READ*, *PROT_WRITE*, *PROT_NONE* nastavit žádaný stupeň ochrany mapované paměti. Například bitová maska *PROT_READ*|*PROT_WRITE* zajistí do paměti přístup pro čtení a zápis. Aby byly provedené změny skutečně zapsány do fyzické paměti, musí být argument *flags* nastaven na *MAP_SHARED*.

V případě neúspěchu je vrácena hodnota *MAP_FAILED* a ze speciální proměnné *errno* je možné získat kód specifikující chybu blíže. Paměť je odmapována buď ukončením procesu, nebo zavoláním funkce *munmap*.

Ukázka funkce provádějící mapování registrů pro ovládání GPIO portů do virtuální paměti procesu:

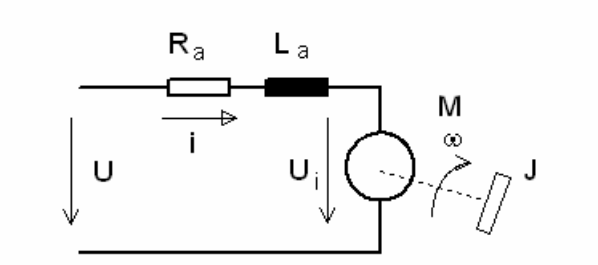
```
void gpio_mapMemory(struct gpio_t *g) {
  #define GP_MAP "[gpio_mapMemory()] "
  int fd_mem;
  fd_mem = open("/dev/mem", O_RDWR | O_SYNC);
    if (fd_mem < 0) errExit(GP_MAP,"Failed to open /dev/mem")
        ;
  g->map_base = (uint32_t *)mmap(0, MAP_SIZE,
        PROT_READ|PROT_WRITE, MAP_SHARED, fd_mem, g->target &
            ~MAP_MASK);
  if (g->map_base == MAP_FAILED) errExit(GP_MAP,"mmap failed"
     );
  else printf("%sMemory at %X mapped as %p\n",GP_MAP,
        (uint32_t)g->target,g->map_base);
  g->per  = (uint32_t *) (g->map_base +
              ((g->target + PIO_PER) & MAP_MASK));
  g->pdr  = (uint32_t *) (g->map_base +
              ((g->target + PIO_PDR) & MAP_MASK));
  ...
  if (close(fd_mem))
    errExit(GP_MAP,"close fd_mem failed");
}
```

Funkce `gpio_mapMemory` otevře pro čtení a zápis soubor `/dev/mem`, který reprezentuje fyzickou paměť. Následně je provedeno mapování tohoto souboru od offsetu daného

proměnnou `target` do virtuální paměti. Adresa virtuální paměti je vrácena do `map_base`. Ukazatele na jednotlivé registry jsou poté získány pomocí ukazatelové aritmetiky - k bázové virtuální adrese je přičten offset `target` a offset jednotlivých registrů `PIO_xxx`. Zavřením souboru `/dev/mem` nedojde k odmapování paměti a proto může být na konci funkce deskriptor `fd_mem` uvolněn.

4 IDENTIFIKACE A ŘÍZENÍ POHONŮ

Pro polohování stojanu jsou použity pohony tvořené malými stejnosměrnými motorky. Ty jsou buzeny permanentními magnety a na výstupu mají převodovku. Jedinou známou informací o použitých motorcích je jejich jmenovité napětí 12 V a maximální otáčky 3800 za minutu. Náhradní schéma stejnosměrného motoru s permanentními magnety je na obr. 4.1.



Obr. 4.1: Náhradní schéma stejnosměrného motoru s konstantním magnetickým tokem[16]

Stejnosměrný motor je matematicky popsán soustavou diferenciálních rovnic a fyzikálně obsahuje dva akumulátory energie: kinetickou energii rotoru a magnetickou energii rotorového vinutí. Pro obvod kotvy platí rovnice reprezentující rovnováhu napětí v obvodu kotvy:

$$U = R_a I_a + L_a \frac{di_a}{dt} + C\phi\omega$$

Rovnováha momentů na hřídeli motoru je pak popsána rovnicí:

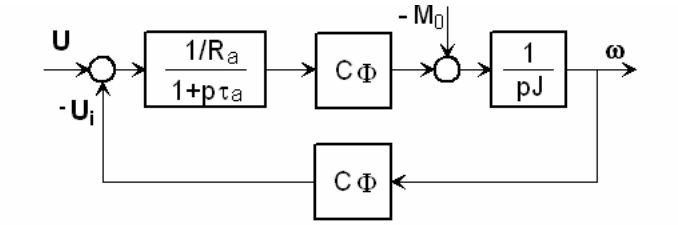
$$C\phi i_a = J\frac{d\omega}{dt} + B\omega + M_0$$

kde J představuje celkový moment setrvačnosti na hřídeli motoru, B koeficient viskózního tření a M_0 moment odporu. S využitím Laplaceovy transformace je možné z uvedených rovnice sestavit operátorový přenos, jehož odvození je provedeno v [16]. Matematický model v Laplaceově transformaci je tvořen přenosy $F_1(p)$ až $F_4(p)$ (viz. 4.2):

$$F_1(p) = \frac{\frac{1}{R_a}}{1 + p\tau_a} \quad \tau = \frac{L_a}{R_a}, \qquad F_2(p) = F_4(p) = C\phi, \qquad F_3(p) = \frac{\omega}{M - M_0}$$

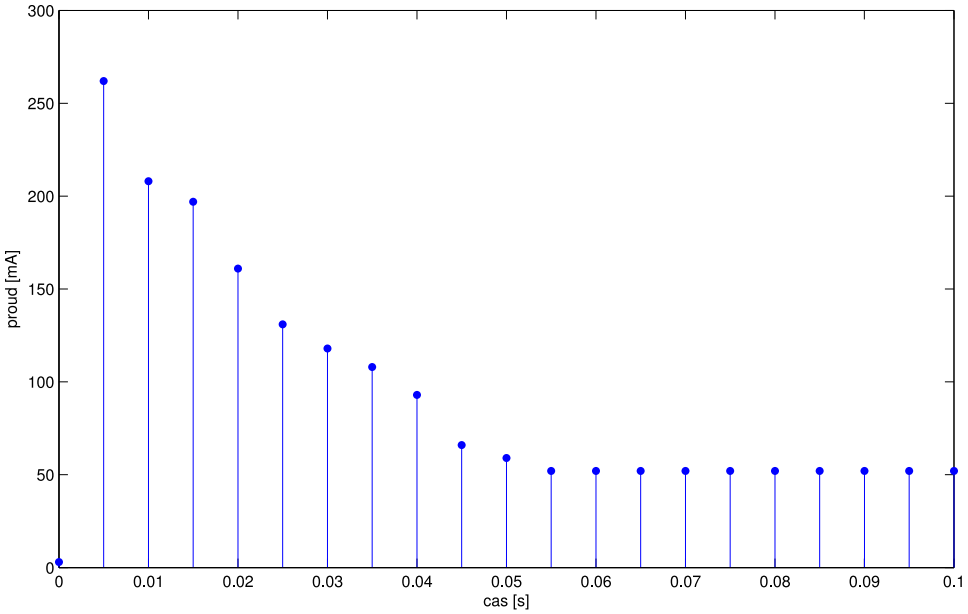
4.1 Experimentální identifikace

Protože parametry motoru ani převodovky nejsou známy, bylo přikročeno k provedení experimentální identifikace. Vstupem regulované soustavy je napětí přivedené na



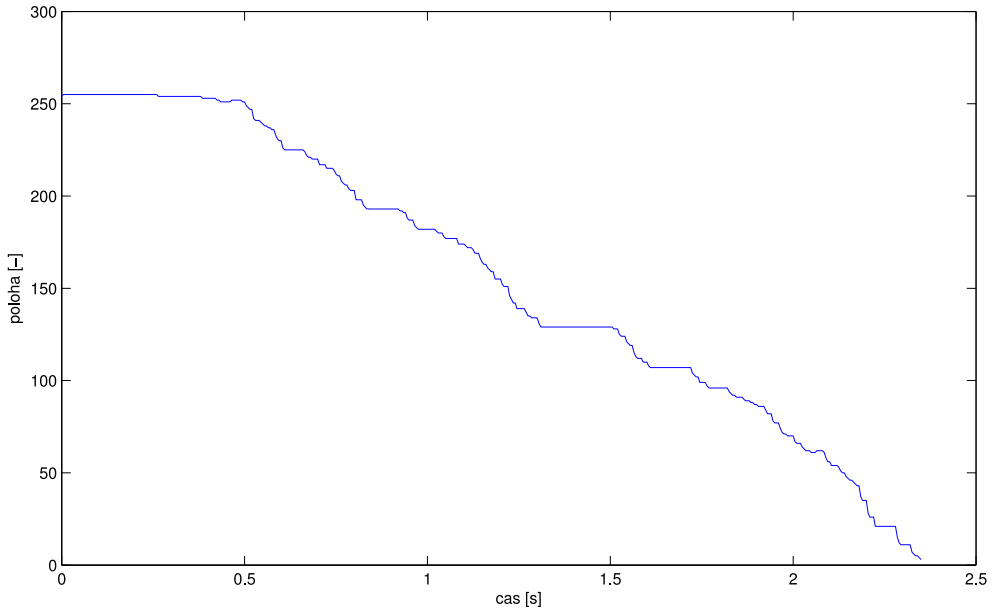
Obr. 4.2: Matematický model stejnosměrného motoru[16]

svorky motoru, které je úměrné činiteli plnění PWM. Výstupem je proud kotvou a napětí na potenciometru snímajícím úhel natočení. Protože známé je vstupní napětí, proud procházející kotvou a poloha, bylo by možné modelovat soustavu jako sériové spojení přenosů $F_{id1} = \frac{proud}{PWM}$ a $F_{id2} = \frac{poloha}{proud}$, nebo při použití zpětné vazby jen od polohy jako $F_{celk} = \frac{poloha}{pwm}$. Pro získání představy o průbězích proudu a polohy byla sejmuta přechodová charakteristika. Na vstup motorku bylo v čase t=0 přivedeno napětí 12 V (činitel plnění PWM 1,0) a výstupem jsou průběhy 4.3 a 4.4, kde perioda vzorkování činí 5 ms.



Obr. 4.3: Přechodová charakteristika - proud

Při získávání modelů za pomocí Identification toolboxu z programu Matlab a jejich simulaci v Simulinku se ukázaly být významným problémem nelinearity soustavy. Vůle převodovky není nezanedbatelná, problémem je ale zejména vůle mezi hřídělí převodovky a prvkem propojujícím ji s potenciometrem měřícím polohu. Další



Obr. 4.4: Přechodová charakteristika - poloha

nelinearita je způsobena faktem, že stejnosměrný motorek překoná odpor převodovky a potenciometru a začne se točit až při činiteli plnění PWM cca 0,5 (přičemž tato hodnota záleží také na aktuální poloze), takže pásmo necitlivosti je poměrně široké. Další nelinearitu představují potenciometry, jejichž odpor také není ideálně lineární. Bohužel tedy nejde ani přibližně o lineární systém a ukázalo se, že jeho matematický popis pomocí lineárních modelů je možný, ovšem při zanedbání nelinearit neodpovídá realitě. Na základě experimentů byl tedy navržen a odzkoušen empirický P regulátor (algoritmus popsán v 5.1.6), který se při praktických testech ukázal jako dostatečně spolehlivý.

5 SOFTWAROVÉ ŘEŠENÍ

Pro účely práce byl vytvořen a nakonfigurován pomocí balíku Buildroot vlastní linuxový systém a vytvořeno prostředí pro křížovou kompilaci. Do linuxového jádra verze 2.6.33.7 byl aplikován patch pro podporu použité vývojové desky dodaný na CD s kitem, patch opravující některé chyby v jádře u mikroprocesorů AT91[1] a patche pro zlepšení RT odezvy popsané blíže v 3.3.1 a 3.3.2. Řízení pohonů stojanu, obsluhu potřebných periferií mikroprocesoru, H-můstků a AD převodníku na vytvořené desce s elektronikou realizuje vlastní program napsaný v C. Pro nahrání jádra a image filesystému do příslušných pamětí, byly použity upravené skripty dodané na CD s kitem. Pro zjednodušení a urychlení práce byl během vývoje filesystém připojen přes NFS a umístěn na PC. Pro streamování obrazu z webkamery je použit program MJPG-streamer šířený pod licencí GPL.

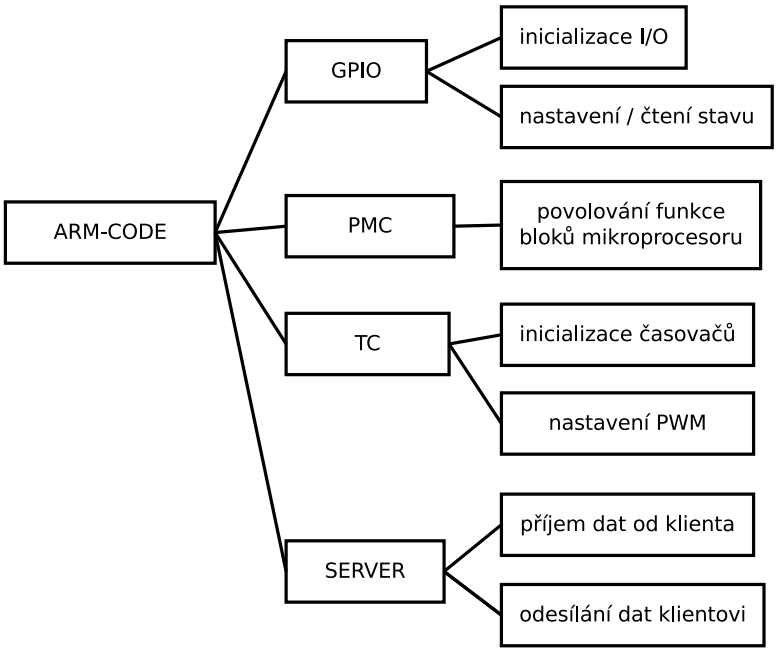
5.1 Program pro ARM

Program zajišťující řízení stojanu je napsán v jazyce C ve vývojovém prostředí Eclipse. Protože využívá ovládání periferií metodou mapování paměti, je nutné ho spouštět s právy superuživatele (root). Rozdělen je do několika modulů. Kromě hlavního modulu *arm-code* jsou to: *gpio* kde je řešena obsluha vstupně výstupních pinů, modul *pmc* zajišťuje povolení funkce potřebných periferií, *server* se stará o komunikaci s klientem (programem na PC) a v modulu *tc* jsou funkce pro obsluhu a nastavení časovačů. Důležité proměnné jsou podle svého účelu sdruženy do struktur, pro snadné předávání do jednotlivých funkcí. Kde to má význam, jsou struktury chráněny mutexy. Bloková struktura modulů vytvořeného SW je znázorněna v obr. 5.1. Moduly implementující ovladače zařízení jsou v práci popsány detailněji, protože jde o využití zajímavé možnosti, jak v OS Linux přistupovat k periferiím mikroprocesoru.

5.1.1 Logování, obsluha chyb

Program při běhu posílá do systémového logu informace o své činnosti, jejichž součástí je čas, jméno funkce a další údaje. Pokud je program spušten s parametrem *-v*, nebo je systémový log nedostupný, jsou logy vypisovány na standardní výstup. V případě výskytu závažné chyby je běh programu ukončen a do standardního výstupu je vypsán důvod chyby. Pro logování a obsluhu chyb jsou definována dvě makra, která jsou následně používána v celém programu, protože zjednodušují odhalení případných chyb.

[1] Ke stažení na http://maxim.org.za/at91_26.html



Obr. 5.1: Blokové schéma modulů programu a jejich účelu

```
#define LOG(...) { char _bf[1024] = {0};
   snprintf(_bf, sizeof(_bf)-1, __VA_ARGS__); syslog(LOG_INFO
      , "%s", _bf); }
#define errExit(fcn,msg)    do { puts(fcn);
   perror(msg); closelog(); exit(EXIT_FAILURE);} while (0)
```

Volání makra `LOG` potom vypadá například takto:

```
LOG("%sI: Pin P%c%d set as input, driven by PIO",GP_IIN,g->id
   ,bit)
```

Makro `errExit` vypíše v programu definovaný text, vysvětlení poslední hodnoty chyby (proměnná *errno*) a ukončí program s chybovou návratovou hodnotou. Jeho volání může vypadat například takto:

```
if (pthread_mutex_init(&(s->stand_info_mutex), NULL))
  errExit(MAIN_SIM,"Failed to initialize stand_info_mutex");
```

5.1.2 Modul gpio

V hlavičkovém souboru *gpio.h* jsou definovány bázové adresy jednotlivých portů mikroprocesoru a offsety registrů sloužících k obsluze těchto portů. Dále je definována struktura typu `gpio_t`, ve které jsou ukazatele na jednotlivé registry, proměnná pro uložení bázové adresy portu, ukazatel na virtuální adresu kam bude prováděno

mapování a proměnná typu `char` pro uložení písmena identifikujícího daný port. Proměnné-ukazatele na jednotlivé porty musí mít modifikátor `volatile`, protože jinak by nebylo zaručeno uložení dat na danou adresu a mohlo by dojít k „vyoptimalizování“ těchto proměnných z výsledného kódu.

```
struct gpio_t {
  volatile uint32_t *per;
  volatile uint32_t *pdr;
  ....
  volatile off_t target;
  volatile void *map_base;
  char id;
};
```

Mapování paměti pro jeden port provádí funkce `gpio_mapMem` (viz. 3.5.3), která má jako parametr ukazatel na strukturu typu `gpio_t`. Tuto strukturu je nejprve nutné alokovat a nastavit požadovanou bázovou adresu. Příslušný kód vypadá následujícím způsobem:

```
#define BA_GPIO_PIOA 0xFFFFF400U
static struct gpio_t gpio_a;
gpio_a.id = 'A';
gpio_a.target = BA_GPIO_PIOA; // port A address
gpio_mapMem(&gpio_a);
```

Opětovné odmapování paměti je řešeno funkcí *gpio_cleanup*, která pouze zavolá *munmap*. Funkce *gpio_mamMem* provede otevření souboru `/dev/mem` a voláním *mmap* získá virtuální adresu na kterou je namapovaná požadovaná fyzická adresa. Pomocí offsetů jednotlivých registrů přičtených k ukazateli na bázovou adresu jsou získány ukazatele na registry. V případě neúspěchu této funkce je ukončen běh celého programu.

Všechny dále popisované funkce slouží pro manipulaci s konkrétním pinem portu. Jako parametry očekávají ukazatel na strukturu typu `gpio_t` a číslo pinu se kterým se má pracovat. Proto budou u těchto funkcí popsány pouze jejich specifické parametry.

Pro správnou funkčnost je nutné správně inicializovat piny daného portu. Pro inicializaci vstupních pinů slouží funkce *gpio_initIn*, kde třetí parametr určuje, jestli má mít daný pin zapnutý pull-up rezistor. Funkce pro všechny vstupní piny zapíná vstupní digitální filtr, zjišťuje hodnotu odpovídajícího bitu v registru PSR[2], kde jednička znamená, že daný pin je řízen PIO kontrolérem. Dále je testováno, jestli byl daný pin skutečně nastaven na vstupní, což je indikováno hodnotou nula příslušného bitu registru OSR[3] a výsledky obou kontrol jsou zapsány do logu.

[2]Peripheral Status Register

[3]Output Status Register

```
void gpio_initIn(struct gpio_t *g,
  unsigned char bit, unsigned char pullup) {

  #define GP_IIN "[gpio_initIn()   ] "

  *(g->per) = (uint32_t)(1<<bit);
  *(g->ifer) = (uint32_t)(1<<bit); // input filter
  *(g->odr) = (uint32_t)(1<<bit); // output disable
  *(g->codr) = (uint32_t)(1<<bit); // clear output reg.

  if (pullup) *(g->puer) = (uint32_t)(1<<bit); // en. pull-up
  else *(g->pudr) = (uint32_t)(1<<bit); // disable pull-up

  if ((((*(g->psr))>>bit) & 0x1)==1) {
    if ((((*(g->osr))>>bit) & 0x1)==0)
        LOG("%sI: Pin P%c%d set as input, driven by PIO",
           GP_IIN,g->id,bit)
      else LOG("%sW: Failed to set pin P%c%d as input",GP_IIN
         ,g->id,bit);
  } else LOG("%sW: Pin P%c%d not driven by PIO",GP_IIN,g->id,
     bit);
}
```

Inicializaci výstupu zajišťuje funkce *gpio_initOut*, jejíž třetí parametr určuje, jestli bude daný pin ovládán PIO kontrolérem, periferií A, nebo B. Funkce kontroluje zda byl pin skutečně nastaven jako výstupní a je řízen požadovanou periferií, nebo PIO kontrolérem.

```
void gpio_initOut(struct gpio_t *g,
  unsigned char bit, unsigned char mode) {
  #define GP_IOUT "[gpio_initOut()  ] "

  if (mode==0) { // set as PIO
    *(g->per) = (uint32_t)(1<<bit);
    *(g->oer) = (uint32_t)(1<<bit);
    *(g->pudr) = (uint32_t)(1<<bit);
    *(g->ower) = (uint32_t)(1<<bit);}

  if (mode==1) { // peripheral A
    *(g->pdr) = (uint32_t)(1<<bit);
    *(g->asr) = (uint32_t)(1<<bit);}
```

```
  if (mode==2) { // peripheral B
    *(g->pdr) = (uint32_t)(1<<bit);
    *(g->bsr) = (uint32_t)(1<<bit);}

  if ((((*(g->psr))>>bit) & 0x1)==1) {

    if ((((*(g->osr))>>bit) & 0x1)==1)
      LOG("%sI: Pin P%c%d set as output, driven by PIO",
         GP_IOUT,g->id,bit)
    else LOG("%sW: Failed to set pin P%c%d as output!",
       GP_IOUT,g->id,bit);
  } else {
    if ((((*(g->absr))>>bit) & 0x1)==1)
      LOG("%sI: Pin P%c%d driven by peripheral B",GP_IOUT,g->
         id,bit)
    else LOG("%sI: Pin P%c%d driven by peripheral A",GP_IOUT,
       g->id,bit);
  }
}
```

Ovládání stavu výstupu zajišťuje funkce *gpio_setOutSt*, které se jako třetí parametr předává požadovaný stav výstupního pinu. Funkce neprovádí žádné kontroly a je označena jako **inline**, aby její provedení bylo co nejrychlejší, protože je mimo jiné použita pro softwarově generované SPI.

```
inline void gpio_setOutSt(struct gpio_t *g,
  unsigned char bit, unsigned char state) {

  if (state) *(g->sodr) = (uint32_t)(1<<bit);
  else *(g->codr) = (uint32_t)(1<<bit);

}
```

Pro načtení stavu vstupu je určena funkce *gpio_readInSt*, která je opět velmi jednoduchá:

```
inline uint8_t gpio_readInSt(struct gpio_t *g,
                             unsigned char bit) {

  return ((*(g->pdsr)>>bit) & 0x1);
}
```

Pro zamezení vzniku zbytečných chyb při práci s GPIO je možné používat následující přehledný zápis, kdy je odkaz na strukturu odpovídající příslušnému

portu a číslo pinu nahrazeno definicí. Díky tomuto zápisu je také možné snadno měnit význam jednotlivých pinů, bez přepisování celého programu.

```
static gpio_t gpio_a;
#define SPI_MOSI &gpio_a, 24
...
gpio_setOutSt(SPI_MOSI,1);
```

5.1.3 Modul tc

V hlavičkovém souboru *tc.h* je definována bázová adresa čítače-časovače 1 - TC1 (TC0 není možné použít, protože je využíván operačním systémem) a dále jsou definovány offsety registrů a umístění významných bitů v nich. V hlavičkovém souboru je také definována struktura pomocí které se přistupuje k jednomu kanálu časovače. Mapování registrů potřebných pro ovládání čítačů-časovačů do paměti zajišťuje funkce *tc_mapMem*, jejíž funkce je obdobná funkci *gpio_mamMem*, takže nebude blíže popsána.

Jediným využívaným módem čítače-časovače je generování PWM. Nastavení do tohoto režimu provádí funkce *tc_mode_pwm*, které se jako parametry předává polarita generovaného signálu pro výstupy a,b. Polarita rovna nule znamená, že daný výstup (TIOxn) bude vynulován při shodě obsahu čítače s registrem RA a nastaven při dosažení hodnoty RC. Časovač je nastaven na čítání směrem nahoru, s vynulováním při dosažení hodnoty RC. Funkce ověřuje zda je generování PWM povoleno čtením hodnoty bitu CLKSTA z registru SR.

```
void tc_mode_pwm(struct tc_t *tc, uint8_t pola, uint8_t polb)
    {

  #define TC_MPW "[tc_mode_pwm()   ] "

  uint32_t tmp;
  tmp = (uint32_t)(1<<WAVE)|(0b10<<WAVSEL)|(TIMER_CLOCK1);

  *tc->bmr = 0b010101; // no external clock

  if (pola==0) tmp |= (uint32_t)(2<<ACPA)|(1<<ACPC);
  else tmp |= (uint32_t)(1<<ACPA)|(2<<ACPC);

  if (polb==0) tmp |= (uint32_t)(2<<BCPB)|(1<<BCPC);
    else tmp |= (uint32_t)(1<<BCPB)|(2<<BCPC);
```

```
  *tc->cmr = tmp;
  *tc->ccr = (uint32_t)(1<<SWTRG)|(1<<CLKEN); // clock-enable

  if ((((*tc->sr)>>CLKSTA) & 0x1)==1)
    LOG("%sI: PWM on channel %d activated",TC_MPW,tc->chan)
  else LOG("%sW: Failed to activate PWM mode on channel %d",
     TC_MPW,tc->chan);
}
```

K nastavení činitele plnění PWM slouží funkce *tc_setpwm*. Parametry *pwma*, *pwmb* udávají činitel plnění v % krát deset (maximum je tedy 1000). Parametr *top* představuje vrchol čítače. Funkce provádí přepočet ze zadané hodnoty na obsah registrů s ohledem na hodnotu vrcholu[4].

```
void tc_setpwm(struct tc_t *tc, uint16_t pwma, uint16_t pwmb,
    uint16_t top) {

  if (top<0xffff) *tc->rc = top;
    else *tc->rc = 0xffff;

  if (pwma<=1000)
    pwma = ((uint32_t)top*pwma+500)/1000;}
  else pwma = top;

  if (pwmb<=1000)
    pwmb = ((uint32_t)top*pwmb+500)/1000;
  else pwmb = top;

  *tc->ra = pwma;
  *tc->rb = pwmb;

}
```

5.1.4 Modul pmc

Účelem modulu *pmc* je zapínat a vypínat hodiny jednotlivým periferiím, což v mikroprocesoru zajišťuje jednotka PMC. Její registry jsou do paměti mapovány stejným způsobem, jako registry předcházejících periferií. Povolení hodin je řešeno ve funkci *pmc_enClk*, které se jako parametr předává identifikační číslo periferie, které je možné najít v [1]. Identifikační čísla periferií použitých v práci jsou definována v hlavičkovém

[4]Pro dosažení přirozeného zaokrouhlování při celočíselných výpočtech je zde, ale i jinde v programu, k dělenci přičtena polovina dělitele.

souboru *pmc.h*. Povolení dané periférie je zajištěno zápisem jedničky na patřičnou pozici registru PCER.

5.1.5 Modul server

Modul server zajišťuje komunikaci s programem v PC. Pro komunikaci jsou použity tzv. sokety a protokol se spolehlivým přenosem TCP. V modulu jsou definována dvě vlákna s RT prioritou a plánovaná v třídě SCHED_FIFO. Vlákno pro obsluhu příjmu dat (`server_tid`) má prioritu nastavenu na 20 a vlákno zajišťující periodické odesílání dat připojenému klientovi na 19. V případě, že se připojí klient a posílá data v příslušném tvaru *[PH:x PV:y]*, kde x a y jsou čísla v rozsahu 0-255 představující žádanou polohu v horizontální a vertikální rovině, jsou tato kopírována do struktury chráněné mutexem a je podle nich prováděna regulace. Pokud je připojen klient, běží zároveň vlákno odesílající údaje o skutečné poloze v obou rovinách, rovněž ve tvaru *[PH:x PV:y]*. Odesílání hodnot probíhá pravidelně s periodou 100 ms. Při odpojení klienta dojde k nastavení žádaných hodnot do poloviny rozsahu. Oznámení o připojení, či odpojení klienta, stejně jako varování při příjmu chybných hodnot jsou zaznamenávána do logu. Pro příjem a odesílání dat jsou použity funkce definované v hlavičkovém souboru *sys/socket.h* a to `recv` pro příjem a `send` pro odesílání.

5.1.6 Hlavní modul

V hlavním modulu *arm-code* jsou ve funkci `main` volány inicializační funkce všech předchozích modulů a především je spuštěno vlákno `thread_reg`, ve kterém probíhá obsluha AD převodníku, filtrace dat a samotná regulace. V inicializační fázi také proběhne parsování parametrů programu, které je možné zadat při spouštění z příkazové řádky (jejich přehled je možné získat spuštěním s parametrem `-h`), zamčení paměti, a registrace obsluhy signálu *SIGINT*, který je třeba obsloužit při „násilném“ ukončení programu (uvolnit alokovanou paměť, zastavit všechna vlákna, vypnout PWM atd.)

Činnosti vykonávané ve vláknu `thread_reg` představují podstatu programu a proto budou popsány podrobněji. Vlákno běží s periodou 10 ms a má nejvyšší možnou prioritu, tj. 99. Do logu jsou po 10 s vypisovány hodnoty minimální, průměrné a maximální latence za toto časové okno. V nekonečné smyčce ve vláknu probíhá: obsluha AD převodníku, přepočet do používaného rozsahu a filtrování, stavový automat zajišťující inicializaci a regulaci pohonů, volitelně výpis údajů pro identifikaci, výpočet statistik.

Obsluha AD převodníku byla původně realizována pomocí standardního linuxového ovladače *spidev*, který zpřístupňuje rozhraní SPI v uživatelském prostoru.

Komunikace však nefungovala spolehlivě, protože ovladač pracoval s výběrovým vodičem (chip-select) způsobem, který nebyl kompatibilní s protokolem AD převodníku. Bylo tedy přistoupeno k odstranění ovladače z jádra a vytvoření vlastního - obdobného ovladačům jiných periferií popsaných dříve. Tato varianta také nefungovala - rozhraní SPI bylo korektně nastaveno, ale ve stavovém registru byla kombinace bitů, která by podle datasheetu nikdy neměla nastat a nedošlo k odeslání ani příjmu dat. dostatečně spolehlivou a rychlou se ukázala varianta komunikace přes SPI pomocí ovládání jednotlivých GPIO (tzv. bit-bang). Hodiny generované tímto způsobem dosahují cca 1,3 MHz, což je nad spodní hranici požadovanou obvodem AD převodníku (0,8 MHz, optimální je však minimálně 3,2 MHz). Funkce realizující SPI komunikaci *spi_trW* očekává jako parametr 16 bitovou proměnou jejíž hodnota bude odeslána a vrací hodnotu přečtenou z AD převodníku. Ukázka kódu odesílajícího a přijímajícího jednotlivé bity:

```
gpio_setOutSt(SPI_CS,0);
for (it=0; it<16; it++) {
  gpio_setOutSt(SPI_SCK,0);
  gpio_setOutSt(SPI_MOSI,(uint8_t)((send>>(15-it)) & 0x1));
  asm("mov r0,r0"); // short delay
  asm("mov r0,r0");
  tmp = gpio_readInSt(SPI_MISO);
  read |= tmp<<(15-it);
  gpio_setOutSt(SPI_SCK,1);
}
gpio_setOutSt(SPI_CS,1);
```

Jako nezbytné se ukázalo zařazení krátkého zpoždění tvořeného vložením dvou instrukcí v assembleru (ekvivalentních instrukci NOP, která však u ARMu není) za sestupnou hranu hodin. Ani přesto však není komunikace bezchybná - občas se stane, že je z AD převodníku přečtena nula, nebo hodnota větší, než 4096, což je maximální přípustná hodnota pro 12 bitový převodník. V takovém případě se program pokusí přenos maximálně třikrát zopakovat. Přenos je opakován také v případě, že je pro daný potenciometr změřena hodnota větší než maximální získaná při inicializaci, nebo menší než minimální.

Hodnoty odpovídající proudu oběma motorky jsou přepočítány na mA následující rovnicí:

$$I = \frac{ADC \cdot 3,3 \cdot 2 \cdot 1000}{4096 \cdot 15,71} \approx \frac{ADC \cdot 412 + 2010}{4021}[mA]$$

Protože přechodový děj při přivedení napětí na motorek je rychlý (vzhledem k použité periodě vzorkování), ukázalo se jako vhodné hodnoty proudu nijak nefiltrovat.

Naopak filtrování se zřejmě z důvodu nekvalitních potenciometrů ukázalo jako nutné pro měření polohy, kde je použit filtr typu klouzavý průměr délky 30. Po filtrování proběhne ještě přepočet do rozsahu 0-255, který je používán v regulaci a komunikaci s klientem:

$$POS = \frac{255 \cdot (AKT - MIN) + \frac{MAX-MIN}{2}}{MAX - MIN}[-]$$

Pro inicializaci měření polohy (zjištění hodnot MIN, MAX) a regulaci byl vytvořen jednoduchý stavový automat. Ten po spuštění programu zajistí zjištění minimálních a maximálních hodnot odpovídajících napětí potenciometrů měřících polohu v krajních polohách. Krajní poloha je detekována mikrospínačem (který je však vždy pouze na jedné straně), nebo zvýšením hodnoty proudu nad nastavenou mez. Pokud je rozdíl hodnot naměřených v krajních polohách příliš malý, proces se opakuje. Po úspěšném změření rozsahu v obou osách je (pokud není požadován výstup pro identifikaci) zahájena regulace. Algoritmus použitého proporcionálního regulátoru ukazuje vývojový diagram 5.2. Pokud je absolutní odchylka větší než nastavená mez (ERR_TH), motorek se otáčí plnou rychlostí. Při menší hodnotě odchylky dochází k jejímu přepočtu do rozsahu daném minimální hodnotou PWM kdy se daný motorek již otáčí a horní hranicí 1000. Takto přepočítaná odchylka je použita jako akční zásah:

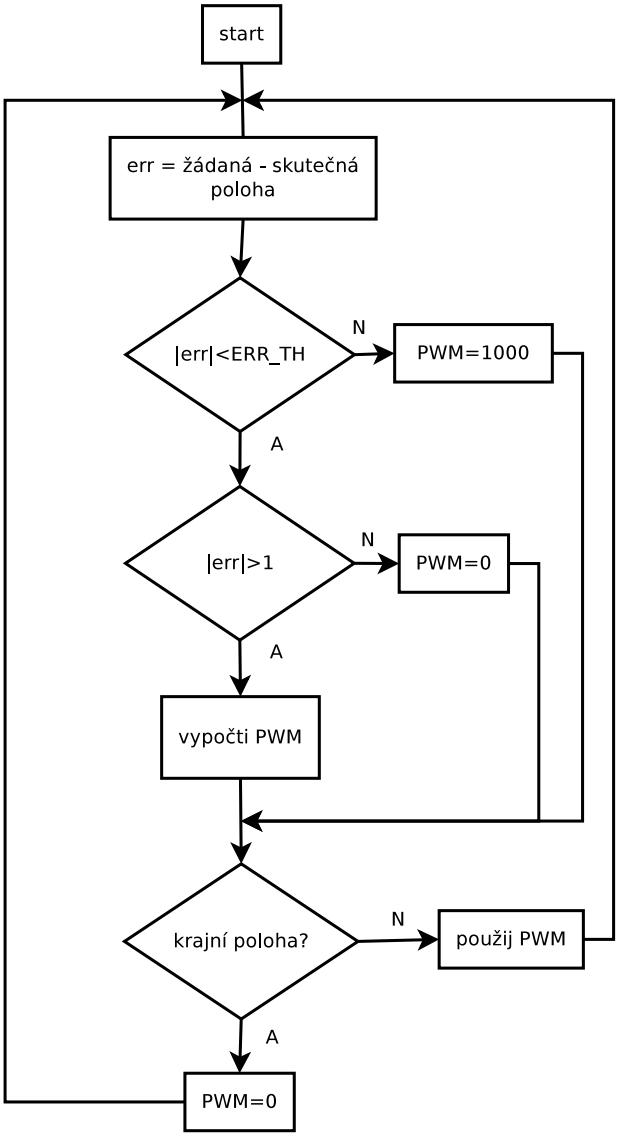
$$PWM = \frac{(1000 - MIN\_PWM) \cdot (\text{žádaná} - \text{aktuální})}{ERR\_TH} + MIN\_PWM$$

Minimální hodnoty PWM pro každý z motorků jsou uloženy v odpovídající struktuře, stejně jako informace o aktuální a žádané poloze atd.

V případě spuštění programu s parametrem `-i` zůstane po inicializaci regulátoru jeden z motorků (který lze změnit ve zdrojovém kódu) v krajní poloze a následně je změřena a vypsána odezva proudu a polohy na vygenerovanou pseudonáhodnou posloupnost, přepočtenou na binární posloupnost v rozsahu 0-1000. Posloupnost je tvořena voláním funkce `rand` a jednoduchým výpočtem:

```
pwm = 1000*(rand()%2);
```

Při opakovaném spuštění programu je generována stejná posloupnost. Pokud je požadována jiná (například pro ověření správnosti vytvořeného modelu), je nutné v programu voláním funkce *srand* nastavit jiný počátek (seed) náhodné posloupnosti. Generování stejné posloupnosti vícenásobném spuštění programu má tu výhodu, že je možné získat více odezev a z nich spočítat průměr, pro potlačení vlivu šumu. Výpis tvořený 512 vzorky je ve formátu CSV, který může být snadno importován např. do programu Matlab, pro provedení experimentální identifikace.



Obr. 5.2: Vývojový diagram algoritmu regulátoru

5.2 MJPG-streamer

Pro streamování obrazu z webkamery je použit program MJPG-streamer, dostupný na SourceForge.net, který vytvořil Tom Stoeveken a je šířen pod licencí GPL. Kromě streamování přes TCP, nebo UDP, ukládání do souboru, vytváří program i jednoduchý webový server, díky němuž je možné obraz sledovat přes internetový prohlížeč. Pracuje s kamerami podporujícími standard UVC, případně podporovanými ovladačem GSPCA. V případě použití kamery standardu UVC poskytující přímo komprimovaný JPEG stream je streamování velmi nenáročné na zdroje mikroprocesoru. V případě, že kamera posílá nekomprimovaný stream, provádí MJPG-streamer jeho kompresi pomocí knihovny libjpeg. Z důvodu chyby ve firmware použité kamery (Genius

Facecam 1000), kdy kamera ovladači UVC zařízení hlásí vyšší požadavky na šířku pásma než by bylo nutné, je paradoxně možné provozovat kameru v režimu s nekomprimovaným proudem dat a nejnižším rozlišení (160x120), kdy je skutečná šířka pásma vyšší než u komprimovaného streamu s vyšším rozlišením. Ovladač UVC (jaderný modul uvcvideo) je možné u tzv. RAW streamu pomocí parametru `quirks` nastavit tak, že požadavek kamery přepočítá. U JPEG streamu to bohužel ve verzi ovladače použité v práci možné není. Komprimace a streamování videa o rozlišení 160x120 pixelů představuje pro mikroprocesor použitého vývojového kitu zátěž cca 20%. Pro spuštění MJPG-streamer byl vytvořen skript v adresáři */etc/init.d*, který při startu systému spustí program s následujícími parametry:

```
LD_LIBRARY_PATH=/usr/local/lib /usr/local/bin/mjpg_streamer -
    b
-i "input_uvc.so -y -r 160x120 -f 30 -d /dev/video0"
-o "output_http.so -w /usr/local/www -p 8090"
```

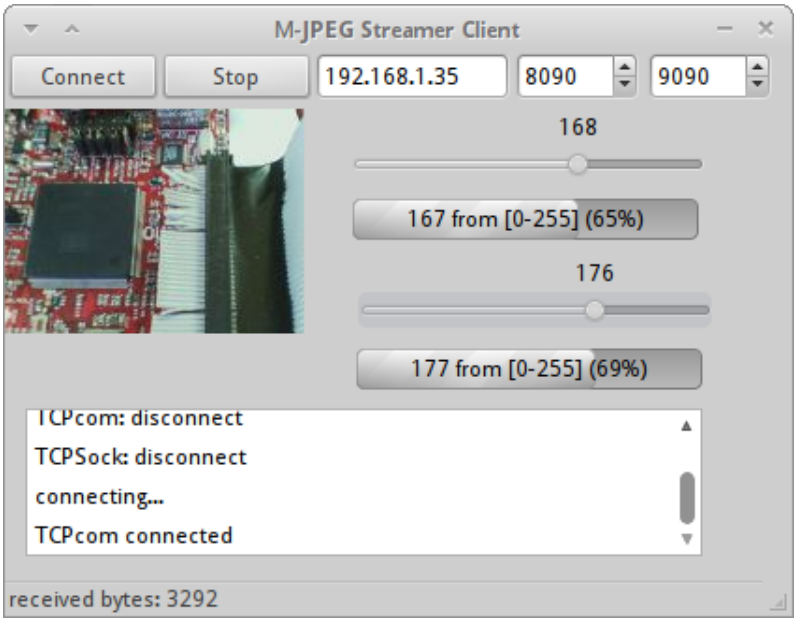
Program MJPG-streamer využívá ke své funkci knihovnu Video For Linux 2, označovanou často jen jako V4L2. Knihovna tvoří abstraktní vrstvu nad multimediálními zařízeními (kamery, rádia, televizní tunery apod). Jednotné API poskytuje modul `videodev`. O funkci konkrétního zařízení se stará další modul (např. `uvcvideo`). V uživatelském prostoru je zařízení přístupné jako `/dev/videoX`. Bližší popis funkcí knihovny je např. v [15]

5.3 Program pro PC

Program pro ovládání stojanu z PC je vytvořen pomocí IDE Lazarus, využívajícího kompilátor Free Pascal. Vzorem pro toto prostředí je IDE Delphi od firmy Borland. Výhodou použití Lazaru je velmi jednoduchý vývoj GUI a také že při kompilaci vzniká binární soubor pro OS Linux a Windows zároveň. Jako základ programu byla použita aplikace šířená společně s balíkem MJPG-streamer zobrazující stream videa, která byla rozšířena o možnost nastavování žádané polohy stojanu a sledování skutečné.

Pro komunikaci protokolem TCP je použita komponenta TLTCP z knihovny lNET. Knihovna lNET je multiplatformní, objektově orientovaná. Jeden soket obsluhovaný komponentou TLTCP je použit pro zachytávání streamu obrazu, druhý pro přenos povelů a informací o skutečné poloze stojanu. Inicializace soketu se provede zavoláním metody *connect*, které se jako parametry předají IP adresa protějšku a číslo portu. V případě úspěchu je vyvolána událost *OnConnect*. Odeslání zprávy přes socket se velice snadno provede zavoláním metody *SendMessage*, které se jako parametr předá

řetězec s obsahem zprávy. Při přijmu dat je vyvolána událost *OnReceive* a v příslušné proceduře je možné metodou *Get* načíst přijatá data do bufferu zadané délky.



Obr. 5.3: Program pro PC

Stream poskytovaný MJPG-streamerem je proudem jednotlivých JPG obrazů. Při otevření soketu zajišťujícího obsluhu tohoto streamu je server požádán o stream a následně probíhá v proudu dat detekce speciálních hodnot označujících začátek JPG obrazu (SOI[5]: 0xFFD8) a konec obrazu (EOI[6]: 0xFFD9). Při přenosu dat pomocí soketu není zajištěno přenesení určitého množství dat naráz, proto jsou příchozí data „akumulována“ do bufferu (instance třídy *TMemoryStream*), ve kterém následně probíhá detekce přijatého obrazu. Pokud je skutečně nalezen kompletní obraz, je tento zobrazen pomocí komponenty typu *TImage*.

Žádaná poloha stojanu je nastavována dvojicí trackbarů, kde horní slouží k nastavení horizontální polohy a spodní k nastavení vertikální polohy. Skutečná poloha je zobrazována dvojicí progressbarů. Poloha je udávána jako osmibitová hodnota, tudíž je možné ji nastavovat v rozsahu 0-255. Jak často jsou přijímána data o skutečné poloze je dáno nastavením serveru na vývojové desce. Četnost odesílání hodnot o žádané poloze je řízena timerem.

[5]Start Of Image - začátek JPEG obrazu

[6]End Of Image - konec JPEG obrazu

6 ZÁVĚR

Pro realizaci ovládání polohovatelného stojanu byl vybrán výkonný mikroprocesor ARM, který umožňuje běh standardních operačních systémů, čímž je ulehčen vývoj uživatelských programů. V rámci diplomové práce byly navrženy dvě varianty elektroniky pro ovládání stojanu zahrnující vybraný mikroprocesor AT91SAM9260 od firmy Atmel. První variantu spočívající v návrhu jediné desky obsahující vše potřebné se bohužel kvůli technologickým obtížím a chybám v návrhu nepodařilo realizovat. Proto byla navržena a oživena druhá varianta spočívající ve využití vývojového kitu Olimex L9260 a vlastní desky s potřebnou elektronikou k buzení motorků, snímání proudu a polohy. Pro vývojový kit založený na mikroprocesoru ARM byl vytvořen pomocí balíku Buildroot linuxový operační systém, jehož jádro bylo upraveno pro získání lepší real-time odezvy. Vytvořený systém startuje výrazně rychleji než systém dodaný výrobcem kitu a používá modernější rozhraní pro komunikaci programů a operačního systému (ABI). Dále byl napsán program zajišťující ovládání stojanu a komunikaci se softwarem v PC. Díky nelinearitám v soustavě tvořené budičem, motorkem a snímacím potenciometrem se nepodařilo vytvořit uspokojivý matematický model řízené soustavy. Řízení polohy bylo realizováno pouze za pomoci polohové zpětné vazby. Nad rámec zadání byla přidána možnost připojení USB webkamery a streamování obrazu z ní. Pro nastavování žádané polohy stojanu, sledování skutečné a zobrazení video streamu v PC byl vytvořen multiplatformní program pro operační systémy Linux a Windows.

Z možných vylepšení hardwaru i softwaru se nabízí například výměna stejnosměrných motorků za krokové, které by umožnily snazší a přesnější řízení natočení. Také je tu možnost připojení kamery pomocí speciálního rozhraní, které je pro tento účel v mikroprocesoru zabudováno. Pokud by bylo požadováno přenášení vyššího rozlišení než 160x120 pixelů z připojené web kamery, bylo by nutné aktualizovat nebo upravit použitý ovladač v jádře OS. Dále by bylo možné do softwaru doplnit podporu pro použití krokových motorků, na jejichž ovládání je elektronika připravena. Také komunikace mezi mikroprocesorem a AD převodníkem by byla spolehlivější při použití hardwarového SPI, které se v práci nepodařilo využít. V neposlední řadě se u podobného zařízení přímo nabízí možnost rozpoznávání pohybu v obraze, ukládání snímků, případně záznamu zvuku na paměťovou kartu apod. Za úvahu by také stála změna použité knihovny jazyka C uClibc, která neimplementuje některé důležité funkce, za standardní Glibc. Zajimavé by mohlo být i porovnání rychlosti obou knihoven.

Pro mě osobně přínos práce spočívá zejména v hlubším pochopení funkce operačního systému Linux, úpravách tohoto systému pro embedded zařízení, programování real-time aplikací a vytváření ovladačů periferií mikroprocesoru.

LITERATURA

[1] Atmel Corporation *AT91 ARM Thumb Microcontrollers - AT91SAM9260* [online]. 2008, [cit. 19. 10. 2010]. Dostupné z URL: <www.atmel.com/at91sam>.

[2] OLIMEX Ltd *SAM9-L9260 development board Users Manual* [online]. 2008, [cit. 4. 4. 2011]. Dostupné z URL: <www.olimex.com/dev/sam9-L9260.html>.

[3] PETAZZONI, Thomas *Building embedded Linux systems with Buildroot. Embedded Linux Training* [online]. 2009, [cit. 7. 4. 2011]. Dostupné z URL: <free-electrons.com/docs/buildroot>.

[4] CALDERON, Andres; CASTILLO, Nelson *Why ARM's EABI matters* [online]. 2007, [cit. 7. 4. 2011]. Dostupné z URL: <www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Why-ARMs-EABI-matters/>.

[5] *Debian Wiki* [online]. 2008, 2010-03-02 [cit. 2011-04-07] ArmEabiPort. Dostupné z URL: <wiki.debian.org/ArmEabiPort>.

[6] *PAL.* In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2001-11-04, aktualizováno 2011-04-07 [cit. 2011-04-09]. Dostupné z WWW: <en.wikipedia.org/wiki/PAL>.

[7] *Composite video.* In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2002-11-22, aktualizováno 2011-04-05 [cit. 2011-04-09] Dostupné z WWW: <en.wikipedia.org/wiki/Composite_video>.

[8] *Maxim* [online]. 2002-09-22 [cit. 2011-04-09]. Understanding Analog Video Signals. Dostupné z WWW: <www.maxim-ic.com/app-notes/index.mvp/id/1184>.

[9] *Linux (jádro).* In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2005-10-30, aktualizováno 2011-03-24 [cit. 2011-04-18]. Dostupné z WWW: <cs.wikipedia.org/wiki/Linux_(jádro)>.

[10] KOCH, Hans-Jürgen; TSIRKIN, Michael S. *The Linux Kernel Archives* [online]. 2006-12-11, 2009-07-16 [cit. 2011-04-18]. The Userspace I/O HOWTO. Dostupné z WWW: <www.kernel.org/doc/htmldocs/uio-howto.html>.

[11] CORBET, Jonathan; RUBINI, Alessandro; KROAH-HARTMAN, Greg. *Linux Device Drivers. third edition.* [online]. O'Reilly Media, 2005. 640 s. Dostupné z WWW: <lwn.net/Kernel/LDD3/>. ISBN 978-0-596-00590-0.

[12] THAYER, Doug; MILLER, Keith. *Four UNIX Programs in Four UNIX Collections : Seeking Consistency in an Open Source Icon. In Midwest Instruction and Computing Symposium Proceedings.* [online]. Morris : University of Minnesota, 2004. [cit. 2011-05-07] Dostupné z WWW: <www.micsymposium.org/mics_2004/proceedings.html>.

[13] *Real-Time Linux Wiki* [online]. 2006 [cit. 2011-05-08] Dostupné z WWW: <https://rt.wiki.kernel.org>.

[14] CHANTEPERDRIX, Gilles; COCHRAN, Richard. *The ARM Fast Context Switch Extension for Linux* [online]. 2009 [cit. 2011-05-09]. Dostupné z WWW: <http://sisyphus.hd.free.fr/ gilles/pub/fcse/>.

[15] VÍT, Jiří. *Možnosti použití operačního systému GNU/Linux v bezpečnostní kameře EYE-02.* Praha, 2009. 39 s. Diplomová práce. České vysoké učení technické v Praze. Dostupné z WWW: <http://support.dce.felk.cvut.cz/mediawiki/images/d/df/Dp_2009_vit_jiri.pdf>.

[16] SKALICKÝ, Jiří. *Elektrické regulované pohony.* Brno : [s.n.], 2007. 123 s.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ABI Application binary interface - rozhraní pro komunikaci mezi aplikačním programem a OS ... 32

BCR Block Control Register ... 16

BMR Block Mode Register ... 16

CCR Channel Control Register ... 17

CMR Channel Mode Register ... 17

CVBS Composite Video, Blanking, and Sync ... 26

DPS deska plošných spojů ... 19

EMI elektromagnetické interference ... 19

EOI End Of Image - konec JPEG obrazu ... 60

ESR Equivalent series resistance ... 24

FCSE Fast Context Switch Extension ... 35

GPIO General Purpose Input/Output ... 8

ISI Image Sensor Interface - rozhraní pro připojení obrazového snímače ... 13

JTAG Joint Test Action Group - rozhraní pro testování a programování integrovaných obvodů ... 19

LPJ loops per jiffy ... 30

MIPS Million instructions per second ... 14

MMU Memory Management Unit ... 14

NFS Network File System - protokol pro sdílení souborů přes síť ... 30

OS operační systém ... 14

OSR Output Status Register ... 50

PAL Phase Alternating Line ... 10

PHY PHYsical Interface - obvod realizující fyzickou vrstvu Ethernetu ... 19

PIO Parallel Input/Output Controller ... 15

PMC Power Management Controller ... 15

POE Power Over Ethernet - napájení přes Ethernet (48 V, 400 mA) ... 10

PSR Peripheral Status Register ... 50

PTZ Pan Tilt Zoom - vyklápění, natáčení, zoom ... 10

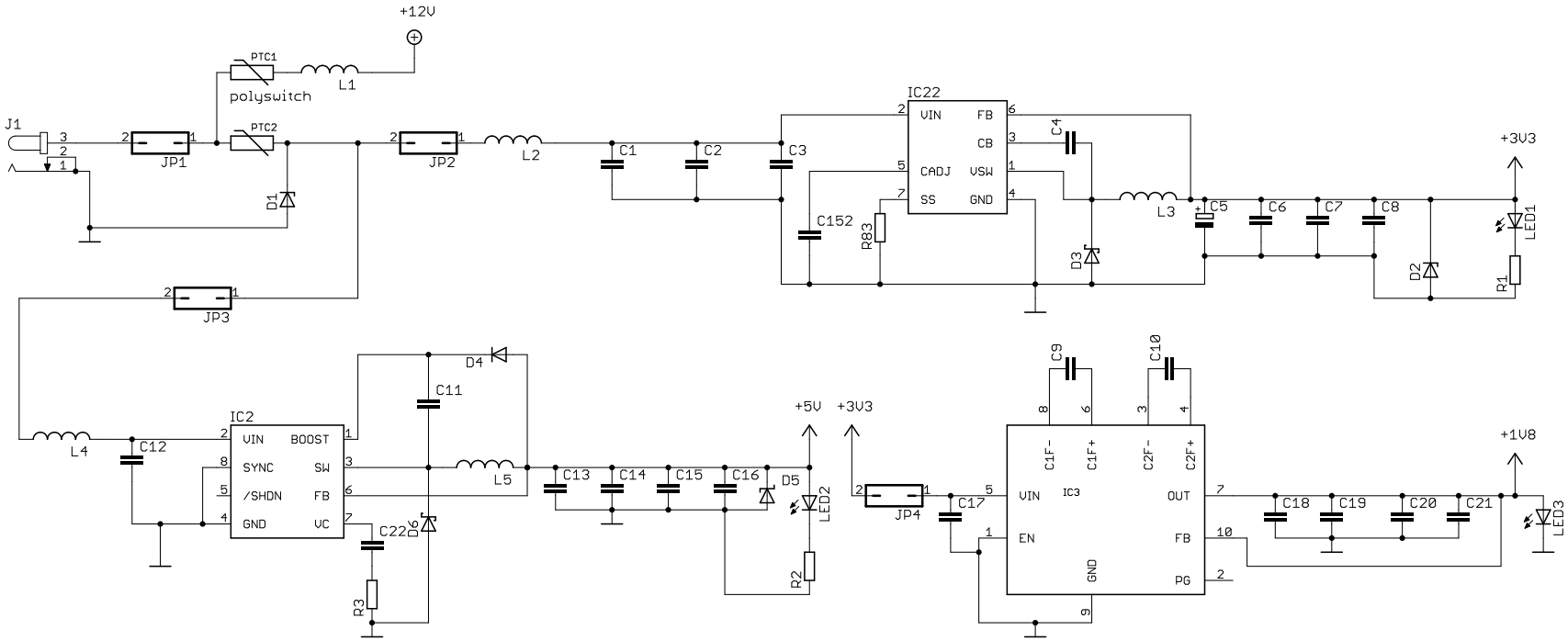
PWM pulse width modulation 15
RMII Reduced Media Independent Interface - propojení Ethernet MAC a PHY 19
RT real-time 28
SOI Start Of Image - začátek JPEG obrazu 60
SPI Serial Peripheral Interface 21
SRAM Static random access memory 19
TC Timer Counter 8
TVS Transient voltage suppressor 24

SEZNAM PŘÍLOH

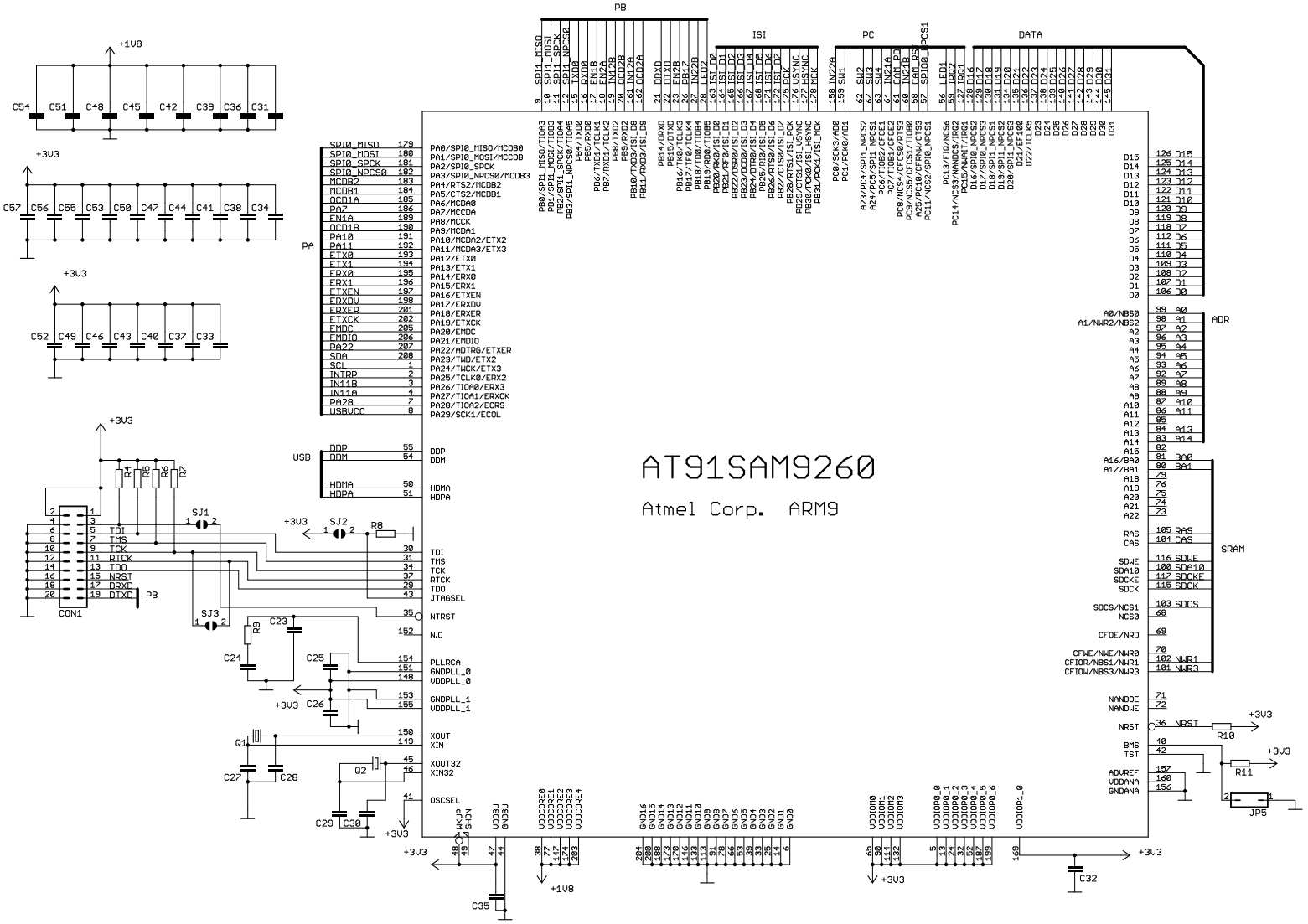
A OBSAH PŘILOŽENÉHO DVD

- text práce v elektronické podobě
- schémata a návrhy DPS ve formátu Eagle, pdf
- obrazy souborového systému a linuxového jádra
- nakonfigurovaný balík Buildroot, včetně prostředí pro křížovou kompilaci
- skripty používané pro přenos obrazů pamětí do kitu
- zdrojové kódy programů

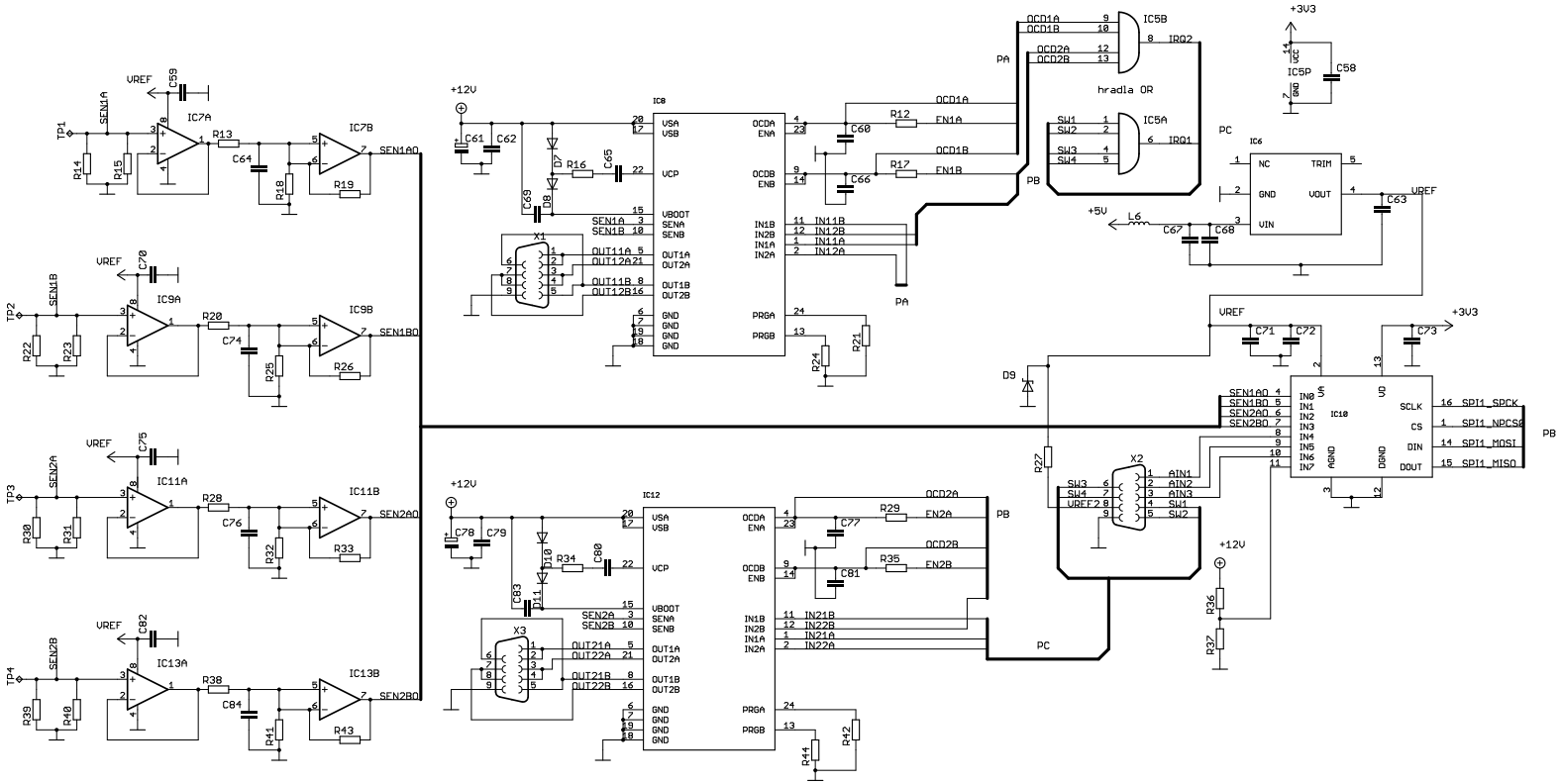
B SCHÉMATA, OSAZOVACÍ PLÁNKY, SEZNAMY SOUČÁSTEK



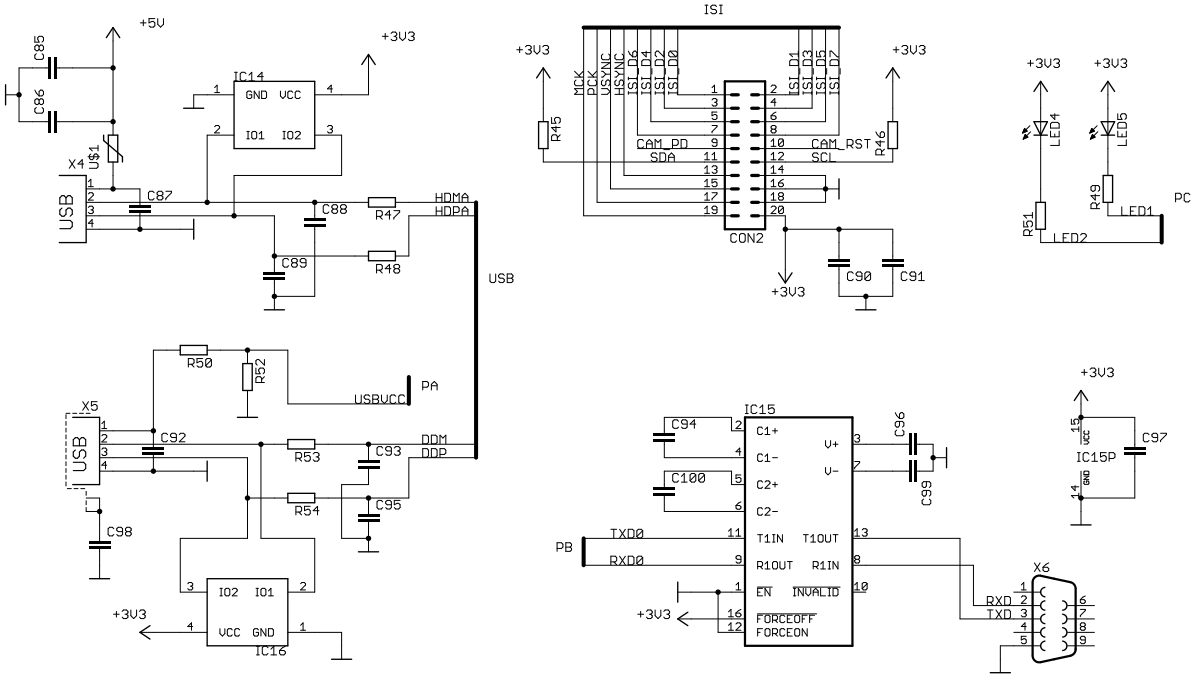
Obr. B.1: První varianta HW: schéma napájení desky



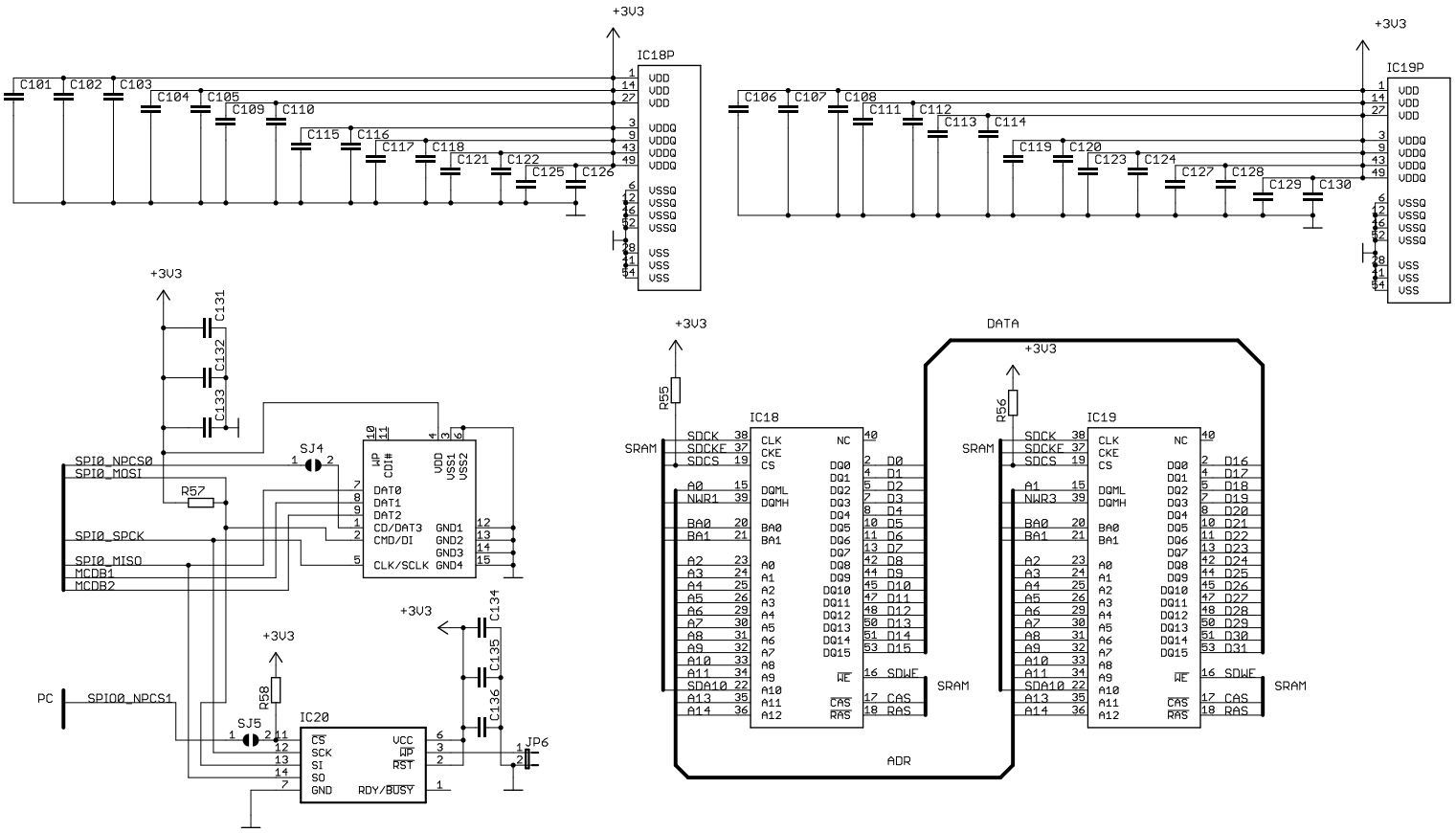
Obr. B.2: První varianta HW: zapojení mikroprocesoru AT91SAM9260



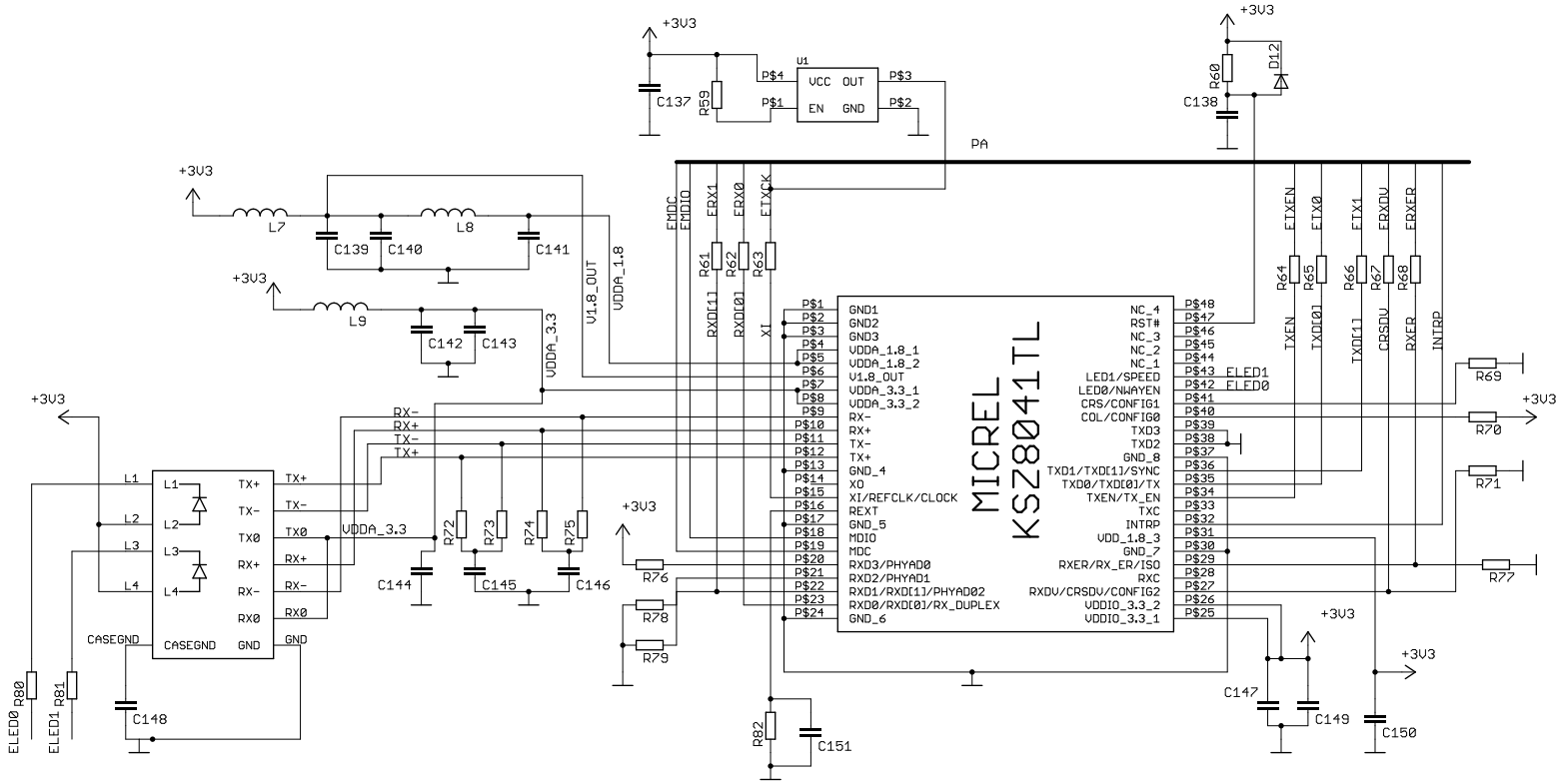
Obr. B.3: První varianta HW: budiče motorů, měření proudu, AD převodník



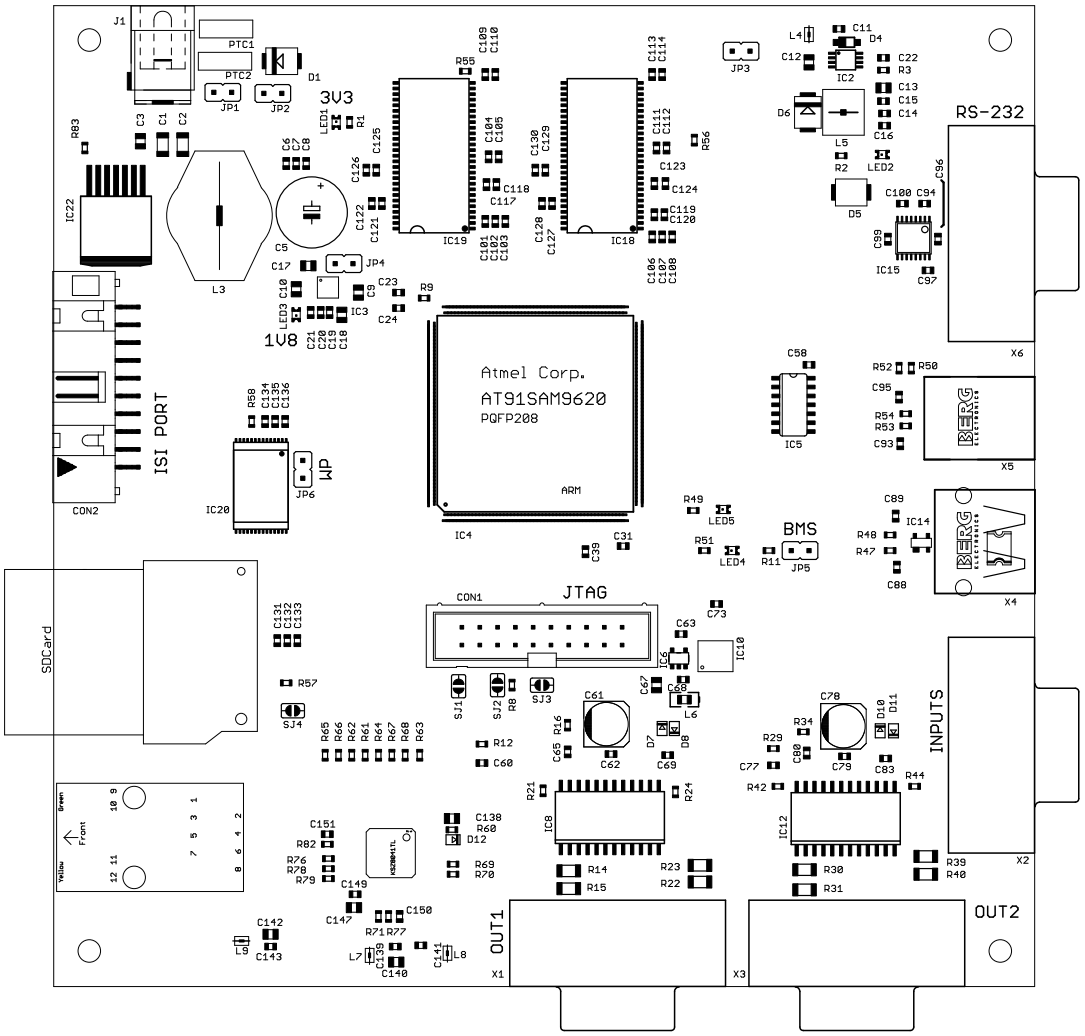
Obr. B.4: První varianta HW: komunikační porty



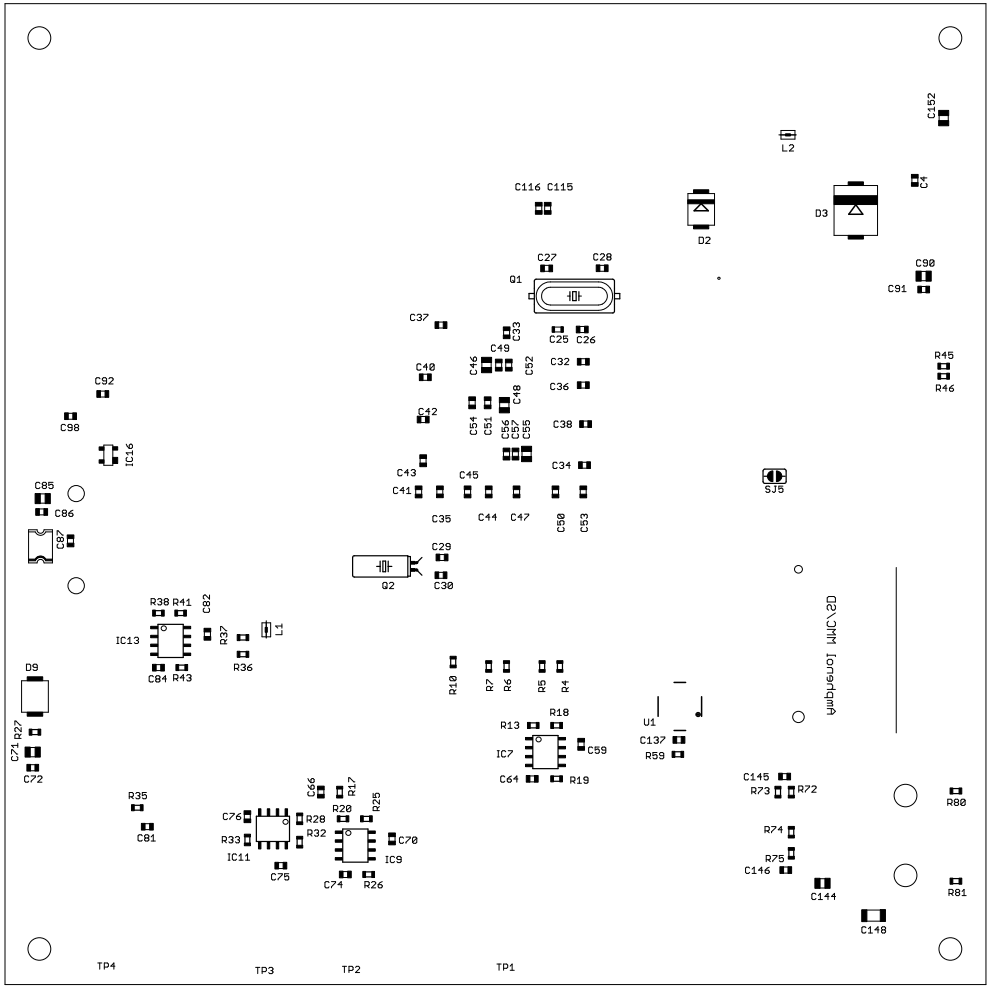
Obr. B.5: První varianta HW: SDRAM, dataflash, SD karta



Obr. B.6: První varianta HW: zapojení Ethernet PHY KSZ8041TL



Obr. B.7: První varianta HW: rozložení součástek na horní straně desky



Obr. B.8: První varianta HW: rozložení součástek na spodní straně desky

Mn.	Součástka	Pouzdro	Označení	Poznámka
71	100N/16V	0603	C6, C11, C14, C19, C25, C26, C31, C32, C33, C34, C35, C36, C37, C38, C39, C40, C41, C42, C43, C44, C45, C47, C50, C53, C58, C59, C63, C64, C67, C69, C71, C72, C73, C74, C75, C81, C83, C85, C90, C91, C92, C94, C96, C97, C98, C99, C100, C103, C105, C108, C110, C112, C114, C116, C118, C120, C122, C124, C126, C128, C130, C133, C136, C137, C139, C141, C143, C145, C146, C149, C150	hmota X7R
27	10N/25V	0603	C4, C7, C15, C20, C24, C49, C51, C56, C65, C79, C86, C102, C104, C107, C109, C111, C113, C115, C117, C119, C121, C123, C125, C127, C129, C132, C135	hmota COG
13	1K	0603	R8, R9, R10, R11, R68, R70, R76, R77, R78, R19, R26, R32, R42	přesnost 1%
11	1N/50V	0603	C8, C16, C21, C23, C52, C54, C57, C101, C106, C131, C134	hmota COG
8	100K	0603	R4, R5, R6, R7, R12, R17, R28, R34	
8	10M/6,3	0805	C13, C18, C46, C48, C55, C84, C89, C144	hmota X7R

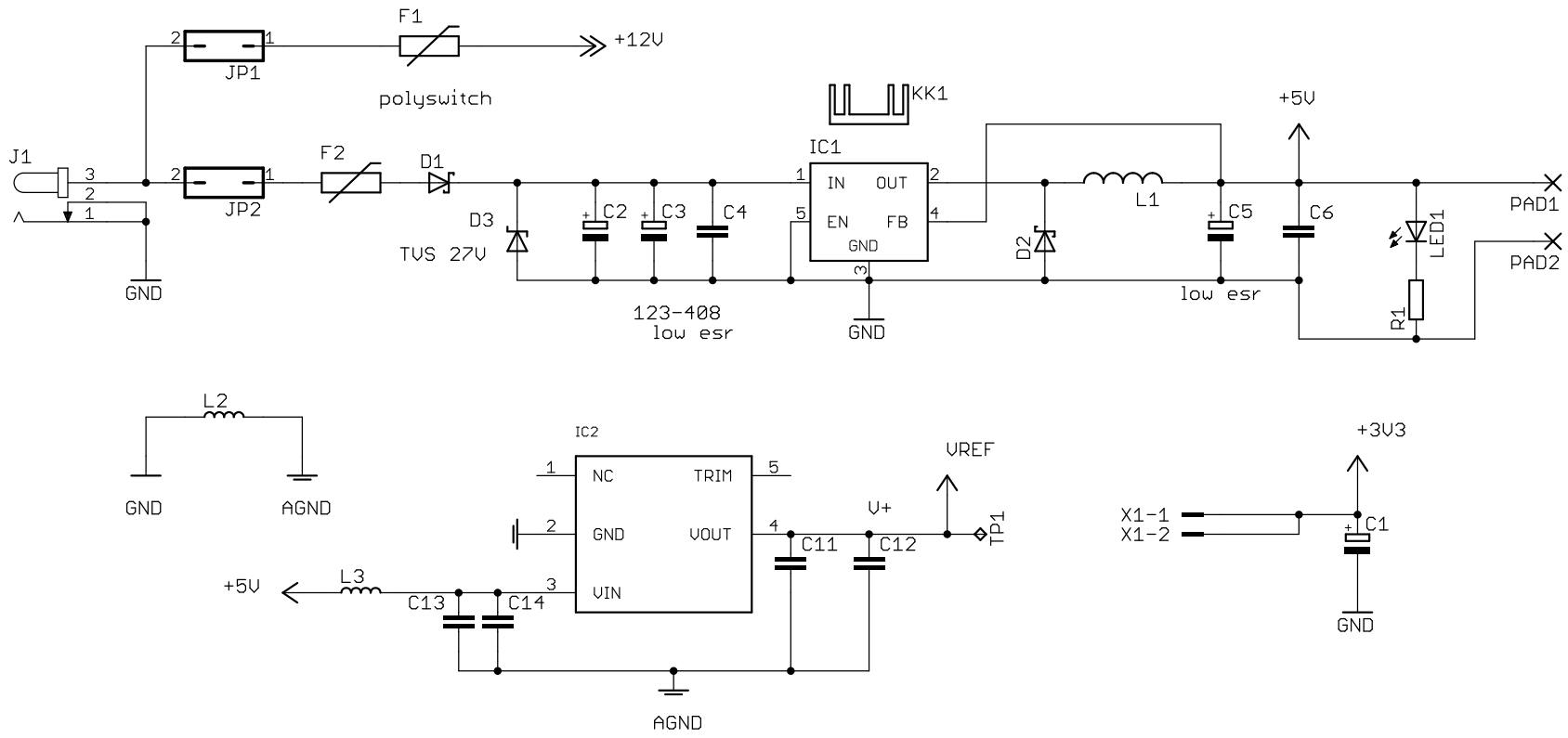
8	33R	0603	R60, R61, R62, R63, R64, R65, R66	
8	R20FI	1206	R14, R15, R22, R23, R29, R30, R38, R39	proudový bočník
6	742792012	0805	L1, L2, L4, L6, L7, L8	feritová perlička
6	10K	0603	R21, R24, R41, R43, R56, R58	
5	1N4148W	SOD323-W	D5 D6 D7 D8 D9	
5	4K7	0603	R3, R44, R45, R69, R75	
5	JP1Q		JP5, JP1, JP2, JP3, JP4	jumper
5	211UTT86K	0805	LED1, LED2, LED3, LED4, LED5	červené LED
4	750R	0603	R13, R20, R27, R37	
4	10P/50V	0603	C27, C28, C29, C30	hmota COG
4	1K2	0603	R18, R25, R31, R40	přesnost 1%
4	22M/6,3V	0805	C138, C140, C142, C147	hmota X5R
4	39R	0603	R46, R47, R52, R53	
4	49R9	0603	R71, R72, R73, R74	
4	5N6	0603	C60, C66, C76, C80	
4	DB9		X2, X3, X4, X7	konektor D SUB
4	TS912ID	SO08	IC6, IC8, IC9, IC11	operační zesilovač
3	680R	0603	R1, R48, R50	
3	1M/10V	0805	C9, C10, C70	hmota X7R
3	470K	0603	R54, R55, R57	
2	22M/25V	1206	C1, C2	X5R
2	100M/25V	6,3X5,8	C61, C77	elektrolytický kond.

2	100N/25V	0603	C62, C78	hmota X7R
2	100R	0603	R16, R33	
2	15P/50V	0603	C93, C95	hmota COG
2	220N	0603	C68, C82	
2	220R	0603	R79, R80	
2	2M2/25V	0805	C17, C12	hmtoa X5R
2	47P	0603	C87, C88	
2	6K49	0603	R59, R81	
2	L6206	SO24W	IC7, IC10	dvojitý H-můstek
2	MT48LC16M16A2P	TSOP54	IC13, IC1	paměť DRAM
2	PRTR5V0U2X	SOT143	U4, $U6$	ESD dioda
2	RGEF500		U1, $U2$	polyswitch
1	AT91SAM9260B	PQFP208	AT91SAM1	mikroprocesor
1	XO91050UITA	SMD7X5	U1	50 MHz oscilátor
1	744052005	TPC_L/LH	L5	akumulační tlumivka
1	100P	0603	C151	
1	10BQ030PBF	SMB	D4	schottkyho dioda
1	10N/500V	1206	C148	hmota X7R
1	15K	0603	R49	
1	18.432 MHz	HC49UP	Q1	krystal
1	1K1	0603	R36	
1	1K5	0603	R2	
1	1N5	0603	C22	

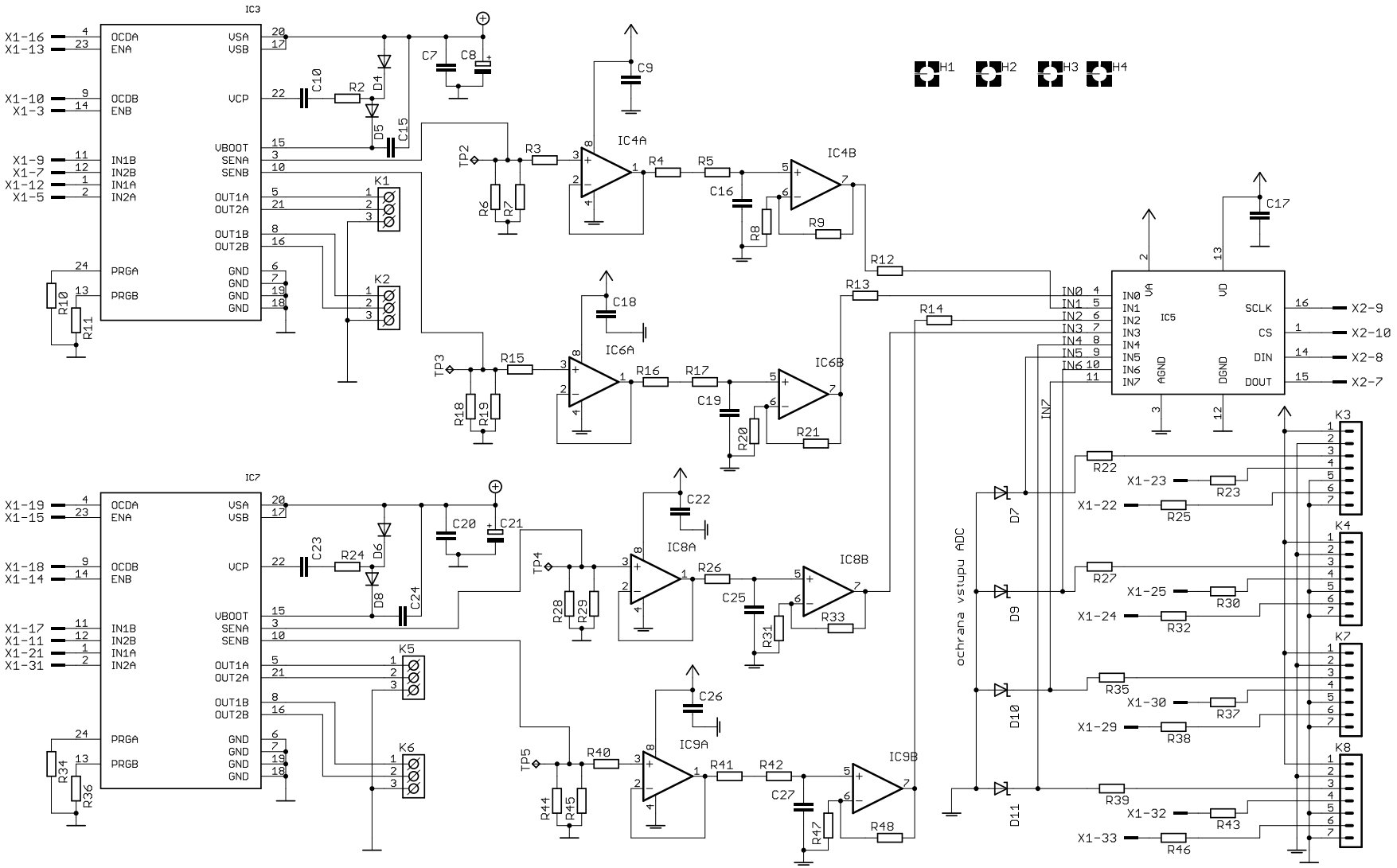
1	22K	0603	R51	
1	30BQ060PBF	SMC	D2	schottkyho dioda
1	32.768 kHz		Q2	krystal
1	3K9	0603	R35	
1	470N/25V	0805	C3	hmota X7R
1	74HC21D	SO14	IC4	hradlo OR
1	ADC128S052	TSSOP16	U$3	AD převodník
1	ADR366	SOT23-5	IC5	napěťová reference
1	AT45DB642D	TSOP28-1	U	sériová flash paměť
1	DL50-33	WE-PD3_X	L3	akumulační tlumivka
1	E470M/16V	E5-10,5	C5	elektrolyt. kond., nízké ESR
1	KSZ8041TL	TQFN	U$8	Ethernet PHY
1	LM2670S	TO263-7	IC1	DC-DC měnič 3,3 V
1	LT1767-5.0	MSOP8	IC2	DC-DC měnič 5 V
1	MAX3221CUE	TSSOP16	IC12	
1	MBR0520LT	SOD123	D3	schottkyho dioda
1	MINISMDC075F		U$5	polyswitch
1	MLW20	MLW20G	CON1	
1	MLW20_90	MLW20A	CON2	
1	PN87520		X5	USB konektor A
1	PN61729-S		X6	USB konektor B
1	SPC21364		J1	napájecí konektor
1	10915W0T1		U$7	slot na SD kartu

1	SI-60062-F		U$9	Eth. MagJack konektor
1	SM6T15A	SMB	D1	TVS - transil
1	TPS60502DGS	DGS	IC3	DC-DC měnič 1,8 V

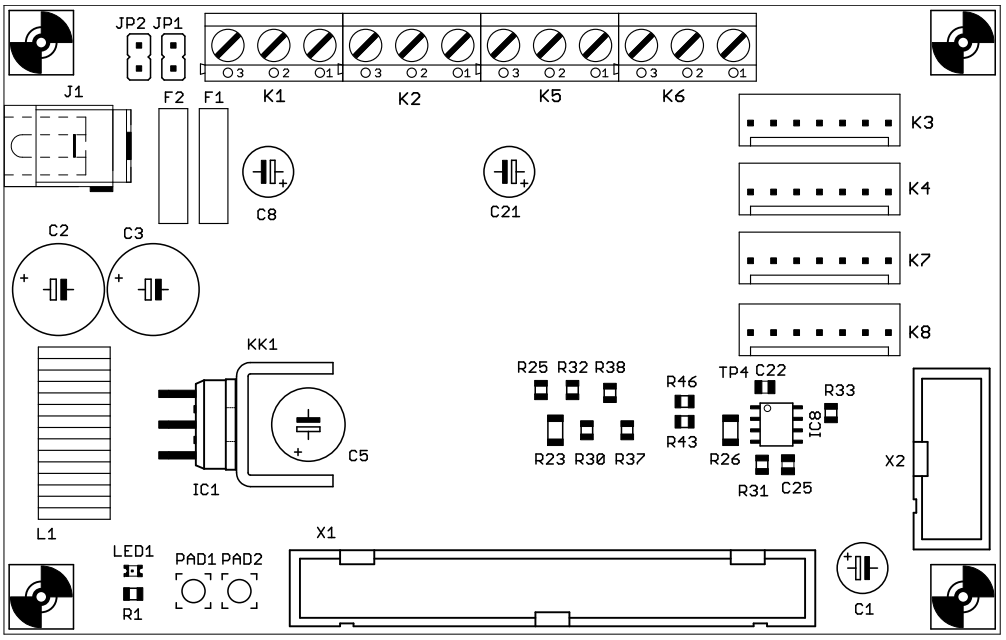
Tab. B.1: První varianta HW: seznam součástek



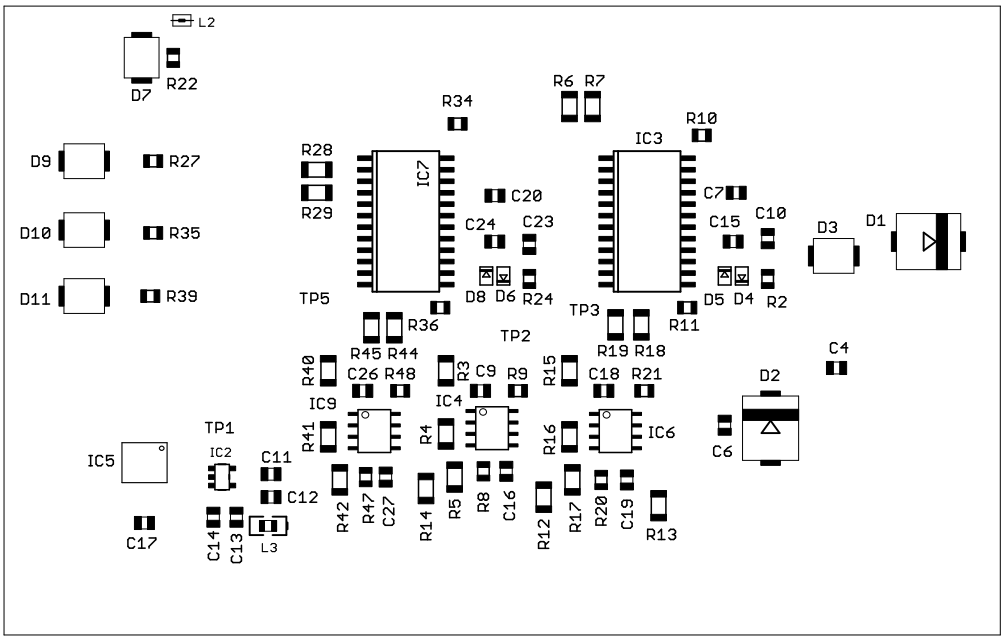
Obr. B.9: Druhá varianta HW: schéma zapojení - napájení.



Obr. B.10: Druhá varianta HW: schéma zapojení - budiče motorků, AD převodník.



Obr. B.11: Druhá varianta HW: rozložení součástek na horní straně desky



Obr. B.12: Druhá varianta HW: rozložení součástek na spodní straně desky

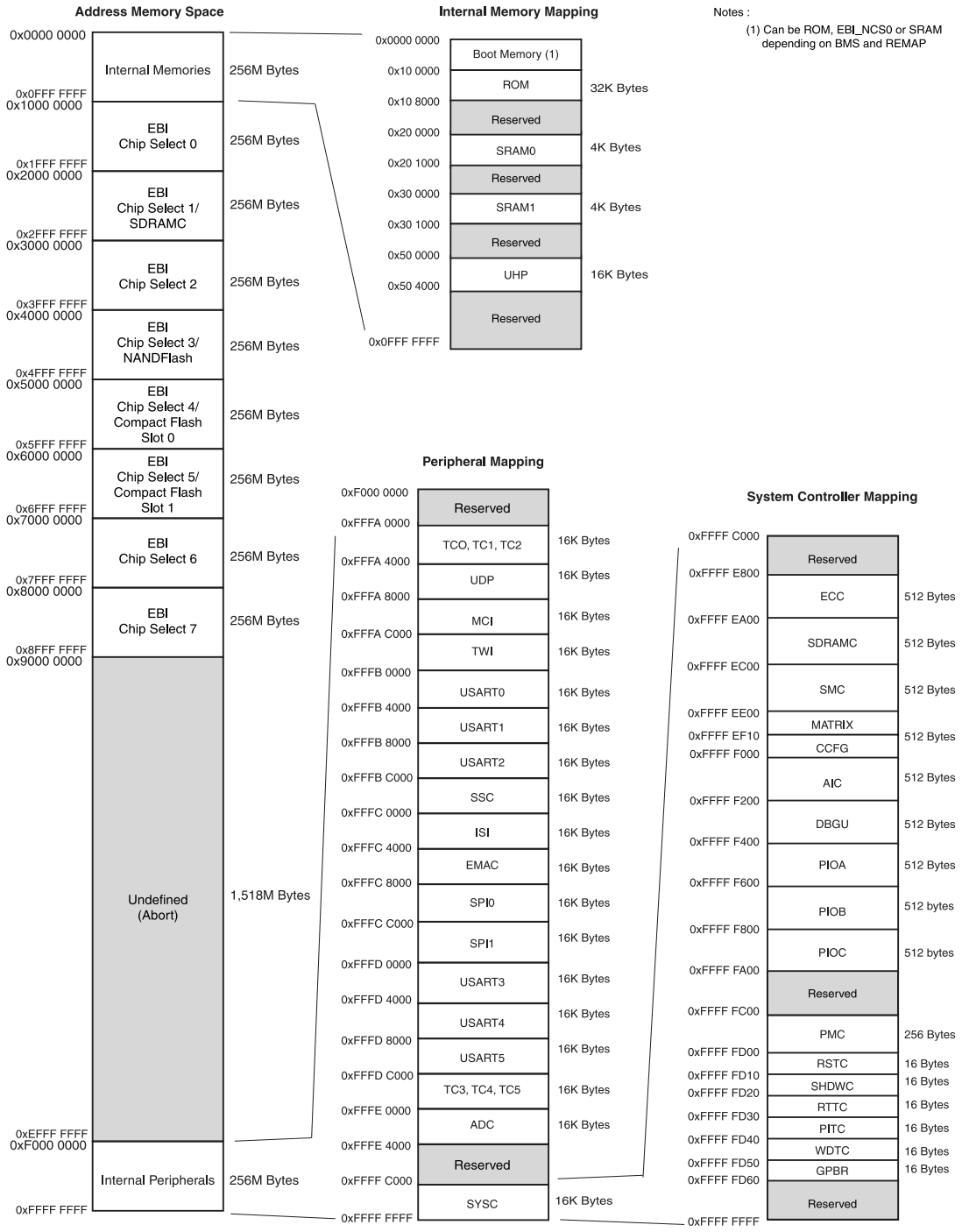
deska s elektronikou	vývojový kit	pin procesoru
X1-3	EXT-5	PB0
X1-5	EXT-3	PC15
X1-7	EXT-13	PB4
X1-9	EXT-8	PC13
X1-10	EXT-10	PC10
X1-11	EXT-14	PC8
X1-12	EXT-12	PC9
X1-13	EXT-7	PB1
X1-14	EXT-11	PB3
X1-15	EXT-31	PB19
X1-16	EXT-16	PC7
X1-17	EXT-17	PB8
X1-18	EXT-18	PC6
X1-19	EXT-20	PC4
X1-21	EXT-21	PB10
X1-22	EXT-22	PB31
X1-23	EXT-23	PB11
X1-24	EXT-24	PB30
X1-25	EXT-25	PB16
X1-29	EXT-29	PB18
X1-30	EXT-30	PB27
X1-31	EXT-15	PB5
X1-32	EXT-32	PB26
X1-33	WXT-33	PB20
X2-7	UEXT-4	PB9
X2-8	UEXT-5	PA24
X2-9	UEXT-9	PB2
X2-10	UEXT-6	PA23

Tab. B.2: Propojení vývojového kitu a desky s vlastní elektronikou po provedených změnách

Mn.	Hodnota	Označení	Pouzdro	Poznámka
15	1K	R9, R21, R33, R48, R22, R25, R27, R30, R32, R35, R37, R38, R39, R43, R46	0805	
14	100N	C4, C6, C7, C9, C11, C14, C16, C17, C18, C19, C20, C22, C25, C26, C27	0805	
9	0R	R3, R5, R12, R13, R14, R15, R17, R40, R42	1206	
8	1R0	R6, R7, R18, R19, R28, R29, R44, R45	1206	
4	TS912	IC4, IC6, IC8, IC9	SO08	operační zesilovač
4	SMBJ5.0A	D7, D9, D10, D11	SMBJ	transil 6,74 V
4	PSH02-07P	K3, K4, K7, K8		konektor
4	ARK500/3	K1, K2, K5, K6		svorky
4	68R	R8, R20, R31, R7	0805	
4	10K	R10, R11, R34, R36	0805	
4	3K3	R4, R16, R26, R41	1206	
4	1N4148W	D4, D5, D6, D8	SOD323	
2	SK56C	D1, D2	SMC	schottkyho dioda
2	PFRA.300	F1, F2		polyswitch 3 A
2	L6206	IC3, IC7	SO24W	dvojitý H-můstek
2	jumper	JP1, JP2		
2	E330M/35V	C2, C3	E5-10,5	elektrolyt, nízké ESR
2	E100M/25V	C8, C21	E2,5-6	elektrolyt
2	220N	C15, C24	0805	

2	100R	R2, R24	0805	
2	10N	C10, C23	0805	
1	SM6T27A	D3	SMBJ	transil 27 V
1	power-jack	J1		napájecí konektor
1	MLW40	X1		kon. pro plochý kabel
1	MLW10	X2		
1	LM2596	IC1	TO220-51	DC-DC měnič 5 V
1	LED	LED1	0805	
1		L2	0805	feritová perlička
1	E470M/10V	C5	E3,5-8	elektrolyt, nízké ESR
1	ADR366	IC2	SOT23-5	napěťová reference
1	ADC128S052	IC5	TSSOP16	AD převodník
1	680R	R1	0805	
1	10M	C13	0805	
1	1uH	L3	0805	tlumivka
1	1M	C12	0805	
1	1K	R23	1206	
1	E220M/10V	C10, C23	E2,5-6	elektrolyt
1	DPU033A3	L1		akumulační tlumivka

Tab. B.3: Druhá varianta HW: seznam součástek



Obr. B.13: Organizace paměti Atmel AT91SAM9260 [1]