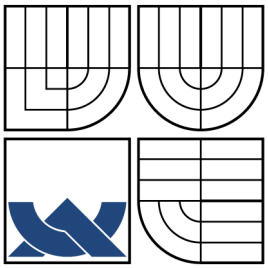
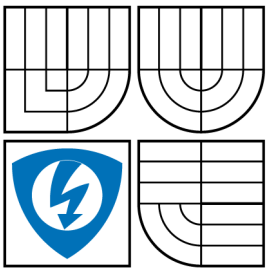


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

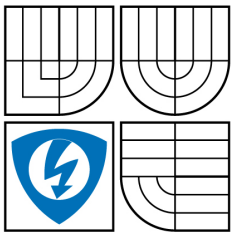
VZDÁLENÁ SPRÁVA JEDNOČIPOVÝCH SYSTÉMŮ
REMOTE MAINTENANCE OF THE MICROCONTROLLER SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MARTIN VÁGNER
AUTHOR

VEDOUCÍ PRÁCE — Ing. Pavel Kučera, Ph.D.
SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student:	Bc. Martin Vágner	***ID:***	78423
Ročník:	2	***Akademický rok:***	2009/2010

NÁZEV TÉMATU:

Vzdálená správa jednočipových systémů

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s mikrokontroléry řady Atmel AVR a prostudujete problematiku jejich programování. Seznamte se s problematikou síťového rozhraní Ethernet a sadou protokolů transportní vrstvy TCP/IP. Navrhněte a realizujte systém pro vzdálenou správu výše uvedené řady procesorů pomocí tohoto síťového rozhraní. Při realizaci se pokuste zohlednit požadavky na jednoduchost (robustnost) systému a jeho zabezpečení.

DOPORUČENÁ LITERATURA:

[1] ATMEL. ATmega644 Preliminary. 2008/07. 376 s. [online]. [cit.2010-01-20]. Dostupné z <http://www.atmel.com/dyn/resources/prod_documents/doc2593.pdf>.
[2] Dostálek, L.: "Velký průvodce protokoly TCP/IP a systémem DNS." 5. vydání. Computer Press, 2008, ISBN: 978-80-251-2236-5.

Termín zadání:	8.2.2010	***Termín odevzdání:***	24.5.2010

Vedoucí práce: Ing. Pavel Kučera, Ph.D.

prof. Ing. Pavel Jura, CSc.
Předseda oborové rady

ABSTRAKT

Práce řeší problematiku vzdálené správy paměti programu (firmware) jednočipových systémů založených na mikrokontrolérech řady Atmel AVR prostřednictvím rozhraní Ethernet a protokolů TCP/IP. Komunikaci zprostředkovává embedded server NE-4100T. V úvodu jsou rozebrány vlastnosti použitého embedded serveru a způsoby programování mikrokontrolérů Atmel AVR. Zvoleným řešením úlohy je pak metoda bootloaderu, pro niž byl navržen komunikační protokol, softwarové vybavení mikrokontroléru a obslužný program pro PC. Po hardwarové stránce řešení zahrnuje návrh obvodů pro propojení serveru NE-4100T s modulem mikrokontroléru, snižujícího měniče napětí, obvodu hodin reálného času a příslušné desky plošných spojů. V projektu se podařilo splnit požadavky na vzdálenou správu paměti programu mikrokontroléru, ale nebylo nalezeno dostatečné řešení stran zabezpečení proti neoprávněnému přístupu a napadení systému.

KLÍČOVÁ SLOVA

Vzdálená správa, mikrokontrolér, zápis a mazání paměti programu, firmware, bootloader, Atmel AVR, Ethernet, TCP/IP, MOXA NE-4100T, ATmega644

ABSTRACT

This thesis deals with methods of remote maintenance of microcontroller systems based on Atmel AVR family over the Ethernet interface and TCP/IP protocols. To create communication through TCP/IP, an embedded server NE-4100T is used. At the beginning, key features of the server and methods of handling with content of a program memory are discussed. The final solution is based on the bootloader method. It includes bootloader firmware and user program for PC. The hardware part covers design of interconnection electronics, DC-DC step down converter, real time clock and printed circuit board. The remote maintenance of program memory has been sucessfully solved, but the embedded server NE-4100T produces a problem with an auhentification without a sufficient solution.

KEYWORDS

Remote maintenance, microcontroller, programming and erasing of the program memory, firmware, bootloader, Atmel AVR, Ethernet, TCP/IP, MOXA NE-4100T, ATmega644

VÁGNER, M.: *Vzdálená správa jednočipových systémů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 92 s. Vedoucí diplomové práce Ing. Pavel Kučera, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Vzdálená správa jednočipových systémů“ 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.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Pavlu Kučerovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne ....

(podpis autora)

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V současné době elektronická zařízení stále ve větším množství využívají namísto mnoha jednoúčelových obvodů integrovaná řešení v podobě mikrokontrolérů. Mezi hlavní důvody patří velké množství integrovaných periferií, rostoucí výkon a díky masové produkci také klesající cena. Integrované periferie přebírají funkci externích podpůrných obvodů, čímž zjednodušují celé zařízení a tak zmenšují jeho rozměry.

Mikrokontroléry bývají nejčastěji vybaveny pamětí programu typu flash, proto může být program opakovaně přepsán. Přepisování obsahu paměti programu ve výsledku dovoluje u stávájících zařízení jednoduše modifikovat programové vybavení, a tak přidávat nové funkce, či odstraňovat případné chyby. Z těchto skutečností logicky vyplývá požadavek na nástroje, které umožní správu těchto systémů.

S rozvojem výpočetní techniky a síťového rozhraní Ethernet dochází k pronikání těchto technologií i do oblasti průmyslové automatizace a embedded zařízení. Rozvoj Ethernetu v těchto odvětvích způsobil především příchod technologie Fast Ethernet s přenosovou rychlostí $100\,\mathrm{Mbit/s}$, která spolu s vhodným komunikačním protokolem dosahuje lepších vlastností z hlediska odezvy a přenosové rychlosti na větší vzdálenosti ve srovnání s běžnými technologiemi jako Profibus DP.

Vzdálená správa pak nachází své místo hlavně z hlediska snazší diagnostiky, minimalizace nákladů na údržbu a rychlejšímu řešení problémů, protože tak často odpadá nutnost fyzické přítomnosti operátora u daného zařízení.

Z principu vzdáleného přístupu na druhou stranu vznikají i nová rizika. Tím je především možnost vzdáleného napadení systému za účelem neoprávněného získání procesních dat, pozměnění jeho činnosti nebo úplného vyřazení z provozu.

Práce si klade za cíl vytvořit systém pro vzdálenou správu modulu s mikrokontrolérem typu Atmel AVR prostřednictvím síťového rozhraní Ethernet a sadou protokolů transportní vrstvy TCP/IP. K implementaci rozhraní Ethernet bude využito embedded serveru NE-4100T. Pro tyto komponenty bude řešení zahrnovat návrh obvodů nutných k jejich vzájemnému propojení a napájení, dále návrh desky plošných spojů a následnou realizaci celého zařízení. Snahou bude především jednoduchost, v rámci možností generalizace řešení na co nejširší počet obvodů z řady Atmel AVR a zabezpečení systému.

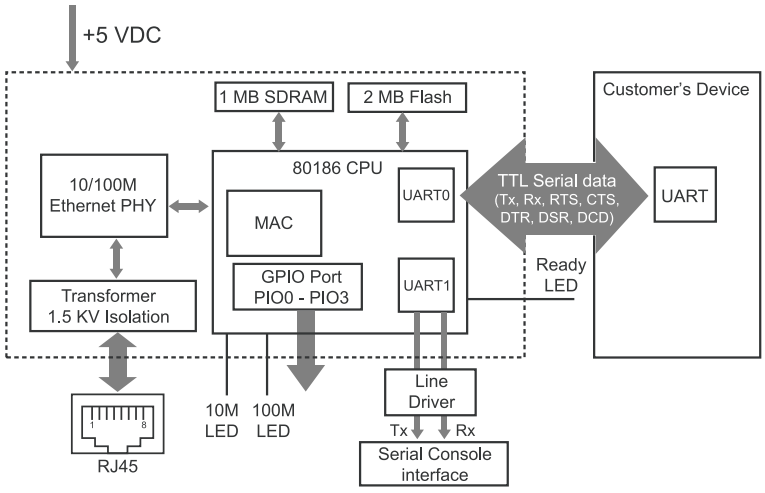
2 Volba koncepce řešení

Úvod práce bude zaměřen na zevrubné vlastnosti požadovaných komponent, kterými jsou především embedded server NE-4100T a mikrokontroléry řady Atmel AVR. Na jejich základě pak budou diskutovány různé způsoby řešení, jejich výhody, nevýhody a nakonec zvoleno jedno konkrétní řešení.

2.1 Rozbor vlastností embedded serveru NE-4100T

Moduly řady NE-4100 představují jednoduchý způsob implementace ethernetového rozhraní a protokolů TCP/IP do jakékoliv aplikace disponující sériovým rozhraním. Jedná se o embedded server, jenž vytváří most mezi rozhraním Ethernet a asynchronním sériovým portem s řadou dalších funkcí. Moduly jsou k dispozici v několika modifikacích lišících se způsobem připojení a montáže do cílového zařízení.

Blokové schéma se nachází na obrázku 2.1. Činnost řídí šestnáctibitový procesor z řady 80186 doplněný o 1 MB paměti dat typu SDRAM, 2 MB paměti programu flash, což umožňuje aktualizaci firmware modulu, a dvě asynchronní sériové linky. Rozhraní Ethernet se skládá z obvodů fyzické a linkové vrstvy. Na ně navazuje implementace TCP/IP komunikace. Navíc jsou k dispozici čtyři digitální signály konfigurovatelné jako vstupy či výstupy pro všeobecné účely a signalizační LED definující stav modulu.



Obrázek 2.1: Blokové schéma NE-4100T [1]

2.1.1 Napájení

Hlavní nevýhodou modulu je požadované napájecí napětí $5\,\mathrm{V} \pm 5\,\%$ a maximální odebíraný proud $290\,\mathrm{mA}$ [1]. Moderní zařízení s nízkou spotřebou obvykle pracují s nižším napájecím napětím (3,3 nebo $1{,}8\,\mathrm{V}$), což znemožňuje snadné využití modulu NE-4100T v těchto případech z důvodu rozdílných logických úrovní.

Výrobcem udávaná nízká spotřeba $1{,}5\,\mathrm{W}$ rovněž není ve srovnání s obdobnými konkurenčními výrobky (EM1000 a EM1202 [4], EZL-50M [5], XT-Nano-XXL [6], FModTCP DB [7]), které se pohybují v oblasti 0,4 až $0{,}7\,\mathrm{W}$, nikterak výjimečná.

2.1.2 Rozhraní

Ethernet

Fyzickou vrstvu vytváří obvod RTL8201BL. Podporovány jsou dnes nejběžněji používané standardy 10Base-T a 100Base-TX s možností poloviního nebo plného duplexu, přičemž nechybí automatická detekce (auto negotiation). Na modulu se dále nachází nezbytný oddělovací transformátor s izolačním napětím $1{,}5\,\mathrm{kV}$, který tvoří galvanické oddělení.

Sériové rozhraní

Modul NE-4100T obsahuje dvě rozdílně vybavená asynchronní sériová rozhraní s logickými úrovněmi TTL. Pro připojení serveru k uživatelskému zařízení slouží plnohodnotné rozhraní UART0. K dispozici jsou signály TxD, RxD, RTS, CTS, DTR, DSR, DCD. Možná nastavení parametrů přenosu dat udává tabulka 2.1.

Rozhraní UART1 zprostředkovává vstup a výstup konfigurační konzole. Parametry přenosu jsou pevně nastaveny výrobcem na $19200\,\mathrm{bit/s}$, 8 datových bitů, 1 stop bit, bez parity a řízení toku. Rozhraní má vyvedeny pouze signály RxD, TxD.

Digitální vstupy/výstupy

Modul dále nabízí čtyři programovatelné vstupně-výstupní porty s logickými úrovněmi typu TTL. U každého DIO může uživatel definovat jeho funkci (vstup či výstup) a v případě výstupu logickou úroveň. Čtení a programování probíhá pomocí příkazů

na definovaném TCP portu ([1], příloha D „DIO Commands“). Nevýhoda spočívá ve skutečnosti, že přístup k těmto funkcím může být omezen pouze povolenými IP adresami ([1], kapitola 6 „Accessible IP Settings“).

Datových bitů	5, 6, 7, 8
Stop bitů	1, 1,5, 2
Parita	žádná, sudá, lichá, space (vždy 0), mark (vždy 1)
Řízení toku dat	vypnuto, hardwarové RTS/CTS, softwarové XON/XOFF
Přenosová rychlost	50 bit/s až 115,2 kbit/s (PCB V1.x) 110 kbit/s až 230 kbit/s (PCB V2.x)
Vyrovnávací paměť	vypnuta, 16 B (FIFO)

Tabulka 2.1: Parametry přenosu dat na rozhraní UART0 [1]

2.1.3 Pracovní módy

Hlavní výhoda modulu NE-4100T spočívá pro jednoduchá zařízení v implementaci komunikace pomocí TCP/IP protokolu, což vede ke zjednodušení cílové aplikace. Uživatel má k dispozici sériové rozhraní, kde data jsou přenášena prostřednictvím Ethernetu a protokolů TCP/IP bez nutnosti zabývat se konkrétní problematikou těchto vrstev.

Ke splnění účelu a požadavků cílové aplikace jsou k dispozici čtyři módy, ve kterých může modul pracovat. Tyto možnosti se snaží postihnout nejběžnější požadavky ze strany uživatelských zařízení. Jedná se o varianty TCP server, TCP klient, UDP a virtuální sériový port.

TCP server

Modul v módu TCP server naslouchá na své IP adrese a zvoleném TCP portu. Na žádost klienta je sestaveno obousměrné spojení, které umožňuje zapisovat a číst data prostřednictvím sériové linky. Princip protokolu TCP [9, 8] garantuje spolehlivé doručení dat ve správném pořadí, nebo informaci o jejich nedoručitelnosti.

V jednom okamžiku dovoluje server připojení až čtyř klientů současně, přičemž jejich počet lze dle potřeby libovolně omezit.

TCP server mód se tedy jeví výhodný především pro samostatně pracující zařízení, dálkovou správu, terminály a uživatelská rozhraní, zvláště je-li požadováno potvrzení o přenesení dat.

TCP klient

Tento mód dovoluje připojení modulu protokolem TCP ke vzdálenému hostiteli. Jeho IP adresa a cílový port musí být předem znám a pevně nastaven. Data se následně přenáší obousměrně a k ukončení spojení dochází po definovaném časovém intervalu, je-li přenos dat neaktivní. Z použití protokolu TCP pak vyplývají stejné vlastnosti jako u módu server.

Sestavení spojení tedy dochází na žádost modulu, a tak data mohou být přenášena v návaznosti na časově nedeterministické události. Tím se snižuje zatížení sítě nadbytečnou komunikací, kterou by způsobovalo udržování spojení či kontrolování změn v režimu server.

UDP mód

Přenos dat se realizuje prostřednictvím protokolu UDP [10, 8]. Hlavní rozdíl UDP módu od módů TCP tedy spočívá v komunikaci bez nutnosti sestavit spojení. Mezi výhody patří menší náročnost, rychlejší přenos dat a možnost snadného doručení dat na velké množství adres (multicasting). Nevýhodou je především absence ověření správného doručení na úrovni protokolu UDP.

Mód se s výhodou používá hlavně k úlohám, které nevyžadují vysokou spolehlivost, jako například vizualizace. Vzhledem k absenci postupů pro spolehlivé doručení, které vnášejí časově nedeterministické chování, je UDP mód vhodný i pro úlohy s požadavky na práci v reálném čase, kdy chybové stavy řeší příslušný nadřazený protokol.

Real COM mód

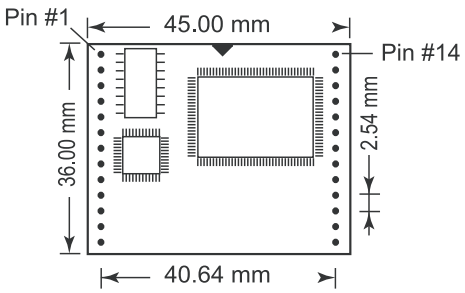
Základ spočívá ve vytvoření virtuálního sériového portu COM na PC pomocí dodávaných ovladačů. Data se pak přenášejí mezi tímto portem a vzdáleným rozhraním modulu NE-4100T. Z hlediska softwarového vybavení PC se port chová jako standardní port instalovaný v PC.

Mód má za cíl především snadné rozšíření stávajících zařízení a software s rozhraním RS232 o komunikaci prostřednictvím Ethernetu. V tomto režimu musí být bráno v potaz dopravní zpoždění způsobené přenosem dat, které může snadno dosáhnout jednotek sekund, a tak následně zapříčinit nefunkčnost dané aplikace.

2.1.4 Mechanické specifikace

Verze NE-4100T je určena k montáži na desku s plošnými spoji jako modul o rozměrech 36 x 45 mm s rozložením vývodů v uspořádání DIL-26 a roztečí 2,54 mm, jak ukazuje obrázek 2.2.

Pracovní podmínky [2] se liší především provozním teplotním rozsahem. Moduly ve standardním provedení jsou určeny pouze pro kladné hodnoty v rozsahu 0 až 55 °C. Modely s rozšířeným teplotním rozsahem (označené T) mohou pracovat v rozsahu teplot −40 až 75°C. Povolený rozsah relativní vlhkosti spadá do rozmezí 5 až 95 %.



Obrázek 2.2: Mechanické uspořádání modulu NE-4100T [1]

Rozložení vývodů udává tabulka 2.2. Signály ETx+, ETx-, ERx+, ERx- náleží rozhraní Ethernet a jsou určeny k vyvedení na standardní konektor typu RJ45. Vstupy GND a +5 V jsou přívody napájecího napětí, PIO0 až PIO3 uživatelské vstupně-výstupní signály a ostatní piny náleží sériovým rozhraním UART0 a UART1. Signály 10M LED a 100M LED udávají aktuální stav ethernetového rozhraní, Ready LED stav modulu.

2.1.5 Konfigurace a softwarové vybavení

Správu a nastavení modulu zprostředkovává rozhraní Network Enabler Configuration Interface (NECI), webové rozhraní, protokol telnet nebo sériová konzole na portu UART1.

1	ETx+ (RJ45 pin 1)	8	RTS	15	PIO1	22	GND
2	ETx- (RJ45 pin 2)	9	CTS	16	PIO2	23	Ready LED
3	ERx+ (RJ45 pin 3)	10	Reset	17	PIO3	24	+5 V
4	ERx- (RJ45 pin 6)	11	GND	18	100M LED	25	+5 V
5	10M LED	12	GND	19	DCD	26	RXD1
6	TXD	13	TXD1	20	DSR	-	-
7	RXD	14	PIO0	21	DTR	-	-

Tabulka 2.2: Rozložení vývodů modulu NE-4100T [1]

Od verze firmware 3.0 je navíc dostupný tzv. „Serial Command Mode“, a tak lze souborem speciálních příkazů [3] modul konfigurovat přímo prostřednictvím datového portu UART0.

Pro rozhraní NECI existuje nástroj s grafickým rozhraním „Network Enabler Administrator“, nebo DLL knihovna pro jednoduchou integraci do uživatelské aplikace.

Zabezpečení

Z hlediska informační bezpečnosti modul NE-4100T dovoluje přístup k nastavením chránit heslem, bohužel dokumentace neuvádí, je-li proces ověření hesla nějakým způsobem chráněn před odposlechem. Mezi bezpečnostní prvky dále patří systém zasílání hlášení na email prostřednictvím SMTP protokolu, kde lze definovat odeslání informační zprávy v případě chybného pokusu o přihlášení, změně IP adresy, hesla či restartu. Na úrovni IP protokolu je možné nastavit autorizované IP adresy, které mají oprávnění s modulem komunikovat.

Až na omezení IP adres klientů tedy není možné dalšími způsoby ošetřit autorizaci ke komunikaci prostřednictvím datového portu, nebo přístupu k uživatelským portům.

Funkční bezpečnost zvyšuje vestavěný časovač watchdog. Úkolem časovače je zabránit permanentnímu zablokování modulu vlivem hardwarové či softwarové chyby, která obvykle vede k uváznutí vykonávání programu, nebo nedefinovanému chování. Není-li časovač patřičným způsobem periodicky resetován, dojde při vypršení časového intervalu k resetování modulu, a tak k obnovení činnosti.

Další funkce

U sériového rozhraní UART0 dovoluje modul NE-4100T nastavit generování emailové zprávy při změně logické úrovně signálů DCD nebo DSR, čehož je možné využít například k odeslání upozornění na chybové stavy.

Při dynamické konfiguraci IP adresy modulu ze serveru DHCP lze aktivovat funkci periodického ohlašování nově získané IP adresy modulu na uživatelsky definovanou adresu v síti.

2.2 Mikrokontroléry Atmel AVR 8-bit

Rodina obvodů Atmel AVR v současné době zahrnuje jednak 8-bitové obvody a jednak řadu 32-bitových mikrokontrolérů. Protože jsou obě skupiny značně odlišné a středem zájmu této práce je především obvod ATmega644, bude dále uvažována pouze řada 8-bitových mikrokontrolérů, do které spadá.

Zástupci této skupiny se pak dále dělí na řady ATmega, ATtiny a AT90. V příloze E byl sestaven přehled zmíněných obvodů a jejich klíčových vlastností pro tuto práci. Z ní je patrné, že nejpočetnější skupinu tvoří právě obvody ATmega, které se zároveň vyznačují největší výbavou v oblasti velikosti paměťových prostorů a periférií.

Obvody ATtiny jsou určeny pro jednoduché aplikace, disponují minimálními velikostmi pamětí programu a dat. Mikrokontroléry AT90PWM jsou pak zaměřeny na speciální aplikace, disponují odlišnými vlastnostmi a vyznačují se horší dostupností.

Z těchto důvodů budou dále uvažovány obvody kategorie ATmega, které jsou nejrozšířenější a zahrnují obvod, jenž bude osazen v cílové aplikaci.

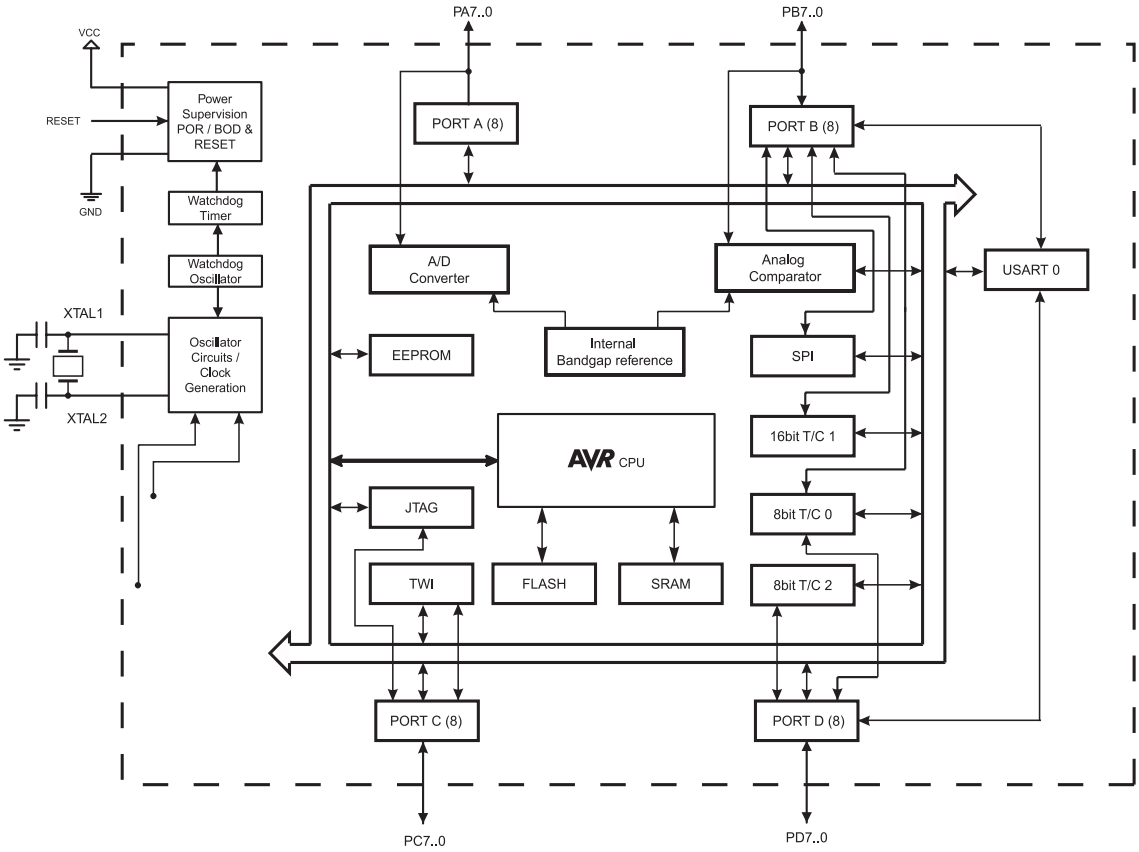
2.2.1 Architektura

Architektura mikrokontrolérů AVR využívá redukovanou instrukční sadu (RISC) spolu s technikou zřetězení instrukcí. Tím je dosaženo vykonání většiny instrukcí v jednom hodinovém cyklu, což zajišťuje vysoký výkon při nízkých taktovacích frekvencích například oproti obvodům řady 8051.

Z hlediska dálkové správy firmware je však důležitější provedení paměťového subsystému. Zde byla zvolena harvardská architektura. Z toho vyplývá, že paměť programu a paměť dat jsou fyzicky odděleny. Přístup se řeší prostřednictvím samostatných adresních, datových a řídících sběrnic. Z programátorského hlediska pak paměti disponují nezávislými adresními prostory.

Mikrokontroléry dále obsahují celou řadu podpůrných obvodů a periferií, jak demonstruje blokové schéma na obrázku 2.3. Z periferních obvodů jsou z pohledu dálkové správy důležitá komunikační rozhraní. Pro sériovou komunikaci v asynchronním režimu bývají mikrokontroléry AVR vybaveny jedním až dvěma rozhraními UART (podporuje pouze asynchronní režim) nebo USART (podporuje synchronní i asynchronní režim), jak udává přehled v příloze E.

Důležitými periferiemi jsou také obvody dohlížející na napájecí napětí a časovač typu watchdog, kterými je vybavena většina mikrokontrolérů.



Obrázek 2.3: Blokové schéma ATmega644 [11]

2.2.2 Typy pamětí

Všechny obvody ATmega obsahují integrovanou nonvolatilní paměť programu typu flash, volatilní paměť dat typu SRAM a nonvolatilní paměť EEPROM pro uživatelská data. Mezi programovatelné prvky dále patří konfigurační bity typu „lock“ a „fuse“.

Paměť programu

Paměť programu je interní typu flash a není možné její další rozšíření (např. připojením externí paměti programu), což dovoluje programování jednotným způsobem. Organizace po šestnácti bitech odpovídá šířce instrukčního slova, kdy jedna instrukce má šířku 16 nebo 32 bitů [12]. Velikost paměti programu se pohybuje v rozmezí 1 až 256 kB (viz. příloha E). Udávaná životnost paměti Flash je 10000 cyklů zápis/vymazání.

Nonvolatilní paměť dat

Uživatelská paměť dat EEPROM s osmibitovou organizací může mít velikost 0 až 4 kB (viz. příloha E). Nezávislý adresní prostor je dostupný pomocí registrů v I/O oblasti prostřednictvím instrukcí *IN*, *OUT*. Udávaná životnost 100000 cyklů.

2.3 Metody programování

Dostupné způsoby programování mikrokontrolérů AVR hrají klíčovou roli ve volbě koncepce řešení dané úlohy, a proto budou dále rozebrány jednotlivé metody, uvedeny jejich vlastnosti, omezení, výhody a nevýhody.

2.3.1 Paralelní

Paralelní programování [11] vyžaduje jeden osmibitový I/O port, osm řídících signálů, RESET vstup, hodinový signál a zvýšené programovací napětí 12 V. Touto metodou lze programovat a číst veškeré paměťové prvky mikrokontroléru, a to s vysokou rychlostí. Podporovány jsou všechny obvody.

Vzhledem k značné náročnosti na propojení I/O signálů a nezbytnosti programovacího napětí 12 V, však není paralelní programování vhodné pro úlohu vzdálené správy firmware.

2.3.2 Sériové

Metodu sériového programování [11] podporují rovněž všechny mikrokontroléry. Fyzická vrstva odpovídá rozhraní SPI. Jsou vyžadovány signály MISO, MOSI, SCK, RESET a standardní napájecí napětí. Pomocí sériového programování lze číst a zapisovat paměť programu flash, paměť dat EEPROM, konfigurační fuse a lock bity. Tato metoda zaručuje při správném obvodovém řešení možnost programování obvodu přímo v zapojení bez nutnosti jeho vyjmutí.

2.3.3 Rozhraní IEEE 1149.1 (JTAG)

Mikrokontroléry, které disponují standardním rozhraním JTAG, mohou být programovány speciální sadou instrukcí [11]. Vyžadovány jsou signály TCK, TMS, TDI, TDO a standardní napájecí napětí.

Kromě programování paměti programu flash, paměti dat EEPROM a konfiguračních fuse a lock bitů, umožňuje rozhraní JTAG navíc další operace. Těmi jsou reset obvodu, přístup do registrů a veškerých typů pamětí. Stejně jako u sériového programování lze obvod programovat přímo v zapojení. Nevýhoda spočívá především v absenci rozhraní JTAG u některých starších typů mikrokontrolérů.

2.3.4 Rozhraní debugWIRE

Nejnovější obvody disponují rozhraním „debugWIRE“. Rozhraní prostřednictvím jediného signálu přístupného jako alternativní funkce pinu RESET zprostředkovává kompletní ladicí funkce. K nim navíc paří i možnost zápisu do paměti programu i nonvolatilní paměti dat EEPROM.

Mezi nevýhody lze řadit hlavně absenci u starších obvodů a nemožnost využití signálu pinu RESET k resetování mikrokontroléru, což znemožňuje připojení externího monitoru napájení a obvodu WDT.

2.3.5 Bootloader

Většina obvodů AVR podporuje funkci „Self Program Memory“ (viz. příloha E), která dovoluje zápis paměti programu flash pomocí instrukce *SPM* [12] a mazání po stránkách. Této funkce může být s výhodou využito pro zaváděcí program (tzv. „bootloader “ [11], s jehož pomocí lze zajistit programování paměti programu flash v daném zapojení pomocí libovolného rozhraní, jenž se v mikrokontroléru nachází (např. UART, I2C). Vzhledem k softwarové implementaci komunikačního protokolu a programovacího algoritmu může zaváděcí program zahrnovat další funkce jako například ošetření stavů I/O linek během programování, kontrolu neporušenosti programu a ochranu před neoprávněným přístupem (heslo, šifrování přenosu).

Kód bootloaderu se nachází ve speciální sekci na konci paměti programu flash. Velikost sekce lze nastavit konfiguračními fuse bity.

Start bootloaderu může být realizován jednak instrukcí skoku na příslušnou adresu, nebo dle konfigurace fuse bitů, vždy po resetu obvodu přesměrováním vektoru nemaskovatelného přerušení RESET z adresy 0000_{hex} na adresu bootloaderu.

Nevýhodami jsou nemožnost měnit hardwarovou konfiguraci mikrokontroléru (fuse bity), zmenšení kapacity paměti programu o kód bootloaderu, nutnost počátečního naprogramování a konfigurace obvodu jednou z výše uvedených metod programování (paralelní, sériové, JTAG, debugWire).

2.4 Rozbor možných způsobů řešení

2.4.1 Bootloader

Z rozboru klíčových vlastností embedded serveru NE-4100T a mikrokontrolérů Atmel AVR vyplývá na první pohled jako nejvhodnější metoda pro implementaci vzdálené správy firmware vytvoření vhodného bootloaderu. Toto řešení ale naráží na zásadní bezpečnostní problém spočívající v jeho spuštění.

Hardwarové řešení resetem mikrokontroléru s využitím jednoho z uživatelských vstupů-výstupů serveru NE-4100T se jeví nevhodné z důvodu, že není možné dostatečně omezit přístup k ovládání stavu toho výstupu, jak bylo uvedeno v odstavci 2.1.5. To

činí systém snadno napadnutelný, protože jej lze trvalým resetem mikrokontroléru jednoduše vyřadit z provozu. Jedinou ochranou může být pouze nastavení povolených IP adres, avšak z principu TCP/IP komunikace [8] se jedná o řešení značně nedostačující. Při použití síťových prvků typu rozbočovač (hub) či jednoduchý přepínač (switch), může být IP adresa snadno podvrhnuta.

Čistě softwarové řešení spuštění bootloaderu má několik nevýhod. Pro funkčnost by bylo nutné kdykoliv akceptovat příkazy přijímané sériovým rozhraním, a to současně při zachování činnosti hlavního programu mikrokontroléru. Obdobná funkce lze realizovat jedině pomocí přerušení (za předpokladu, že bootloader pracuje jako nadřazený systém nad hlavním programem). Z toho však vyvstávají další problémy. Sériové rozhraní má být využíváno hlavním programem, a proto by bootloader musel být integrován do hlavního programu. Dále by vektory přerušení musely být umístěny v sekci bootloaderu (přesměrování vektorů přerušení bitem *IVSEL* v registru *MCUCR* [11]), funkčnost dálkové správy by závisela na správné konfiguraci sériového kanálu a pomocných registrů, které mohou být hlavním programem kdykoliv změněny. Ostatní vektory přerušení by bylo nutné přesměrovat zpět na základní adresy, aby byla zachována funkčnost hlavního programu. Zpracování přijatých příkazů v přerušení sériového kanálu navíc vykazuje nezanedbatelnou režii, která zatěžuje mikrokontrolér a při nesprávném návrhu může způsobit chybnou funkci hlavního programu. V neposlední řadě je nutné zmínit, že jsou-li naprogramovány ochranné lock bity sekce bootloaderu a současně přesměrovány vektory přerušení, dojde k zakázání přerušení ve chvíli, kdy je vykonáván kód hlavního programu [11].

2.4.2 SPI, JTAG, debugWIRE

Protože embedded server NE-4100T disponuje pouze asynchronním sériovým rozhraním, vyžadovalo by pro programování prostřednictvím rozhraní SPI, JTAG nebo debugWIRE přídavný převodník.

Konverzi UART/SPI nabízí například obvod MAX3100. Nevýhodou je jednak zvýšení ceny celého zařízení, ale především by nebylo možné zajistit správnou činnost sériového rozhraní v rámci hlavního programu a taktéž další funkce, například ochranu před neoprávněným přístupem.

Z uvedených důvodů by řešení využívající server NE-4100T a rozhraní SPI, JTAG či debugWIRE vyžadovalo doplnit obvod dalším mikrokontrolérem, který by se staral o programovací proces a jeho zabezpečení. Bohužel, aby byla zachována funkčnost sériového rozhraní vzhledem k hlavnímu MCU, musel by být signál pro vysílání dat přepínán mezi hlavním a pomocným mikrokontrolérem. To přináší problém s rozhodováním, kdy má být signál přepnut na pomocný MCU. Přepnutí by bylo možné pouze na základě pasivního příjmu vyhrazené sekvence pro vstup do programovacího režimu. Z důvodu zabezpečení by sekvence musela představovat formu určitého hesla (tj. velmi dlouhá a u každého zařízení nejlépe unikátní), protože po vstupu do programovacího režimu by došlo nutně odpojení vysílacího signálu od cílového zařízení, což by mohlo být opět zneužito k omezení funkčnosti. Další nevýhodou je především nutnost naprogramovat pomocný mikrokontrolér.

2.4.3 Jiný server než NE-4100T

Jako další možnost připadá v úvahu změna embedded serveru na typ, který splňuje požadavky na zabezpečení přístupu k I/O linkám, což by umožňovalo hardwarové resetování mikrokontroléru, a tím implementaci bootloaderu se splněním požadavků na zabezpečení.

Z nalezených dostupných produktů (např. EM1000 a EM1202 [4], EZL-50M [5], XT-Nano-XXL [6], FMod-TCP DB [7]) však žádný neuspokojuje toto kritérium, protože funkce jsou obdobné jako u NE-4100T.

Nejvýhodnější by bylo vytvořit server na míru potřebným požadavkům. Toto řešení je však značně náročnější. Z důvodu zahlcení serveru DoS útoky nelze využít softwarové řešení TCP/IP stacku, což klade vyšší nároky na hardware. Hardwarové řešení MAC vrstvy a TCP/IP stacku implementuje například obvod W3150A [13], který není dostupný v kusovém množství.

3 Řešení

Analýzou vlastností mikrokontrolérů Atmel AVR a embedded serveru NE-4100T v první části práce bylo zjištěno, že s NE-4100T nelze uspokojivě splnit požadavky na zabezpečení systému. Adekvátního řešení nelze dosáhnout jak ve variantě s bootloaderem, tak při využití rozhraní JTAG, SPI či debugWIRE a přídavného mikrokontroléru.

Protože nebyla nalezena vhodná náhrada za NE-4100T, která by řešila dané problémy, připadá v úvahu jako plnohodnotné řešení pouze konstrukce vlastního serveru s patřičnými vlastnostmi.

Po dohodě s vedoucím práce byla zvolena varianta bootloaderu, protože se jedná o řešení, které nevyžaduje dodatečný hardware a nevýhody jsou obdobné jako u ostatních značně složitějších variant. Zabezpečení pak bude realizováno na úrovni aktivních prvků sítě Ethernet a protokolů TCP/IP mimo rámec této práce.

V této části budou stanoveny funkce bootloaderu a podrobně rozebrány vlastnosti modulu CPU unit se zaměřením na vlastnosti, které je nezbytné zohlednit při návrhu. Následně budou popsány principy činnosti bootloaderu u mikrokontrolérů Atmel AVR a na základě těchto poznatků bude stanoven komunikační protokol, softwarové vybavení mikrokontroléru a nakonec uživatelská aplikace pro PC.

3.1 Funkce bootloaderu

Pro konečné řešení bylo rozhodnuto vybavit bootloader následujícími funkcemi:

- Zápis uživatelské aplikace
- Mazání uživatelské aplikace
- Čtení konfiguračních bytů MCU
- Čtení podpisu MCU
- Zápis konfiguračních bitů paměti programu
- Ověření integrity strojového kódu uživatelské aplikace
- Podpora interního a externího watchdog časovače

Zápis a mazání patří mezi nezbytné funkce, protože jsou podstatou celé aplikace. Čtení konfiguračních bytů a podpisu MCU může na dálku poskytnout cenné informace

o vzdáleném systému, a tak ověřit správnou konfiguraci.

Další funkce si kladou za cíl zvýšení funkční bezpečnosti. Zápis konfiguračních bitů paměti programu dává možnost uzamknout uživatelskou aplikaci a chránit ji tak před přepsáním, vymazáním a případně vyčtením. Ověření integrity strojového kódu uživatelské aplikace dává jednak informaci o správném zapsání a také dovoluje identifikovat poškození obsahu paměti programu a zabránit spuštění poškozeného kódu. Běh takového kódu by mohl způsobit nedefinované chování systému, čemuž se musí nutně zamezit.

Jelikož je bootloader softwarovou záležitostí, je podpora watchdog časovače nezbytnou součástí bootloaderu dovolující jeho správnou činnost v aplikacích, které jsou WDT vybaveny.

3.2 Požadovaný hardware

Ve finální verzi má být úloha vzdálené správy firmware v mikrokontroléru Atmel AVR řešena pro modul *CPU unit*, jehož úplné schéma zapojení se nachází v příloze A, a embedded server NE-4100T ve funkci ethernetového rozhraní.

3.2.1 CPU unit

Deska je osazena mikrokontrolérem ATmega644 [11] doplněným o podpůrné obvody. Těmi jsou monitor napájení kombinovaný s watchdog časovačem typu MB3773 [14] a lineární stabilizátor napětí LP2950CZ-3.3 [15] jako napěťová reference pro AD převodník integrovaný v mikrokontroléru.

Na desce se dále nachází konektor JTAG rozhraní pro připojení programátoru, který slouží k zavedení bootloaderu a počáteční konfiguraci MCU. S dalšími obvody spojují desku dva konektory po dvaceti pinech v uspořádání DIL. Zde jsou vyvedeny všechny vstupně-výstupní porty MCU, signály $\overline{RESET}$ a $\overline{E\_RESET}$, napěťová reference AD převodníku a vstup napájecího napětí.

Taktování mikrokontroléru zajišťuje externí krystal s frekvencí 18,432 MHz, na což je nutné brát ohled při návrhu bootloaderu z hlediska časovaných operací.

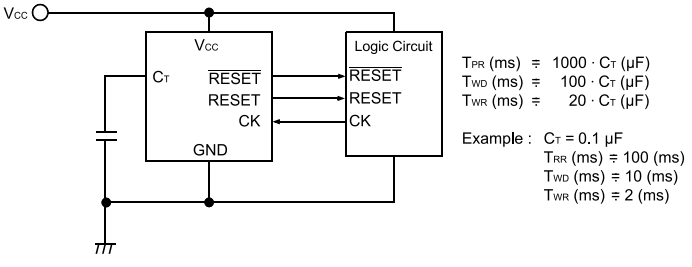
Modul pracuje se stejnosměrným stabilizovaným napětím 5 V, které se při hodinovém kmitočtu nad 10 MHz musí pohybovat v povoleném pracovním rozsahu 4,5 až 5,5 V [11]. Nároky modulu na odběr proudu udává především MCU. Předpokládáme-li maximální povolené proudové zatížení vstupně-výstupních portů, nabývá maximální odběr MCU 200 mA. Monitor napájení MB3773 má deklarován pracovní proud menší než 1 mA při aktivním watchdog timeru [14] a uvažujeme-li minimální zatížení referenčního zdroje pro AD převodník (v řádu jednotek mA), musíme počítat s maximálním odběrem modulu 210 mA.

3.2.2 Monitor napájení a watchdog timer

Integrovaný obvod MB3773 má za úkol detekovat správnou úroveň napájecího napětí a generovat resetovací signál o deklarovaných parametrech. Cílem je zabezpečení požadované činnosti mikrokontroléru jednak ve fázi zapnutí napájení, kdy dochází k náběhu napájecího napětí na požadovanou úroveň, a jednak v průběhu činnosti, kdy může výpadek způsobit pokles napětí pod potřebnou úroveň. V takových případech je nutné držet vstup $\overline{RESET}$ MCU v log. úrovni 0, aby nedošlo k chybnému běhu programu, což by mohlo zapříčinit nesprávnou činnost zařízení.

Druhou částí činnosti obvodu je watchdog časovač, který zabezpečuje reset MCU v případě, že dojde zastavení běhu programu. Příkladem může být například uváznutí v nekonečné smyčce vlivem špatného návrhu software, či hardwarovou chybou.

Na obrázku 3.1 se nachází základní zapojení monitoru napětí MB3773. Vstupem je monitorované napájecí napětí, které musí být shodné resetovaným obvodem, a signál pro nulování watchdog timeru CK. Vstup CK reaguje na sestupnou hranu signálu. Výstupy jsou přímý a negovaný resetovací signál pro hlídaný obvod.



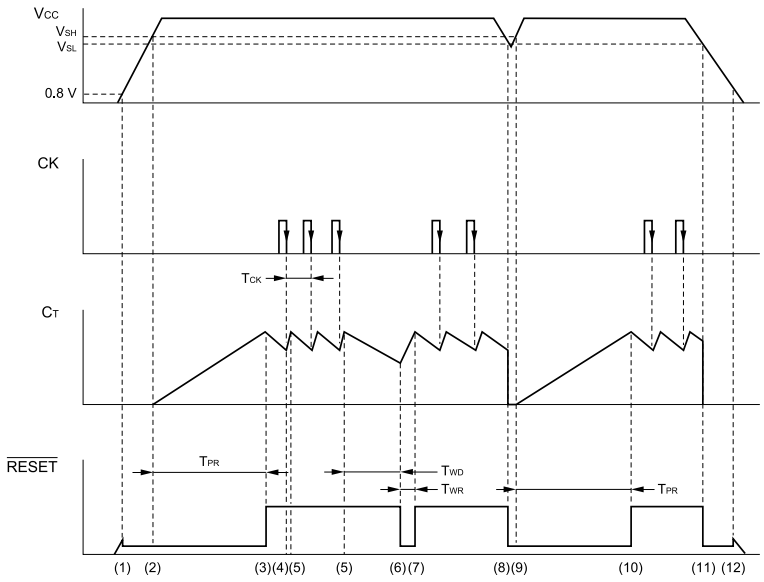
Obrázek 3.1: Základní zapojení MB3773 [14]

Grafy na obrázku 3.2 definují časové průběhy monitoru napětí a watchdog časovače. Délky jednotlivých časových úseků závisí na velikosti kapacity kondenzátoru C_T, který spolu s interním zdrojem proudu tvoří generátor lineárně narůstajícího napětí. Od jeho velikosti jsou pak odvozeny patřičné časové intervaly. Těmi jsou T_{PR}, T_{WD} a T_{WR}.

Interval T_{PR} udává dobu, po kterou je držen hlídaný obvod ve stavu reset po té, co se napájecí napětí ustálilo na požadované úrovni. K tomuto stavu dochází po zapnutí napájení, či poklesu v průběhu činnosti.

Časový úsek T_{WD} definuje dobu vypršení watchdog časovače. Čas se měří od sestupné hrany posledního nulovacího impulzu na vstupu CK. V tomto intervalu je nutné časovač nulovat, jinak dojde k resetování hlídaného zařízení.

Interval T_{WR} odpovídá době, po kterou je držen reset v aktivní úrovni, dojde-li k vypršení watchdog časovače.



Obrázek 3.2: Časové průběhy MB3773 [14]

Signál $\overline{RESET}$, který se nachází na propojovacím konektoru modulu, je přímo spojen s pinem $\overline{RESET}$ mikrokontroléru. Ten ovládá monitor napájení MB3773. Jedná se tedy o výstup resetovacího signálu, jehož aktivní úroveň odpovídá log. úrovni 0. Funkce vstupního signálu $\overline{E\_RESET}$ závisí na konfiguraci zkratovací propojky JP2. Není-li zkratovací propojka osazena, nemá signál vliv. Je-li propojka v poloze *1-2*, přivádí se externí resetovací signál na vstup monitoru napájení, který zajistí vygenerování resetovacího im-

pulzu o deklarované délce. Propojka v poloze *2-3* způsobí přemostění resetovacího obvodu, a tak je signál $\overline{E\_RESET}$ veden přímo na vstup mikrokontroléru.

3.3 Rozbor funkcí a činnosti bootloaderu

Mikrokontroléry Atmel AVR disponují funkcí *Read-While-Write Self-Programming*, která dovoluje za určitých podmínek současně vykonávat kód uložený v paměti programu flash a zároveň tuto paměť po jednotlivých stránkách mazat a zapisovat nová data. Tato vlastnost umožňuje vytvořit speciální část programu, která zajistí potřebné úkony k příjmu kódu hlavního programu prostřednictvím jakýchkoliv periférií MCU a jeho zapsání do paměti programu flash. Tím vznikne program zvaný *bootloader* , zajišťující programování mikrokontroléru v daném zapojení bez nutnosti jeho vyjmutí, či připojení dodatečného hardware. Při vhodném návrhu může bootloader dokonce přepsat sám sebe novou verzí, nebo se úplně vymazat z paměti.

3.3.1 Paměťový prostor

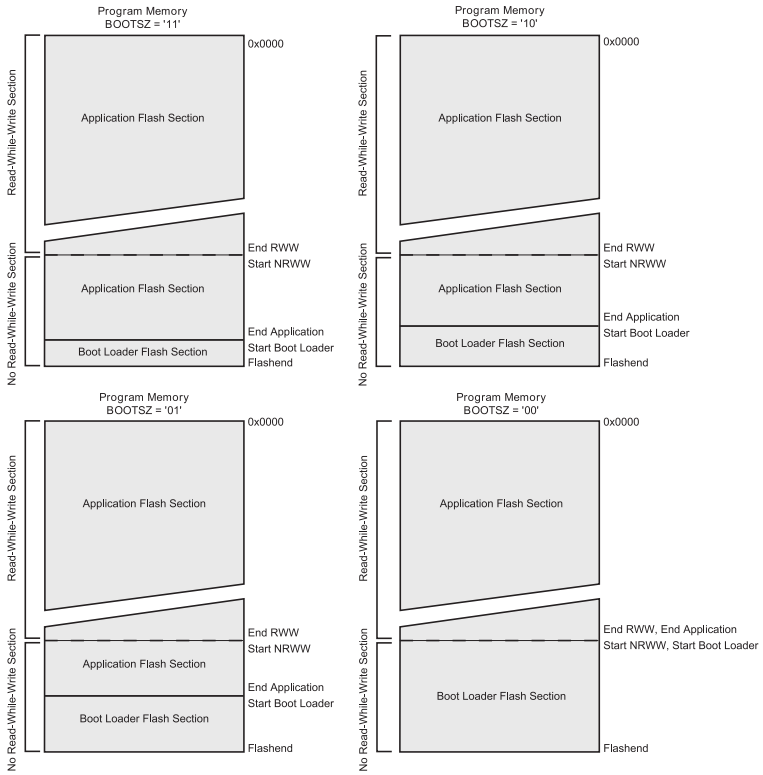
Prostor paměti programu mikrokontrolérů AVR s podporou funkce *Read-While-Write Self-Programming* se skládá z dvou částí. Těmi jsou Read-While-Write (RWW) a No Read-While-Write (NRWW) sekce. Ty se od sebe liší způsobem činnosti MCU v případě, že je programována stránka paměti v rámci těchto sekcí.

Je-li prováděn zápis (mazání) stránky v rámci RWW oblasti, může mikroprocesor vykonávat kód programu uložený v oblasti NRWW. Během zápisu je však zakázáno číst či vykonávat program z oblasti RWW, což by mohlo vést k nedefinovanému stavu mikroprocesoru. Z tohoto důvodu se musí zajistit blokování přerušení v průběhu programování, jsou-li přerušení mapována do sekce RWW.

Je-li vykonávána operace zápisu do sekce NRWW, dojde v jejím průběhu k pozastavení činnosti mikroprocesoru.

Mapování jednotlivých sekcí v paměti udává obrázek 3.3. Oblast RWW vždy začíná na adrese 0000_{hex}, kde se také nachází vektor přerušení RESET, vstupní bod uživatelského programu. Začátek a velikost sekce NRWW závisí na konkrétním MCU a zároveň odpovídá maximální velikosti bootloaderu. Konkrétní velikost sekce bootloaderu definují

bity *BOOTSZ* v rámci konfiguračního bytu fuses low [11]. Rozložení sekcí dle konfigurace uvádí tabulka 3.1. Zde je také patrné, že kód bootloaderu se musí umístit na správné místo v paměti vzhledem ke konfiguraci bitů *BOOTSZ*.



Obrázek 3.3: Rozdělení paměti Flash ATmega644 [11]

BOOTSZ	Boot Size [words]	Pages	Application Section [words]	Boot Loader Section [words]	End Application [words]	Boot Reset [words]
11	512	4	0000_{hex} - $7DFF_{hex}$	$7E00_{hex}$ - $7FFF_{hex}$	$07DF_{hex}$	$7E00_{hex}$
10	1024	8	0000_{hex} - $7BFF_{hex}$	$7C00_{hex}$ - $0FFF_{hex}$	$7BFF_{hex}$	$7C00_{hex}$
01	2048	16	0000_{hex} - $77FF_{hex}$	7800_{hex} - $7FFF_{hex}$	$77FF_{hex}$	7800_{hex}
00	4096	32	0000_{hex} - $6FFF_{hex}$	7000_{hex} - $7FFF_{hex}$	$6FFF_{hex}$	7000_{hex}

Tabulka 3.1: Konfigurace velikosti sekce bootloaderu u ATmega644 [11]

3.3.2 Konfigurační bity paměti programu

Z důvodu zvýšení spolehlivosti jsou mikrokontroléry Atmel AVR vybaveny systémem ochrany kódu v paměti programu flash. Pomocí konfiguračních bitů *Boot Loader Lock Bits* (tabulky 3.2, 3.3) lze nezávisle na obou sekcích paměti nastavit tři stupně zabezpečení, které povolují nebo zakazují operace zápisu a čtení nad těmito sekcemi. Aktivace ochran je možná jakoukoliv metodou programování včetně softwarové instrukcí *SPM*. Je-li zabezpečení nastaveno, lze jej deaktivovat pouze příkazem *Chip Erase* prostřednictvím rozhraní JTAG, paralelním nebo sériovým programátorem. Tím současně dojde k vymazání veškerého obsahu paměti programu flash.

Můžeme tak efektivně ochránit sekci bootloaderu před náhodným poškozením vlivem chybného vykonání instrukce zápisu *SPM*, což by mohlo způsobit nemožnost načtení nové verze uživatelského programu, a tím i nefunkčnost cílového zařízení. Druhou možností je blokování čtení sekce bootloaderu instrukcí *(E)LPM* vykonávané ze sekce uživatelského programu. Tím lze zabránit přečtení kódu bootloaderu, a tak například prolomení ochranných prvků jako je šifrování.

Stejně tak aplikační sekci lze chránit před přepsáním nebo čtením prostřednictvím bootloaderu. Tím lze zamezit dalším změnám uživatelského programu, či jeho zpětné přečtení.

Pro aplikační sekci nebude ve výchozí konfiguraci nastaven žádný mód ochrany, protože bootloader musí být schopen v této oblasti zapisovat i číst (z důvodu verifikace). Pro bootloader sekci bude nastaven mód 3, kde je zakázán zápis i čtení bootloader sekce aplikací. Zákaz zápisu zabrání náhodnému poškození bootloaderu a čtení není pro aplikaci potřebné.

3.3.3 Spuštění bootloaderu

Vzhledem k softwarové podstatě bootloaderu, musí být zajištěno vykonání jeho kódu v případě, že má být zapsán program do aplikační sekce paměti programu. Ve výchozím nastavení leží vektor přerušení RESET, a tak vstupní bod programu, na adrese 0000_{hex}. Vstupní bod bootloaderu se však nachází na adrese, která závisí na nastavení bitů *BOOTSZ* dle tabulky 3.1.

Mód	BLB02	BLB01	Funkce
1	1	1	Žádná omezení.
2	1	0	Zakázán zápis do aplikační sekce.
3	0	0	Zakázán zápis a čtení aplikační sekce bootloaderem. Zakázána přerušení, jsou-li vektory umístěny v sekci bootloaderu a vykonáván kód z aplikační sekce.
4	0	1	Zakázáno čtení aplikační sekce bootloaderem. Zakázána přerušení, jsou-li vektory umístěny v sekci bootloaderu a vykonáván kód z aplikační sekce.

Tabulka 3.2: Konfigurační bity paměti programu - aplikační sekce [11]

Mód	BLB12	BLB11	Funkce
1	1	1	Žádná omezení.
2	1	0	Zakázán zápis do sekce bootloaderu.
3	0	0	Zakázán zápis. Zakázáno čtení bootloaderu z aplikační sekce. Zakázána přerušení, jsou-li vektory umístěny v aplikační sekci a vykonáván kód bootloaderu.
	0	1	Zakázáno čtení bootloaderu z aplikační sekce. Zakázána přerušení, jsou-li vektory umístěny v aplikační sekci a vykonáván kód bootloaderu.

Tabulka 3.3: Konfigurační bity paměti programu - sekce bootloaderu [11]

Toto chování lze ovlivnit konfiguračním bitem *BOOTRST*. Je-li nastaven na hodnotu 1 (výchozí nastavení), leží vektor přerušení RESET na adrese 0000_{hex}. Je-li nastaven na hodnotu 0, dojde k přesměrování přerušení RESET na adresu bootloaderu, a to v souladu s nastavením bitů *BOOTSZ*.

Pro účely bootloaderu bude využito přesměrování vektoru přerušení RESET, čímž se zajistí vždy jako první vykonání kódu bootloaderu. Proto můžeme po startu MCU softwarově rozhodnout, je-li splněna podmínka pro vstup do hlavní části bootloaderu, nebo bude-li spuštěn uživatelský program.

Podmínku pro spuštění bootloaderu lze v tomto projektu v zásadě definovat dvěma způsoby. Jednak příkazem na úrovni komunikačního protokolu na rozhraní Ethernet, nebo na základě stavu DIO výstupu převodníku NE-4100T.

Vzhledem k časovému nedeterminismu komunikace prostřednictvím Ethernetu by muselo být definováno časové okno, ve kterém by program čekal na přijetí příkazu, což by mělo za následek zpoždění startu uživatelského programu v řádu jednotek sekund. Z tohoto důvodu byla zvolena varianta využívající stavu DIO signálu. Výhodou je vyhodnocení v rámci několika instrukcí, nevýhodou především obsazení jednoho vstupu MCU, který bude propojen s převodníkem NE-4100T.

3.3.4 Vektory přerušení

Umístění vektorů přerušení závisí na hodnotě bitu *IVSEL* v konfiguračním registru *MCUCR*. Vektory přerušení lze tedy přesměrovat z aplikační sekce do sekce bootloaderu, která je dostupná i v průběhu vykonávání operace *SPM* na aplikační sekci. Tato vlastnost dovoluje návrh bootloaderu s využitím přerušení.

V případě, že jsou vektory přerušení umístěny v aplikační sekci, je nezbytné během operace *SPM* veškerá přerušení zakázat. V opačném případě by mohlo dojít k nedefinovanému stavu MCU v důsledku o čtení kódu přerušení z aplikační sekce.

IVSEL	**Adresa vektorů přerušení**
0	0002_{hex}
1	Adresa počátku bootloaderu (tabulka 3.1) + 0002_{hex}

Tabulka 3.4: Konfigurace umístění vektorů přerušení [11]

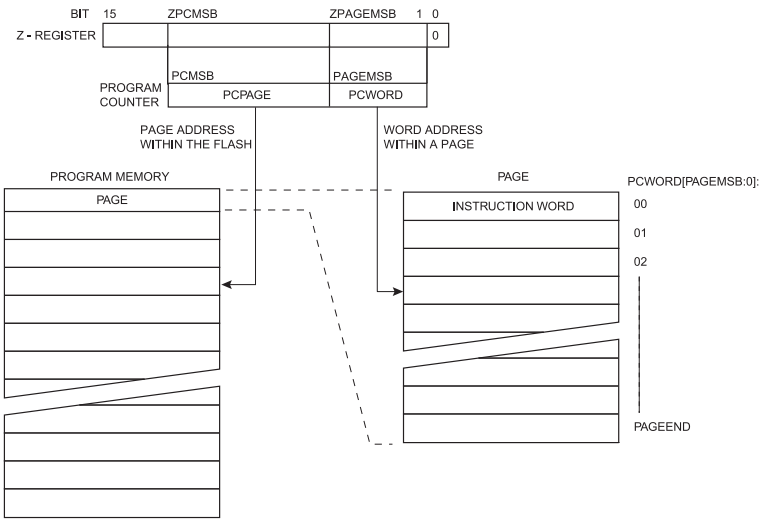
3.3.5 Instrukce *SPM*, registr *SPMCSR*

Instrukce *SPM* zprostředkovává veškeré operace s úpravami obsahu paměti programu. Těmi jsou mazání stránky paměti, zápis stránky paměti a nastavení konfiguračních bitů paměti programu. Při vhodném nastavení registru *SPMCSR* je navíc možné instrukcí *LPM* čtení konfiguračních bytů, identifikačních bytů MCU a kalibračního bytu RC oscilátoru.

Její chování definuje osmibitový konfigurační a stavový registr *SPMCSR*, obsah šestnáctibitového adresovacího registru *Z* a datových registrů *R0* a *R1*.

V průběhu mazání či zápisu stránky paměti programu probíhá adresování dle obrázku 3.4. Adresní registr *Z* se fyzicky skládá ze dvou osmibitových registrů *R30* a *R31*. Při adresování jej lze v zásadě rozdělit na dvě části. Horní bity udávají adresu programované stránky a spodní bity adresu bytu uvnitř stránky. Počet bitů připadajících na adresu bytu uvnitř stránky závisí na její velikosti dle konkrétního MCU, stejně tak počet bitů adresy stránky vyplývá z velikosti paměti programu.

Datové registry *R0* a *R1* jsou využívány při plnění stránky novými daty, kde reprezentují jedno slovo. Registr *R0* dále definuje hodnoty při nastavování konfiguračních bitů a obsahuje výsledek při čtení.



Obrázek 3.4: Adresování instrukce *SPM* [11]

Časování instrukce *SPM* při zápisu udává tabulka 3.5. Časové intervaly nezávisí na frekvenci hodinového kmitočtu MCU, jelikož časování zápisu vychází z frekvence in-

terního RC oscilátoru. V jeden čas může být aktivní pouze jedna operace *SPM*, a proto je nezbytné dodržet tento časový interval, aby nedošlo k poškození obsahu paměti. K indikaci ukončení operace instrukce *SPM* slouží bit *SPMEN* v registru *SPMCSR*. Ten při hodnotě log. 1 inicializuje operaci a vrací se do log. 0 až po jejím dokončení. Kontrolou stavu bitu *SPMEN* tak lze minimalizovat dobu čekání na dokončení operace, a tím dosáhnout co nejvyšší rychlosti zápisu.

S bitem *SPMEN* má návaznost přerušení instrukce *SPM*. Ke generování tohoto přerušení dochází je-li bit *SPMEN* v log. úrovni 0, což indikuje dokončení operace *SPM*. Přerušení je maskováno bitem *SPMIE*. Dále musí být bráno v úvahu, kde leží vektory přerušení (odstavec 3.3.4), protože zápisem nebo mazáním stránky dojde k zablokování čtení z aplikační sekce.

Operace	**Min. čas [ms]**	**Max. čas [ms]**
Mazání stránky, zápis stránky, nastavení (zápis) konfiguračních bitů	3,7	4,5

Tabulka 3.5: Časování zápisu instrukcí *SPM* [11]

3.3.6 Mazání stránky paměti programu

Vymazání stránky paměti dané adresou v registru *Z* se zajistí voláním instrukce *SPM*, při registru *SPMCSR* s nastavenými bity *SPMEN* (Store Program Memory Enable), *PGERS* (Page Erase) a volitelně bitem *SPMIE* (SPM Interrupt Enable) v případě využití přerušení. Adresa v registru *Z* udává adresu stránky, a tak mají vliv pouze bity *PCPAGE* (obrázek 3.4). Vývojový diagram této operace se nachází na obrázku 3.5(b). Po úspěšném provedení jsou jednotlivá slova vymazané stránky nastavena na hodnotu $FFFF_{hex}$.

Dobu trvání operace mazání uvádí tabulka 3.5, přičemž při mazání více stránek se musí čekat na dokončení aktuálně prováděné operace před započetím další, jak je uvedeno v odstavci 3.3.5.

Během mazání nedojde k narušení obsahu dočasného bufferu pro zápis stránky, což je důležité, je-li požadována změna pouze určité části stránky. V takovém případě se nejdříve načte její obsah do dočasného bufferu, provede se instrukce mazání a modifikovaná data se zapíší zpět instrukcí zápisu.

3.3.7 Zápis stránky paměti programu

Operace zápisu stránky se skládá ze dvou kroků. Data pro zápis musí být nejdříve načtena prostřednictvím instrukce *SPM* do interního bufferu, jehož velikost odpovídá velikosti stránky. V druhé fázi se instrukcí *SPM* inicializuje vlastní zápis dat. Postup popisuje vývojový diagram na obrázku 3.5(a).

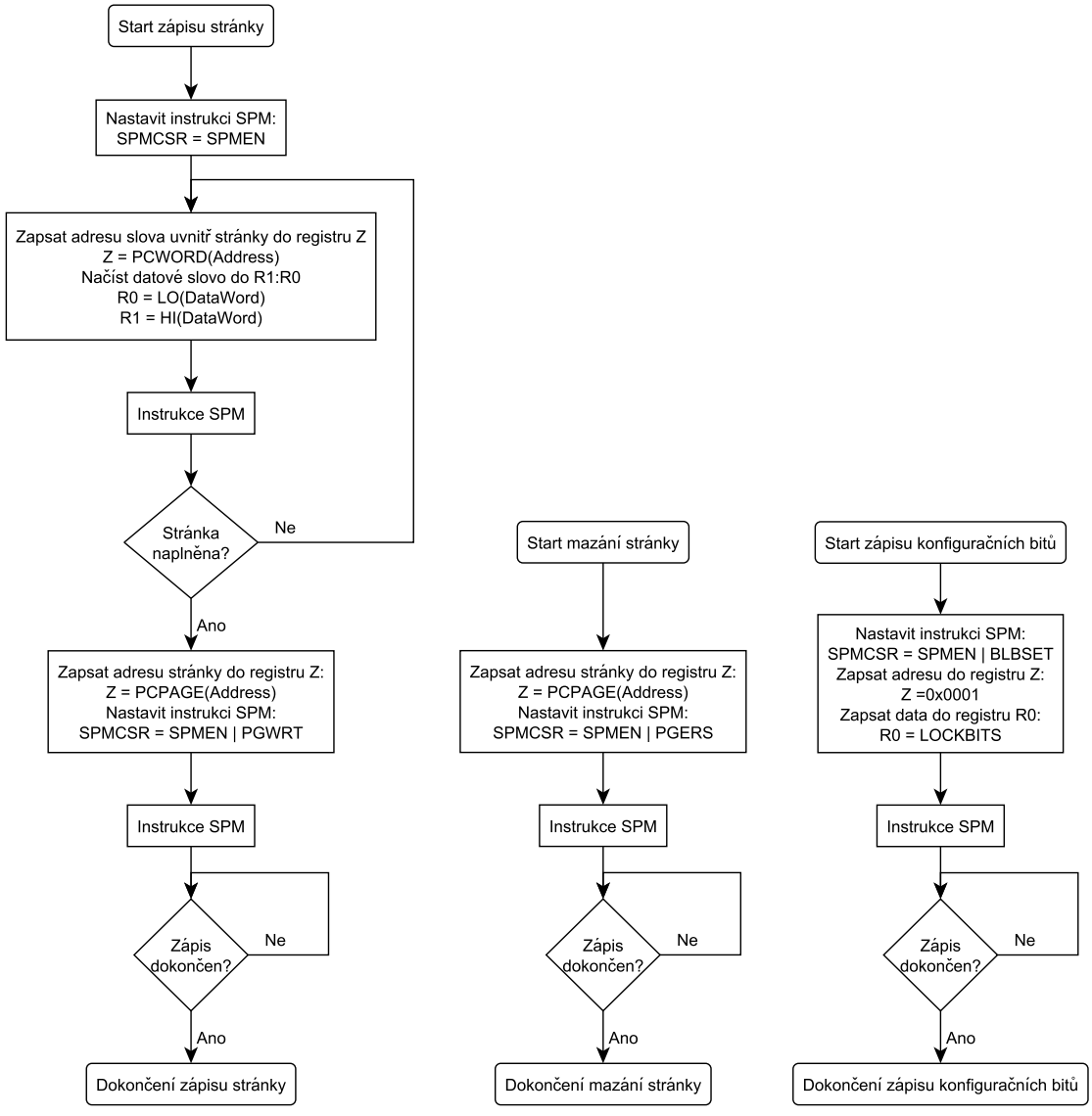
Jako první krok se nakonfiguruje instrukce *SPM* pro plnění interního bufferu, a to zápisem hodnoty, která odpovídá nastavenému bitu *SPMEN*, do registru *SPMCSR*. Adresovací registr *Z* se naplní adresou slova uvnitř stránky, přičemž vliv mají pouze bity *PCWORD* (obrázek 3.4), a do registrů *R0* a *R1* se umístí požadované datové slovo. Následným voláním instrukce *SPM* dojde k přenesení obsahu registrů *R0* a *R1* do dočasného bufferu, který je obrazem zapisované stránky. Tímto postupem lze naplnit celý buffer, nebo jen jeho část. Neinicializované části odpovídají vymazaným oblastem paměti flash a mají hodnotu FFFF_{hex}.

Naplněný dočasný buffer můžeme zapsat do paměti programu opět instrukcí *SPM*, a to při hodnotě konfiguračního registru *SPMCSR* s nastavenými bity *SPMEN*, *PGWRT* (Page Write) a volitelně *SPMIE*. Adresa stránky, do které má být obsah bufferu zapsán, v takovém případě odpovídá hodnotě uložené v registru *Z*. Vliv mají pouze bity *PCPAGE* (obrázek 3.4), ostatní bity (adresa uvnitř stránky) musí být nastaveny na hodnotu nula.

Časování zápisu je obdobné jako u operace mazání. Rozbor způsobů detekce ukončení byl proveden v odstavci 3.3.5 a je nutné jej dodržet. Spolu s dokončením zápisu dojde navíc k vymazání obsahu dočasného bufferu, a proto není možné jej opakovaně zapsat do více stránek.

3.3.8 Zápis konfiguračních bitů paměti programu

Vývojový diagram operace je na obrázku 3.5(c). K zápisu konfiguračních bitů instrukcí *SPM* dojde při hodnotě registru *SPMCSR* s nastavenými bity *SPMEN*, *BLBSET* (Boot Lock Bit Set) a adrese 0001_{hex} v registru *Z*. Dle požadavků aplikace může být nastaven i bit *SPMIE*. Zapisovaná data odpovídají hodnotě registru *R0*, jehož mapování na konfigurační bity udává tabulka 3.7. Dokončení operace je indikováno stejně jako u zápisu či mazání stránky paměti.



(a) Čtení konfiguračních bytů (b) Čtení podpisu MCU (c) Zápis konfiguračních bitů paměti programu

Obrázek 3.5: Vývojové diagramy užití instrukce *SPM*

3.3.9 Čtení konfiguračních bytů

Čtení konfiguračních bytů probíhá prostřednictvím instrukce *LPM* (*Load Program Memory*) s nastavenými bity *SPMEN* a *BLBSET*. Adresa v registru *Z* dle tabulky 3.6 pak definuje, jaká hodnota bude načtena. Výsledek je uložen do registru v souladu s parametry instrukce *LPM* [12]. Mapování jednotlivých bitů do cílového registru udává tabulka 3.7.

Funkce	**Registr *Z***
Fuse Low	0000_{hex}
Lock	0001_{hex}
Extended Fuse	0002_{hex}
Fuse High	0003_{hex}

Tabulka 3.6: Hodnoty registru *Z* pro čtení konfiguračních bytů [11]

Bit Rx:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Lock	1	1	BLB12	BLB11	BLB02	BLB01	LB2	LB1
Fuse Low	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0
Fuse High	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0
Extended Fuse	EFB7	EFB6	EFB5	EFB4	EFB3	EFB2	EFB1	EFB0

Tabulka 3.7: Mapování konfiguračních bitů do registru [11]

3.3.10 Čtení podpisu MCU a kalibrace RC oscilátoru

Čtení podpisu mikrokontroléru a kalibračního bytu RC oscilátoru je obdobné s bodem 3.3.9 s tím rozdílem, že registru *SPMCSR* musí být nastaveny bity *SPMEN* a *SIGRD* (Signature Row Read). Adresy dostupných údajů pak udává tabulka 3.8.

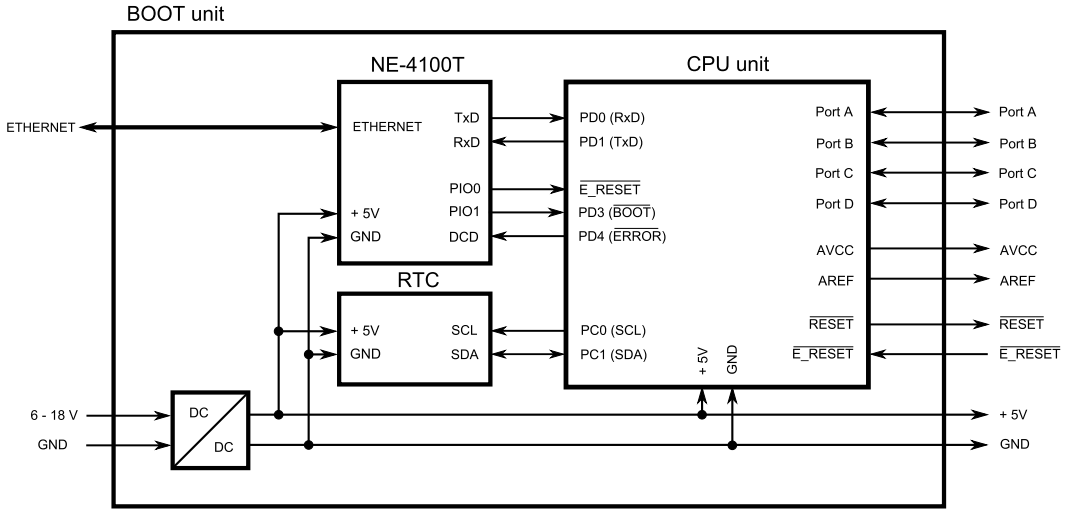
3.4 Hardware

Z hlediska vybrané varianty řešení prostřednictvím bootloaderu se hardwarová část redukuje na napájecí zdroj a obvody zajišťující propojení embedded serveru NE-4100T s modulem CPU unit. Jako dodatečný požadavek bylo požadováno doplnění zapojení o hodiny

Funkce	Registr Z
Signature Byte 1	0000_{hex}
RC Oscilator Calibration Byte	0001_{hex}
Signature Byte 2	0002_{hex}
Signature Byte 3	0004_{hex}

Tabulka 3.8: Hodnoty registru Z pro čtení podpisu MCU a kalibrace RC oscilátoru [11]

reálného času. Celé zapojení tak popisuje blokové schéma na obrázku 3.6.



Obrázek 3.6: Blokové schéma zapojení modulu BOOT unit

Mezi požadavky je navíc snaha o co největší rozsah pracovních teplot. Pokud možno má být dosaženo průmyslových požadavků od -40 do $85\,^{\circ}\mathrm{C}$, což se promítne na volbě použitých součástek.

Napájecí zdroj tvořený pulzním snižujícím měničem vytváří napětí $5\,\mathrm{V}$ a napájí server NE-4100T, modul CPU unit i obvod hodin reálného času. Napájení je dále vyvedeno na konektor modulu BOOT unit. Tomu musí odpovídat proudové dimenzování zdroje.

Embedded server NE-4100T a modul CPU unit propojují signály RxD, TxD sériového rozhraní pro přenos dat. Ty jsou sdílené pro uživatelskou aplikaci a bootloader. Pro účely bootloaderu dále slouží signály $\overline{E\_RESET}$, $\overline{BOOT}$ a $\overline{ERROR}$.

Pro komunikaci s obvodem hodin reálného času byla zvolena sběrnice I2C. Důvodem je především hardwarová podpora na straně mikrokontrolérů AVR a o proti SPI nižší počet vodičů, který potřebujeme k propojení. Mezi výhody I2C patří i možnost multimaster

konfigurace, a tak může být obvod jednoduše sdílen dalšími zařízeními typu master na sběrnici.

K propojení modulu BOOT unit s okolím slouží signály vyvedené na vstupně výstupní konektory. Mezi ně patří vstup stejnosměrného napájecího napětí, rozhraní Ethernet a veškeré signály modulu CPU unit v nezměněné podobě.

3.4.1 Změny modulu CPU unit

Oproti původní verzi modulu CPU unit musela být provedena jedna změna v zapojení. V původním návrhu byl signál pro resetování externího watchdog časovače propojen na signál PD0 mikrokontroléru. Signál PD0 však zároveň sdružuje funkci vstupu přijímače rozhraní UART, který je potřebný ke komunikaci.

Počítat s resetováním WDT prostřednictvím komunikace na sériové lince není možné, protože nelze zajistit neustálý tok dat. A to především z důvodu komunikace protokolem TCP, kdy vlivem interních bufferů může dojít ke značné časové prodlevě v případě opakovaného přenosu při vzniku chyby.

Z těchto důvodů byl resetovací signál WDT přepojen na pin mikrokontroléru PD2.

3.4.2 Propojení NE-4100T a modulu CPU unit

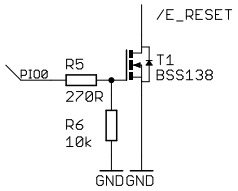
Vstupně-výstupní signály serveru NE-4100T disponují logickými úrovněmi TTL, které spadají do odpovídajících rozsahů logických úrovní mikrokontroléru ATmega644.

Dle doporučeného zapojení [16] jsou datové signály RxD a TxD přímo propojeny bez dalších dodatečných obvodů, které by upravovaly logické úrovně, pouze s ohledem na směr toku dat.

Signály $\overline{BOOT}$ a $\overline{ERROR}$ byly doplněny o zvedací rezistory R1 a R2, které definují logické úrovně v průběhu resetu a inicializace. Sériové rezistory R3 a R4 omezují maximální proud, a tím strmost hran. Hodnoty byly převzaty z doporučeného zapojení [16].

Ovládání signálu $\overline{E\_RESET}$ počítá s osazením zkratovací propojky JS1 na modulu CPU unit do pozice *1-2* (funkce popsána v odstavci 3.2.2). Zapojení resetovacího tlačítka a případně dalších externích obvodů na signálu $\overline{E\_RESET}$ vyžaduje výstup typu otevřený kolektor. Při tvrdé logické úrovni 1 by došlo při stisku tlačítka ke konfliktu a proudovému

přetížení výstupu serveru NE-4100T. Jeho výstupy nedovolují konfiguraci typu otevřený kolektor. Jistou možností by bylo přepínat funkci příslušného pinu mezi vstup (obdoba otevřeného kolektoru) a výstup v logické úrovni 0, což se ale nejeví jako ideální řešení. Výstup byl proto doplněn o pomocný tranzistor T1, který vytváří požadovaný otevřený kolektor.



Obrázek 3.7: Ovládání signálu $\overline{E\_RESET}$

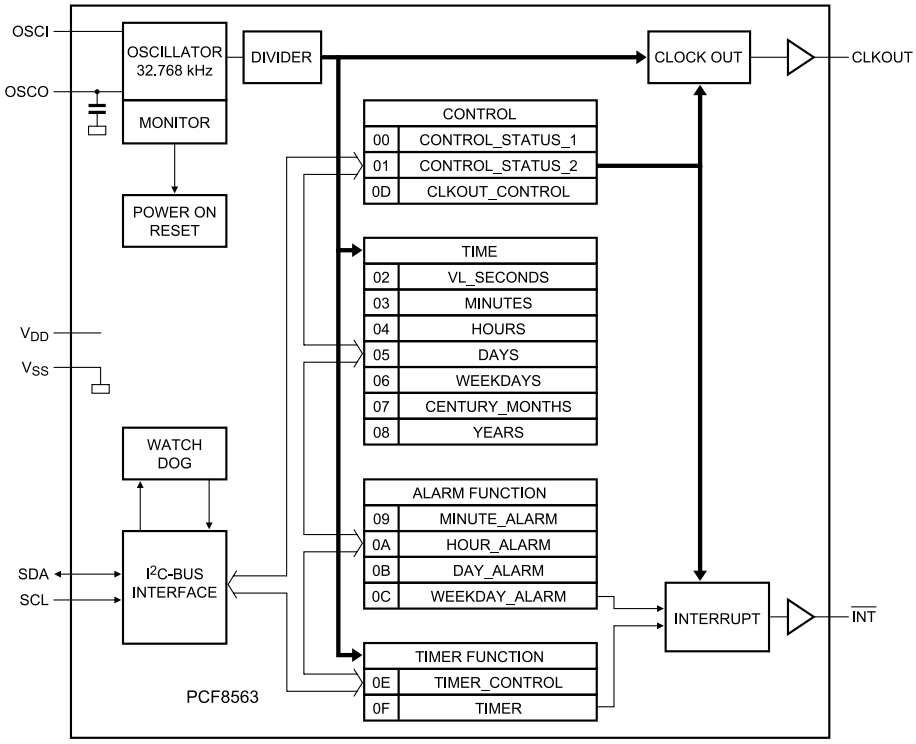
Protože pomocný obvod pracuje ve statickém režimu, je kladen důraz především na nízkou spotřebu v ustálených stavech. Z tohoto důvodu byl zvolen unipolární tranzistor MOSFET s kanálem N. Dalším požadovaným parametrem je nízké prahové napětí U_{GS}. Dle těchto požadavků a dostupnosti byl vybrán typ BSS138. Zapojení pomocného obvodu se nachází na obrázku 3.7. Rezistor R6 definuje klidovou úroveň, rezistor R5 omezuje maximální proud hradlem při přechodových jevech.

3.4.3 Hodiny reálného času

Z hlediska ceny a dostupnosti byl pro hodiny reálného času vybrán osvědčený obvod PCF8563 [17], jehož blokové schéma se nachází na obrázku 3.8. Disponuje širokým rozsahem napájecího napětí 1,8 až 5,5 V, což vyhovuje napájecímu napětí 5 V a zálohovacímu napětí 3 V ze standardní lithiové baterie. Pro zálohování z baterie je výhodou i nízká spotřeba 0,25 μA při napětí 3 V. Spolehlivou funkci zaručují monitorovací a resetovací obvody typu watchdog.

Specifikace rozhraní sběrnice I2C dovolují využití plné rychlosti při taktovacím kmitočtu 400 kHz. Sedmibitová slave adresa zařízení je pevně definována na hodnotu 51_{hex}.

Poskytované informace jsou rok, měsíc, den, den v týdnu, počet hodin, minut, sekund a stavový bit *VL*, který potvrzuje validitu uvedených údajů. Časovou základnu tvoří krystalový oscilátor s externím krystalem o frekvenci 32768 Hz. Přídavné funkce jako výstup

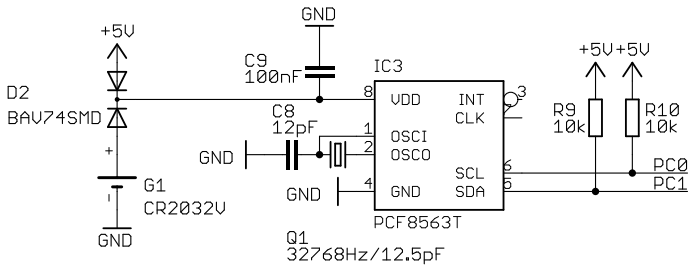


Obrázek 3.8: Blokové zapojení hodin reálného času PCF8563 [17]

referenčního kmitočtu, nebo programovatelné přerušení nejsou v tomto zapojení využity.

Schéma zapojení bloku hodin reálného času se nachází na obrázku 3.9. Dvojitá dioda D2 zajišťuje zálohování chodu hodin při vypnutém napájecím napětí z lithiové baterie typu CR2032 o jmenovitém napětí $3\,\mathrm{V}$ s nízkým samovybíjením. U diody D2 je kladen důraz především na nízký závěrný proud, který způsobuje vybíjení záložní baterie. Z těchto důvodů byl zvolen typ BAV74 se závěrným proudem $30\,\mathrm{nA}$ při napětí $25\,\mathrm{V}$ a teplotě přechodu $25\,^{\circ}\mathrm{C}$ [19]. Keramický kondenzátor C9 s kapacitou $100\,\mathrm{nF}$ slouží k blokování napájecího napětí a měl by být umístěn co nejblíž k vývodům integrovaného obvodu IC3.

Rezistory R9 a R10 jsou zvedací odpory sběrnice I2C. Protože není předem známo využití modulu, počet připojených zařízení na sběrnici a ani její délka, není tím pádem známa kapacita sběrnice. Vzhledem k modulárnímu řešení se dá předpokládat, že kapacita bude relativně vysoká. Uvažujeme-li průměrnou kapacitu $7\,\mathrm{pF}$ na zařízení (maximální kapacita deklarovaná u PCF8563 je $7\,\mathrm{pF}$ [17], u ATmega644 $10\,\mathrm{pF}$ [11]) a kapacitu sběrnice



Obrázek 3.9: Schéma zapojení hodin reálného času

10 pF vychází pro tři zařízení a frekvenci 400 kHz hodnota [18]:

$$R_{9,10} = \frac{t_r}{0{,}8473 \cdot C_b} = \frac{300 \cdot 10^{-9}}{0{,}8473 \cdot 34 \cdot 10^{-12}} \approx 10\,\mathrm{k\Omega} \tag{3.1}$$

Frekvenci integrovaného oscilátoru Piercova typu udává krystal Q1 o frekvenci 32768 Hz doplněný o zatěžovací kapacitu C8. Dle požadavků na co nejvyšší teplotní rozsah byl vybrán SMD krystal MC-306 [20] s teplotním rozsahem -40 až $85\,^\circ\mathrm{C}$ a zatěžovací kapacitou $C_L = 12{,}5\,\mathrm{pF}$. Bohužel se v tomto teplotním rozsahu nepodařilo najít jiný dostupný krystal s nižší zatěžovací kapacitou, což má za následek vyšší spotřebu [18].

Protože nejsou kladeny žádné speciální požadavky na přesnost hodin reálného času, byl kondenzátor C8 zvolen jako pevný s kapacitou $C8 = C_L = 12{,}5\,\mathrm{pF} \approx 12\,\mathrm{pF}$ a materiálem NPO, čímž za cenu menší přesnosti odpadá nutnost kalibrace. Ta by vyžadovala ruční nastavení frekvence měřitelné na výstupním pinu CLK na přesnou hodnotu.

3.4.4 Napájecí zdroj

Na napájecí zdroj jsou kladeny vysoké nároky především v ohledu rozsahu vstupního napětí a pracovních teplot. Požadovaný rozsah vstupních napětí odpovídá hodnotám 6 až 18 V. Teplotní rozsah by měl splňovat průmyslové požadavky od -40 do $85\,^\circ\mathrm{C}$.

Jak bylo popsáno v úvodu, musíme u embedded serveru NE-4100T počítat s maximálním proudem 290 mA a u modulu CPU unit s maximálním odběrem 210 mA. Tím dostáváme maximální proud dodávaný zdrojem 500 mA.

Z těchto parametrů je patrné, že při použití lineárního regulátoru bude velký problém s chlazením. Pro nejhorší variantu dostáváme výkonovou ztrátu 6,5 W. Pro napětí 12 V ve středu pracovního rozsahu a odhadovanou efektivní hodnotu proudu 400 mA dostáváme ztrátový výkon 2,8 W.

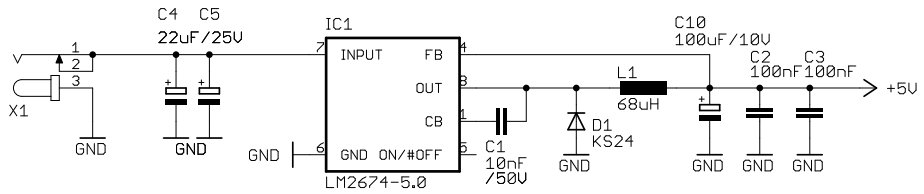
U varianty snižujícího měniče vzniká problém především s dodržením spodní hranice napájecího napětí 6 V. U běžně dostupných obvodů s výstupním napětím 5 V bývá minimální vstupní napětí 6,5 V.

Vzhledem k požadavku na pracovní teplotní rozsah byla zvolena varianta snižujícího měniče. Lineární regulátor by kladl velké nároky na rozměry chladiče a dosahoval by nízké účinnosti.

Na základě těchto údajů byl zvolen obvod LM2674-5.0 [21]. Jedná se o integrovaný snižující měnič pracující s pevnou frekvencí 260 kHz, rozsahem vstupního napětí 6,5 až 40 V a garantovaným výstupním proudem 500 mA. Při vstupním napětí 12 V dosahuje účinnosti 90%.

Na obrázku 3.10 se nachází schéma zapojení použitého snižujícího měniče. Při volbě hodnot součástek je třeba začít u indukčnosti cívky L1. Ta je definována maximálním vstupním napětím a maximálním proudem. Z grafu ([21], figure 5) tak dostáváme hodnotu indukčnosti L1 68 μH. Pro volbu konkrétního typu cívky je v zájmu účinnosti požadován co nejmenší ohmický odpor vinutí. Z hlediska minimalizace elektromagnetického rušení jsou pak vhodné typy s uzavřeným magnetickým obvodem. Na základě těchto požadavků byla vybrána cívka MATSUTA SC75F-680 [22].

Dalším krokem návrhu je volba kapacity výstupního kondenzátoru C10. Pro indukčnost 68 μH vychází dle tabulky ([21], figure 10) hodnota 100 μF pro napětí 10 V. U kondenzátoru se navíc požaduje malá vnitřní indukčnost, a proto nevyhovují běžné elektrolytické kondenzátory a bude použit kondenzátor tantalový.



Obrázek 3.10: Schéma zapojení napájecího zdroje

V třetím kroku přichází na řadu volba diody D1. Ta přebírá vedení proudu zátěží ve chvíli, kdy dojde k rozepnutí tranzistoru uvnitř integrovaného obvodu IC1. Indukčnost cívky L1 se snaží udržet směr a velikost proudu zátěží. Jedinou možností je, že dojde ke změně polarity napětí na cívce L1 a proud se uzavře přes diodu D1. Vzhledem

k pracovní frekvenci 250 kHz musí být dioda rychlá a z požadavku na co nejmenší ztráty (nejvyšší účinnost) vyplývá co nejnižší napětí v propustném směru. Tyto požadavky vedou na Schottkyho diodu. Proudové dimenzování dle [21] odpovídá maximálnímu proudovému omezení obvodu LM2674, které má hodnotu 1,25 A. Závěrné napětí musí být minimálně 1,2krát větší než maximální vstupní napětí. Na základě těchto hodnot byla vybrána dioda SK24 se závěrným napětím 40 V a jmenovitým proudem v propustném směru 2 A. Úbytek napětí v propustném směru je menší než 0,4 V při proudu 0,5 A.

V posledním kroku se volí kondenzátor na vstupu měniče. Napěťové dimenzování musí odpovídat hodnotě minimálně $1{,}25 \cdot U_{in,max}$. Konkrétní typ kondenzátory se volí dle proudové zatížitelnosti. Ta by měla dle [21] odpovídat přibližně 1/2 výstupního proudu. Bohužel u dostupných kondenzátorů není tento parametr uveden. Byla zvolena paralelní kombinace dvou tantalových kondenzátorů C4 a C5 s kapacitou 22 μF pro napětí 25 V.

Kondenzátor C1, který spolu s vnitřními obvody IC1 napomáhá k rychlejšímu otevření výkonových tranzistorů měniče, má dle doporučení [21] hodnotu 10 nF a je dimenzován na napětí 50 V.

3.4.5 Vstupy a výstupy

Na obrázku 3.4.5 se nachází zapojení vstupně-výstupních konektorů modulu BOOT unit. V levé části leží konektory X5 a X6, což jsou dutinkové lišty pro osazení modulu CPU unit. Kolíkové lišty po stranách modulu reprezentují konektory X4 a X7. Ze schématu je patrné, že bylo zachováno rozložení vývodů jako u modulu CPU unit a dostupné jsou veškeré signály poskytované tímto modulem.

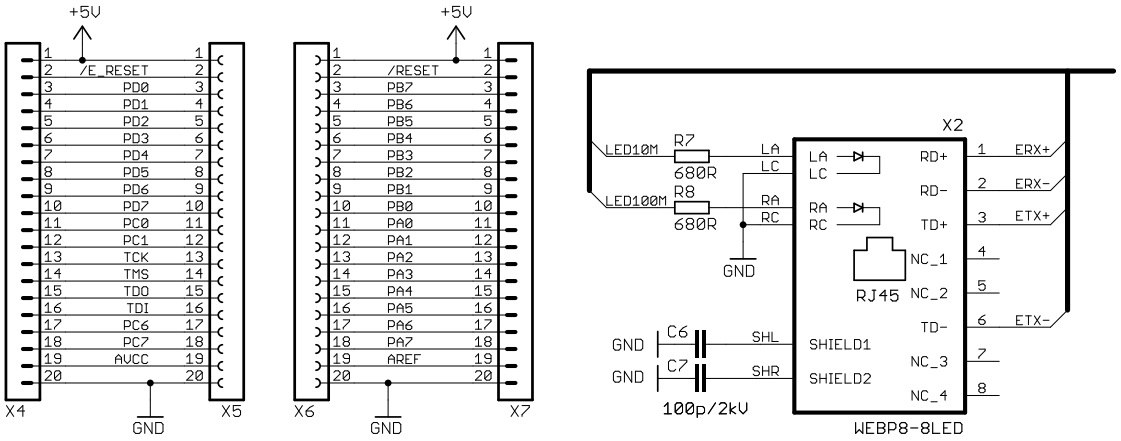
Na konektoru je taktéž vyvedeno napětí 5 V pro případné napájení dalších obvodů. Zde však musí být brán ohled na maximální proudové zatížení zdroje, jak bylo navrhnuto v bodě 3.4.4.

Pravá část schématu zobrazuje zapojení ethernetového konektoru RJ-45. V něm jsou integrovány dvě signalizační LED diody, které udávají aktuální stav ethernetového rozhraní. Konektor byl zvolen typ RJHS-5381 [23] s nízkopříkonovými LED diodami a stíněním. Parametry diod jsou 2,1 V a 2 mA v propustném směru. Výstupní napětí serveru NE-4100T pro LED diody odpovídá hodnotě 3,3 V. Z těchto parametrů lze určit

hodnotu rezistorů R7 a R8:

$$R_{7,8} = \frac{U_{out} - U_d}{I_d} = \frac{3{,}3 - 2{,}1}{2 \cdot 10^{-3}} \approx 680\,\Omega \tag{3.2}$$

Kondenzátory C6 a C7 mají za úkol vyrovnávat elektrostatické napětí (ESD), které vzniká na stínění kabelu. Kapacita kondenzátorů je $100\,\mathrm{pF}$ a napěťové dimenzování musí odpovídat hodnotě $2\,\mathrm{kV}$, což vychází z doporučeného zapojení výrobce [16].



(a) Vstupně-výstupní signály (b) Konektor rozhraní Ethernet

Obrázek 3.11: Schéma zapojení vstupů a výstupů

3.4.6 Deska plošných spojů

Modul BOOT unit je realizován na jedné dvouvrstvé prokovené desce plošných spojů o rozměrech 85 x $58\,\mathrm{mm}$. Při návrhu byly zohledněny technologické požadavky zvoleného výrobce, a proto deska vyhovuje následujícím parametrům:

- minimální šířka spoje $0{,}3\,\mathrm{mm}$
- minimální šířka izolační mezery $0{,}2\,\mathrm{mm}$
- minimální průměr vrtání $0{,}6\,\mathrm{mm}$

Na materiál desky nejsou kladeny žádné speciální nároky, a proto byl vybrán standardní materiál FR4 o tloušťce $1{,}5\,\mathrm{mm}$. Aby byla garantována pájitelnost, je povrch desky upraven metodou žárového cínování HAL.

Většina součástek je v provedení SMD. Rozměr rezistorů a kondenzátorů odpovídá velikosti 1206. Tato volba vychází ze snahy využití stejných druhů součástek, jako jsou použity v modulu CPU unit.

Výjimky tvoří rezistory R3, R4 a kondenzátory C6, C7. U rezistorů R3 a R4 bylo zvoleno klasické provedení o rozměru 0207, a to z důvodu snazší realizace spojů na DPS. Kondenzátory C6 a C7 jsou v klasickém provedení, aby bylo možné zajistit napěťové dimenzování na 2 kV. S ohledem na toto napětí jsou proto u patřičných spojů použity větší izolační mezery.

Z hlediska zásad pro návrh DPS snižujícího měniče jsou kondenzátory C1, C4, C5, C10, dioda D1 a cívka L1 umístěny co nejblíže vývodům integrovaného obvodu IC1. Důvody jsou minimalizace ohmického odporu a indukčnosti spojů. Kvůli těmto požadavkům byly navíc u patřičných spojů použity větší tloušťky.

Lithiová baterie G1 k zálohování hodin reálného času byla z důvodu dosažení co nejmenšího rozměru DPS zvolena typu CR2032 pro vertikální montáž.

U modulu CPU unit a embedded server NE-4100T se počítá s osazením do dutinkových lišt. Důvodem tohoto řešení je především úspora místa na DPS, protože pod moduly mohou být umístěny součástky s vyšším profilem (cívka snižujícího měniče, tantalové kondenzátory). Spodní strana DPS není pro vyšší součástky využitelná, protože se zde nacházejí kolíkové lišty X4 a X7.

Kompletní schéma zapojení, deska plošných spojů a rozložení součástek jsou umístěny v přílohách B a C.

3.4.7 Požadavky na součástky

Tolerance hodnot rezistorů a kondenzátorů není kritická. Mohou být použity rezistory s přesností 5 % nebo lepší. Žádný z rezistorů není výkonově namáhán, a proto postačí standardní dimenzování 0,25 W pro SMD velikost 1206.

U keramických kondenzátorů je požadována tolerance alespoň 10 % a materiál X7R. Výjimku tvoří kondenzátor C8 v obvodu hodin reálného času. Zde potřebujeme co nejlepší teplotní stabilitu, a proto byl vybrán materiál NPO s tolerancí 5 % či lepší. Není-li uvedeno jinak, jsou keramické kondenzátory dimenzovány na napětí alespoň 25 V. Parametry tantalových kondenzátorů C4, C5 a C10 byly diskutovány v odstavci 3.4.4.

Kompletní seznam součástek, jejich parametry a typ pouzdra obsahuje příloha D.

3.5 Komunikační protokol

Cílové mikrokontroléry Atmel AVR disponují omezenými hardwarovými zdroji, a to především v oblastech velikosti paměti programu vyhrazené pro bootloader, velikosti paměti dat a výpočetním výkonu. Tato omezení musí být proto zohledněna již při návrhu vhodného komunikačního protokolu tak, aby bylo zajištěno optimální využití, a tak minimalizovány vlivy bootloaderu na uživatelskou aplikaci.

Z rozboru principů úloh poskytovaných bootloaderem vyplývá, že jisté nároky na objem přenesených dat klade pouze zápis do paměti programu. Z přehledu MCU v příloze E vidíme, že jde řádově o desítky až stovky kB. Přenést celý obraz paměti programu v jednom bloku a následně jej zapsat není možné, protože MCU nedisponuje dostatečně velkou pamětí dat. Programová data se však zapisují po blocích, které odpovídají velikosti stránky, a ta dosahuje maximálně velikosti 256 B. Této skutečnosti lze využít k dekompozici úlohy na jednodušší celky a přenášet data v menších blocích.

Převodník NE-4100T používá k přenosu dat i ovládání obecných vstupně-výstupních portů protokol TCP na pozici transportní vrstvy, což zaručuje dle specifikací [9] do značné míry spolehlivé doručení dat ve správném pořadí. Toho je dosaženo především využitím kontrolních součtů, číslování paketů a opakováním nedoručených paketů. Přenos dat v menších blocích má pozitivní dopad i na detekci chyb. Při menším objemu dat přenášených TCP rámcem stoupne pravděpodobnost detekce chyby, protože i kontrolní součet je pak počítán pro menší blok dat. Z těchto důvodů není dále potřeba řešit zabezpečení přenosu proti chybám na úrovni aplikační vrstvy, kterou tvoří komunikace s bootloaderem. To vede na značné zjednodušení firmware a nižší nároky na výpočetní výkon.

Protože k zápisu jedné stránky dochází v nezanedbatelném čase, jak uvádí tabulka 3.5, musí být při komunikaci realizován mechanizmus řízení toku dat. Tím se zajistí, že nedojde k přetečení přijímacího bufferu bootloaderu, což by vedlo ke ztrátě dat a selhání procesu programování.

Z rozdělení paměťového prostoru (kapitola 3.3.1) je vidět, že kód bootloaderu zabírá místo v paměti programu na úkor uživatelské aplikace a jeho velikost je omezena. Proto se stává přirozeným požadavkem minimalizovat jeho velikost. Není tak na místě složitý

komunikační protokol, čímž lze vyloučit textově orientované protokoly, které preferují čitelnost a flexibilitu nad množstvím přenášených dat a složitostí zpracování.

Z těchto důvodů byl zvolen jednoúčelový binární protokol s komunikací typu master-slave. Firmware bootloaderu tvoří podřízenou stranu (slave) a uživatelská aplikace nadřízenou stranu (master). Každý příkaz bude definován pevným formátem žádostí a odpovědí.

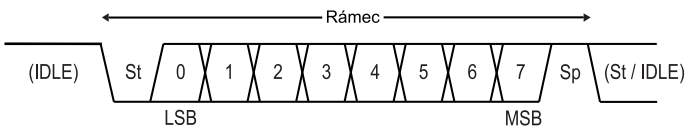
3.5.1 Formát přenosu

Pro sériovou komunikaci mezi mikrokontrolérem a převodníkem NE-4100T je nezbytné určit vhodný formát přenosu. Ten definuje počet datových bitů, stop bitů, parita, řízení toku dat a bitová rychlost.

Bitová rychlost byla vybrána 115200 $\mathrm{bit/s}$ s ohledem na standardní řadu hodnot a rychlost přenosu.

Pro počet datových a stop bitů jsou nejvýhodnější standardní hodnoty osm datových a jeden stop bit. Volba datových bitů vychází z šířky datové sběrnice MCU. Prodloužený stop bit se u tohoto druhu sériové komunikace využíval v případě, že hardware nebyl schopen přijatý byte zpracovat v dostatečně krátkém čase. Vzhledem k postačující rychlosti příjmu a zpracování dat není v tomto případě stop bit potřeba prodlužovat. Parita nebude využita a řízení toku bude řešeno softwarově na úrovni aplikačního protokolu. Hardwarové řízení toku není výhodné jednak proto, že vyžaduje další vstupně-výstupní signály, ale hlavně v nemožnosti přesně určit množství přijatých dat. To by komplikovalo implementaci přijímacího bufferu.

Formát přenosu je naznačen na obrázku 3.12. Začátek přenosu jednoho bytu zahajuje start bit *St*, osmibitová data se vysílají v pořadí od nejnižšího bitu k nejvyššímu bitu a sekvenci uzavírá stop bit *Sp*.



Obrázek 3.12: Formát přenosu

3.5.2 Start bootloaderu

Start bootloaderu po provedení resetu mikrokontroléru se signálem $\overline{BOOT}$ v logické úrovni 0 signalizuje vyslání identifikační sekvence. Ta se skládá z konstantní části a verze bootloaderu, jak popisuje tabulka 3.9. Verzi označuje šestnáctibitové číslo bez znaménka s organizací typu nejméně významný byt první (little-endian). Odesláním identifikační sekvence přechází bootloader do stavu, ve kterém čeká na příkazy, jejichž formáty jsou popsány dále.

k	**Příkaz** (master)	**Odpověď** (slave)	**Popis**
1	RESET		Reset MCU, $\overline{BOOT} = 0$
2		41_{hex} 45_{hex} 42_{hex} $4C_{hex}$	Konstantní sekvence
3		V[0..1]	Verze (uint16)

Tabulka 3.9: Formát identifikační sekvence

3.5.3 Princip potvrzení operací mazání a zápisu

Aby se snížilo riziko náhodného vymazání aplikace, nebo zápisu neplatných hodnot například při chybě komunikace, či připojení neplatného klienta, musí být tyto operace potvrzeny.

Mechanizmus potvrzení pracuje na principu vyslání klíče a ověření odpovědi, která je vypočítána z odeslaného klíče definovaným způsobem.

Generování sekvence klíče se odvodí od časového intervalu vymezeného startem bootloaderu a přijetím příkazu, který má být potvrzen. Praktická realizace bude implementována pomocí volně běžícího šestnáctibitového časovače s vysokým časovým rozlišením. Tím bude zajištěn do jisté míry náhodný charakter daný časovým nedeterminismem komunikace a vznikne tak sekvence dvou bytů, která se odešle aplikaci žádající o provedení příkazu.

Z této dvoubytové sekvence aplikace vypočítá odpověď jako standardní kontrolní součet CRC16-CCITT daný rovnicí $x^{16} + x^{12} + x^5 + 1$. Důvodem této volby spočívá použití stejného CRC algoritmu pro ověření integrity uživatelské aplikace, která bude

popsána dále. Snahou je tedy umožnit optimalizaci velikosti kódu bootloaderu sdílením stejné funkce.

Ověření operace tedy probíhá dle následujícího postupu:

1. Inicializace CRC16 hodnotou $FFFF_{hex}$
2. Výpočet nové hodnoty CRC16 z prvního bytu klíče K0
3. Výpočet nové hodnoty CRC16 z druhého bytu klíče K1
4. Odeslání nižšího bytu CRC16 (odpověď R0)
5. Odeslání vyššího bytu CRC16 (odpověď R1)

3.5.4 Vymazání aplikační sekce

Mazání aplikační sekce je uvozeno kódem $0E_{hex}$. Aby se snížilo riziko náhodného vymazání, používá operace metodu potvrzení popsanou v odstavci 3.5.3. Bootloader odpoví sekvencí dvou bytů, které dávají klíč k potvrzení operace. Pro potvrzení vymazání musí master odpovědět způsobem popsaným v předchozím odstavci.

Komunikaci popisuje tabulka 3.10. Operace mazání má čtyři návratové kódy. Hodnota 00_{hex} označuje úspěšné vymazání aplikační sekce. Odpověď 02_{hex} signalizuje nesprávné potvrzení operace pomocí klíče. Hodnota 03_{hex} informuje o tom, že funkce mazání není u daného systému povolena. Návratová hodnota 10_{hex} znamená, že mazání aplikační sekce proběhlo neúspěšně, protože při verifikaci byl nalezen jeden nebo více bytů, které nemají hodnotu FF_{hex} odpovídající vymazané oblasti.

k	**Příkaz** (master)	**Odpověď** (slave)	**Popis**
1	$0E_{hex}$		Žádost o vymazání aplikační sekce
2		K[0..1]	Klíč pro potvrzení (uint16, K0 = LSB, K1 = MSB)
3	R[0..1]		Odpověď na klíč K (uint16, R0 = LSB, R1 = MSB)
4		00_{hex}	Aplikační sekce úspěšně vymazána
		02_{hex}	Neplatný klíč
		03_{hex}	Funkce zablokována
		10_{hex}	Verifikace se nezdařila, jeden či více bytů má hodnotu různou od FF_{hex}

Tabulka 3.10: Formát komunikace při mazání aplikační sekce

3.5.5 Zápis do aplikační sekce

Kód příkazu pro zápis do aplikační sekce odpovídá hodnotě $0F_{hex}$ a je následován parametrem, který udává velikost obrazu aplikace v bytech. Parametr má formu celého čísla bez znaménka o délce 32 bitů s organizací typu nejméně významný byt první (little-endian).

Pro potvrzení příkazu platí stejné pravidlo jako v případě mazání. Odpovědí na klíč je potvrzení či zamítnutí příkazu. Tím může být nesouhlas klíče, nebo chybná velikost obrazu (větší než aplikační sekce).

Je-li příkaz potvrzen, odešle bootloader velikost svého přijímacího bufferu jako celé číslo bez znaménka o délce 16 bitů. Tento parametr definuje velikost bloků, po kterých bude přenášen obraz aplikace při zápisu.

Master tak vždy odešle blok dat a čeká na potvrzení jeho zápisu. Délka bloku odpovídá velikosti přijímacího bufferu. Menší může být pouze, je-li zbývající počet dat menší, než velikost bufferu.

Po zápisu posledního bloku dat očekává bootloader kontrolní součet obrazu aplikace. Ten je přenášen celočíselnou hodnotou bez znaménka o délce 16 bitů. Jako kontrolní součet byl zvolen standardní polynom CRC16-CCITT daný rovnicí $x^{16} + x^{12} + x^5 + 1$. Kontrolní součet se vypočítá z binárního obrazu uživatelské aplikace, přičemž na začátku výpočtu musí být kontrolní součet inicializován hodnotou $FFFF_{hex}$.

Odpovědí na kontrolní součet je buď potvrzení úspěšného zápisu, nebo chyba ověření CRC, jak popisuje tabulka 3.11.

3.5.6 Čtení konfiguračních bytů

Kódem pro čtení konfiguračních bytů je hodnota 10_{hex}. Odpovědí jsou čtyři byty v pořadí Fuse Low, Lock, Extended Fuse a Fuse High. Není-li některý z konfiguračních bytů u daného MCU dostupný, vrací bootloader hodnotu FF_{hex}.

3.5.7 Čtení podpisu MCU

Kódem ke čtení podpisu mikrokontroléru je hodnota 11_{hex} a pořadí odpovědi definuje tabulka 3.13. U mikrokontrolérů, které neumožňují čtení signature bootloaderem (například ATmega16), vrací funkce hodnotu $FFFFFF_{hex}$.

k	**Příkaz** (master)	**Odpověď** (slave)	**Popis**
1	$0F_{hex}$		Příkaz zápisu aplikace
2	S[0..3]		Velikost aplikace (uint32, S0 = LSB, S3 = MSB)
3		K[0..1]	Klíč pro potvrzení (uint16, K0 = LSB, K1 = MSB)
4	R[0..1]		Potvrzení příkazu
5		00_{hex} 02_{hex} 20_{hex}	Velikost přijata Neplatný klíč Velikost odmítnuta
6		B[0..1]	Velikost přijímacího bufferu B (uint16, B0 = LSB, B1 = MSB)
7	D[N]		Blok dat o velikosti N N = B je-li zbývající počet dat >= B, jinak N = zbývající počet dat
8		00_{hex}	Blok dat zapsán
9	Kroky 7 a 8 se opakují, dokud není zapsána celá aplikace		
10	CRC[0..1]		Kontrolní součet aplikace (uint16, CRC0 = LSB, CRC1 = MSB)
11		00_{hex} 21_{hex}	Aplikace úspěšně zapsána Zápis aplikace se nezdařil (nesouhlasí kontrolní součet)

Tabulka 3.11: Formát komunikace při zápisu do aplikační sekce

k	**Příkaz** (master)	**Odpověď** (slave)	**Popis**
1	10_{hex}		Žádost o čtení konfiguračních bytů
2		FL	Konfigurační byt Fuse Low
3		L	Konfigurační byt Lock
4		EF	Konfigurační byt Extended Fuse
5		FH	Konfigurační byt Fuse High

Tabulka 3.12: Formát komunikace při čtení konfiguračních bytů

k	**Příkaz** (master)	**Odpověď** (slave)	**Popis**
1	11_{hex}		Žádost o čtení konfiguračních bytů
2		S1	Signature Byte 1
3		S2	Signature Byte 2
4		S3	Signature Byte 3

Tabulka 3.13: Formát komunikace při čtení podpisu MCU

3.5.8 Zápis konfiguračních bitů paměti programu

Zápis konfiguračních bitů paměti programu má kód 12_{hex} Za kódem příkazu následuje hodnota, která má být nastavena. Z principu funkce lock bitů je možné pouze nastavit bity s hodnotou 1 na hodnotu 0 (viz. odstavec 3.3.2), a proto chování odpovídá logické funkci AND. Z toho vyplývá, že hodnota FF_{hex} neprovede žádnou změnu a hodnota 00_{hex} nastaví všechny dostupné lock bity.

Z důvodu zachování funkce uživatelské aplikace musí být blokováno nastavení bitu *BLB02*. Jeho aktivování má za následek nemožnost čtení aplikační sekce bootloaderem, což vede k selhání systému ověření CRC. V konečném výsledku by tak nastavení bitu *BLB02* zabránilo spuštění uživatelské aplikace, což znamená nefunkčnost celého systému. Tento bit proto nebude bootloaderem nikdy nastaven.

Komunikaci a návratové kódy popisuje tabulka 3.14. Jelikož se jedná o zápis, musí být operace potvrzena obvyklým způsobem.

k	**Příkaz** (master)	**Odpověď** (slave)	**Popis**
1	12_{hex}		Žádost o zápis konfiguračních bitů
2	LB		Zapisovaná hodnota
3		K[0..1]	Klíč pro potvrzení (uint16, K0 = LSB, K1 = MSB)
4	R[0..1]		Odpověď na klíč K (uint16, R0 = LSB, R1 = MSB)
5		00_{hex}	Operace úspěšně provedena
		02_{hex}	Neplatný klíč
		03_{hex}	Funkce zablokována

Tabulka 3.14: Formát komunikace při zápisu konfiguračních bitů paměti programu

3.5.9 Neplatný příkaz

Není-li kód příkazu rozpoznán, odpovídá bootloader chybovým kódem 01_{hex} a vrací se do stavu čekání na příkaz. Popis komunikace popisuje tabulka 3.15.

k	Příkaz (master)	Odpověď (slave)	Popis
1	E		E = neplatný kód příkazu
2		x01	Neplatný příkaz

Tabulka 3.15: Odpověď na neplatný příkaz

3.6 Firmware

Pro vývoj firmware mikrokontroléru byl zvolen jazyk C. Oproti assembleru vykazuje nesrovnatelně jednodušší vývoj aplikace s ohledem na případné změny, rozšíření a především snadnější kontrolu řešení. Výhodou programovacího jazyka na vyšší úrovni je také menší návaznost na hardware, a tak lze snadněji dosáhnout použití stejného kódu pro různé typy MCU. Cenou za to je hlavně mírné zvětšení výsledného strojového kódu a nižší stupeň optimalizace.

Vzhledem k úzké návaznosti problematiky bootloaderu na hardware musí být brán ohled na specifické vlastnosti překladače. Z tohoto důvodu bude zdrojový kód firmware navržen pro konkrétní překladač. Z volně dostupných byl vybrán nejrozšířenější překladač AVR-GCC, protože podporuje většinu mikrokontrolérů Atmel AVR, je stále aktivně rozvíjen a disponuje dobrou dokumentací.

3.6.1 Struktura projektu

Na základě zvoleného komunikačního protokolu, principu funkcí bootloaderu a požadavků diskutovaných v předchozích částech projektu byl vytvořen firmware bootloaderu.

Kód řešení byl v zájmu přehlednosti dekomponován na elementární funkce a rozdělen dle logické struktury na několik samostatných částí. Celý projekt se tak skládá z devíti souborů, jejichž názvy a hrubý popis udává tabulka 3.16.

3.6.2 Start bootloaderu

Start popisuje vývojový diagram hlavní funkce programu na obrázku 3.13. Vstupním bodem je vektor resetu mikrokontroléru nastavený konfiguračním bitem BOOTRST do

Soubor	Popis
Makefile	Skript pro překlad projektu
aebl.c	Hlavní zdrojový kód. Implementace funkcí bootloaderu.
uart.c	Zdrojový kód komunikačních funkcí.
watchdog.c	Zdrojový kód funkcí pro obsluhu externího WDT.
aebl.h	Hlavní hlavičkový soubor bootloaderu.
options.h	Hlavičkový soubor s konfiguračními parametry bootloaderu.
rand.h	Hlavičkový soubor s inline funkcemi generátoru klíče.
uart.h	Hlavičkový soubor komunikačních funkcí.
watchdog.h	Hlavičkový soubor funkcí WDT.

Tabulka 3.16: Struktura projektu firmware

sekce bootloaderu. Zde leží hlavní funkce programu s prototypem:

```
void __attribute__((noreturn,naked,section(".vectors"))) main(void);
```

Atribut *section(".vectors")* zaručuje umístění funkce linkerem na požadovanou adresu. Atributy *noreturn* a *naked* optimalizují velikost výsledného kódu. Protože hlavní funkce bootloaderu nikdy nevrací hodnotu a nepotřebuje standardní inicializaci, jsou tyto vlastnosti zablokovány.

Z důvodu optimalizace nejsou použity ani standardní inicializační funkce zprostředkované překladačem, čehož je dosaženo parametrem linkeru *-nostartfiles*. Jako první krok se tedy musí inicializovat důležité registry MCU, o jejichž nastavení se obvykle stará překladač. Těmi jsou v případě překladače AVR-GCC registr *R1*, stavový registr *SREG* a ukazatel zásobníku *SP*.

Registr *R1* je překladačem využíván jako nulový registr, proto musí být inicializován na hodnotu 00_{hex}. Jelikož přístup k těmto registrům není z jazyka C přímo možný, byla vložena instrukce *CLR* (*Clear Register*) prostřednictvím inline assembleru. Parametr *volatile* zabraňuje překladači instrukci vypustit v rámci optimalizace. Stavový registr *SREG* počítá ve výchozím stavu s nulovou hodnotu. Zde již použití assembleru není nutné. Nejdůležitějším krokem je inicializace ukazatele zásobníku *SP*. Mikrokontroléry AVR používají zpětný posun ukazatele při ukládání na zásobník (adresa je dekrementována).

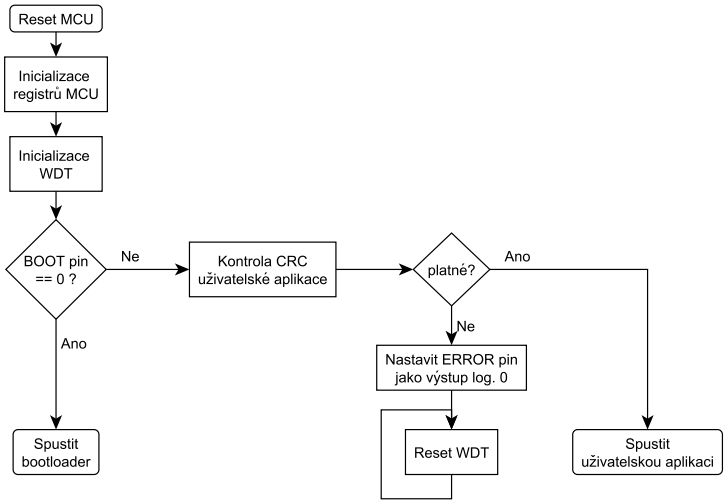
Překladač AVR-GCC tak ve výchozím nastavení počítá s počátkem zásobníku na poslední adrese paměti dat.

Po inicializaci watchdog časovače přichází na řadu kontrola podmínky pro spuštění uživatelské aplikace. Ta je definována jako logický signál na vstupním pinu MCU. Je-li hodnota 0, pokračuje se ve spouštění bootloaderu. V opačném případě se požaduje spuštění uživatelské aplikace, což je podmíněno správným kontrolním součtem.

V případě platného kontrolního součtu dojde ke spuštění uživatelské aplikace skokem na adresu 0000_{hex}. Toho je docíleno deklarací ukazatele na imaginární funkci ležící na požadované adrese a následným voláním této funkce:

```
void (*jump_to_app)(void) = 0x0000;
jump_to_app();
```

Je-li kontrolní součet neplatný, generuje se příznak chyby nastavením pinu $\overline{ERROR}$ jako výstup s logickou hodnotou 0 a spuštění uživatelské aplikace není dovoleno. Program končí v nekonečné smyčce, ve které dochází k periodickému resetování WDT, který by jinak způsobil neustálé resetování MCU.



Obrázek 3.13: Vývojový diagram spuštění bootloaderu

3.6.3 Čekání na příkaz

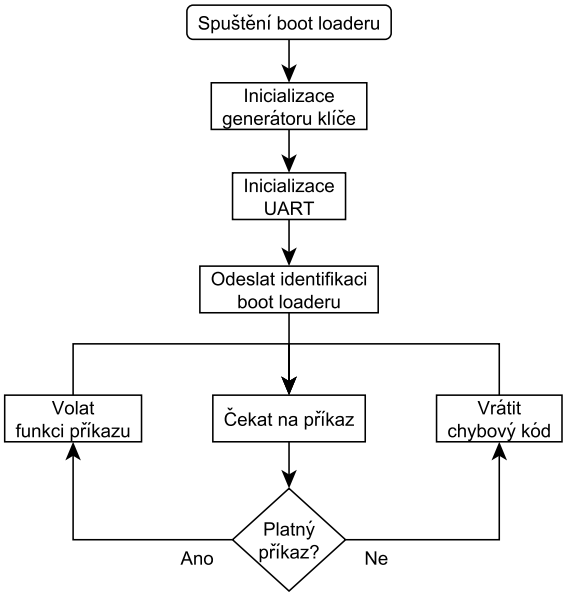
Tuto úlohu zajišťuje funkce (vývojový diagram na obrázku 3.14):

```
static inline void WaitForCommand(void);
```

Jako první krok po vstupu do bootloaderu se inicializuje generátor potvrzovacích klíčů. Dále musí být zahájena komunikace s klientským software prostřednictvím sériového kanálu. To vyžaduje inicializaci sériového kanálu UART na požadované parametry přenosu. Úspěšné dokončení inicializace je signalizováno odesláním identifikačního bloku dat. Ten obsahuje definovanou konstantní sekvenci a verzi firmware bootloaderu, jak bylo specifikováno v odstavci 3.5.2.

Následně bootloader vstupuje do nekonečné smyčky, kde čeká na přijetí příkazu od klientské aplikace, na jehož základě jsou volány patřičné funkce. Je-li přijat neplatný příkaz, vrací bootloader chybový kód. Každá platná funkce dále vrací kód úspěšného provedení, případně specifický chybový kód identifikující důvod selhání.

Z hlavní smyčky bootloaderu již není možné softwarově spustit uživatelskou aplikaci. Tato strategie byla zvolena, aby po zavedení uživatelské aplikace bylo nutné provést hardwarový reset mikrokontroléru, což zaručuje uvedení jeho stavu do výrobcem definovaného výchozího stavu.



Obrázek 3.14: Vývojový diagram čekání na příkaz

3.6.4 Vymazání aplikační sekce

Vývojový diagram, který specifikuje jednotlivé elementární kroky při mazání aplikační sekce paměti programu je na obrázku 3.15. V programu mu odpovídá funkce:

```
static inline void Flash_Erase(void);
```

Před započetím mazání se musí nejdříve odeslat a přijmout ověřovací klíč. Porovnáním se potvrdí provedení operace. V opačném případě je generován chybový kód a funkce ukončena.

Celý proces se dále skládá ze dvou kroků. V prvním dojde k postupnému vymazání příslušných stránek paměti programu a v druhém probíhá ověření úspěšnosti této operace.

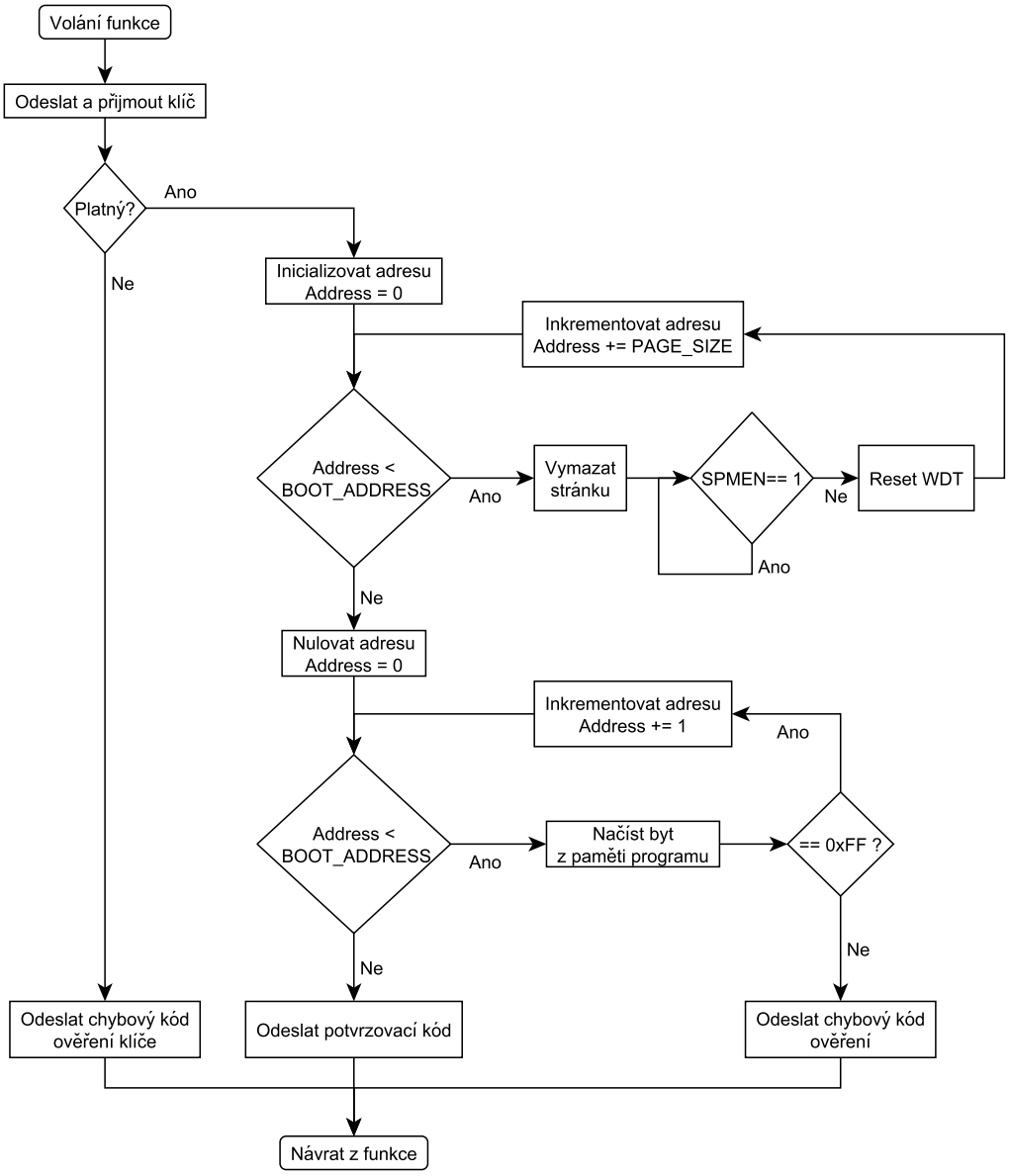
Vlastní proces mazání s využitím instrukce *SPM* se realizuje makrem *boot_page_erase()* z hlavičkového souboru *boot.h*, který je součástí standardních knihoven překladače. Makro obsahuje příslušné instrukce v inline assembleru odpovídající popisu v sekci 3.3.6. Následuje čekání na dokončení mazání prostřednictvím čtení stavu bitu *SPMEN* (viz. sekce 3.3.5). Po vymazání stránky je nezbytné okamžitě resetovat WDT, protože operace mazání trvá řádově 5 ms.

Použitím instrukce mazání dojde k zablokování čtení z RWW sekce paměti. Čtení musí být obnoveno voláním makra *boot_rww_enable()*. Fáze ověření pak spočívá ve zpětném načtení obsahu paměti programu. Je-li vymazána, musí všechny paměťové buňky obsahovat hodnotu FF_{hex}. V takovém případě funkce vrací potvrzovací kód, jinak chybový kód mazání paměti.

3.6.5 Zápis do aplikační sekce

Zápis obrazu uživatelské aplikace do paměti programu mikrokontroléru můžeme opět dekomponovat na jednodušší celky, jak udává vývojový diagram na obrázku 3.16. Prakticky je úloha rozložena do funkcí:

```
static inline void Flash_Write(void);
static inline void Flash_ReceiveAndWriteImage(aeblcnt_t DataToWrite);
static inline void Flash_WriteImageInfo(const uint32_t ImageSize, const
    uint16_t CRC);
```



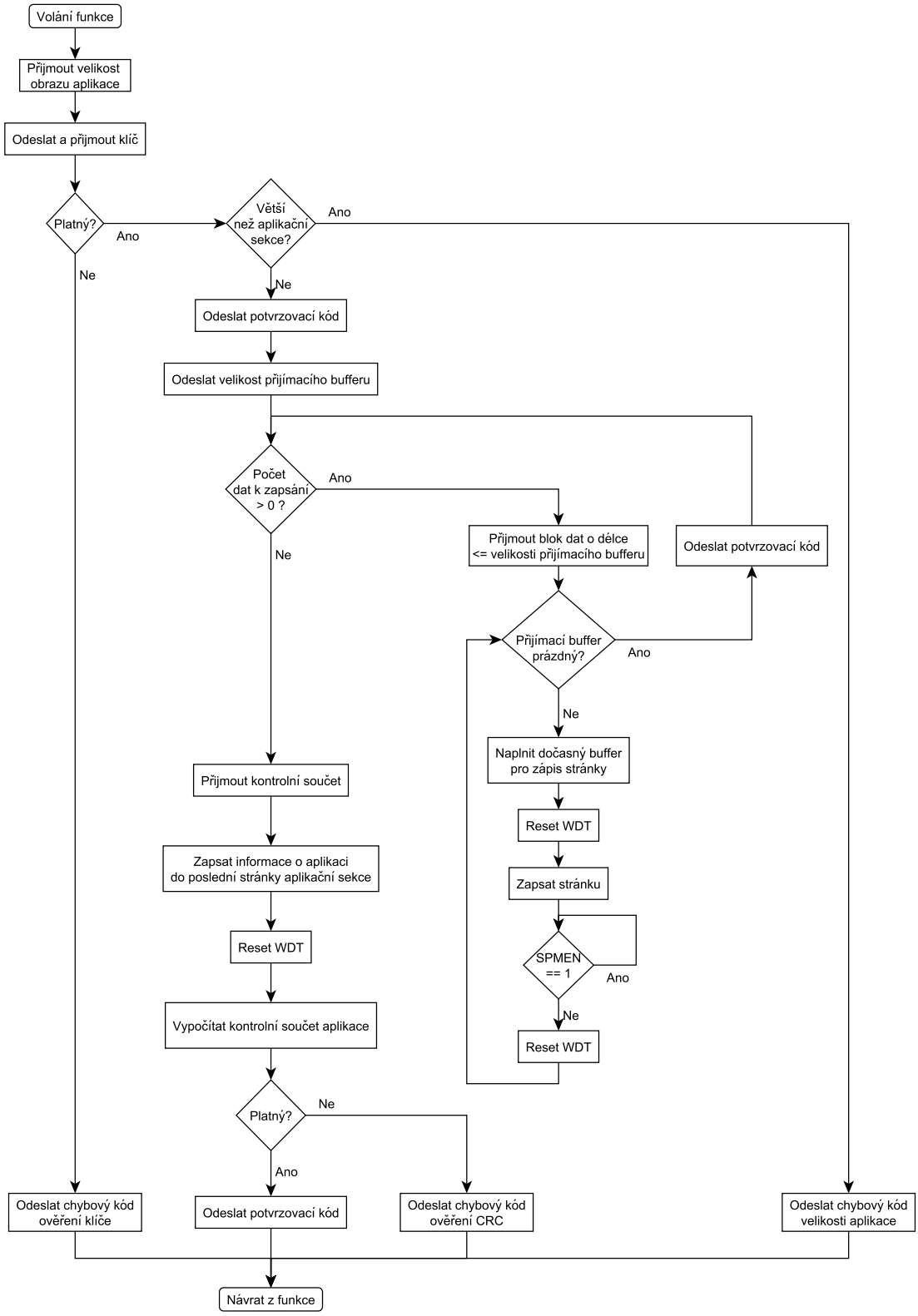
Obrázek 3.15: Vývojový diagram vymazání aplikační sekce

V úvodu je prvně přijata informace o velikosti obrazu aplikace. Jako odpověď odesílá bootloader potvrzovací sekvenci a po přijetí její odpovědi generuje příslušný kód. Pro pokračování zápisu musí souhlasit odeslaný klíč s přijatým a velikost aplikace s velikostí paměti. Nesouhlasí-li klíče, nebo je-li aplikace větší než dostupná paměť, dojde k vygenerování chybového kódu a operace ukončena. V opačném případě dojde k potvrzení velikosti a odeslání informace o přijímacím bufferu bootloaderu.

Velikost přijímacího bufferu je důležitá pro další komunikaci, kdy bootloader přijímá obraz aplikace v blocích o maximální délce odpovídající velikosti bufferu. Je-li počet zbývajících bytů pro zapsání větší než nula, je očekáván blok dat. A to o velikosti přijímacího bufferu v případě, že zbývající počet bytů je větší nebo roven jeho velikosti. Obsah bufferu je následně po jednotlivých stránkách zapsán do paměti a dokončení potvrzeno příslušnou odpovědí. Klientská aplikace tak čeká na potvrzení zápisu, čímž je dosaženo řízení toku dat.

Plnění dočasného bufferu pro zápis stránky provádí makro *boot_page_fill()*, jenž obsahuje příslušné volání instrukce *SPM* v inline assembleru. Po naplnění stránky musí být proveden reset WDT a teprve potom inicializován zápis stránky makrem *boot_page_write()*. Na dokončení zápisu se čeká obdobným způsobem jako v případě mazání stránky, a to čtením stavu bitu *SPMEN*. Po zapsání stránky je opět resetován WDT.

Následně za potvrzením úspěšného zapsání poslední stránky očekává bootloader přijetí kontrolního součtu obrazu aplikace, který slouží k ověření zápisu. Přijatý kontrolní součet se nejdříve zapíše spolu s velikostí obrazu aplikace do vyhrazené poslední stránky aplikační sekce. Dále se musí obnovit schopnost čtení ze sekce RWW voláním makra *boot_rww_enable()*. Teprve potom se může vypočítat kontrolní součet zpětným přečtením paměti programu a porovnat jej s přijatou hodnotou. V případě souladu je odesláno potvrzení správného zápisu, v případě rozdílného kontrolního součtu se vrací chybový kód.



Obrázek 3.16: Vývojový diagram zápisu do aplikační sekce

3.6.6 Čtení konfiguračních bytů

Činnost funkce

```
static inline void Config_Read(void);
```

vychází z diagramu na obrázku 3.17(a). Na základě adresy odpovídající tablulce 3.6 jsou postupně čteny jednotlivé konfigurační byty instrukcí *LPM* (s patřičným nastavením registru *SPMCSR*) a průběžně odesílány. Funkce využívá inline assembleru.

3.6.7 Čtení podpisu MCU

Vývojový diagram funkce

```
static void Signature_Read(void);
```

znázorňuje obrázek 3.17(b). Rozdíl oproti čtení konfiguračních bitů spočívá pouze v nastavení registru *SPMCSR* a inkrementování adresy.

3.6.8 Zápis konfiguračních bitů paměti programu

Princip funkce

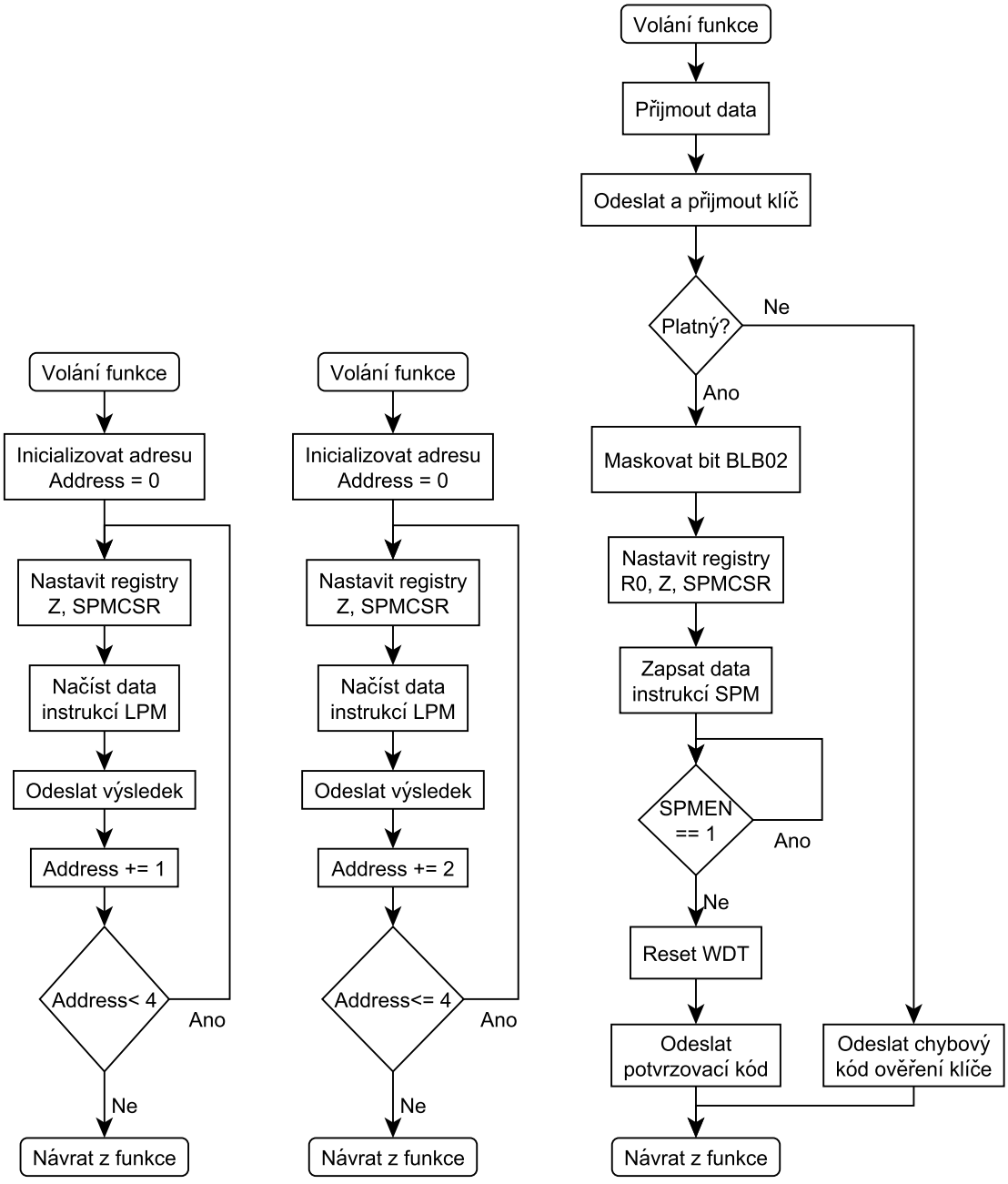
```
static void Lock_Write(void);
```

znázorňuje diagram na obrázku 3.17(c). Stejným způsobem, jako u mazání paměti a zápisu paměti programu, se potvrdí provedení operace. Následně je z důvodů popsaných v odstavci 3.5.8 maskován bit *BLB02*. Instrukce *SPM* inicializuje zápis hodnoty a následuje čekání na dokončení operace. Nezbytné úkony uzavírá reset WDT. Konfiguraci registrů *SPMCSR*, *Z* a vložení instrukce *SPM* zprostředkovává makro *boot_lock_bits_set()*.

3.6.9 Pomocné funkce

Výpočet kontrolního součtu

Tato skupina pomocných funkcí obstarává výpočet kontrolního součtu kódu uživatelské aplikace. Do ní spadají funkce:



(a) Čtení konfiguračních bytů

(b) Čtení podpisu MCU

(c) Zápis konfiguračních bitů paměti programu

Obrázek 3.17: Vývojové diagramy čtení konfiguračních bytů, podpisu MCU a zápisu konfiguračních bitů paměti programu

```
static uint16_t GetFlashCRC(aeblcnt_t Length);
static inline bool Flash_CheckCRC(void);
```

První funkce vrací kontrolní součet paměti programu, a to pro blok začínající na adrese 0000_{hex} o délce specifikované parametrem *Length*, který vyjadřuje počet bytů. Je sdílena funkcemi *Flash_CheckCRC()* a *Flash_Write(void)*. Využívá makra *_crc_ccitt_update()* z hlavičkového souboru *crc16.h*, který obsahuje balík překladače AVR-GCC. Makro vkládá optimalizovaný výpočet bez použití lookup tabulky v inline assembleru. Kód se vyznačuje malými nároky na paměť programu (ve srovnání s tabulkovou metodou) a relativně vysokým výkonem. Volba byla provedena na základě testů, jejichž výsledky popisuje tabulka 3.17. Srovnání bylo provedeno pro blok dat o délce 63232 B, což odpovídá maximální velikosti aplikační sekce u ATmega644. Počet strojových cyklů odpovídá při taktu MCU 1 MHz době výpočtu v μs. Zde je patrné, že implementace kontrolního součtu o vyšším počtu bitů naráží na nízký výpočetní výkon MCU a vede na neúnosnou délku výpočtu. Jako kompromisní řešení mezi rychlostí výpočtu a velikostí výsledného kódu byl zvolen výpočet CRC-16-CCITT bez lookup tabulky. Při taktu 18,432 MHz, který používá modul CPU unit, tak dostáváme dobu výpočtu 125 ms.

Typ CRC	Implementace	Strojových cyklů	Velikost kódu
CRC-16-CCITT	C (lookup tabulka)	1 965 628	804 B
CRC-16-CCITT	crc16.h (avr-gcc)	2 358 744	314 B
CRC-32-IEEE 802.3	assembler (výpočet)	6 158 531	226 B
CRC-32-IEEE 802.3	C (výpočet)	9 303 029	320 B

Tabulka 3.17: Srovnání nároků výpočtů kontrolního součtu (63232 B, ATmega644)

Druhá funkce ověřuje kontrolní součet uživatelské aplikace oproti údajům zapsaným v poslední stránce aplikační sekce. Funkci volá pouze hlavní funkce *main()*, a proto byla označena atributem inline, čímž se eliminuje volání a návrat z funkce.

Potvrzení operace

Úlohu potvrzení provedení operace, kterou vyžadují funkce *Flash_Erase()*, *Flash_Write()* a *Lock_Write()*, byla sdružena do pomocné funkce:

```
static bool ConfirmOperation(void);
```

Ta je navázána na generování klíče funkcemi:

```
static inline void aebl_Rand_Init(void);
static inline uint16_t aebl_Rand(void);
```

První spouští volně běžící šestnáctibitový časovač při startu bootloaderu. Druhá generuje hodnotu na základě stavu časovače v okamžiku volání funkce *ConfirmOperation()*. Dle dříve popsaného algoritmu se následně vypočítá odpověď na vygenerovaný klíč a ověří proti přijaté odpovědi.

Watchdog časovač

Z hlediska možnosti volby interního nebo externího WDT využívají tyto funkce hojně podmíněný překlad, čímž je dosaženo přizpůsobení bootloaderu na míru danému WDT.

K resetování WDT slouží v celém programu makro *WDT_Reset()*. Na reset interního WDT stačí pouze jedna instrukce *WDR*, a tak se vkládá přímo jako příkaz inline assembleru. Resetování externího WDT typu MB3773 z hlediska časování (odstavec 3.2.2) vychází značně složitější, a tak je implementováno jako funkce:

```
extern void WDT_ResetExt(void);
```

Obvod MB3773 reaguje podle obrázku 3.2 na sestupnou hranu signálu CK. Ze specifikací [14] vychází nejkratší možná perioda signálu CK $T_{CK} = 20\,\mu\mathrm{s}$ a nejmenší šířka impulzu $T_{CKW} = 3{,}0\,\mu\mathrm{s}$. V návaznosti na přenosovou rychlost sériové komunikace byla prodlevami nastavena minimální perioda signálu CK $T_{CK} = 30\,\mu\mathrm{s}$ a šířka impulzu $T_{CKW} = 15\,\mu\mathrm{s}$. Při bitové rychlosti $115200\,\mathrm{bit/s}$ a deseti bitech jednoho rámce (1 start, 8 datových a 1 stop bit) tak připadají dva resetovací impulzy na časový interval potřebný k jejich přenesení.

Úkony nezbytné k inicializaci časovače při startu bootloaderu a současně první reset obsahuje funkce:

```
static inline void WDT_Init(void);
```

U externího WDT MB3773 spočívá inicializace v nastavení pinu MCU pro ovládání vstupu CK do funkce výstupu. Interní WDT inicializaci nevyžaduje.

Komunikace

Obsluhu sériového portu pro přenos dat zprostředkovávají pomocné funkce:

```
extern void uart_WriteByte(const uint8_t Data);
extern void uart_Init(void);
static inline void uart_WriteWord(const uint16_t Data);
extern uint8_t uart_ReadByte(void);
static inline uint16_t uart_ReadWord(void);
static inline uint32_t uart_ReadDWord(void);
extern void uart_WriteBlockP(uint16_t Bytes, const void *pBuffer);
```

Komunikace probíhá v blokujícím režimu bez využití přerušení. Vzhledem k nezanedbatelnému času, který je nutný na odeslání či příjmu jednoho bytu, musí být brán ohled na součinnost s watchdog časovačem. Základem všech funkcí jsou elementární *uart_WriteByte()* pro vysílání a *uart_ReadByte()* pro příjem. Stačí tak ovládání WDT implementovat v nich.

3.6.10 Konfigurace bootloaderu

Při implementaci bootloaderu byla snaha o možnost jeho použití i v jiném hardwarovém řešení, než je modul CPU unit. Pro přizpůsobení danému hardwaru slouží konfigurační soubor *options.h* a skript překladače *Makefile*.

Soubor *Makefile* obsahuje tři důležitá nastavení, kterými jsou:

```
MCU = atmega644p
F_CPU = 18432000
BOOT_ADDRESS = 0xF800
```

Jak již názvy napovídají, parametr *MCU* definuje konkrétní typ použitého mikrokontroléru a *F_CPU* jeho taktovací kmitočet v Hz. Takt závisí na použitém oscilátoru a případně nastavení předděličky. Parametr *BOOT_ADDRESS* odpovídá počáteční adrese bootloaderu v **bytech**, která se odvíjí od nastavení konfiguračních bitů *BOOTSZ*.

Ostatní nastavení definuje soubor *options.h*. První skupinu tvoří konfigurace sériové komunikace:

```
#define UART                    0
#define UART_BAUD_RATE          115200
```

```
#define UART_USE_U2X
```

Parametr *UART* má význam pouze u mikrokontrolérů vybavených více rozhraními UART (například ATmega644P) a vybírá, které rozhraní bude použito ke komunikaci bootloaderu. Parametr *UART_BAUD_RATE* nastavuje komunikační rychlost. Snižování rychlosti přenosu nemá omezení, ale při zvyšování se musí počítat s časem potřebným k resetování externího watchdog časovače, je-li použit. Při příliš velké přenosové rychlosti tak může dojít k přetečení přijímacího bufferu. Parametr *UART_USE_U2X* definuje, je-li nastaven bit *U2X* v konfiguračním registru rozhraní UART.

Druhá skupina nastavení definuje použitý watchdog časovač. Může být vybrána pouze jedna z možností:

```
#define WATCHDOG_INTERNAL
#define WATCHDOG_MB3773
```

Dále následuje volba mapování vstupně-výstupních signálů, jenž závisí na konkrétním zapojení. Pin $\overline{ERROR}$ signalizuje chybný kontrolní součet uživatelské aplikace:

```
#define AEBL_ERROR_PIN      4
#define AEBL_ERROR_PORT     PORTD
#define AEBL_ERROR_DDR      DDRD
```

Parametr *AEBL_ERROR_PIN* udává číslo pinu, *AEBL_ERROR_PORT* výstupní registr a *AEBL_ERROR_DDR* registr nastavující funkci pinu. Druhým závazným signálem je $\overline{BOOT}$, definující podmínku vstupu do bootloaderu:

```
#define AEBL_BOOT_PIN       3
#define AEBL_BOOT_PORT      PIND
```

Parametr *AEBL_BOOT_PIN* odpovídá číslu pinu a *AEBL_BOOT_PORT* příslušnému vstupnímu registru. Poslední signál CK musí být nastaven pouze v případě, je-li použit externí WDT typu MB3773:

```
#define MB3773_CK_PIN       2
#define MB3773_CK_DDR       DDRD
#define MB3773_CK_PORT      PORTD
```

MB3773_CK_PIN odpovídá číslu pinu, *MB3773_CK_PORT* výstupnímu registru a parametr *MB3773_CK_DDR* konfiguračnímu registru.

Poslední sekce dovoluje povolení či zakázání mazání uživatelského programu a zápisu konfiguračních bitů paměti programu:

```
#define ENABLE_ERASE_FLASH
#define ENABLE_WRITE_LOCK
```

3.6.11 Kompilace

Kód byl překládán a testován ve verzi překladače AVR-GCC 4.3.2 s parametry, které jsou definovány ve skriptu *Makefile*. Podstatné parametry překladače popisuje tabulka 3.18, parametry linkeru pak tabulka 3.19.

Zdrojový kód je psán v jazyce C, který využívá rozšíření *C99* (deklarace, inline funkce, typ bool), *GNU* (inline assember) a parametry specifické pro překladač. Jejich podporu zapíná parametr *-std=gnu99*.

Cílem ostatních parametrů je především umístění strojového kódu bootloaderu na příslušnou adresu v paměti programu a minimalizace jeho výsledné velikosti. Toho bylo dosaženo především vypuštěním linkování standardních knihoven a nevyužitých funkcí, které mohou vzniknout následkem podmíněného překladu v závislosti na konfiguraci.

Parametr	**Popis**
-std=gnu99	povoluje standard C99 s GNU rozšířeními
-Os	optimalizace na velikost výsledného kódu
-funsigned-char	definuje typ *char* jako neznaménkový
-ffunction-sections	umístí každou funkci do samostatné sekce
-fdata-sections	umístí každou nelokální proměnnou do samostatné sekce

Tabulka 3.18: Parametry překladače AVR-GCC

3.6.12 Konfigurace mikrokontroléru

Ke správné funkci bootloaderu musí být patřičně nakonfigurován použitý MCU prostřednictvím svých fuse bitů. Pro všechny typy MCU platí nastavení bitu $BOOTRST = 0$, který po resetu zajistí start MCU na adrese bootloaderu.

Parametr	Popis
–section-start=.text=	umístí počátek kódu na požadovanou adresu
–gc-sections	odstraní nevyužité funkce a proměnné
-nostartfiles	zakáže vkládání standardních inicializačních sekcí
-nodefaultlibs	zakáže vkládání výchozích knihoven
-nostdlib	zakáže vkládání standardních knihoven

Tabulka 3.19: Parametry linkeru AVR-GCC

Na konkrétním typu MCU závisí nastavení bitů *BOOTSZ*, které definují velikost sekce bootloaderu a současně její počátek. Bootloader vyžaduje sekci o velikosti 2048 B, což u ATmega644 odpovídá nastavení bitů $BOOTSZ1 = 1$ a $BOOTSZ0 = 0$.

Pro zvýšení funkční bezpečnosti by měly být patřičně nastaveny i konfigurační bity paměti programu (lock), a to s cílem zamezit poškození bootloaderu uživatelskou aplikací. Zápis i čtení zakáže nastavení bitů $BLB11 = BLB12 = 0$.

Je-li požadována ochrana proti modifikaci či čtení obsahu paměti programu a nonvolatilní paměti dat programátorem (paralelním či sériovým), mohou být nastaveny bity $LB1 = LB2 = 0$. Bit *LB1* brání zápisu, *LB2* čtení. Tato funkce spadá do problematiky informační bezpečnosti, protože můžeme jednoduše zamezit zpětnému získání strojového kódu uživatelské aplikace z MCU.

Doporučené nastavení konfiguračních bitů mikrokontroléru ATmega644 v modulu CPU unit udává tabulka 3.20. Zohledněn je krystalový oscilátor, externí WDT a požadavky na zabezpečení.

3.6.13 Konfigurace NE-4100T

Nezbytný krok k uvedení celého systému do provozu spočívá v nastavení embedded serveru NE-4100T. Toho lze docílit jednou z metod zmíněných v odstavci 2.1.5. Parametry NE-4100T můžeme rozdělit do dvou základních skupin na nezbytné a volitelné, podle jejich návaznosti na funkčnost bootloaderu. První skupina parametrů musí být vždy správně nakonfigurována a jejich nastavení udává tabulka 3.21. Z pohledu bezpečnosti patří k nezbytným opatřením nastavení povolených IP adres, které mají právo komunikovat se ser-

Parametr	Hodnota	Popis
Fuse Low	11010111_{bin}	Oscilátor s plným rozkmitem. Doba náběhu 16K/14 CK.
Fuse High	10011100_{bin}	Bootloader 2048 B; SPI; JTAG;
Extended Fuse	11111111_{bin}	BOD vypnut.
Lock	11001100_{bin}	Zakázáno čtení i zápis programátorem. Zakázán zápis uživatelské aplikace do bootloaderu.

Tabulka 3.20: Konfigurace MCU ATmega644 v modulu CPU unit

verem, a hesla pro změnu jeho konfigurace.

Volitelné parametry mohou být nakonfigurovány podle potřeby uživatele. Do této skupiny patří především volitelné zasílání hlášení na email či prostřednictvím SNMP. Povolením události na změnu signálu DCD tak můžeme dostat informaci v případě, že bootloader zablokuje spuštění uživatelské aplikace v důsledku chybného kontrolního součtu. Aby zpráva obsahovala platnou časovou značku, musí být nakonfigurován i vestavěný klient časového serveru.

3.7 Softwarové vybavení PC

Cílem aplikace pro PC je zpřístupnění funkcí vzdálené správy, které poskytuje bootloader, koncovému uživateli. Aby se nejednalo o samoúčelné řešení, byl celý problém rozdělen na několik hierarchicky navazujících částí.

První část představuje knihovnu základních funkcí pro Ethernetovou komunikaci protokolem TCP/IP a na ni navazuje knihovna k ovládání uživatelských vstupů-výstupů embedded serveru NE-4100T. Na jejich rozhraní pak staví nejpodstatnější část, kterou reprezentuje knihovna k ovládání bootloaderu. Ta vytváří jednotné aplikační rozhraní. Knihovnu nakonec využívá výsledná uživatelská aplikace.

Výhoda tohoto dělení spočívá v možnosti jednoduše využít knihovnu v jiném projektu, nebo ji portovat na odlišný operační systém. Hierarchické dělení navíc napomáhá ke zvýšení přehlednosti kódu a v neposlední řadě jednoduššímu ladění.

Network Settings

dle požadavků uživatele

Serial Settings - Port 1

Baud Rate	115200 (nebo dle uživatelské konfigurace bootloaderu)
Data Bits	8
Stop Bits	1
Parity	None
Flow Control	None
FIFO	Enable

Operating Settings - Port 1

Operation mode	TCP Server Mode
TCP alive check time	1 min
Max connection	1
Local TCP port	4001 (nebo dle požadavků uživatele)
Delimiter 1, 2	Off

Digital IO - DIO Settings

TCP Port	4002 (nebo dle požadavků uživatele)
DIO 0	OUT - LOW
DIO 1	OUT - HIGH
SW Reset Function	Disable
Serial Command Mode	Disable

Accessible IP Settings, Password

dle požadavků uživatele

Tabulka 3.21: Konfigurace NE-4100T - povinné parametry

Jelikož tvorba uživatelského rozhraní není zcela jádrem tohoto projektu, byla vytvořena pouze jednoduchá konzolová aplikace, která však dovoluje využít veškeré funkce bootloaderu.

Pro praktickou realizaci byl zvolen jazyk C++ s překladačem Microsoft Visual Studio 2008 a jako cílový port operační systém Microsoft Windows.

3.7.1 Knihovna pro TCP komunikaci

Základem celé aplikace je právě knihovna pro komunikaci TCP protokolem. Cílem bylo vytvořit jednotné rozhraní pro zápis a čtení bloku dat. To reprezentuje třída *CBlockTCP*, jejíž veřejné metody jsou:

```
void Connect(const char *pAddress, WORD Port);
void Disconnect();
void Recv(char *pBuffer, int Length, int TimeOut = INFINITE);
int  Send(char *pBuffer, int Length, int TimeOut = INFINITE);
```

Třída *CBlockTCP* vychází z bázové třídy *CWinSock*, která zaručuje inicializaci a uvolnění systémové knihovny *WinSock*.

Jak vyplývá z názvů, metody *Connect()* a *Disconnect()* implementují připojení a odpojení serveru na dané adrese a portu. Metody *Recv()* a *Send()* pak zprostředkovávají příjem a odeslání bloku dat definovaného ukazatelem a délkou v bytech. Třetím parametrem je maximální časový interval v milisekundách pro vypršení operace.

Knihovna tedy pracuje v blokujícím režimu, protože návrat z funkce proběhne až v okamžiku, kdy je přenesen požadovaný blok dat, nebo dojde k chybě. Chybové stavy jsou ošetřeny formou výjimek reprezentovaných třídou *EWSAError*, která oproti bázové třídě *exception* uchovává navíc chybový kód.

Vnitřně komunikační metody využívají funkce *WSASend()* a *WSARecv()* v tzv. „overlapped“ režimu, který dovoluje definování časového intervalu pro vypršení operace.

3.7.2 Knihovna k ovládání DIO portu NE-4100T

Ovládání vstupně-výstupních portů serveru NE-4100T vyžaduje zasílání příkazů ve formátu definovaném v uživatelském manuálu [1] na patřičném TCP portu. Formát

příkazu a odpovědi specifikuje tabulka 3.22. Odpověď se liší pouze v poli *status*, které obsahuje návratový kód a u příkazu nemá význam. V případě správného vykonání příkazu vrací server hodnotu 00_{hex}, ostatní hodnoty znamenají chybu, jak popisuje tabulka 3.23.

#	Název	Hodnota	Popis
1	příkaz	2	Identifikátor příkazu (konstanta)
2	verze	2	Verze (konstanta)
3	status	[1..6, 255]	Návratový kód v případě odpovědi
4	délka dat	3	Délka dat (konstanta)
5	data	[0..3]	Číslo DIO kanálu
6	data	0, 1	Funkce (0 = vstup, 1 = výstup)
7	data	0, 1	Úroveň výstupu (0 = low, 1 = high)

Tabulka 3.22: Formát příkazu a odpovědi pro nastavení DIO signálu NE-4100T [1]

Status	Popis
00_{hex}	Příkaz úspěšně vykonán
01_{hex}	Neznámá chyba příkazu
02_{hex}	Příkaz není podporován touto verzí
03_{hex}	Nesouhlasí délka dat
04_{hex}	Neplatný mód
05_{hex}	Paket je příliš krátký
06_{hex}	Neplatné číslo DIO portu
FF_{hex}	Neznámá chyba

Tabulka 3.23: Návratové kódy příkazu pro nastavení DIO signálu NE-4100T [1]

Na základě popsané komunikace pracuje třída *CMoxaGPIO*, která slouží k ovládání DIO signálů NE-4100T veřejnými metodami:

```
void Connect(const char *pAddress, WORD Port);
void Disconnect(void);
void SetDIO(char Port, GPIOMode Mode, GPIOState State, int TimeOut =
    INFINITE);
```

Metody *Connect()* a *Disconnect()* zajišťují sestavení a uzavření spojení se serverem. Metoda *SetDIO()* má na starosti nastavení stavu DIO signálu určeného parametrem *Port*. Funkci signálu definuje parametr *Mode*, který může nabývat hodnot *CMoxaGPIO::Output* a *CMoxaGPIO::Input*. V případě výstupního módu pak nabývá na funkčnosti parametr *State*, který určuje stav výstupu *CMoxaGPIO::Low* nebo *CMoxaGPIO::High*.

Chybové stavy jsou opět ošetřeny metodou výjimek třídy *EMoxaGPIOError*.

3.7.3 Knihovna rozhraní bootloaderu

Aplikační rozhraní třídy *CAEBLtcp* pro ovládání bootloaderu obsahuje dvě skupiny funkcí. První skupina sdružuje metody, které mohou být volány pouze, není-li bootloader aktivní:

```
void SetTimeOut(int Value);
void ResetTarget(const char *pAddress, WORD GPIOPort);
WORD EnterBootLoader(const char *pAddress, WORD GPIOPort, WORD DataPort
    );
```

Metoda *SetTimeOut()* nastavuje globální dobu vypršení komunikace, kterou specifikuje parametr *Value* v ms. Metoda *ResetTarget()* vyvolá reset vzdáleného systému bez vstupu do bootloaderu.

Nejdůležitější metodou je *EnterBootLoader()* pro spuštění bootloaderu. Parametry jsou adresa serveru NE-4100T, datový port a port uživatelských vstupů-výstupů. Návratová hodnota funkce udává verzi bootloaderu vzdáleného systému.

Ostatní metody mohou být použity pouze po úspěšném volání *EnterBootLoader()*. V opačném případě je generována patřičná výjimka. Do této skupiny patří:

```
void RunApplication(void);
void FlashErase(void);
void FlashWrite(char *pImage, int ImageSize);
void LockWrite(BYTE Lock);
TAVRConfig ConfigRead(void);
DWORD SignatureRead(void);
```

Metoda *RunApplication()* je párový příkaz k *EnterBootLoader()*, který zajistí ukončení bootloaderu a spuštění uživatelské aplikace. Z tohoto důvodu musí být vždy vykonán.

Metody *FlashErase()* a *FlashWrite()* slouží ke správě obsahu paměti programu. První provádí její vymazání, druhá zápis binárního obrazu, který je dán ukazatelem *pImage* a velikostí *ImageSize* v bytech.

Metoda *LockWrite()* provádí úlohu nastavení konfiguračních bitů paměti programu na hodnotu danou parametrem *Lock*. Význam bitů odpovídá chování uvedenému v katalogovém listu, kdy hodnota 0 znamená aktivovaný bit a hodnota 1 neaktivovaný.

Zbývající metody slouží ke čtení konfigurace a podpisu MCU. Konfiguraci reprezentuje struktura *TAVRConfig* s patřičnými položkami.

V případě chybových stavů bootloaderu je generována výjimka třídy *EAEBLError*, nebo některá z dříve uvedených, dojde-li k chybě na nižší úrovni.

3.7.4 Konzolová aplikace

Finální konzolová aplikace „aebltool“ tedy využívá rozhraní třídy *CAEBLtcp* a je vytvořena jako čistá Win32 aplikace. Veškeré příkazy lze specifikovat z příkazové řádky, a to ve formátu:

`aebltool.exe <target_address> [options] <action>`

kde povinný parametr *target_address* specifikuje IP adresu cílového serveru NE-4100T. Druhou možností je zadání adresy ve tvaru doménového jména, které se následně přeloží na IP adresu prostřednictvím systému DNS.

Nepovinné parametry *options* udává tabulka 3.24. Jsou jimi čísla TCP portů serveru NE-4100T v případě, liší-li se od standardních hodnot používaných bootloaderem.

Třetím parametrem je příkaz *action*, který definuje činnost požadovanou od bootloaderu. Dostupné příkazy uvádí tabulka 3.24 a vždy musí být definován pouze jeden příkaz.

Při zápisu uživatelské aplikace následuje za příkazem název souboru, který má být použit jako obraz aplikace. Obsahuje-li název souboru mezery, musí být opatřen uvozovkami. Aplikace dovoluje použít obraz v binárním nebo Intel HEX formátu, přičemž formát souboru je automaticky rozpoznán. Rozpoznání typu souboru není založené na příponě souboru, ale pracuje na principu testování jeho obsahu. Není-li první řádek správně dekódován, jako záznam typu Intel HEX, považuje se soubor za binární. U formátu Intel HEX je z rozšířených adresování (pro obsah větší než 2^{16} B) podporován pouze záznam typu *Start Segment Address Record* s kódem 03_{hex}.

Příkaz zápisu konfiguračních bitů paměti programu vyžaduje za příkazem hodnotu, která bude zapsána v hexadecimálním formátu. Zápis konfiguračních bitů musí být navíc následně opětovně potvrzen uživatelem, aby nedošlo k nechtěnému zápisu, a tak zablokování funkcí bootloaderu.

Příkaz (action)

-r	Reset vzdáleného zařízení
-e	Vymazání uživatelské aplikace
-f:<filename>	Zápis uživatelské aplikace
-ef:<filename>	Vymazání a zápis uživatelské aplikace
-c	Čtení konfigurace MCU
-s	Čtení podpisu MCU
-wl:<HEX value 00 to FF>	Zápis konfiguračních bitů paměti programu

Volby (options)

-gp:<port_number>	DIO port serveru NE-4100T (výchozí: 4002)
-dp:<port_number>	Datový port serveru NE-4100T (výchozí: 4001)
-t:<timeout>	Doba vypršení komunikace [ms] (výchozí: 5000)

Tabulka 3.24: Parametry příkazové řádky uživatelské aplikace

Příklady použití

Vymazání uživatelské aplikace bootloaderu s IP adresou 192.168.127.254:

```
aebltool.exe 192.168.127.254 -e
```

Zápis obrazu uživatelské aplikace ze souboru „demo.hex“ ve formátu Intel HEX:

```
aebltool.exe 192.168.127.254 -f:"demo.hex"
```

Vymazání aplikace a zápis obrazu ze souboru „demo.bin“ v binárním formátu:

```
aebltool.exe 192.168.127.254 -ef:"demo.bin"
```

Nastavení konfiguračních bitů paměti programu na hodnotu CC_{hex}:

```
aebltool.exe 192.168.127.254 -wl:CC
```

Nestandardní nastavení portů (datový port 5000, DIO port 6000):

```
aebltool.exe 192.168.127.254 -dp:5000 -gp:6000 -ef:"demo.bin"
```

4 Výsledky

Na základě zvoleného řešení bylo vytvořeno softwarové vybavení PC, firmware mikrokontroléru a deska plošných spojů hardwarové části. Deska plošných spojů byla následně osazena a zařízení oživeno.

Při testování prototypu byly odhaleny drobné nedostatky na desce plošných spojů. Prvním problémem bylo chybné zapojení signalizačních LED diod rozhraní Ethernet z důvodu použití nesprávné knihovny definující zapojení LED na konektoru RJ-45. Druhý nedostatek spočíval v nedefinovaném stavu signálu, který ovládá pomocný tranzistor pro resetování cílového mikrokontroléru, v průběhu resetu modulu NE-4100T. Finální zapojení bylo z těchto důvodů patřičně upraveno správným zapojením signalizačních LED a doplněním o rezistor R6, který definuje klidovou úroveň tranzistoru T1.

U modulu NE-4100T byly při testování pozorovány nepříjemné vlastnosti. Za nejzávažnější lze považovat hodnoty logických úrovní výstupů. Ačkoliv má být modul napájen stabilizovaným napětím $5\,\mathrm{V}$ a dokumentace všude uvádí úrovně TTL [1], má logická úroveň 1 hodnotu $3{,}3\,\mathrm{V}$. Mezi další nevýhody patří vysoký příkon, který téměř nezávisí na činnosti serveru a blíží se v klidu maximálnímu deklarovanému příkonu $1{,}5\,\mathrm{W}$ [2]. Modul se navíc znatelně zahřívá.

Funkce bootloaderu byly prakticky ověřeny na modulu CPU unit s mikrokontrolérem ATmega644 a s externím watch dog časovačem, dále na MCU typu ATmega644 v konfiguraci s interním WDT a nakonec na mikrokontroléru ATmega16 bez WDT. Poslední MCU navíc pracoval s nižším taktovacím kmitočtem o hodnotě $3{,}6864\,\mathrm{MHz}$. Vyzkoušeno bylo i snížení přenosové rychlosti na hodnotu $9600\,\mathrm{bit/s}$.

Komunikace byla testována jednak při přímém propojení se síťovou kartou PC a jednak při vzdáleném spojení prostřednictvím veřejné sítě Internet.

K ověření funkčnosti systému kontrolujícího integritu uživatelské aplikace byl použit sériový programátor. Paměť programu typu flash dovoluje vícenásobný zápis do stejné oblasti, ale lze měnit pouze stav bitu s hodnotou „1“ na hodnotu „0“. S využitím tohoto principu byl pozměněn kód aplikace bez zásahu do ostatních částí.

Ve všech případech bootloader pracoval dle navržených požadavků. Úspěšně fungoval při rekonfiguraci na HW s jiným MCU, jiným taktovacím kmitočtem i přenosovou rych-

lostí. Úspěšný byl i test při komunikaci sítí Internet, kdy se na rozdíl od přímého spojení síťových karet značně prodlužují časové odezvy, protože může dojít i k poškození či zahození některých rámců aktivními prvky na trase. Tyto stavy musí být vyřešeny transportním protokolem TCP, což ale ovlivní časovou odezvu.

V případě pozměnění uživatelské aplikace došlo ke správné identifikaci rozdílného kontrolního součtu, a tak k zablokování jejího spuštění. Při správné konfiguraci monitorování signálu DCD, který signalizuje chybu kontrolního součtu, zasílá server NE-4100T hlášení o změně na požadovanou emailovou adresu.

5 Závěr

V projektu byla rozebrána problematika vzdálené správy firmware mikrokontrolérů řady Atmel AVR s ohledem na zadané požadavky. Na jejich základě pak byl navržen požadovaný hardware a software.

Hardwarovou část tvoří modul propojující embedded server NE-4100T s deskou mikrokontroléru. Ten navíc obsahuje snižující měnič napětí pro napájení zbytku zařízení a hodiny reálného času, které doplňují funkce modulu mikrokontroléru. Hardwarová část proto úzce souvisí dodaným modulem CPU unit.

Softwarové vybavení se skládá z firmware mikrokontroléru a uživatelského programu pro PC. Princip vzdálené správy firmware byl založen na funkci „Self Program Memory“, kterou disponuje většina mikrokontroléru Atmel AVR. Jedná se o princip mazání a zápisu do paměti programu prostřednictvím instrukce *SPM* vykonávané z vyhrazené oblasti programem zvaným bootloader.

Bootloader dovoluje prostřednictvím konfiguračních parametrů pro překlad adaptaci na různé typy mikrokontrolérů rodiny Atmel AVR a s hardwarem je spjat pouze způsobem propojení se serverem NE-4100T. Teoreticky je tedy možné použít kterýkoliv typ vybavený funkcí „Self Program Memory“ s velikostí sekce bootloaderu $2\,\mathrm{kB}$ a sériovým rozhraním. Prakticky byly testovány obvody ATmega644P a ATmega16.

Realizované řešení dovoluje mazání a zápis paměti programu, zápis konfiguračních bitů (Lock Bits), čtení konfiguračních bitů (Lock, Fuse Low, Fuse High, Extended Fuse) a čtení podpisu mikrokontroléru. Doplňkovou funkcí je kontrola integrity uživatelské aplikace ověřením kontrolního součtu. Cílem tohoto opatření je zamezit spuštění poškozeného programu, a tak zvýšit funkční bezpečnost. Bootloader dále dovoluje použití časovače typu watch dog. Tím může být integrovaná periferie, nebo externí obvod typu MB3773.

Z hlediska zabezpečení proti neoprávněnému přístupu nebylo nalezeno uspokojivé řešení. Problém spočívá především v embedded serveru NE-4100T, který nedovoluje vyšší úroveň zabezpečení než filtrování povolených IP adres, což tvoří nejslabší článek. Veškerá uvažovaná dodatečná opatření pak již nedokáží bezpečnost zvýšit.

Reference

[1] MOXA. *NE-4100 Series User's Manual*. 2008/06. 121 s. [online]. [cit. 2009-03-03] URL: <http://www.moxa.com/doc/man/NE-4100_Series_Users_Manual_v9.pdf>

[2] MOXA. *NE-4100 Series Datasheet*. 2009/11. 2 s. [online]. [cit. 2010-01-03] URL: <http://www.moxa.com/doc/specs/NE-4100_Series.pdf>

[3] MOXA. *NE-4100 Series Serial Command Mode User's Guide*. 2004/11. 25 s. [online]. [cit. 2009-03-03] URL: <http://www.moxa.com/doc/manual/NE/4100-CMD/NE-4100_Serial_Command_Mode_User_Guide_V1.pdf>

[4] Tibbo Technology. *Programmable Hardware Manual*. 2010. 214 s. [online]. [cit. 2010-03-05] URL: <http://www.tibbo.com/downloads/open/phm_manual.pdf>

[5] Soalle Systems. *EZL-50M Datasheet*. 2010. 2 s. [online]. [cit. 2010-03-05] URL: <http://www.sollae.com/en/Support/datasheet/ds_ezl50men.pdf>

[6] AK-NORD. *AK XXL Product Line Manual*. Version 1.0. 91 s. [online]. [cit. 2010-03-05] URL: <http://www.ak-nord.de/en/daten/manual_xxl_ts.pdf>

[7] FiveCo. *FMod-TCP DB Datasheet*. 2007/12. 1 s. [online]. [cit. 2010-03-05] URL: <http://www.fiveco.ch/datafiles/support/tcp/FMod-TCP_DB_v2.5.pdf>

[8] Dostálek, L. *Velký průvodce protokoly TCP/IP a systémem DNS*. 5. vydání. Computer Press, 2008, ISBN: 978-80-251-2236-5

[9] INFORMATION SCIENCE INSTITUTE. *Transmission Control Protocol*. RFC 793, 1981. [online]. [cit. 2010-04-09] URL: <http://www.ietf.org/rfc/rfc793.txt>

[10] INFORMATION SCIENCE INSTITUTE. *User Datagram Protocol*. RFC 768, 1980. [online]. [cit. 2010-04-09] URL: <http://www.ietf.org/rfc/rfc768.txt>

[11] ATMEL. *ATmega644 Preliminary*. 2008/07. 376 s. [online]. [cit.2010-01-20]. URL: <http://www.atmel.com/dyn/resources/prod_documents/doc2593.pdf>

[12] ATMEL. *AVR Instruction Set*. 2007/09. 155 s. [online]. [cit.2010-01-20]. URL: <http://www.atmel.com/dyn/resources/prod_documents/doc0856.pdf>

[13] WIZnet. *W3150A Datasheet*. 2004. 65 s. [online]. [cit. 2009-03-03] URL: <http://www.wiznet.co.kr/en>

[14] FUJITSU SEMICONDUCTOR. *Power Supply Monitor with Watch-Dog Timer MB3773*. 2004/09. 29 s. [online]. [cit. 2009-03-03] URL: <http://www.fujitsu.com/downloads/MICRO/fma/pdf/e427401.pdf>

[15] NATIONAL SEMICONDUCTOR. *LP2950/LP2951 Series of Adjustable Micropower Voltage Regulators*. 2009/06. 26 s. [online]. [cit.2010-01-20]. URL: <http://www.national.com/ds.cgi/LP/LP2950.pdf>

[16] MOXA. *NE-4100T recommend circuit design*. 2004/05. 8 s. [online]. [cit. 2009-03-03] URL: <http://www.moxa.com/doc/manual/NE/4100/Circuit/4100T/v1.2/NE4100Tcircuit_12.pdf>

[17] NXP Semiconductors. *PCF8563 Product data sheet*. 2008/02. 32 s. [online]. [cit. 2010-03-21] URL: <http://www.nxp.com/documents/data_sheet/PCF8563.pdf>

[18] NXP Semiconductors. *User Manual for NXP Real Time Clocks PCF85x3, PCA8565 and PCF2123, PCA2125*. 2008/12. 32 s. [online]. [cit. 2010-03-21] URL: <http://www.nxp.com/documents/user_manual/UM10301.pdf>

[19] NXP Semiconductors. *BAV74 High-speed double diode*. 2004/01. 9 s. [online]. [cit. 2010-03-21] URL: <http://www.nxp.com/documents/data_sheet/BAV74.pdf>

[20] EPSON TOYOCOM. *MC-306 Datasheet*. 2 s. [online]. [cit. 2010-03-21] URL: <http://www.epsontoyocom.co.jp/english/product/Crystal/set01/mc306/index.html>

[21] National Semiconductor. *LM2674 Datasheet*. 2005/02. 26 s. [online]. [cit. 2010-03-04] URL: <http://www.national.com/ds/LM/LM2674.pdf>

[22] MATSUTA. *SMD Power Inductor SC/SCD Series*. 2010. 1 s. [online]. [cit. 2010-03-04] URL: <http://www.matsuta.com/file/file/50.pdf>

[23] Amphenol. *Modular Jacks*. 2010. 29 s. [online]. [cit. 2010-03-04] URL: <http://www.amphenolcanada.com/ProductSearch/pdf/Modular_CAT.pdf>

Seznam symbolů, veličin a zkratek

DHCP Dynamic Host Control Protocol

DIL Dual In Line

DIO digitální vstup/výstup

DLL dynamicky linkovaná knihovna

DoS Denial Of Service

ESD Electrostatic Discharge

HAL žárové cínování

JTAG Joint Test Action Group

MAC Media Access Control

MCU mikrokontrolér

NECI Network Enabler Configuration Interface

NRWW No Read-While-Write

RISC redukovaná instrukční sada

RWW Read-While-Write

SPI Serial Peripheral Interface

SMTP Simple Mail Transfer Protocol

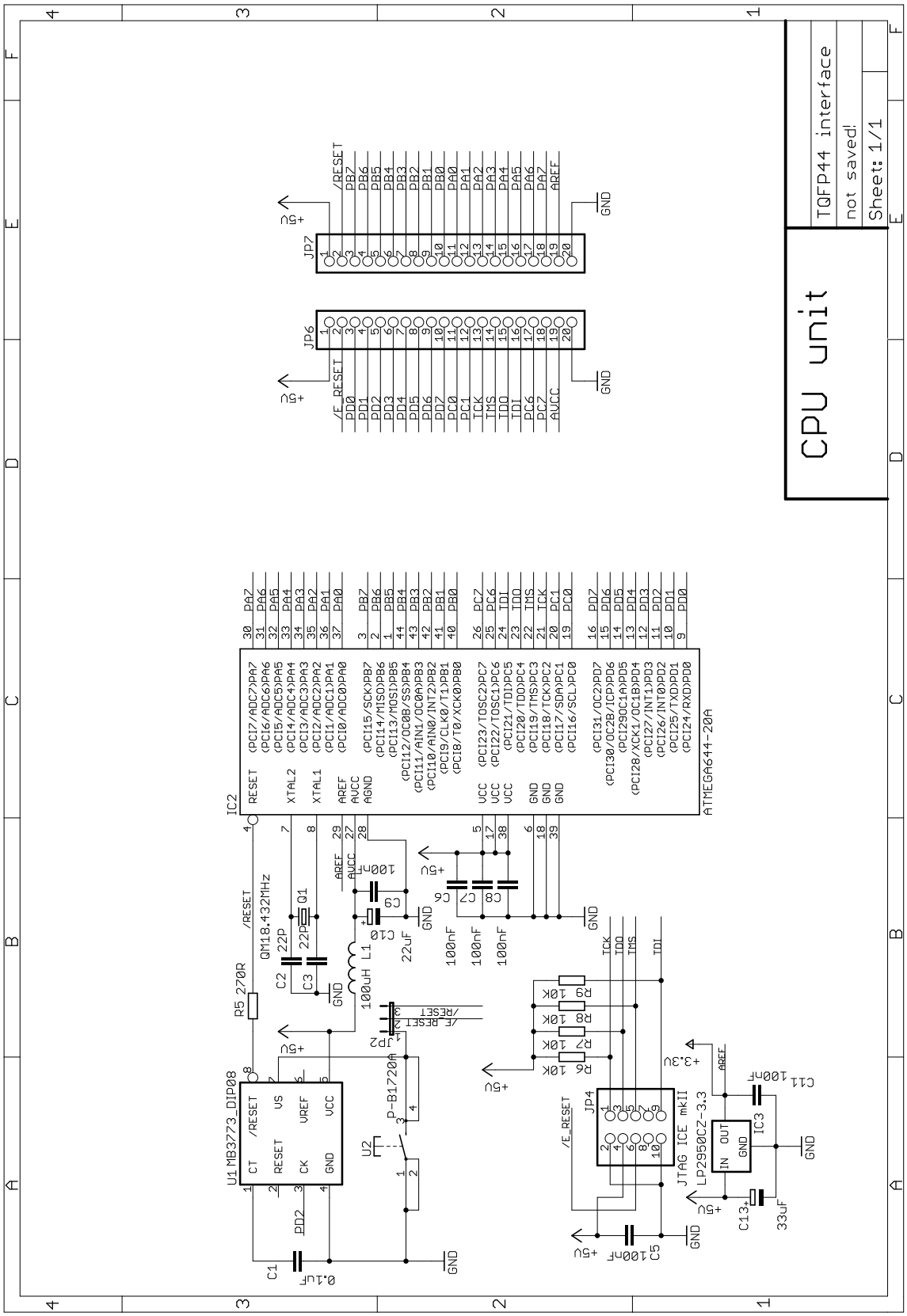
SNMP Simple Network Management Protocol

UART univerzální asynchronní rozhraní

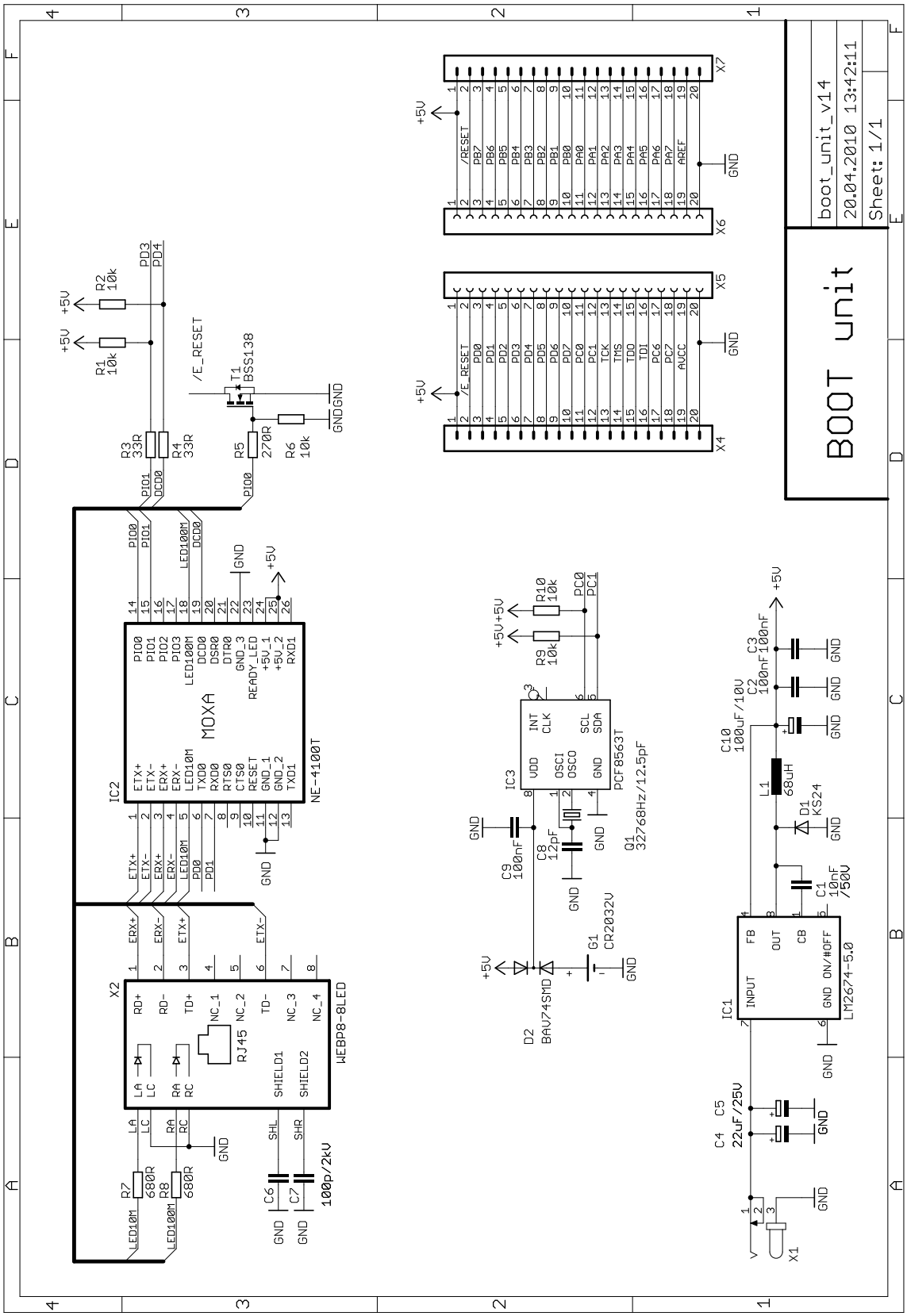
WDT Watchdog Timer

Seznam příloh

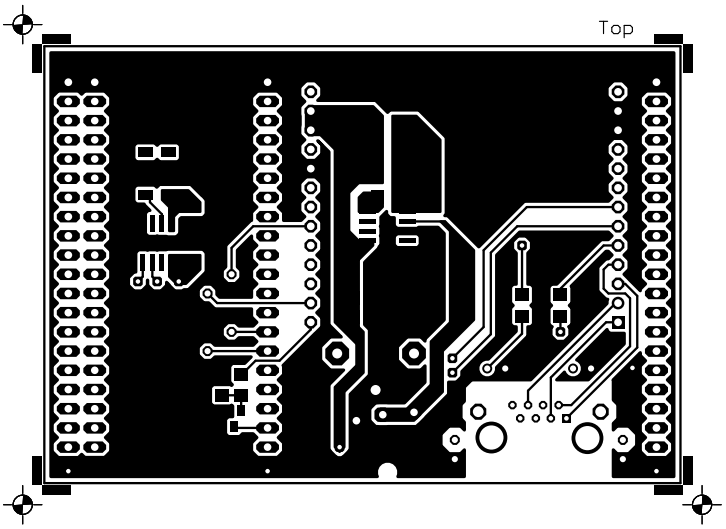
A Schéma zapojení modulu CPU unit



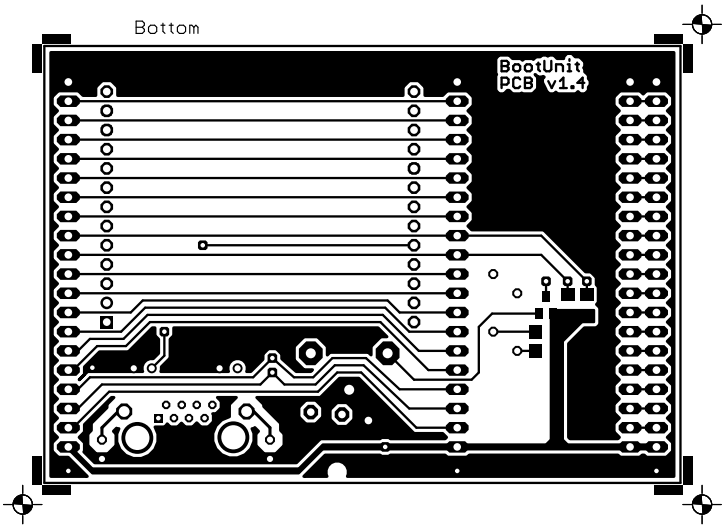
B Schéma zapojení modulu BOOT unit



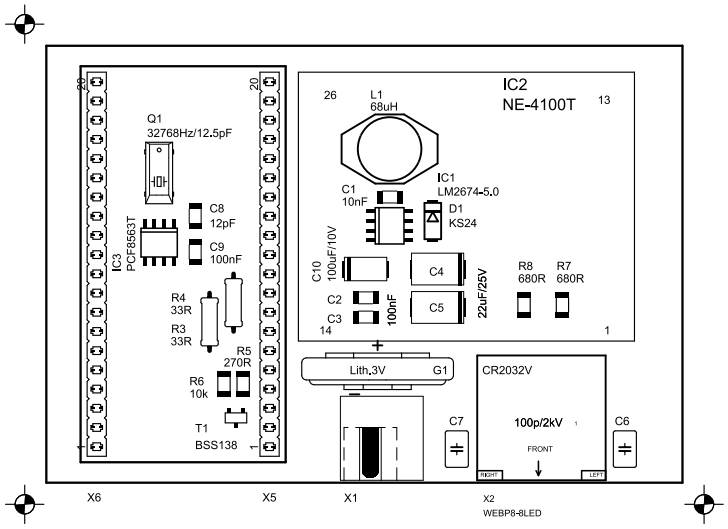
C Desky plošných spojů modulu BOOT unit



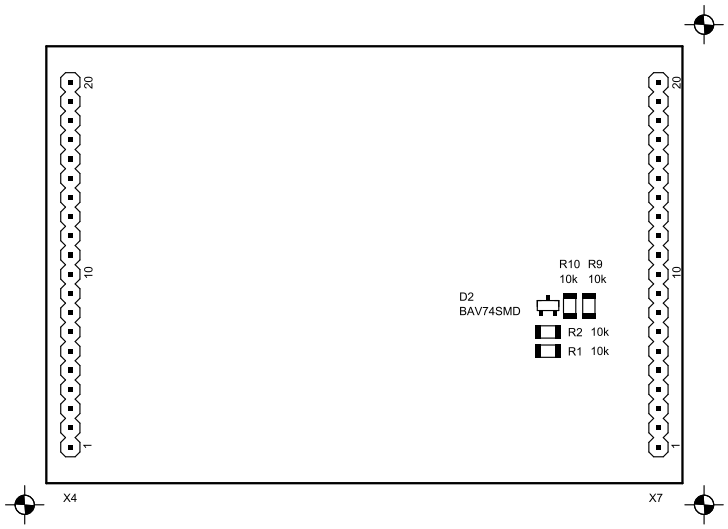
Obrázek C.1: Deska plošných spojů - vrstva „Top“



Obrázek C.2: Deska plošných spojů - vrstva „Bottom“



Obrázek C.3: Rozložení součástek - vrstva „Top“



Obrázek C.4: Rozložení součástek - vrstva „Bottom“

D Seznam součástek

Označení	Součástka	Pouzdro
R3, R4	rezistor 33Ω 5% $0,25\mathrm{W}$	0207
R5	rezistor 270Ω 5% $0,25\mathrm{W}$	SMD 1206
R7, R8	rezistor 680Ω 5% $0,25\mathrm{W}$	SMD 1206
R1, R2, R6, R9, R10	rezistor $10\mathrm{k}\Omega$ 5% $0,25\mathrm{W}$	SMD 1206
C8	kondenzátor $12\mathrm{pF}$ 5% 25V NP0	SMD 1206
C6, C7	kondenzátor $100\mathrm{pF}$ $2\mathrm{kV}$ keramický	rozteč $5\mathrm{mm}$
C1	kondenzátor $10\mathrm{nF}$ 10% 50V X7R	SMD 1206
C2, C3, C9	kondenzátor $100\mathrm{nF}$ 10% 25V X7R	SMD 1206
C4, C5	kondenzátor $22\mu\mathrm{F}$ 25V tantalový	SMD-D
C10	kondenzátor $100\mu\mathrm{F}$ 10V tantalový	SMD-C
L1	cívka MATSUTA SC75F-680 ($68\mu\mathrm{H}$, $0,22\Omega$)	SC75F
IC1	LM2674-5.0	SO8
IC2	MOXA NE-4100T + 2x dutinková lišta 1x13	
IC3	PCF8563T	SO8
T1	tranzistor N-MOSFET typ BSS138	SOT23
D1	Schottkyho dioda KS24	DO-214AA
D2	dvojitá dioda BAV74	SOT23
Q1	krystal MC-306 32768Hz $C_L = 12,5\mathrm{pF}$	MC-306
G1	lithiová baterie CR2032 do DPS vertikální	CR2032V
X1	napájecí konektor, vidlice $2,1\mathrm{mm}$ do DPS	SCD-016
X2	zásuvka RJ45 typ RJHS-5381	RJHS-5381
X4 až X7	dutinková lišta 1x20 pinů	

E Přehled MCU řady Atmel AVR 8-bit

Mikrokontrolér	**Flash** [kB]	**EEPROM** [kB]	**SRAM** [B]	**Self Program Memory**	**UART**	**JTAG**
ATtiny24 Automotive	2	0.125	128	Ano	0	Ano
ATtiny25 Automotive	2	0.125	128	Ano	0	Ne
ATtiny261 Automotive	2	0.125	128	Ano	0	Ne
ATtiny44 Automotive	4	0.25	256	Ano	0	Ne
ATtiny45 Automotive	4	0.25	256	Ano	0	Ne
ATtiny461 Automotive	4	0.25	256	Ano	0	Ne
ATtiny84 Automotive	8	0.5	512	Ano	0	Ne
ATtiny85 Automotive	8	0.5	512	Ano	0	Ne
ATtiny861 Automotive	8	0.5	512	Ano	0	Ne
ATmega16HVA	16	0.256	512	Ano	0	Ano
ATmega16HVB	16	0.512	1024	Ano	0	Ano
ATmega32HVB	32	1	2048	Ano	0	Ano
ATmega406	40	0.5	2048	Ano	0	Ne
ATmega4HVD	4	0.256	512	Ano	0	Ne
ATmega8HVA	8	0.256	512	Ano	0	Ne
ATmega8HVD	8	0.256	512	Ano	0	Ne
AT90PWM1	8	0.5	512	Ano	0	Ne
ATtiny12	1	0.0625	0	Ne	0	Ne
ATtiny13A	1	0.0625	64	Ano	0	Ne
ATtiny24	2	0.125	128	Ano	0	Ne
ATtiny24A	2	0.125	128	Ano	0	Ne
ATtiny25	2	0.125	128	Ano	0	Ne
ATtiny261	2	0.125	128	Ano	0	Ne
ATtiny28L	2	–	32	Ne	0	Ne
ATtiny43U	4	0.0625	256	Ne	0	Ne
ATtiny44A	4	0.25	256	Ano	0	Ne
ATtiny45	4	0.25	256	Ano	0	Ne
ATtiny461	4	0.25	256	Ano	0	Ne
ATtiny48	4	0.0625	256	Ano	0	Ne
ATtiny84	8	0.5	512	Ano	0	Ne
ATtiny85	8	0.5	512	Ano	0	Ne
ATtiny861	8	0.5	512	Ano	0	Ne

Mikrokontrolér	**Flash** [kB]	**EEPROM** [kB]	**SRAM** [B]	**Self Program Memory**	**UART**	**JTAG**
ATtiny88	8	0.0625	512	Ano	0	Ne
ATmega168 Automotive	16	0.5	1024	Ano	1	Ne
ATmega169P Automotive	16	0.5	1024	Ano	1	Ano
ATmega32C1 Automotive	32	1	2048	Ano	1	Ano
ATmega32M1 Automotive	32	1	2048	Ano	1	Ano
ATmega48 Automotive	4	0.25	512	Ano	1	Ne
ATmega64C1 Automotive	64	2	4096	Ano	1	Ano
ATmega64M1 Automotive	64	2	4096	Ano	1	Ano
ATmega88 Automotive	8	0.5	1024	Ano	1	Ne
ATtiny167 Automotive	16	0.5	512	Ano	1	Ne
ATmega169P	16	0.5	1024	Ano	1	Ano
ATmega329	32	1	2048	Ano	1	Ano
ATmega3290	32	1	2048	Ano	1	Ano
ATmega3290P	32	1	2048	Ano	1	Ano
ATmega329P	32	1	2048	Ano	1	Ano
ATmega649	64	2	4096	Ano	1	Ano
ATmega6490	64	2	4096	Ano	1	Ano
AT90PWM2	8	0.5	512	Ano	1	Ne
AT90PWM216	16	0.5	1024	Ano	1	Ne
AT90PWM2B	8	0.5	512	Ano	1	Ne
AT90PWM3	8	0.5	512	Ano	1	Ne
AT90PWM316	16	0.5	1024	Ano	1	Ne
AT90PWM3B	8	0.5	512	Ano	1	Ne
ATmega165P	16	0.5	1024	Ano	1	Ano
ATmega168	16	0.5	1024	Ano	1	Ne
ATmega168P	16	0.5	1024	Ano	1	Ne
ATmega16A	16	0.5	1024	Ano	1	Ano
ATmega325	32	1	2048	Ano	1	Ano
ATmega3250	32	1	2048	Ano	1	Ano
ATmega3250P	32	1	2048	Ano	1	Ano
ATmega325P	32	1	2048	Ano	1	Ano
ATmega328P	32	1	2048	Ano	1	Ne
ATmega32A	32	1	2048	Ano	1	Ano
ATmega48PA	4	0.25	512	Ano	1	Ne

Mikrokontrolér	**Flash** [kB]	**EEPROM** [kB]	**SRAM** [B]	**Self Program Memory**	**UART**	**JTAG**
ATmega644	64	2	4096	Ano	1	Ano
ATmega645	64	2	4096	Ano	1	Ano
ATmega6450	64	2	4096	Ano	1	Ano
ATmega8	8	0.5	1024	Ano	1	Ne
ATmega8515	8	0.5	512	Ano	1	Ne
ATmega8535	8	0.5	512	Ano	1	Ne
ATmega88P	8	0.5	1024	Ano	1	Ne
ATmega88PA	8	0.5	1024	Ano	1	Ne
ATtiny2313	2	0.125	128	Ano	1	Ne
AT90CAN128 Automotive	128	4	4096	Ano	2	Ano
AT90CAN32 Automotive	32	1	2048	Ano	2	Ano
AT90CAN64 Automotive	64	2	4096	Ano	2	Ano
ATmega164P Automotive	16	0.5	1024	Ano	2	Ano
ATmega324P Automotive	32	1	2048	Ano	2	Ano
ATmega644P Automotive	64	2	4096	Ano	2	Ano
ATmega128	128	4	4096	Ano	2	Ano
ATmega1281	128	4	8192	Ano	2	Ano
ATmega1284P	128	4	16K	Ano	2	Ano
ATmega162	16	0.5	1024	Ano	2	Ano
ATmega164P	16	0.5	1024	Ano	2	Ano
ATmega164PA	16	0.5	1024	Ano	2	Ano
ATmega2561	256	4	8192	Ano	2	Ano
ATmega324P	32	1	2048	Ano	2	Ano
ATmega324PA	32	1	2048	Ano	2	Ano
ATmega64	64	2	4096	Ano	2	Ano
ATmega644P	64	2	4096	Ano	2	Ano
ATmega1280	128	4	8192	Ano	4	Ano
ATmega2560	256	4	8192	Ano	4	Ano
ATmega640	64	4	8192	Ano	4	Ano