Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Návrh a implementace HW a SW smart zařízení pro komunikaci v inteligentním domě

Diplomová práce

Autor: Jan Štěpán
Studijní obor: Aplikovaná informatika
Vedoucí práce: Mgr. Josef Horálek Ph.D.

Hradec Králové duben 2015

PROHLÁŠENÍ

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne

……

Jan Štěpán

PODĚKOVÁNÍ

Děkuj vedoucímu diplomové práce Mgr. Josefu Horálkovi Ph.D. za cenné rady, nápady a čas, které mi při psaní této práce věnoval. Také bych zde rád poděkoval mým přátelům a rodině za podporu během celého mého studia.

ANOTACE

Diplomová práce se zabývá návrhem a implementací nového systému pro řízení inteligentního domu. Na základě poznatků současných vědeckých přístupů a dostupných komerčních řešení jsou stanoveny požadavky na nový systém. Je představena třívrstvá architektura nového řešení a jsou detailně rozepsány vlastnosti a parametry dvou nižších vrstev, nodů a subsystému. Dále jsou shrnuty požadavky na hardwarové vybavení těchto vrstev. Jsou navrženy a vyrobeny prototypy vybraných nodů a také implementována aplikace pro subsystém.

ANOTATION

Title: Design and implementation of smart device's hardware and software for communication in smart home

This master's thesis goal is to design and implement new solution for controlling smart homes. Demands for new system are made on the basis of current scientific research and commercially available solutions. New three layer architecture is presented and details of parameters of two lower layers, nodes and subsystem are described in detail. Requirements for these layers are summarized. Hardware and applications are made for selected nodes and subsystem.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

1. ÚVOD

Žijeme v době rozvoje inteligentních domů. Inteligentním domem je myšlen takový dům, který je postaven z moderních materiálů umožňujících nízkoenergetický provoz. Je cíleně situován na místě, kde dlouho svítí slunce, je kvalitně izolován od venkovního prostředí. Dále obsahuje hardwarové a softwarové prostředky umožňující automatizovat běžné úkony a tím ještě více šetřit provozní náklady a zvyšovat svým obyvatelům životní komfort. Hardwarovými prostředky jsou moduly rozmístěné po domě, které jsou propojeny (drátově nebo bezdrátově) s řídící jednotkou. Softwarem jsou pak jednoduché programy v modulech a řídící logika uvnitř jednotky. Tyto prostředky také ulehčují ovládání domu. Je možné jej ovládat tradičními prostředky, jako jsou tlačítka nebo spínače a zároveň využívat například dotykové panely rozmístěné v domě, chytré telefony, tablety nebo počítače, případně notebooky. Dům je možné ovládat také na dálku a například při odjezdu ze zaměstnání zapnout vytápění v domě.

Cílem práce je navrhnout a implementovat systém, tedy hardware i software, pro řízení inteligentního domu. Práce je logicky strukturovaná od návrhu k implementaci. Druhá kapitola popisuje dostupná komerční řešení a vědecké přístupy, a poté analyzuje jejich nedostatky. Z nich jsou stanoveny požadavky na nový systém, především možnost nechat obyvatele domu vytvářet si vlastní pravidla po řízení domu a zvýšení bezpečnosti oproti stávajícím systémům. V třetí kapitole je popsána logická architektura navrženého tří vrstvého systému. Čtvrtá kapitola popisuje nejnižší vrstvu systému obsahující nová zařízení, nahrazující moduly, zvané nody. Jsou popsány jejich společné vlastnosti a parametry. Je definována vnitřní struktura nodů a požadavky na jejich hardware. Pátá kapitola popisuje subsystém, což je prostřední část architektury systému. Jedná se o zařízení zvyšující bezpečnosti inteligentního domu. Subsystém je popsán z hlediska programového vybavení, logické struktury a požadavků na jeho funkcionalitu. Kapitola šest se věnuje výběru vhodného hardwaru pro subsystém a nody. Dále popisuje schématická zapojení vybraných nodů a podpůrných komponent. Sedmá kapitola představuje komunikační protokol mezi nody a subsystémem a také komunikační protokol mezi aplikacemi v subsystému. Také popisuje program navržených aplikací a popisuje průběh a výsledky testování. V závěru práce je shrnuto, co bylo úspěšně implementováno a jak je možné systém v budoucnu dále rozšířit.

2. REŠERŠE KONKURENČNÍCH PŘÍSTUPŮ

V první podkapitole jsou shrnuty poznatky vědeckých prací, které se zabývají problematikou chytrých domů. Druhá podkapitola popisuje vybraná komerčně dostupná řešení. Poslední podkapitola pak stanovuje požadavky na nový systém, který je třeba navrhnout a implementovat.

2.1. Vědecké přístupy

V článku [**1**] je navrhnut a implementován bezdrátový komunikační protokol, který využívá transmittery ZigBee [**2**]. Výsledná komunikační sít by mohla velice usnadnit šíření informací mezi moduly v inteligentním domě. V praxi to znamená, že pokud je nějaký modul příliš vzdálený od řídící jednotky a nemůže s ní komunikovat, ale dokáže komunikovat s jiným modulem, dokáže navržené řešení doručit informace k řídící jednotce a zpět.

Článek [**3**] popisuje algoritmus využitelný pro automatické učení systému chytrého domu. Představený adaptivní algoritmus CASAS hledá vzorce v každodenním využívání domu a poté se snaží tyto kroky sám provádět bez interakce lidí.

Dalším směrem, kterým je možné směřovat vývoj, popisuje článek [**4**]. Pokud jsou v domě rozmístěny bezpečnostní kamery, může v článku navržený algoritmus detekovat pád seniorů. Když je senior v domě sám, neměl by u sebe mobilní telefon a upadl, tak by se pomoc nemusela dostavit včas. Díky navrženém řešení může systém inteligentního domu odeslat upozornění ostatním rodinným příslušníkům, nebo zavolat záchrannou službu.

Současným trendem je tedy se zabývat návrhem komunikačních sítí a pokročilých algoritmů, které se samy učí z chování obyvatel domu, případně zvyšováním bezpečnosti rezidentů. Bohužel tyto algoritmy se zatím nepromítají do komerčně dostupných řešení. Ty si lidé mohou zakoupit a vybavit jimi během rekonstrukce stávající dům nebo s nimi počítat při návrhu novostavby. Všechna řešení se skládají z řídící jednotky, která ovládá a vyčítá stavy z modulů rozmístěných v domě. Moduly jsou s jednotkou propojeny buď metalickým médiem (tedy kabelem), nebo bezdrátově. Některá řešení umožňují obě varianty kombinovat.

2.2. Komerčně dostupná řešení

Na trhu se vyskytují desítky společností, které nabízejí své řešení pro chytré domy. Zde jsou vybrány čtyři systémy od různých společností, jejichž produkty je možné na českém trhu zakoupit a nainstalovat.

2.2.1. Systém Fibaro

Systém Fibaro vyrábí [5] společnost Fibar Group a nabízí výběr z následujících komponent:

- Home Center 2,
- Home Center Lite,
- detektor pohybu,
- detektor kouře,
- detektor zaplavení,
- bateriový magnet,
- LED stmívač,
- ovládání,
- zásuvka,
- uzávěr vody.

Home Center 2 je centrální bod jejich řešení a obsluhuje všechny ostatní komponenty. Podporuje také připojení VoIP telefonů, bezpečnostních kamer a využívání webových služeb. Uživatel může systém spravovat skrze webové rozhrání a nebo přes mobilní aplikace pro platformy Android, iOS nebo Windows Phone. Přes webové rozhrání si může uživatel vytvářet svá vlastní pravidla. Například pokud za předchozích 12 hodin nepršelo, tak zalít trávník. Pokud v domě není nikdo přítomen, tak uzavřít termostatické hlavice topení. Home Center Lite je odlehčená varianta Home Center 2 a může obsluhovat stejné moduly. Není ale vybavena tak výkonným procesorem, takže má zhoršenou odezvu. Nepodporuje také VoIP a bezpečnostní kamery.

Fibaro je kvalitně navržený jak po hardwarové tak i softwarové stránce. Moduly komunikují bezdrátově na proprietárním protokolu. Mohou pracovat z baterie i ze sítě. Fibaro má ale velkou nevýhodu. Jedná se o uzavřený ekosystém, který není možné rozšířit o jiné moduly než ty od Fibar Group. Pokud uživatel chce jiný modul, než má Fibar Group ve svém portfoliu, nemůže jej do domu integrovat. Toto proprietární uzamčení platí i pro

všechny další systémy. Jednotlivé moduly i řídící jednotka jdou zakoupit samostatně a uživatel si je může nainstalovat sám. Samozřejmě musí mít k montáži potřebnou kvalifikaci [**6**], jelikož se jedná o úpravu a zapojení elektroinstalace domu, tedy práci se síťovým napětím 230V.

2.2.2. Systém Ego-n

Systém Ego-n [**7**] vyrábí společnost ABB, která se také zabývá vývojem komponent pro elektroinstalace v klasických domech. Je možné kombinovat ovládání klasickými tlačítky a spínači spolu s dotykovými panely umístěnými v místnostech domu a také mobilní aplikací pro platformy Android a iOS. Ego-n lze využít pro regulaci osvětlení, vytápění a klimatizaci a také pro ovládání rolet a vrat. Obsahuje také funkce pro simulaci přítomnosti v domě. Pokud je rodina delší dobu pryč, systém dokáže sám regulovat osvětlení a rolety, a tím tak předstírat, že v domě jsou lidé. Bohužel ale neobsahuje žádné environmentální senzory, které monitorují kvalitu ovzduší ani bezpečnostní detektory, jako například detektor zaplavení a detektor kouře. Moduly je možné propojit s řídící jednotkou po kabelu, a vybrané moduly také bezdrátově.

Velkou nevýhodou systému Ego-n je nemožnost provést instalace jinak než přes autorizovaného dodavatele a nemožnost uživatelského zásahu do řídící jednotky. Jednotka je naprogramována při instalaci systému a bez zásahu servisního technika není možné změnit chování domu.

2.2.3. iNELS Bus system

Dalším systémem pro ovládání a řízení inteligentního domu je iNELS [**8**] od společnosti ELKO EP. Umožňuje vzdálený dohled nad děním v domě přes bezpečnostní kamery, vyhodnocování alarmu, ovládání rolet, spínání a stmívání světel, ovládání příjezdových vrat apod. Systém je možné ovládat přes chytré telefony, televize nebo přes dotykový panel. Bohužel neumožnuje tradiční ovládání nástěnnými spínači a regulátory, ani uživatelům vytvářet vlastní pravidla pro ovládání domu. Moduly společnost vyrábí v drátových i bezdrátových variantách a je možné je kombinovat.

2.2.4. Systém Loxone

Posledním vybraným systémem je Loxone [**9**] od společnosti Loxone s.r.o. Ta nabízí jak bezdrátové, tak i tradiční kabelové moduly. Nabízí také možnost programovat řídící jednotku a vytvářet tak nová pravidla pro chování domu přes aplikaci Loxone Config. Všechny moduly, které jsou zmíněny v předchozích systémech, obsahuje Loxone také.

2.3. Definice problému

Na jedné straně ukazují vědecké práce ohromné množství směrů, kam se vývoj inteligentních domů bude ubírat a jaké nové funkce bude mít. Na druhé straně existují běžně prodávané systémy, u kterých nejde měnit pravidla chování domu a ani je nelze po instalaci jednoduše rozšiřovat o další moduly. Žádný z komerčních systémů také neřeší následující otázku:

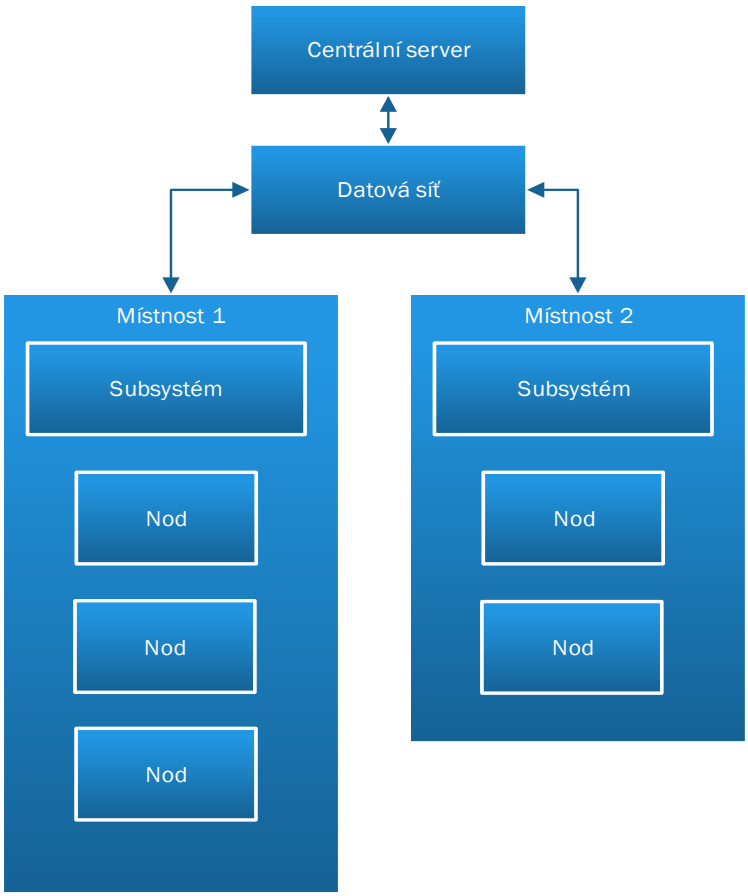
Co se stane při výpadku řídící jednotky?

Protože systémy pro řízení inteligentního domu jsou centralizované, výpadek nebo fatální porucha řídící jednotky bude mít velký dopad na obyvatele domu. Pokud je dům propojen se systémem úplně, a řízeno je veškeré vybavení domu, včetně osvětlení, vytápění, a i klasické ovládací prvky, jako spínače na stěnách jsou připojeny k jednotce, přestane být dům při poruše ovladatelný. Dokonce může být i neobyvatelný, a to do té doby, než je řídící jednotka nahrazena nebo opravena.

Z toho důvodu je navržena nová architektura pro řízení inteligentního domu. Prvním požadavkem, který musí splnit je zachování alespoň částečné funkcionality domu při poruše technického vybavení domu. Systém nesmí být uzavřený a všechny jeho části musí být dokumentovány a k dispozici pro další rozvoj. Dalším požadavkem je možnost nechat uživatele vytvářet vlastní pravidla pro ovládání domu a také umožnit jednoduché rozšiřování systému o nové moduly. Ty musí být konstrukčně jednoduché a široce využitelné. Systém také musí mít možnost tradičního ovládání pomocí nástěnných spínačů a regulátorů, protože ne každý člen domácnosti chce k ovládání domu používat dotykové panely nebo mobilní telefon.

3. PŘEDSTAVENÍ ARCHITEKTURY SYSTÉMU

Z předchozích kapitol jsou patrné nedostatky komerčně dostupných řešení. Většinou se jedná o řešení uzavřená, která nelze dostatečně modifikovat. Při výpadku řídící jednotky se dům může stát neobyvatelným, dokonce i nebezpečným. Z těchto důvodů byla navržena nová architektura chytrého domu, která se snaží tyto nedostatky a problémy eliminovat.



Obrázek 1 - Obecná architektura navrženého řešení [autor]

Na obrázku 1 je znázorněna obecná architektura nového řešení. Stávající řešení využívají řídící jednotku přímo komunikující bezdrátově anebo po metalickém vodiči s moduly. Jedna se tedy o dvouvrstvou architekturu. Oproti tomuto řešení je navržená architektura rozdělena do tří logických vrstev.

Nejnižší vrstva obsahuje nody. Pojmenování nod[1] bylo zvoleno pro odlišení od pojmu modul, který se u komerčních řešení používá nejčastěji. Každý výrobce totiž může pod pojmem modul představovat jiné zařízení. Někdy je modulem myšlen přímo nějaký ovládací prvek, senzor nebo aktuátor, který musí řídící jednotka obsluhovat. Modul také může obsahovat více různých prvků, případně mikrokontrolér a provádět některé činnosti samostatně. Proto byl zaveden termín nod. Nod je fyzické zařízení schopné obsluhovat ovládací prvky, manipulovat s aktuátory nebo vyčítat ze senzorů. Každý nod bude muset splňovat definované chování, přijímat a odesílat zprávy. Přesné vlastnosti všech nodů budou definovány v následující kapitole Logický návrh nodů.

Druhou, prostřední vrstvou je subsystém, ke kterému budou nody připojeny (ať již drátově anebo bezdrátově). Právě subsystém rozšiřuje stávající architekturu o nový prvek. V každé místnosti domu bude umístěn právě jeden subsystém. V případě opravdu malých místností (například toalety), nebo místností nevhodných pro provoz elektronických zařízení je samozřejmě možné použít jeden subsystém pro řízení více místností. Jeho primárním cílem je z nodů dle jejich typu vyčítat nebo nastavovat data nebo stavy. Tyto informace bude předávat nejvyšší vrstvě, centrálním serveru. Subsystém je možné dále rozdělit do dílčích logických celků, jak je popsáno v kapitole Logický návrh subsystému.

Na nejvyšší logické vrstvě je centrální server, který nahrazuje řídící jednotku. Jeho úkolem je řízení celého domu, vystavování webového rozhrání a webových služeb pro uživatele. Přes datovou síť je připojen k subsystémům, které jsou obsaženy v jednotlivých místnostech domu. Centrální server bude realizován jako běžný počítač postavený na architektuře x86-64. Výhodou tohoto řešení je snadná výměna komponent v případě jejich poruch. Uživatel tak nemusí spoléhat na podporu konkrétního proprietárního řešení výrobce. Spojení centrálního serveru se subsystémy bude probíhat po běžné datové síti.

V rámci této diplomové práce jsou navrženy, realizovány a konstruovány nody a také nižší vrstva subsystému. Nižší vrstvou subsystému je myšlen jeho samotný hardware a také řídící aplikace. Zbylá část subsystému a server jsou navrženy a realizovány kolegou diplomantem Martinem Vanclem. Realizuje je v programovacím jazyce Java [**10**] s využitím Spring Framework [**11**], MySQL databáze [**12**] a REST api [**13**] pro komunikaci s mobilními platformami a externími programy.

[1] Z anglického slova node, které značí bod v síti.

4. LOGICKÝ NÁVRH NODŮ

V předchozí kapitole bylo řečeno, že nody jsou nejnižší logická vrstva navrhnutého systému. Nod je hardwarové zařízení komunikující po sběrnici se subsystémem a má k sobě přímo připojené periférie. Je také vybaveno softwarovými prostředky pro komunikaci s vyčítání hodnot z periférií. Těmi mohou být například různé senzory, tlačítka, spínače, světla apod. Tato kapitola popisuje vlastnosti nodů a definuje jejich chování. Dále bude detailně popsána vnitřní struktura nodů.

4.1. Vlastnosti a parametry nodů

Každý nod musí mít stanoveny následující vlastnosti a parametry:

- Typ,
- počet kanálů
- adresu.

Typem je myšleno, jaké periférie (senzory, ovládací prvky nebo aktuátory) se k nodu připojují, jaká data vracejí. Typ nodu má také stanoveny parametry a vlastnosti:

- Okamžitý nebo zpožděný přístup k datům či stavům[2],
- unikátní identifikátor,
- čtení nebo zapisování dat z periférií,
- počet dat,
- šířka dat v bitech.

Typ nodu tedy může být například spínač. Ze spínače je možné stav pouze číst, nikoliv zapisovat. Spínač může mít pouze dva stavy, sepnutý nebo rozepnutý. Dva stavy lze reprezentovat pouze jedním bitem. Šířku dat má tedy jedna. Počet dat je také roven jedné, protože žádné další informace spínač nemůže poskytnout. Stav tlačítka má nod k dispozici okamžitě a může jej ihned poskytnout subsystému.

Dalším příkladem typu nodu může být barevná žárovka. Z žárovky nod nemůže žádná data nebo stav získat, pouze nastavit jakou má mít barvu. Moderní led žárovky mají v sobě tři samostatné LED diody. Červenou, zelenou a modrou. Typ žárovka tedy bude mít

[2] Podle podstaty připojené periférie je vhodné mluvit buď o datech, nebo o stavech. Z teploměru se vyčítají data, ale ze spínače je vhodnější mluvit o vyčítání stavu.

počet dat 3 o šířce 8 bitů (hodnoty 0 – 255). Hodnota udává intenzitu svitu pro každou barevnou složku.

Oba výše zmíněné příklady jsou okamžité typy nodů. To znamená že nod může neustále vyčítat stav, nebo jej nastavovat. U některých typů dokonce nod musí stav vyčítat neustále. Příkladem budiž typ s tlačítky. Pokud by se stav tlačítek vyčítal pouze jednou za sekundu a uživatel mezitím vyčítáním tlačítko krátce stiskl, nebyl by tento stav zaznamenán. Proto musí nody stav vyčítat, co nejčastěji mohou.

Druhou skupinou jsou nody zpožděné. Jedná se o nody obsluhující takové periférie, které potřebují určitý časový rámec k naměření a zpracování hodnot. Jsou to tedy nody, ze kterých lze stav pouze vyčítat. Nejčastěji se jedná o digitální environmentální senzory (teploměry, vlhkoměry, senzory kvality ovzduší apod.) které komunikují po sběrnicích. Pokud by se nod neustále snažil vyčítat data, periférie by vracela buď žádné, nebo nesmyslné hodnoty. Proto zpožděný nod čeká na povel k naměření hodnot od subsystému. Subsystém poté čeká (resp. provádí jiné činnosti) určitý čas a po jeho uplynutí si naměřená data nebo hodnoty z nodu přečte.

Typy nodů je tedy možné rozdělit do čtyř skupin dle přístupu k datům nebo stavům a jejich vyčítáním nebo zapisováním:

- Okamžité, pouze pro čtení
- okamžité, pro čtení i zápis.
- okamžité, pouze pro zápis,
- zpoždění, pouze pro čtení.

V tabulce 1 je uveden přehled běžných typů nodů, které se mohou v chytrém domě vyskytnout. Nejedná se o kompletní seznam, protože díky navržené architektuře systému je možné na centrálním serveru definovat nové typy nodů a systém tak rozšířit o nové nody a další funkcionalitu. ID značí unikátní identifikátor typu nodu, O/Z značí, zdali je nod okamžitého nebo zpožděného typu.

ID	Název	O/Z	Šířka dat	Počet dat	Čtení / zápis
1	Tlačítko	O	1	1	R
2	Spínač	O	1	1	R
3	Světlo	O	1	1	RW
4	Stmívané světlo	O	8	1	RW
5	Stmívané RGB světlo	O	8	3	RW
6	Teploměr	Z	8	2	R
7	Vlhkoměr	Z	8	1	R
8	Pohybový senzor	O	1	1	R
9	Alarm	O	1	1	R
10	Stmívač	O	8	1	R
11	Ventilátor	O	1	1	RW
12	Regulovaný ventilátor	O	16	1	RW
13	Termostatická hlavice	O	1	1	RW
14	Dveřní zámek	O	1	1	R

Tabulka 1 - Přehled běžných typů nodů [autor]

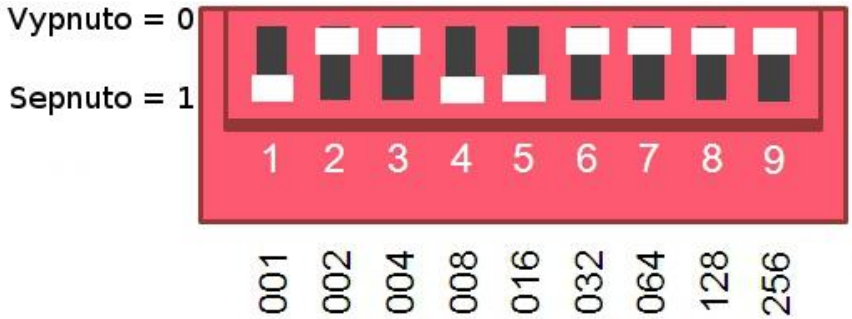
Další vlastností nodu je počet kanálů. Typ nodu pouze specifikuje, jaká periférie je k nodu připojena, respektive jakou periférii nod obsahuje. Pokud by nod ale obsahoval například jenom jedno tlačítko, nebo jedno světlo, bylo by v domě potřeba ohromné množství nodů. Proto může nod obsahovat více periférií, ovšem vždy stejného typu. Počet periférií je pak počtem kanálů, který nod obsahuje. Například osmikanálový spínačový nod, nebo dvoukanálový teploměr.

Poslední vlastností nodu je adresa. Aby mohl subsystém pracovat s konkrétním nodem, musí subsystém znát jednoznačný identifikátor tohoto nodu. Tím je jeho adresa. Existující řešení chytrého domu také umožnují mít v domě více modulu stejného typu a adresaci řeší dvěma odlišnými přístupy.

Modul má unikátní adresu z výroby a řídící jednotka buď obsahuje potřebnou funkcionalitu pro zjištění této adresy, nebo je adresa vytištěna na modulu a technik ji musí do řídící jednotky manuálně naprogramovat. Unikátní adresa od výrobce ale také znamená, že při výrobě modulu se adresa musí nahrát, což znamená, že výrobní linka musí každý modul naprogramovat jiným programem. Toto řešení je nejvíce spolehlivé, ale také z hlediska nákladů na výrobu nejdražší.

Druhý způsob spočívá v přítomnosti několika spínačů, které jsou vyvedeny buď vně modulu anebo skryty uvnitř[3]. Spínače reprezentuje číslo ve dvojkové soustavě. Pokud tedy má modul n spínačů a, které reprezentují buď, 0 nebo 1, lze adresu spočítat dle vzorce:

$$adr = a_0 * 2^0 + a_1 * 2^1 + \cdots + a_{n-2} * 2^{n-2} + a_{n-1} * 2^{n-1}.$$



Obrázek 2 - Manuální adresace modulů [autor]

Na obrázku 2 je demonstrováno nastavení modulu s devíti bitovou adresou a desátým přepínačem, který může měnit funkcionalitu modulu. V tomto konkrétním případě je adresa spočtena takto:

$$adr = 1 * 2^0 + 0 * 2^1 + 0 * 2^2 + 1 * 2^3 + 1 * 2^4 + 0 * 2^5 + 0 * 2^6 + 0 * 2^7 + 0 * 2^8,$$

$$adr = 1 + 0 + 0 + 8 + 16 + 0 + 0 + 0 + 0,$$

$$adr = 25.$$

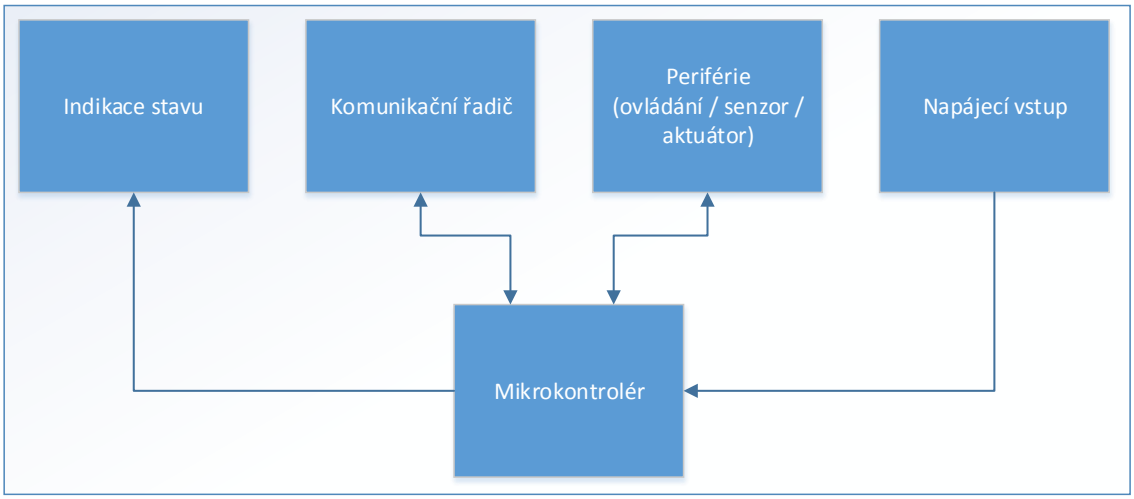
Adresa modulu tedy je 25. Tento způsob adresace má výhodu v ceně výroby modulu. Každý je osazen spínači a až při instalaci v domě je zvolena vhodná adresa. Oba dva přístupy ale mají nevýhody. Unikátní adresa znamená vysoké náklady při výrobě a spínače vyžadují buď odbornou montáž, nebo uživatelovu znalost dvojkové soustavy. Navíc v situaci, kdy se omylem dva moduly nastaví na stejnou adresu, může být chování domu chybové, v nejhorším případě lze očekávat i problémy s řídící jednotkou.

Z těchto důvodů byl navržen nový, třetí přístup. Nový nod nemá unikátní adresu ani přepínače k jeho nastavení. Místo toho se využije vnitřní logika nodu a subsystému pro dynamické přidělení adresy při prvním zapojením nodu do systému chytrého domu.

[3] Pro nastavení adresy je tedy nutné modul nejprve rozmontovat.

4.2. Vnitřní struktura nodu

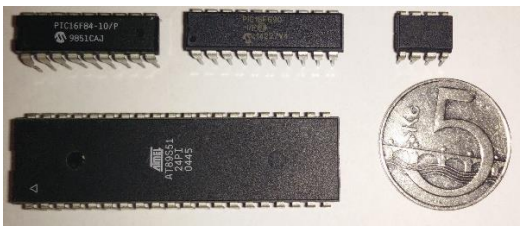
V předcházející podkapitole byly řešeny vlastnosti a parametry nodů. Nyní je třeba představit vnitřní strukturu nodu. Tím je myšleno uspořádání fyzických komponent nodu.



Obrázek 3 - Vnitřní struktura nodu [autor]

Jak je vidět z obrázku 3, centrálním bodem nodů je mikrokontrolér [**14**] neboli jednočipový počítač. Jedná se o většinou o monolitický integrovaný obvod obsahující kompletní mikropočítač (tedy mikroprocesor, operační paměť, programovou paměť a vstupně výstupní piny). V mikrokontroléru je uložen program zodpovědný za vyčítání stavů z periférií, kterými mohou být buď ovládací prvky, senzory nebo aktuátory. Ke každému nodu je připojena periférie dle jeho definovaného typu.

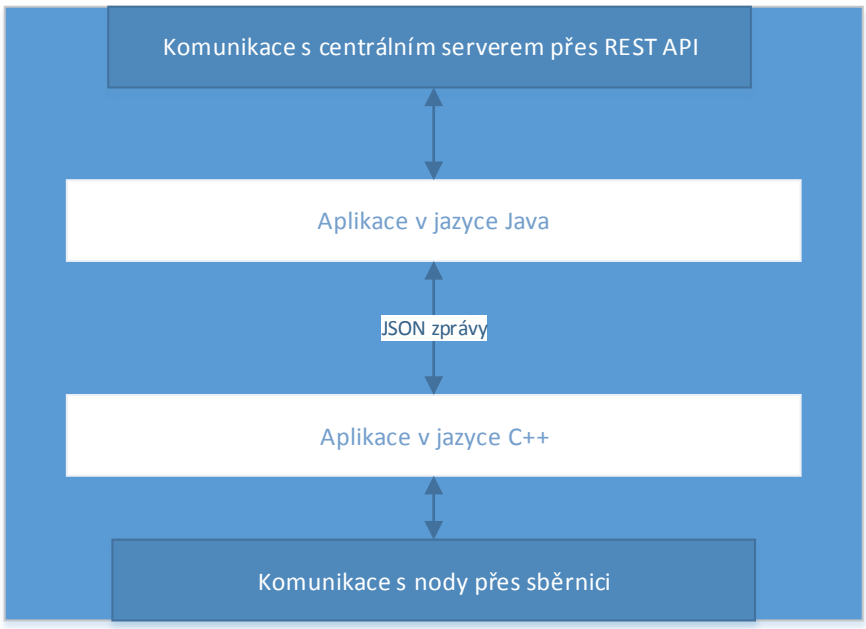
Program v mikrokontroléru dále využívá jeho hardwarové prostředky pro komunikaci se subsystémem skrze komunikační řadič. Komunikační řadič může být buď další samostatná elektronická součástka, nebo může být dle zvolené komunikační technologie a mikrokontroléru jeho součástí. Mikrokontrolér také indikuje stav nodu. Tím může být například, zdali má adresu, případně jestli je periférie funkční apod. Aby mikrokontrolér mohl pracovat, potřebuje mít přivedeno elektrické napětí skrze napájecí vstup. Na obrázku 4 jsou ukázány různé modely mikrokontrolérů.



Obrázek 4 - Ukázka různých mikrokontrolérů [autor]

5. LOGICKÝ NÁVRH SUBSYSTÉMU

Subsystém je prostřední vrstva navržené architektury systému. Bude realizován pomocí vhodně zvoleného počítače. Právě přítomností subsystému se navržená architektura liší od již existujících řešení. Hlavním cílem subsystému je vyčítat a zapisovat stavy a data nodů. Druhým cílem subsystému je zvýšení spolehlivosti a bezpečnosti chytrého domu. Při výpadku nebo poruše centrálního serveru přebírá subsystém v omezené míře jeho funkcionalitu.



Obrázek 5 - Logická struktura subsystému [autor]

Na obrázku 5 je znázorněna logická struktura subsystému. Protože každý programovací jazyk je vhodný pro odlišné typy aplikací, skládá se subsystém ze dvou odlišných aplikací. Ty mezi sebou komunikují pomocí zpráv v populárním formátu JSON [**15**].

Aplikace v jazyce Java není v této práci implementována, ani navrhována. Vytváří ji kolega diplomant Martin Vancl. Její zdrojový kód a funkcionalita bude do značné míry kopírovat funkcionalitu aplikace, kterou vyvíjí pro centrální server. Pokud dojde k výpadku serveru, bude aplikace v subsystému provádět obsluhu nodů dle uživatelsky definovaných pravidel. Těmi může být například, že stisk určitého tlačítka vyvolá obrácení stavu světel, otočení regulátoru otočí motorem apod. Pravidla na centrálním serveru mohou pracovat s nody z různých subsystémů. Pokud nebude centrální server dostupný, aplikace má uloženy pravidla pro nody k němu připojené. Nebudou tedy fungovat všechny pravidla z centrálního

serveru, ale pravidla pro lokálně připojené nody ano. Tato aplikace komunikuje s centrálním serverem přes populární REST API, po kterém stejně jako při komunikaci z aplikací v jazyce C++ [**16**] využívá zprávy ve formátu JSON.

Druhou částí je aplikace v programovacím jazyce C++, které je zodpovědná za fyzickou obsluhu nodů. V podstatě bude měřit a nastavovat stavy nebo data na základě požadavků aplikace v jazyce Java (dále již odkazovanou jako vyšší vrstva). Vyšší vrstva buď pošle nový stav nebo data nodu pro zápis či pro zápis a čtení. Nebo při změně stavu či dat nod pošle tato aplikace nový stav či data vyšší vrstvě. Nejdůležitějším požadavkem na výslednou aplikaci je její rychlost.

Programovací jazyk C++ byl zvolen proto, že zjednodušeně řečeno rozšiřuje jazyk C [**17**] o objektově orientované programování. Je tak zachována výhoda jazyku C, což je nízkoúrovňovost. Jedinou ještě nižší alternativou je psaní ve strojovém jazyce (tedy assembleru). V jazyce C jsou psány i různé operační systémy. Je tedy vhodný pro psaní aplikací, které využívají různé hardwarové prostředky, v tomto konkrétním případě se jedná o komunikaci s nody. Je to jazyk kompilovaný přímo do strojového kódu daného hardwaru. I v jazyce C je možné psát objektově, ale tento jazyk pro objektové programování nebyl navrhován a proto je výsledný kód sice objektový ale nikoliv jednoduše čitelný a upravovatelný. Jazyk C++ pak rozšiřuje vlastnosti jazyka C o nativní podporu objektů a o moderní programovací komponenty jako vektory, mapy, listy nebo například textové řetězce. Tato aplikace již bude nadále nazývána pouze jako aplikace, nikoliv aplikace z jazyce C++. Třídy jsou v C++ rozděleny na hlavičkový soubor s příponou .h, který obsahuje pouze definice metod a ostatních objektů. Třídy jsou implementovány s souborech se stejným názvem jako hlavičkové soubory, ale s příponou .cpp.

Subsystém je tedy fyzická komponenta inteligentního domu běžící na počítači. Může se zdát jako naprosto zbytečná, protože její přínos spočívá pouze při poruše nebo výpadku centrálního serveru. Výrobci současně prodávaných řešení většinou možnost fatální poruchy svých výrobků nepřipouštějí. Realita je ale odlišná, protože vzhledem ke komplikovanosti dnešních součástek je porucha velice pravděpodobná. Pokud nastane porucha subsystému, centrální server bude notifikovat uživatele. Opět ale zůstane zachována funkcionalita ostatních subsystémů i centrálního serveru. Subsystém i centrální server budou používat operační systém GNU/Linux [**18**].

6. VÝBĚR VHODNÝCH HW PLATFOREM

Předchozí části této práce se věnovaly logickému návrhu systému. Tato kapitola se zabývá výběrem a také implementací reálného hardwaru. Nejdříve je vybrána vhodná technologie pro komunikaci mezi nody a subsystémem. Poté jsou vybrány vhodné mikrokontroléry pro nody a jejich podpůrné komponenty. Následně je vybrán minipočítač na platformě ARM pro subsystémy. V poslední části kapitoly jsou navrženy a popsány schématická zapojení pro vybrané typy nodů.

6.1. Výběr komunikačního média a techonologie

Subsystém musí komunikovat s nody a k tomu je nutné stanovit, po jakém médiu bude komunikace probíhat. Po výběru média je třeba vhodně zvolit konkrétní technologii pro komunikaci. Nejprve je nutné určit minimální rychlost komunikace. Protože nody budou odesílat data v řádech jednotek bajtů, není nutné požadovat rychlosti v jednotkách megabitů za sekundu. Například osmikanálový nod s tlačítky odesílá nejčastěji jeden bajt se svým stavem a svojí adresou. Dále bude odesílat několik dalších bajtů v závislosti na použitém komunikačním protokolu (identifikátor, kontrolní součet apod.). Pokud bude rychlost komunikace 100 kbit/s, tedy 12,5 kB/s, tak odeslání jednoho bajtu bude trvat 80 mikrosekund. Teoreticky je tedy možné u subsystému s deseti nody, které přijímají 5 bajtů a odesílají 10 bajtů provést jejich obsluhu až osmdesátkrát za sekundu. Při takové frekvenci vyčítání lze spolehlivě zachytit i velice krátký stisk tlačítka. Rychlost 100 kbit/s lze považovat za dostatečnou.

Komunikace může probíhat buď po metalickém, optickém, nebo bezdrátovém médiu. Zdaleka nejvyšší rychlosti lze dosáhnou s optickým médiem. Optický kabel je ale velice nákladný a také křehký a špatně ohebný. V dnešní době jsou velice populární bezdrátové technologie Wi-Fi, Bluetooth a WiMaxV. Problém těchto technologií spočívá hlavně v bezpečnosti. Pokud by nody využívali Wi-Fi rozhrání, bylo by možné sledovat na běžném počítači veškerou komunikaci mezi nody a subsystémem. Pokud by byly nody využity i pro odemykání a zamykání domu, představovala by Wi-Fi velkou zranitelnost. To samé platí i pro Bluetooth a WiMax, kde lze komunikaci také jednoduše odchytávat. Pro bezdrátovou komunikaci je tedy nejvhodnější navrhnout nové bezdrátové přijímače a vysílače ideálně na takové frekvenci, kterou žádná výše zmíněná bezdrátová technologie

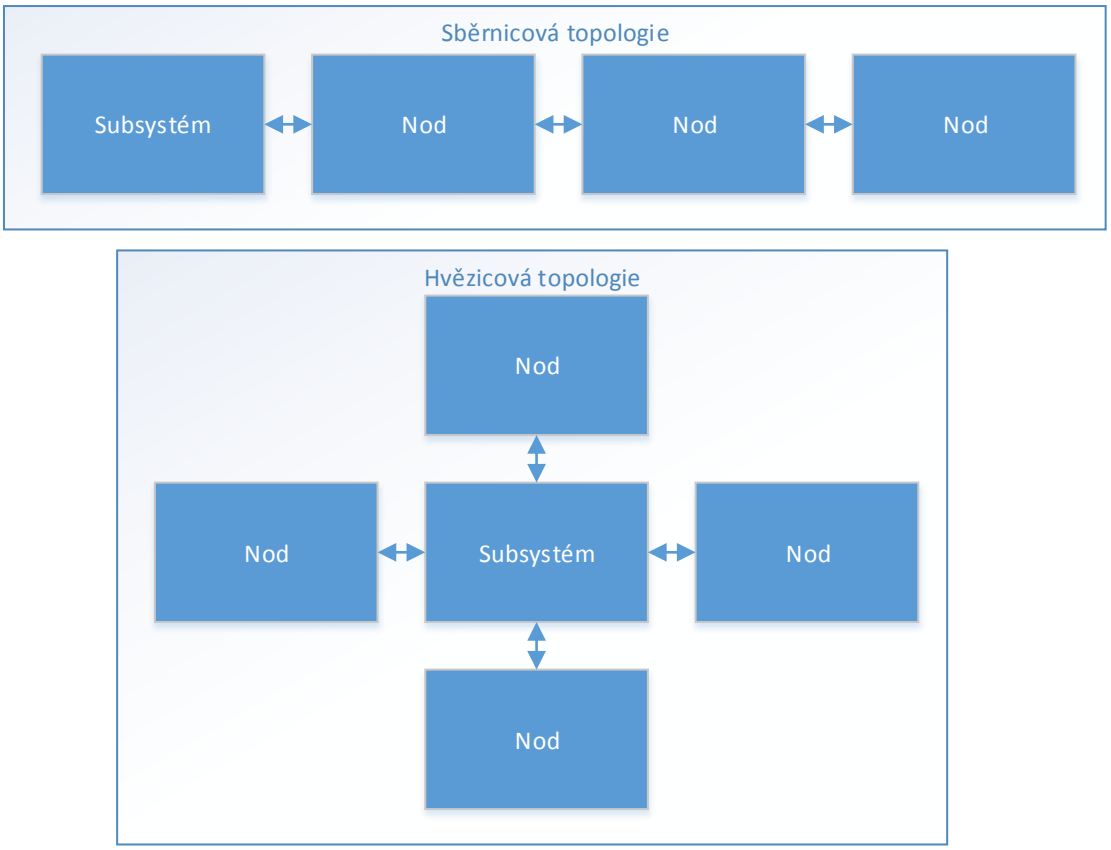
nevyužívá. U Wi-Fi se jedná o frekvence v nelicencovaném[4] pásmu 2,4 GHz a 5GHz. Bluetooth a WiMax také využívají pásmo 2,4 GHz. Pro bezdrátovou komunikaci v chytrém domě je tedy nejlepším řešením navrhnout vlastní přijímače a vysílače aby nebylo možné komunikaci jednoduše odposlouchávat. Vývoj takové technologie je ale velmi nákladný a zdlouhavý proces, při kterém je nutné dodržet velké množství mezinárodních a státních norem, předpisů a vyhlášek.

Z hlediska časové a cenové náročnosti implementace je tedy vhodné zvolit rozhrání, které komunikuje po metalickém médiu (nejčastěji měděný vodič). Toto řešení je časově i nákladové nejúspornější. Metalické médium je také nebezpečnější a nejodolnější. Jediný způsob jak odposlouchávat komunikaci po kabelu, je připojení vodičů k logickému analyzátoru dat. Lze předpokládat, že veškerá kabeláž a nody budou v domě pevně zabudovány ve zdi a pouze tradiční ovládací prvky budou vně. Tím odpadá nutnost řešit šifrování komunikace mezi nody a subsystémem.

Při výběru vhodné metalické technologie je nutné brát v potaz následující parametry:

- Fyzickou topologii,
- počet vodičů,
- cenu integrace.

[4] Nelicencované pásmo znamená, že je možné na frekvencích toto pásma bezplatně využívat homologované zařízení pro komunikaci.



Obrázek 6 - Rozdíl mezi sběrnicovou a hvězdicovou topologií [autor]

Fyzickou topologií je myšleno, jakým způsobem jsou jednotlivá zařízení mezi sebou propojena. Dvě nejčastější používané topologie, sběrnicová a hvězdicová, jsou znázorněny na obrázku 6. U sběrnicové topologie je veden kabel ze subsystému k prvnímu nodu. Z něj pak vede kabel do dalšího a tak dále. Jedná se tedy o sériové zapojení. Výhodou této topologie je výrazná úspora kabeláže. K subsystému jsou přivedeny vždy jen tři kabely. První vede k nodům, druhý k centrálnímu serveru a třetí je kabel k napájecímu adaptéru. Nevýhoda této topologie spočívá v situaci, kdy je kabel omylem přerušen nebo poškozen. Další nevýhodou je, že pokud subsystém pošle zprávu konkrétnímu nodu, zachytí jej i všechny ostatní na sběrnici. Stejně tak zachytí ostatní nody odpověď směrem k subsystému. Tento problém řeší adresa nodu. Pokud nod zachytí zprávu, která není právě jemu určená, bude ji ignorovat.

U hvězdicové topologie jsou všechny nody umístěny přímo k subsystému. Výhodou tohoto přístupu je zachování funkčnosti všech ostatních nodů, pokud je kabel k jednomu nodu omylem přerušen nebo poškozen. Nevýhodou je velký nárůst kabeláže. Ke každému subsystému povede tolik kabelů, kolik bude připojených nodů plus také kabel k subsystému a napájecí kabel.

Obě topologie mají své výhody a nevýhody. Nelze jednoznačně prohlásit, která z nich je lepší a proto bude vybrána vhodná technologie bez ohledu na to, jakou topologii využívá.

Dalším parametrem je počet vodičů, neboli počet jednotlivých drátů (žil) v kabelu. Toto kritérium ovlivňuje hlavně konektor, který je možné použít. Čím je větší počet vodičů, tím větší, složitější a zároveň dražší konektor je třeba použít. Cenou integrace je myšleno jak nákladné je danou technologii reálně implementovat do nodů a subsystému.

Pro metalické médium se v současnosti nejvíce využívají tři technologie. Ethernet, CAN bus a RS-485.

6.1.1. Ethernet

Ethernet [**19**] je souhrnný název pro technologii nejčastěji používanou pro komunikaci v LAN a WAN sítích. Realizuje fyzickou a linkovou vrstvu sedmivrstvého OSI modelu. Linková vrstva zajišťuje předávání dat mezi zařízeními a sama komunikaci zabezpečuje proti chybám v komunikaci. Ethernet může dosahovat rychlosti až několika gigabitů za sekundu. Ethernet, pokud pomineme zastaralý standart, kde byla zařízení zapojená do sběrnice po koaxiálním kabelu, využívá hvězdicovou topologii. Z každého nodu by musel být vedený jeden kabel obsahující čtyři páry kroucené dvoulinky do síťového přepínače. Pokud by tedy jeden subsystém obsahoval například 30 nodů, tak by do přepínače vedlo 30 kabelů a jeden do subsystému. Přepínače s takovým počtem portů jsou velmi drahé. Navíc kabeláž v takovém množství je velmi nákladná a obnáší obtížnou instalaci v domě.

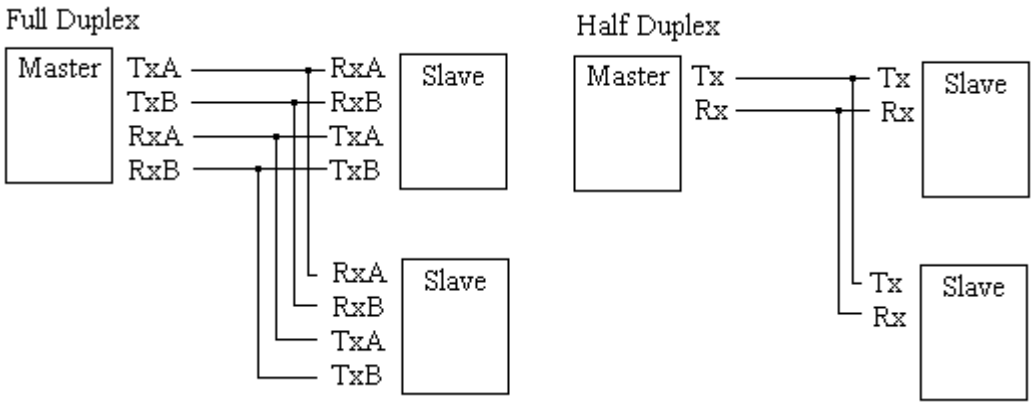
6.1.2. CAN bus

CAN [**20**] (Controller area network) sběrnice je navržená původně pro komunikaci součástí v automobilu. Stejně jako ethernet realizuje jak fyzickou tak i linkovou vrstvu. Zařízení jsou připojena dvěma vodiči sériově a komunikují rychlostí až 125kbit/s na vzdálenost 500 metrů. Parametry sběrnice CAN jsou vhodné pro použití, ale cena jednoho řadiče několikanásobně převyšuje cenu všech ostatních komponent v nodu.

6.1.3. RS-485

RS-485 [**21**] není na rozdíl od výše zmíněných standardů technologie, ale pouze norma definující napěťové úrovně na vodičích a specifikující způsob komunikace mezi zařízeními, tedy fyzickou vrstvu. Používá se pro komunikaci typu Master/Slave, kdy jedno zařízení Master řídí komunikaci a více zařízení typu Slave odpovídají až po výzvě. Je možné

využít zapojení s jedním párem krouceného vodiče (nazývané A a B) a komunikovat v Half-Duplex módu. Pak jsou Slave zařízení trvale v režimu příjmu a až po výzvě se přepínají do režimu vysílání. Ihned poté přechází zpět do režimu přijímaní. Master je naopak v režimu vysílání a po odeslání výzvy některému Slave zařízení se přepíná do režimu příjmu a čeká na odpověď. Druhá varianta je se dvěma páry kroucených vodičů (nazývaných TxA, TxB, RxA a RxB) komunikujících ve Full-Duplex režimu (viz obrázek 7). Na prvním páru je Master trvale v režimu vysílání a na druhém v režimu příjmu. Na prvním páru přijímají všechny Slave zařízení a čekají na výzvu. Když je některý vyzván, odpoví po druhém páru. Výhodou Full-Duplex módu je tedy odpadnutí nutnosti přepínat režimy a s tím spojené režie. Nevýhodou je vyšší počet vodičů a cena převodníků. Protože přepnutí režimů trvá pouze několik nanosekund, je vhodnější použít Half-Duplex mód.



Obrázek 7 - Rozdíl mezi Half a Full duplex komunikací [21]

RS-485 dokáže spolehlivě přenášet data o rychlosti až 20 Mbps na vzdálenost 10 metrů. Při snížení přenosové rychlosti na 100 Kbps vzroste vzdálenost na 1200 metrů. Tato přenosová rychlost je pro komunikaci nodu dostatečná a snižuje riziko chyb v komunikaci. Vzdálenost 1200 metrů dovoluje připojit i nody velmi vzdálené od subsystému. Ceny řadičů RS-485 jsou velice přijatelné a proto je tento standart nejvhodnější pro použití v navrženém systému.

Protože RS-485 nespecifikuje linkovou vrstvu a komunikační protokol, musí být využit buď existující protokol třetích stran, nebo navrhnut nový protokol speciálně pro komunikaci nodů se subsystémem.

6.2. Výběr vhodných mikrokontrolérů

Již bylo řečeno, že mikrokontrolér je centrálním prvkem každého nody. Proto je nutné zvolit vhodné modely mikrokontroléru. Problémem je, že mikrokontroléry vyrábí velké množství výrobců[5] v tisících různých variantách. Mikrokontroléry lze dělit dle desítek možných kritérií. Pro zjednodušení je lze rozdělit dle architektury mikroprocesoru (výpočetní jednotky mikrokontroléru) a šířky datové sběrnice.

Architektura mikroprocesoru může být buď RISC [**22**] (anglicky Reduced Instruction Set Computing) nebo CISC (anglicky Complex Instruction Set Computing). CISC znamená, že mikrokontrolér dokáže vykonávat velké množství různých instrukcí. Především obsahují velkou škálu matematických a aritmetických instrukcí, také logické operace. Široká instrukční data usnadňuje vývoj programů, ovšem v dnešní době se ve strojovém jazyce programuje již minimálně a využívají se vyšší programovací jazyky jako C, C++ nebo například Java. CISC architektura znamená také složitější vnitřní zapojení a vyšší riziko konstrukčních vad. Protože instrukce se u CISC architektury velmi rozmanitá, mohou se některé provádět déle než jiné instrukce. Typickým představitelem CISC architektury jsou procesory x86 vyráběné společnostmi Intel a AMD. RISC architektura naopak obsahuje pouze nezbytně nutnou množinu instrukcí. Například neobsahují instrukce pro násobení. Ta se dá jednoduše vytvořit sledem instrukcí sčítání a bitového posunu. Paradoxně může být takový sled instrukcí rychlejší než specializovaná instrukce na architektuře CISC. Většina mikrokontrolérů je postavena z konstrukčních důvodů na architektuře RISC.

Šířka datové sběrnice u mikrokontrolérů může být 8 bitů, 16 bitů nebo 32 bitů (u mikroprocesorů ještě 64 bitů). Udává, jak velké číslo může mikrokontrolér v rámci jednoho paměťového bloku paměti uložit a poté s ním v jedné instrukci pracovat. Dle šířky datové sběrnice se mluví o 8bitovém, 16bitovém nebo o 32bitovém mikrokontroléru. 8bitový mikrokontrolér může tedy v jedné instrukci pracovat s čísly od 0 do 255, resp. od -127 až 128 při použití znaménkového datového typu. 16bitový s čísly 0 – 65535, resp. -32767 – 32768. 32bitový potom s čísly 0 – 2^{32} nebo $-2^{31} + 1$ až 2^{31}. Neznamená to ale, že 8bitový mikrokontrolér nemůže pracovat s většími čísly než 255. Větší číslo je možné uložit do více paměťových bloků a využít kombinaci více instrukcí pro jeho manipulaci. Šířka datové sběrnice také nesouvisí se šířkou adresní sběrnice. 8bitový mikrokontrolér může mít

[5] Mezi největší výrobce mikrokontrolérů patří například společnosti Texas Instruments, Microchip, Analog Devices, Atmel Corporation, STMicroelectronics apod.

například 10bitovou šířku pro operační paměť a 12bitovou šířku pro programovou paměť. Je tedy možné v operační paměti alokovat až 2^{10}bloků pamětí, tedy 1024 osmibitových bloků a mít program, který čítá 2^{12}, tedy 4096 programových slov[6].

V současné době se nejčastěji využívají 8bitové mikrokontroléry, pro svoji jednoduchou architekturu a strmou učící křivku a 32bitové mikrokontroléry postavené na architekturách společnosti ARM. Samotná společnost ARM nevyrábí mikrokontroléry [**23**] ani mikroprocesory, pouze definuje různé specifikace vnitřní architektury, které musí konkrétní výrobek splňovat. Ostatní společnosti si pak licencují práva na jejich výrobu. V případě mikrokontroléru se jedná nejčastěji o architektury Cortex-M0, Cortex-M3 a Cortex-M4. Tyto mikrokontroléry nabízejí velké množství funkcí, jako například operace v plovoucí desetinné čárce, ladění kódu, podporu USB přímo na mikrokontroléru apod. Obě dvě architektury mají svá pro a proti. 8bitové mikrokontroléry jsou dobré pro nízkoúrovňové použití a v některých případech jsou rychlejší než jejich technologicky nadřazenější konkurenti. 32bitové mikrokontroléry zase mají výhodu s širokém množství sběrnic, které můžou obsluhovat, velké množství jak operační tak i programové paměti apod.

Program nodů nebude příliš komplikovaný. Jeho účelem je uložit si systémem přidělenou adresu a poté vyčítat nebo nastavovat stav nebo data periférie k němu připojené. Tyto informace pak na žádost subsystému zasílá. Takový program dokáže bez problémů vykonávat mikrokontrolér 8bitové architektury. Ty, stejně jako mikrokontroléry ostatních architektur vyrábí velké množství výrobců, ale někteří z nich se soustředí nekomerční sektory nebo na velice úsporná řešení případně poskytují své produkty pouze pro své partnery. V komerčním sektoru jsou nejčastěji použity mikrokontroléry od společností ATMEL nebo Microchip. Obě dvě nabízejí velice obdobně vybavené mikrokontroléry a zvolení jednoho výrobce závisí spíše na osobních preferencích. Čipy od společnosti ATMEL mají často větší operační a programovou paměť, čipy společnosti Microchip mívají za stejnou cenu podporu více sběrnic, případně nižší spotřebu. Mikrokontroléry od ATMEL mají k dispozici levnější a univerzálnější programátory (zařízení určené k nahrání programu vytvořeném v počítači do mikrokontroléru), pro čipy od Microchip je k dispozici lepší vývojové prostředí. Takto je možné pokračovat dále. Pro implementaci nodů byly zvoleny

[6] Šířka programových slov se také může lišit od šířky dat. Některé mikrokontroléry mají 8bitovou šířku operační paměti, ale

mikrokontroléry Microchip. Ta vyrábí široké množství[7] 8bitových mikrokontrolérů [**24**] a některé vybrané modely jsou uvedeny v tabulce 2.

	10F200 [**25**]	**12F617** [**26**]	**16F690** [**27**]	**16F767** [**28**]	**18F86J50** [**29**]
Počet pinů	6	8	20	28	80
Paměť FLASH	375 B	3500 B	7 KB	14 KB	64 KB
Paměť RAM	16 B	128 B	256 B	368 B	3904 B
Paměť EEPROM	X	HEF	256	X	HEF
Frekvence interního oscilátoru	4 MHz	4, 8MHz	32 kHz, 8 MHz	8 MHz	32 kHz, 8 MHz
Počet analogových komparátorů	0	1	2	2	2
Počet AD převodníků	0	4	12	11	12
Rozlišení AD převodníků	0	10 bit	10 bit	10 bit	10 bit
Počet 8 bitových časovačů	1	2	2	2	2
Počet 16 bitových časovačů	0	1	1	1	3
Počet UART	0	0	1	1	2
Počet I2C sběrnic	0	0	1	1	2
Počet SPI sběrnic	0	0	1	1	2
Minimální provozní napětí	2 V	2 V	2 V	2 V	2 V
Maximální provozní napětí	5,5 V	5,5 V	5,5 V	5,5 V	3,6 V
Dostupná pouzdra	SOT-23 PDIP DFN	PDIP SOIC MSOP DFN	PDIP SOIC SSOP QFN	PDIP SOIC SSOP QFN	TQFP

Tabulka 2 - Výběr 8bitových mikrokontrolérů společnosti Microchip [autor]

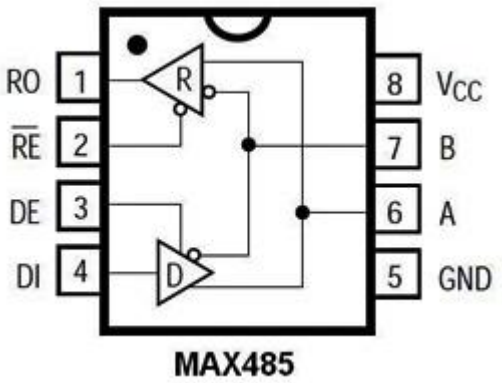
Výhodou takové produktové rozmanitosti je, že pro každý typ nodu je možné vybrat vhodný mikrokontrolér. Navíc je cenová politika společnosti Microchip pozitivně nakloněna i malonákladové výrobě. Tím lze uspořit náklady i při vývoji prototypů.

[7] K datu 18.4 2015 nabízí společnost Microchip 367 8bitových mikrokontrolérů a 20 nových produktů připravuje.

6.2.1. Komunikace se subsystémem

Vybrané mikrokontroléry musí vždy obsahovat alespoň jedno rozhraní UART (anglicky Universal Asynchronous Receiver/Transmiter). Jedná se o univerzální komunikační rozhraní [**30**], které umožňuje vysílat a přijímat data vybranými rychlostmi. Z níže uvedené tabulky je tedy možné použít pouze modely PIC16F690, PIC16F767 nebo 18F86J50.

Protože žádný z mikrokontroléru neobsahuje zabudovaný řadič pro RS-485, bude k mikrokontroléru připojen převodník společnosti Maxim, MAX485 [**31**], který signály převádí signály z rozhraní UART na napěťové úrovně RS-485. Obsahuje také dva piny RE a DE, kterými se řídí směr komunikace. Oba piny jsou spojeny, a pokud jsou v úrovní logické 0, tak se data přijímají a v úrovní logické 1 data vysílají. Pin GND je zem převodníku, pin VCC je vstupní napětí +5V, pin RO přijímá data ze sběrnice do mikrokontroléru, pin DI naopak vysílá data z mikrokontroléru na sběrnici. Pin A a B se pak připojují do sběrnice RS485. Zapojení převodníku MAX485 je ukázáno na obrázku 8.



Obrázek 8 - Převodník MAX485 [31**]**

6.2.2. Rychlost komunikace přes RS-485

Dále je nutné stanovit přesnou přenosovou rychlost [**32**] rozhraní UART. V předchozích částech práce byl definován požadavek na rychlost alespoň 100kbit/s. UART nemůže mít libovolnou rychlost, ale musí komunikovat jednou z následujících přenosových rychlostí.

110, 150, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800 nebo 921600 baud/s

Tyto rychlosti jsou v jednotkách baudů za sekundu. Baud je jednotka modulační rychlosti udávající počet změn stavu na přenosovém médiu. U RS-485 platí, že jeden baud je roven jednomu bitu. Ze seznamu se jeví jako ideální přenosová rychlost 115200 kbaud/s. Vyšší rychlost by snižovala maximální délku kabelu RS-485.

Většina mikrokontrolérů Microchip obsahuje interní oscilátor, který svým taktem udává rychlost zpracování instrukcí (tedy výpočetní výkon). 8bitové mikrokontroléry mají nejčastěji oscilátor o frekvenci 4 a 8MHz. Je také přes binární děličky využíván k získání frekvence taktu UART. Tato hodnota může mít odchylku od požadované rychlosti. Odchylky jsou uvedeny v tabulce pro různé přenosové rychlosti v závislosti na frekvencích oscilátoru. Pro přímou komunikaci dvou mikrokontrolérů je přijatelná odchylka do 2% [**33**]. Protože mikrokontroléry mají na UART připojen převodník na napěťové úrovně RS-485 a vzdálenost mezi subsystémem a nejvzdálenějším node může být až 1,2 km, nelze za přijatelnou odchylku brát jinou než nulovou procentuální odchylku.

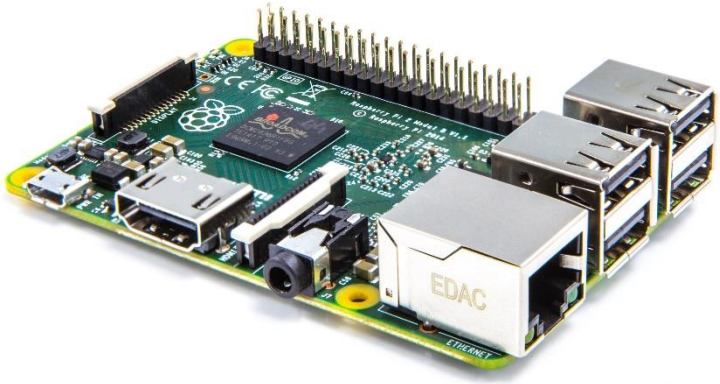
	Frekvence oscilátoru (MHz)						
Přenosová rychlost (kbps)	8	10	11.0592	12	16	18.432	20
300	0,0	0,0	0,0	0,0	0,0	0,0	0,0
600	0,0	0,0	0,0	0,0	0,0	0,0	0,0
1200	0,0	0,0	0,0	0,0	0,0	0,0	0,0
2400	0,2	0,2	0,0	0,2	0,0	0,0	0,0
4800	0,2	0,2	0,0	0,2	0,2	0,0	0,2
9600	0,2	0,2	0,0	0,2	0,2	0,0	0,2
14400	0,8	0,9	0,0	0,2	0,6	0,0	0,2
19200	0,2	1,4	0,0	0,2	0,2	0,0	0,2
38400	0,2	1,7	0,0	2,4	0,2	0,0	1,4
57600	3,7	1,4	0,0	0,2	2,1	0,0	1,4
115200	7,8	7,8	0,0	7,5	3,7	0,0	1,4
230400	7,8	10,6	0,0	7,8	7,8	0,0	7,8

Tabulka 3 - Závislost frekvence krystalu na chybě komunikace [34]

Z tabulky 3 je vidět že pro komunikaci bez odchylky je nutné nastavit interní oscilátor na frekvence buď 11,0592 MHz, nebo 18,432 MHz. Většina interních oscilátorů na tyto konkrétní frekvence nemůže být nastavena, a proto bude muset být k mikrokontroléru připojen externí krystal.

6.3. Výběr počítače pro subsystém

V kapitole Logický návrh bylo řečeno, že subsystém bude ke svému běhu využívat počítač na platformě ARM. Tyto počítače zatím nenabízejí výkon srovnatelný s klasickou architekturou x86 nebo AMD-64, ale jsou velice úsporné a kompaktní. Také většinou nabízejí vstupně výstupní piny (GPIO, anglicky General Purpose Input/Output) a hardwarová rozhraní vyvedené přímo z jejich mikroprocesoru. Díky tomu je možné k nim připojit různé periférie. U subsystému se bude jednat o převodník MAX485 připojený přes UART k počítači. Je také potřeba aby měl počítač alespoň jeden GPIO pin, který bude také připojený k převodníku MAX485 a řídit směr komunikace.



Obrázek 9 - Raspberry Pi 2. generace [35**]**

Tabulka 4 srovná parametry komerčně dostupných ARM počítačů. Všechny ke svému běhu využívají distribuce operačního systému GNU/Linux. V současné době je velmi populární první generace Raspberry Pi [**36**]. Jejím nedostatkem je ale přítomnost pouze jedno jádrového procesoru. Pokud by nad operačním systémem běžela pouze aplikace v jazyce C++ zodpovědná za obsluhování nodů, byl by výkon procesoru dostatečný. Dle logického návrhu ale v subsystému poběží také aplikace v jazyce Java a tím pádem i běhové prostředí Javy. Obě aplikace poběží souběžně a mohla by nastat situace, kdyby C++ aplikace nedokázala, z důvodu vytížení CPU obsluhovat nody dostatečně rychle. Proto byl za počítač vhodný pro provoz subsystému vybrán Raspberry PI druhé generace (viz obrázek Obrázek 9 - Raspberry Pi 2. generace9), který obsahuje čtyřjádrový procesor a operační paměť o velikost 1 GB. Obsahuje také potřebné UART rozhrání a dostatek GPIO pinů, takže je možné subsystém v budoucnosti rozšířit o další funkcionalitu. Přes sběrnici DSI je například možné připojit dotykový LCD panel k ovládání chytrého domu. Počítače Beagle Bone Black a Odroid U3 nebyly zvoleny, protože jsou oproti Raspberry Pi 2 téměř dvounásobně dražší.

Jako operační systém subsystému byla zvolena Linuxová distribuce Raspbian [**37**]. Jedná se o modifikaci distribuce Debian [**38**] přímo určenou pro Raspberry Pi.

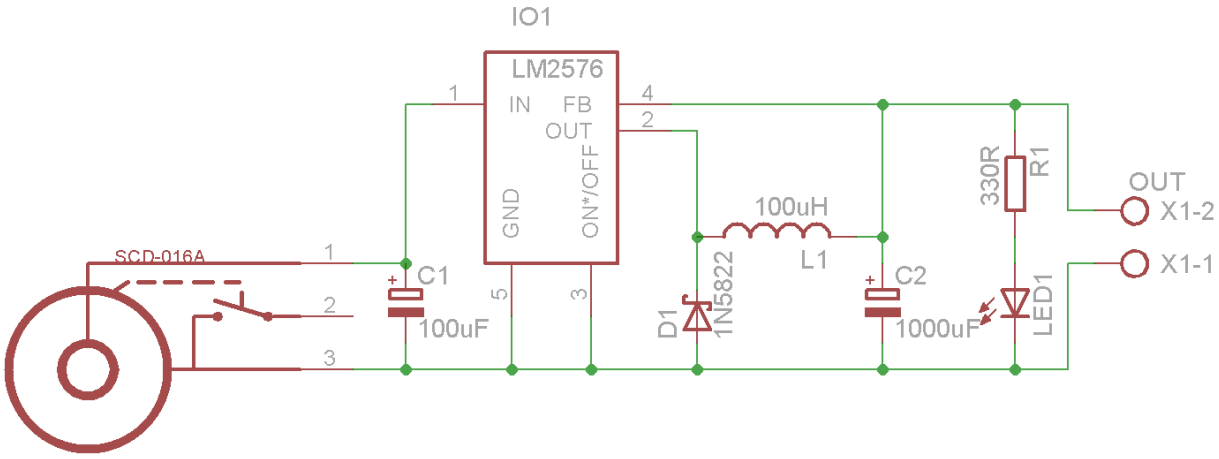
	Raspberry Pi [**36**]	**Raspberry PI 2** [**35**]	**Beagle Bone Black** [**39**]	**Odroid U3** [**40**]
Procesor	BCM2835	BCM2836	AM335x	Exynos 4412
Počet jader procesoru	1	4	1	4
Velikost operační paměti	256 / 512 MB	1 GB	512 MB	2 GB
Paměťové úložiště	SD karta	SD karta	4GB MMC	SD karta
Počet GPIO pinů	12	40	92	16
Sběrnice	UART SPI I2C	UART SPI I2C CSI DSI	UART SPI I2C	UART I2C
Rozměry	85 x 56 mm	85 x 56 mm	86 x 53 mm	83 x 48 mm

Tabulka 4 - Vybrané počítače architektury ARM [autor]

6.4. Implementace vybraných nodů

Protože jsou již vybrány vhodné řady mikrokontrolérů, je stanovená komunikační technologie, její přenosová rychlost i požadavky na taktovací frekvence, je možné se již začít věnovat implementacím [**41**] jednotlivých nodů. Jednotlivé elektrické obvody nodů byly navrhovány [**42**] tak, aby byly co nejvíce univerzální a pouhou změnou řídícího programu bylo možné upravit jejich funkcionalitu. Ve všech realizovaných nodech byl použit stejný mikrokontrolér, a to model PIC16F690 z důvodu snížení délky vývoje. Je nutné také zmínit, že se jedná pouze o prototypy, ne o produkční vzorky. Schémata jsou kresleny v bezplatné verzi programu EAGLE [**43**] [**44**].

6.4.1. Schéma a popis zapojení napájecího zdroje



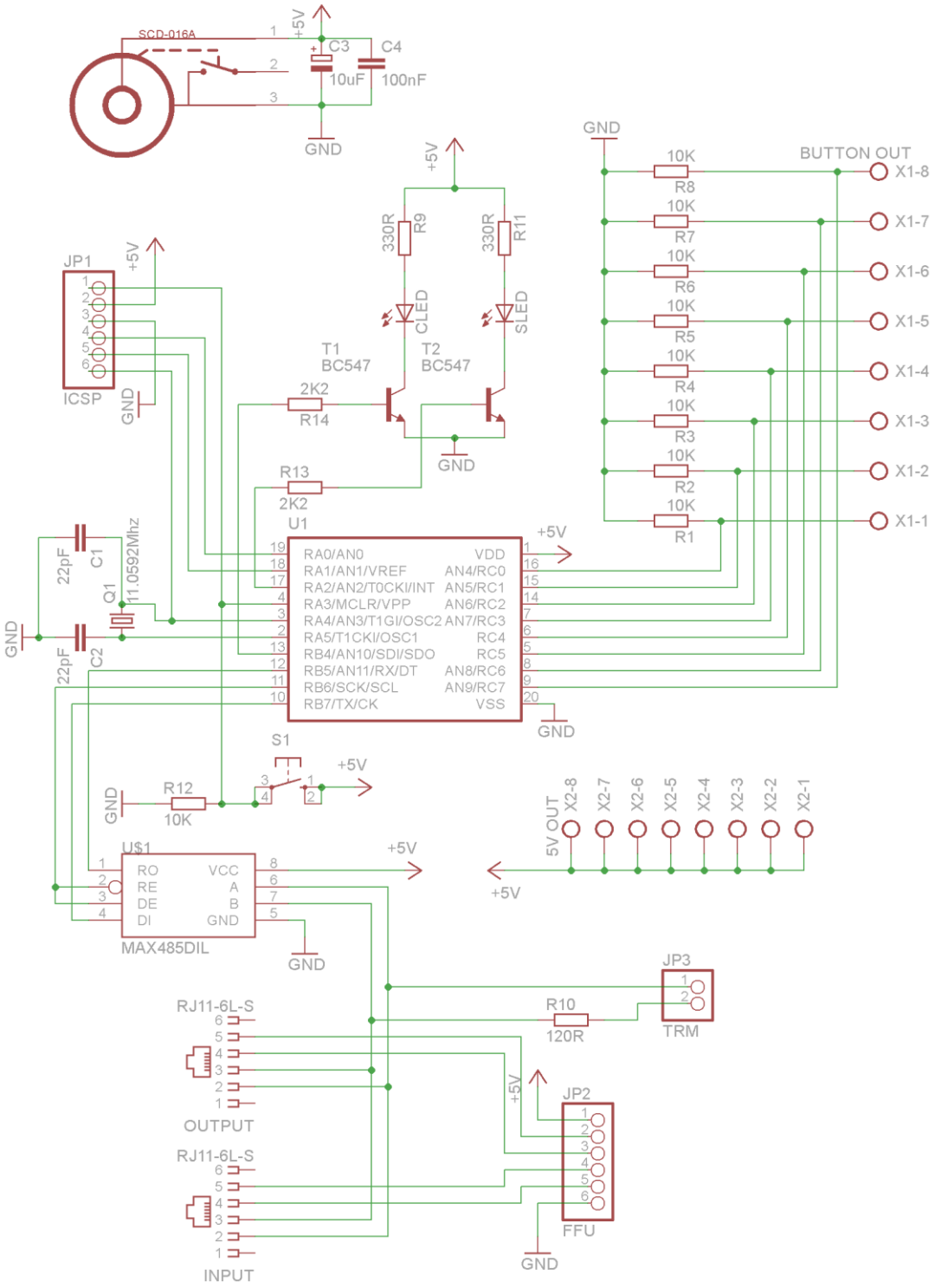
Obrázek 10 - Schéma zapojení napájecího zdroje [autor]

Prvním navrhnutým elektrickým obvodem (viz obrázek 10) není nod, ale napájecí zdroj pro nody. Protože PIC16F690 korektně pracuje při napětí od 2 do 5,5V, převodník MAX485 požaduje přesně napětí 5V, byl navržen zdroj, který je možné napájet běžně dostupnými síťovými adaptéry.

Využívá monolitický spínaný stabilizátor LM2576 [**45**], který má vysokou účinnost 88%. Dokáže stabilizovat vstupní napětí od 7 do 40V. Maximální výstupní proudová zatížitelnost je 3A. Ke zdroji je tedy možné připojit až 15W zátěž. Jeden napájecí zdroj tedy dokáže napájet desítky nodů zároveň. Ke korektnímu chodu vyžaduje několik pasivních součástek. Elektrolytický kondenzátor [**46**] C1 je použit pro nízkofrekvenční filtraci vstupního napětí. Cívka L1 je použita pro zpětnovazební smyčku na výstupu a schottkyho dioda [**47**] D1 jako protizkratová ochrana. Elektrolytický kondenzátor C2 je umístěn paralelně na výstupu a slouží k vyrovnávání poklesu napětí při velkém náhlém zvýšení proudové zátěže na výstupu. LED dioda LED1 je v sérii s rezistorem R1 a jsou paralelně připojeny na výstupu. LED dioda slouží k indikaci funkčnosti zdroje.

Jako vstupní konektor je použit souosý napájecí konektor SCD-016A s průměrem 2,1mm (samice). Běžně dostupné síťové adaptéry mají jako konektor protikus a lze je přímo použít bez nutnosti měnit jejich výstupní konektor. Výstup navrženého zdroje je vyveden na svorkovnici.

6.4.2. Schéma a popis zapojení vstupního nodu



Obrázek 11 - Schéma zapojení spínacího nodu [autor]

V kapitole Logický návrh nodů bylo řečeno, že každý nod musí mít jednoznačně stanovený svůj typ. Neznamená to ale, že není možné navrhnout takové schématické zapojení a následně plošný spoj, který by byl univerzální. Stačí přehrát program

mikrokontroléru a nod bude mít jiný typ. Toto schéma může obsluhovat jakoukoliv periférii resp. 8 periférií, které vrací dva stavy. Lze tedy použít například pro tlačítka, spínače, dveřní magnety, hladinové snímače apod. V každém případě se vždy bude jednat o osmikanálový nod s typem pouze pro čtení, šířkou dat 1 a počtem dat 1. Schéma zapojení je na obrázku 11.

Obvod předpokládá vstupní napětí 5V na konektoru SCD-016A. K němu jsou paralelně připojeny dva kondenzátory. Keramický kondenzátor C4 pro odrušení případných nežádoucích vysokých frekvencí ve vstupním napětí a elektrolytický kondenzátor C3 pro odrušení nízkých frekvencí. Vstupní napětí je přivedeno k mikrokontroléru PIC16F690, k převodníku MAX485 a do pinové lišty JP2.

Vstupy z periférií jsou připojeny přes pull-down rezistory R1 až R8 k zemi, a pokud nebude nějaký vstup připojen, mikrokontrolér bude trvale měřit logickou nulu na svém vstupu. Periférie jsou k mikrokontroléru připojeny na pinech RC0 až RC7. K mikrokontroléru je také připojeno přes pull-down rezistor tlačítko S1. To slouží k testovacím účelům při vývoji programu. Piny TX, RX a RB6 jsou připojeny k MAX485. Pin TX vysílá data do převodníku, po pinu RX mikrokontrolér přijímá data z převodníku a pin RB6 řídí směr komunikace. Aby mohl mikrokontrolér signalizovat svoje stavy, jsou na piny RB4 a RA2 připojeny LED diody CLED a SLED. Připojeny jsou přes NPN [**48**] tranzistory T1 a T2 aby nedošlo k překročení maximální proudové zatížitelnosti pinů. Maximální proud, který lze v součtu odebírat ze všech pinu mikrokontroléru 16F690 je 25mA. Běžné LED diody mají příkon 5mA. Pokud by byly připojeny přímo, mohlo by dojít k překročení této hranice a poškození mikrokontroléru. Protože se LED diody spínají přes bázi tranzistorů, které vyžadují pouze několik nA, je tento problém zcela eliminován. Program mikrokontroléru bude využívat CLED k indikaci problému v komunikaci a SLED pro indikaci stavu. Pokud nod nemá ještě přidělenou adresu, bude dioda blikat. Po adresaci bude svítit neustále. K pinům RA4 a RA5 je připojen krystal Q1 paralelně s uzemněnými keramickými kondenzátory C1 a C2.

Pro komunikaci se subsystémem a dalšími nody jsou vodiče z převodníku MAX485 přivedeny paralelně na RJ11 konektory INPUT a OUTPUT. Paralelně je také připojen rezistor R10, který je možné připojit přidáním propojky na pinovou lištu JP3. Jedná se o terminační rezistor a nod, který je na sběrnici poslední jej musí mít připojen, ostatní nody naopak odpojeny. Úlohou terminačního rezistoru je zastavit možný odraz signálu na vodiči.

Pinová lišta JP2 je zde přítomna pro budoucí využití. Bude sloužit k testování obvodu, který bude řešit připojení terminačního rezistoru na posledním nodu automaticky za uživatele.

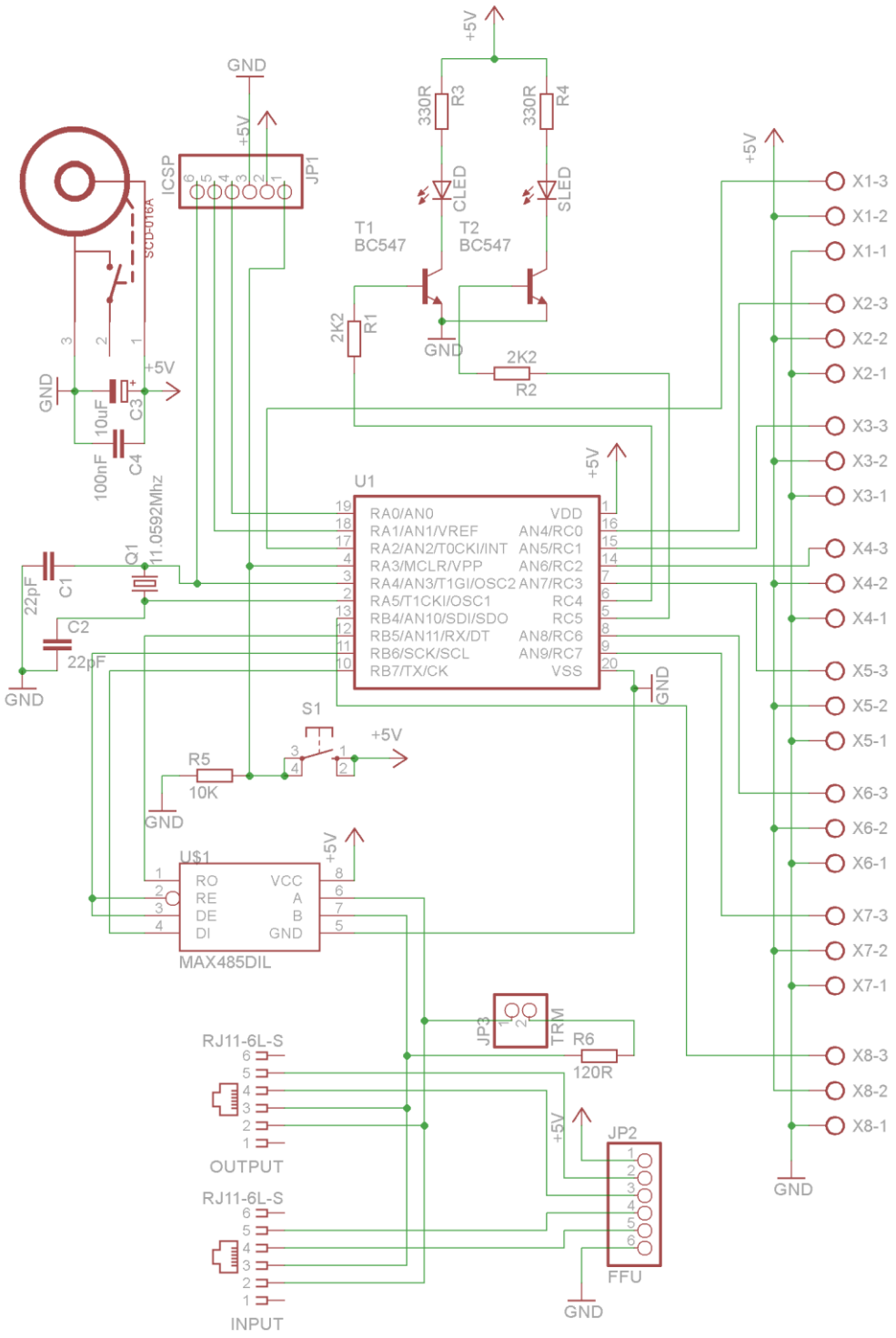
Konektory RJ11 byly zvoleny proto, že propojovací kabel mezi nody a subsystémem není třeba pájet, ale stačí využít krimpovací kleště. Tento přístup značně usnadní montáž nodů v domě.

Poslední částí schématu je pinová lišta JP1, která obsahuje společnou zem, výstup +5V a propojení s výstupy RA0, RA1, RA3 a RA4. Slouží pro připojení programátoru PicKit 2 nebo PicKit 3 (viz obrázek 12). Díky této liště lze do nodu nahrát nový program, aniž by bylo nutné s mikrokontrolérem manipulovat.

Obrázek 12 - Programátor PICkit3 [49]

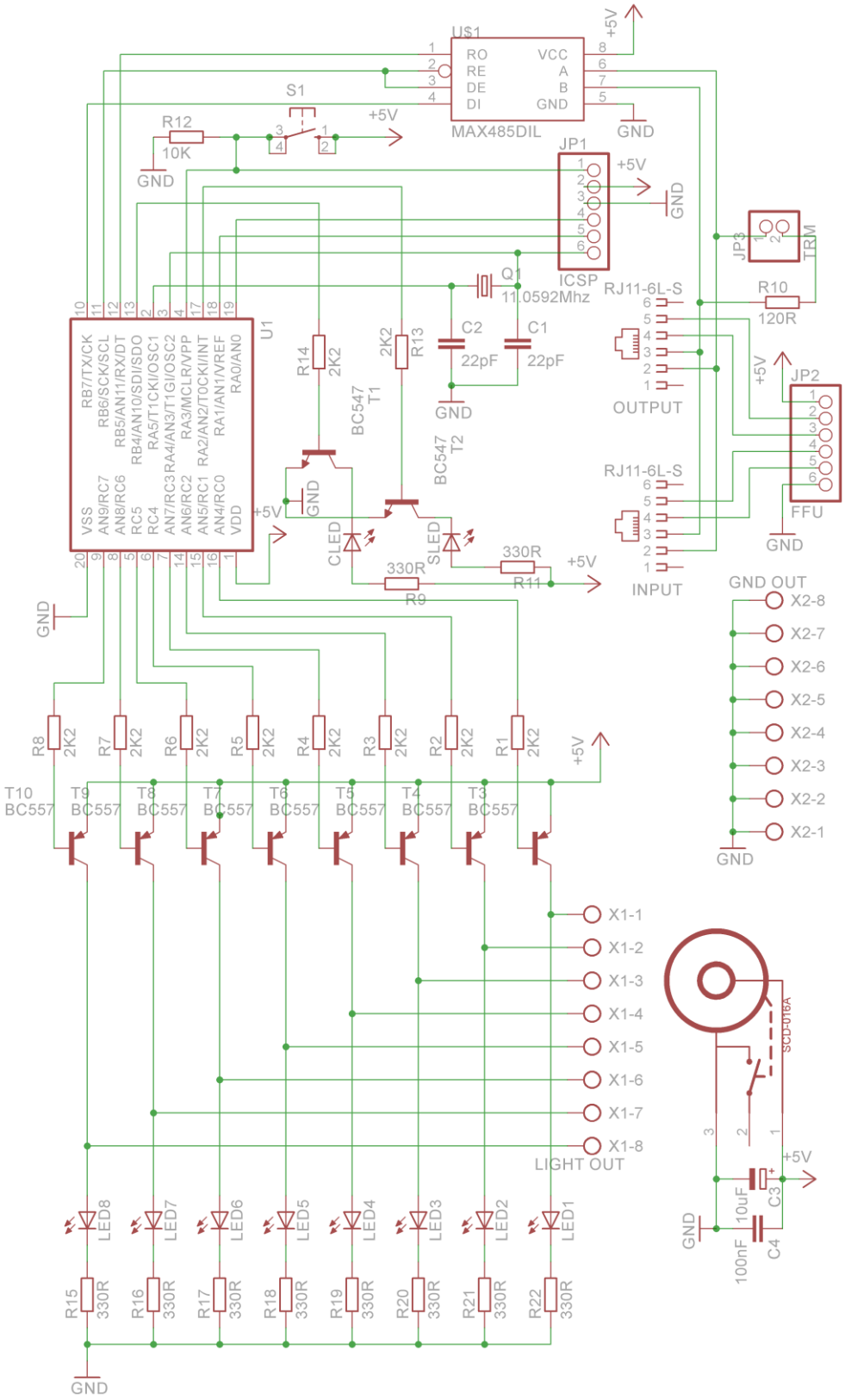
6.4.3. Schéma a popis zapojení analogového nodu

Stejně jako vstupní nod může být i analogový nod změnou programu použit pro více účelů. Využívá osm analogových vstupů mikrokontroléru (viz obrázek 13), takže je možné k němu připojit například stmívače, potenciometry a analogové sensory (teploměr, vlhkoměr nebo PIR čidlo). S předchozím schématickým zapojením sdílí velké množství součástek. Obsahuje stejné filtrační kondenzátory C3 a C4, stejně je připojen krystal, převodník MAX485 i tlačítko S1 a stejně připojené pinové lišty J1, J2 a J3. Indikační LED diody jsou ale připojeny k pinům RC4 a RC5 aby zbytečně nevyužívaly analogové vstupy mikrokontroléru. Analogovými vstupy, které měří veličiny ze svorkovnic X1 až X8 jsou piny AN2 až AN10.



Obrázek 13 - Schéma zapojení analogového nodu [autor]

6.4.4. Schéma a popis zapojení výstupního nodu



Obrázek 14 - Schéma zapojení výstupního nodu [autor]

Toto schématické zapojení nodu, znázorněné na obrázku 14, lze využít pro ovládání dvoustavových periférií. Těmi mohou být světla, motory, ventilátory apod. Protože jako předchozí návrhy i tento pracuje s napájecím napětím +5V, je nutné pro ovládání výkonových periférií umístit za nod relé. Jako výstupy jsou použity piny RC0 až RC7 mikrokontroléru, nod bude tedy osmikanálový a typu pro čtení i zápis (je možné zpětně ověřit stav nodu) s délkou dat jedna a šířkou dat jedna. Výstupy nejsou z důvodů proudové zatížitelnosti spínány napřímo, ale přes PNP [**50**] tranzistory T3 až T10. Báze tranzistorů jsou spojeny přes rezistory R1 až R8 s výstupy mikrokontroléru, emitory spojené k napájení a kolektory směrem ke svorkovnicím. Ke kolektorům jsou také připojeny přes rezistory R15 až R22 LED diody LED1 až LED8. Nod tedy může indikovat stav výstupů (sepnutý nebo vypnutý) bez přítomnosti periférie. Všechny ostatní části schématu jsou realizovány stejně jako u vstupního nodu.

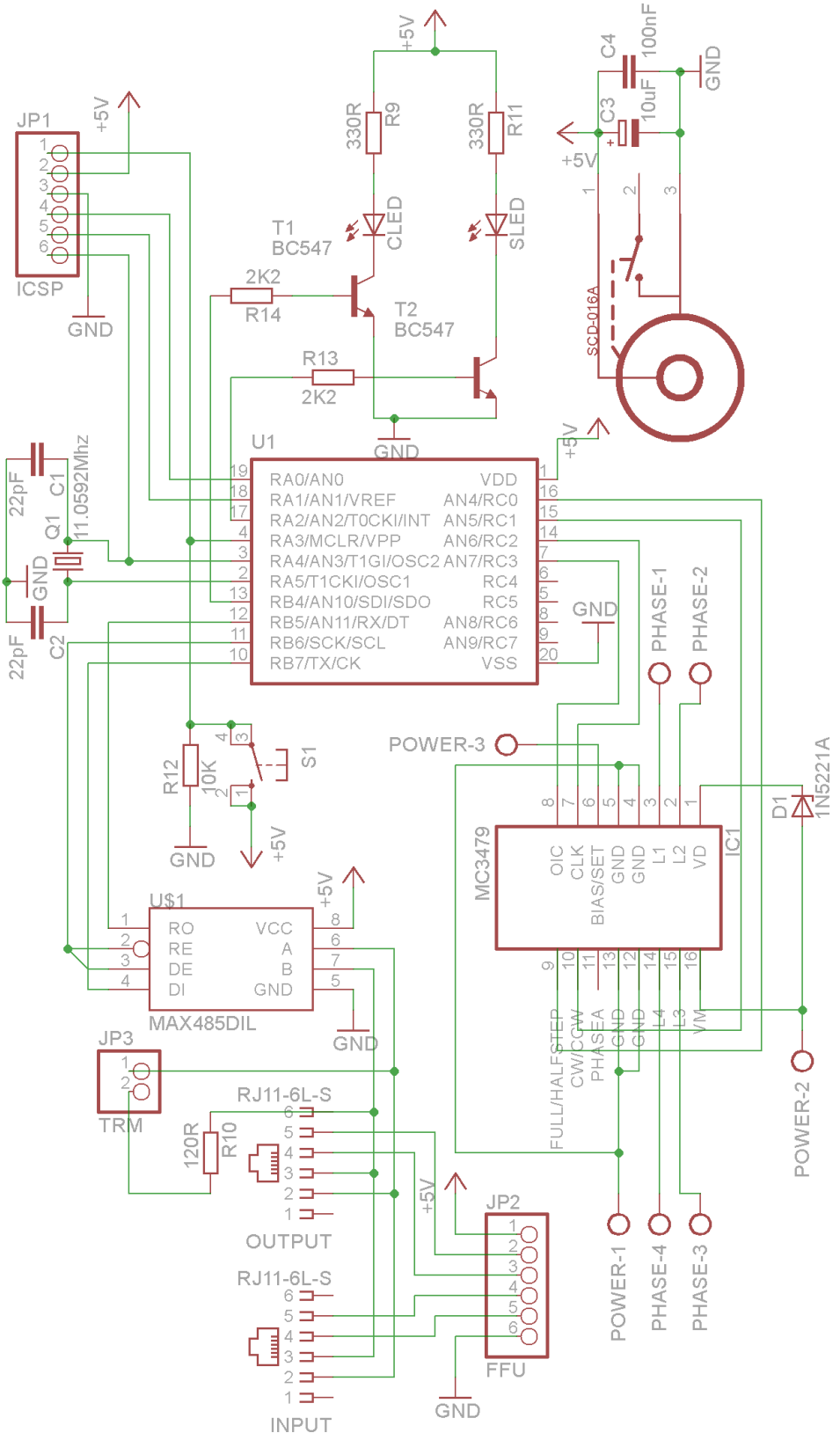
6.4.5. Schéma a popis zapojení motorového nodu

Dalším schématem, které bylo navrženo, je schéma zapojení motorového nodu. Toto schéma nodu není na rozdíl od předchozích schémat univerzální a slouží pouze k řízení otáčení krokového motoru. Využívá integrovaného obvodu MC3479 [**51**], což je driver krokového motoru. Driver je takový integrovaný obvod, který přijímá řídící signály a na jejich základě dle své vnitřní logiky provádí řízení jiných obvodů.

K mikrokontroléru je připojen MC3479 přes piny RC0, RC1, RC2 a RC3 (viz obrázek 15). Pin RC0 je připojen k pinu FULL/HALFSTEP driveru. Tím se volí, zdali má driver budit motor v plných krocích, nebo polovičních krocích. RC1 je připojen k CW/CCW pinu, který volí směr otáčení po, nebo proti směru hodinových ručiček. RC2 je připojen k pinu CLOCK, který řídí kroky motoru. Při každé náběžné hraně je motor otočen o jeden krok nebo půl krok ve zvoleném směru. Poslední pin RC3 vede k pinu OIC driveru a nastavuje impedanci krokového motoru. Dioda D1 slouží jako protizkratová ochrana při poruše motoru.

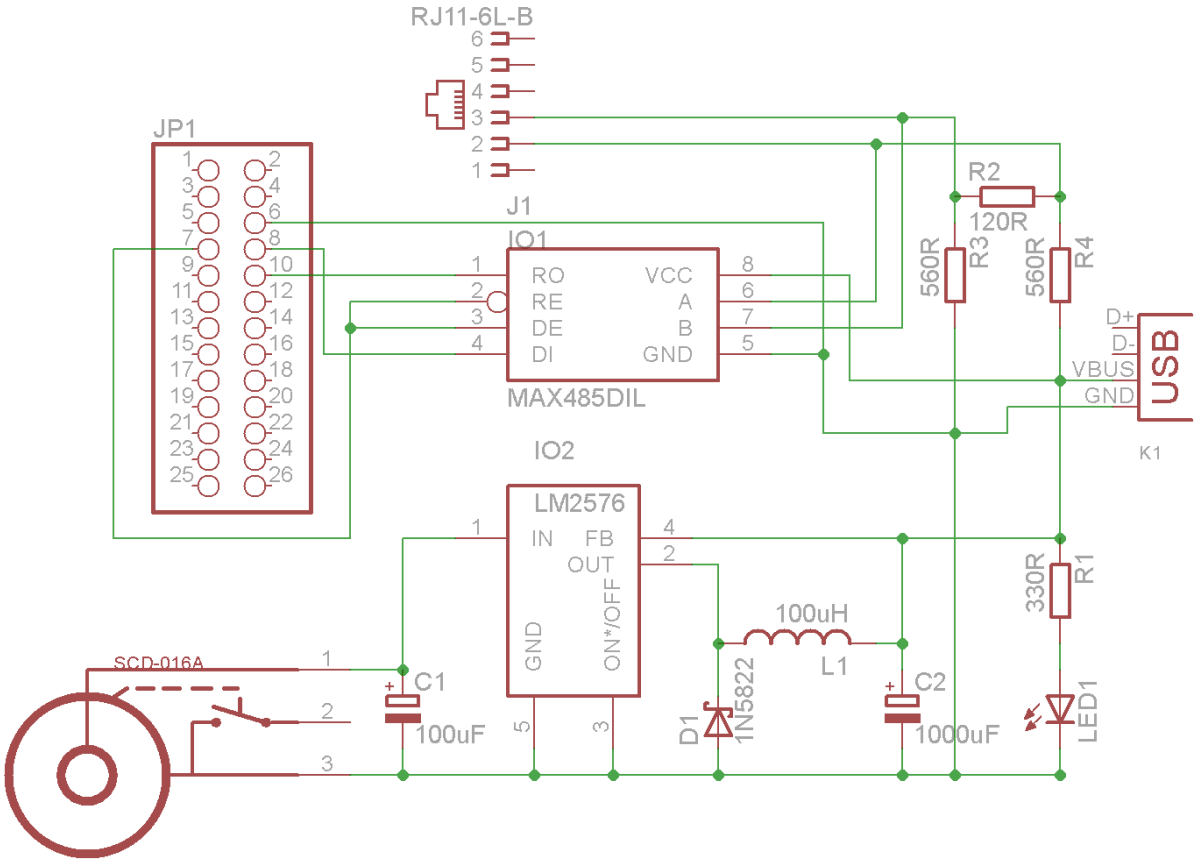
Motor se k driveru připojuje svorkami L1 až L4, které pak korespondují s vinutím konkrétního motoru. Protože MC3479 pracuje v napěťovém rozsahu 7,2 – 16V, musí být toto napětí přivedeno z externího zdroje svorkami POWER-1 a POWER-2. Svorka POWER-3 nemusí být zapojena a slouží ke sledování proudu protékajícího motorem. Všechna

schémata využívají lištu JP1 k nahrávání programu. Při svém běhu pak využívají neuzemněné analogové vstupy AN0 a AN1 k zachytávání šumu.



Obrázek 15 - Schéma zapojení motorového nodu [autor]

6.4.6. Schéma a popis zapojení pro Raspberry Pi shield



Obrázek 16 - Schéma zapojení Raspberry Pi driveru [autor]

Protože Raspberry Pi 2 neobsahuje řadič pro RS-485, bylo navrženo schéma (viz obrázek 16), které jej zároveň s napájecím zdrojem přidává. Název shield se používá pro takové plošné spoje, které se montují nad nebo pod existující výrobek. Výsledná deska bude mít rozměry identické s rozměry Raspberry Pi a pomoci distančních můstku se připevní nad něj.

Napájecí zdroj je schematicky identický se zdrojem pro nody. Pouze neobsahuje svorkovnici, ale USB konektor K1. Raspberry Pi také vyžaduje +5V napájení přes micro USB konektor. To bude do desky přivedeno krátkým propojovacím kabelem.

Napětí ze stabilizátoru také napájí převodník MAX485. Pro propojení převodníku z Raspberry Pi je použita pinová lišta JP1, kam vede společná zem, výběr směru komunikace RE a DE, příjem dat ze sběrnice RI a odesílání dat DI. Lišta JP1 pak bude také kabelem propojena přímo s Raspberry Pi. Výstup z převodníku je vyveden na konektor RJ11. Datový vodič A je dle specifikace pro RS-485 připojen přes rezistor R4 k +5V a datový vodič B přes

rezistor R3 ke společné zemi. Protože má shield pouze jeden konektor RJ11, bude subsystém vždy prvním bodem sběrnice. Z toho důvodu je paralelně na sběrnici připojen rezistor R3 jako terminátor.

Ke všem schématickým zapojením byly vytvořeny návrhy dvouvrstvých plošných spojů. Jejich schématické nákresy jsou uvedeny v příloze. Tyto plošné spoje byly také vyleptány a osazeny součástkami. Fotografie osazených plošných spojů jsou také uvedeny v příloze.

7. PŘEDSTAVENÍ REALIZOVANÉHO ŘEŠENÍ

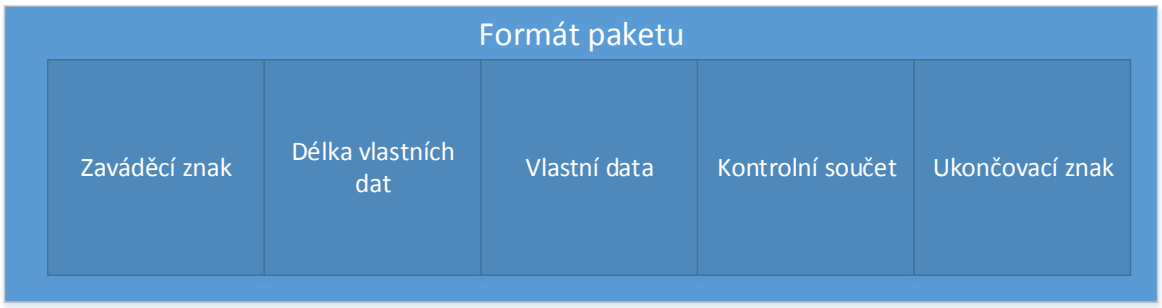
Tato kapitola se již nezabývá návrhem ale implementací definovaného řešení. V první podkapitole je představen komunikační protokol mezi nody a subsystémem resp. s aplikací v jazyce C++. Jsou popsány všechny možné zprávy, které se mohou odesílat. Druhá podkapitola pak popisuje všechny zprávy mezi aplikací v jazyce C++ a Java. Ke každé zprávě je uvedena její ukázka. Třetí kapitola pak popisuje diagram tříd aplikace. Účel každé třídy je popsán. Ve čtvrté kapitole jsou popsány všechny kroky, které aplikace provádí od svého spuštění až do přechodu k plánování. Pátá kapitola pak rozebírá princip implementovaného plánovače nodů. V šesté kapitole jsou popsány společné charakteristiky programů pro mikrokontroléry, které nody používají. Sedmá kapitola se zabývá optimalizací distribuce Raspbian. Poslední kapitola popisuje, jak se implementované řešení testovalo.

Pro odlišení samostatných ukázek zdrojového kódu je použito písmo `Courier`, kód přímo v odstavci je psán *kurzívou* a komentáře v kódu jsou vždy označeny dvěma lomítky //.

7.1. Komunikační protokol mezi nody a substémem

Pro RS-485 již existuje několik dostupných komunikačních protokolů. Mezi dva nejčastěji používané patří Modbus [**52**] a Profibus [**53**]. Modbus je možné kromě standardu RS-485 využívat také při rádiovém přenosu, optickém přenosu, Ethernetu, nebo na sběrnicích RS-232C a RS-422. Modbus vznikl v polovině 70. let 20. století jako protokol pro komunikaci a automatizaci výrobních linek a strojů ve fabrikách. Jeho struktura je velice jednoduchá. Profibus je novější protokol a vznikl na počátku 90. let 20. století. Stejně jako Modbus je využíván pro komunikaci ve výrobních linkách.

Oba dva protokoly jsou téměř průmyslovým standardem v oblasti řízení a automatice. Nicméně pro použití v navrženém systému jsou nevhodné. Předpokládají, že všechny komponenty na sběrnici mají od začátku svoji unikátní adresu. Nody ale při prvním zapnutí adresu nemají. Je jim přidělena až uživatel spustí na centrálním serveru detekci nových nodů. Z tohoto důvodu byl navržen a implementován vlastní komunikační protokol. Formát datového paketu je znázorněn na obrázku 17.



Obrázek 17 - Formát paketu [autor]

Každá zpráva mezi nody a subsystémem je obalena v paketu. První bajt každého paketu obsahuje zaváděcí znak, který je pokaždé stejný. Jedná se o binární číslo 35, v ASCII vyjádřené znakem #. Následuje bajt s informací o délce dat. Každý paket musí obsahovat alespoň jeden bajt vlastních dat. Minimální délka vlastních dat je tedy 1 bajt a maximální 255 bajtů. Poté jsou v paketu bajty vlastní zprávy. Předposlední bajt obsahuje kontrolní součet CRC8. Počítá se pro délku vlastních dat a vlastní data. Poslední bajt je ukončovací znak s binární hodnotou 36, v ASCII vyjádřený znakem $. Celý paket musí mít vždy délku alespoň 5 bajtů. Teprve pro přijetí pátého bajtu začínají nody vyhodnocovat, zdali se jedná o validní paket a kterému nodu je určen.

Adresa subsystému se ve zprávách odesílá jako jeden bajt a adresa nodů jako dva bajty. Maximální počet subsystémů v domě je tedy omezen na 256 a počet nodů teoreticky na 65535 nodů. Protože RS-485 dovoluje maximálně 32 komponent na sběrnici, je počet nodů u jednoho subsystému omezen na 31. Reálný počet připojitelných nodů v domě je tedy 256 x 31, neboli 7936 nodů celkem. Takový počet není možné ve skutečném domě vyčerpat a systém je tak dostatečně robustní a naddimenzovaný.

Všechny zprávy posílané mezi nody a subsystémem jsou pro označeny prefixem MESSAGE a mají přidělen celočíselný identifikátor (viz tabulka 5). Pokud zpráva neobsahuje žádná další data, je identifikátor jediným bajtem vlastních dat.

Paket	Celočíselný identifikátor
MESSAGE_REQUEST	10
MESSAGE_GET	11
MESSAGE_SET	12
MESSAGE_MEASURE	13
MESSAGE_PING	15
MESSAGE_PING_RESPONSE	16
MESSAGE_RESET	20
MESSAGE_DISCOVERY	0
MESSAGE_DISCOVERY_RESPONSE	1
MESSAGE_DISCOVERY_ADDRESS	2
MESSAGE_DISCOVERY_CHECK	4
MESSAGE_DISCOVERY_CHECK_OK	5

Tabulka 5 - Seznam zpráv a jejich identifikátorů [autor]

Zprávy lze rozdělit do dvou logických skupin. První skupina zpráv se používá pro adresaci nových nodů a druhá skupina pro běžnou komunikaci.

7.1.1. Zprávy pro adresaci

Adresaci vždy musí vyvolat uživatel na centrálním serveru. Následující zprávy jsou seřazeny v pořadí, ve kterém skutečně probíhají.

MESSAGE_DISCOVERY

Touto zprávou zahajuje subsystém adresaci. Zpráva neobsahuje žádná další data, slouží pouze jako pokyn pro nové nody, aby odeslaly svoje data.

MESSAGE_DISCOVERY_RESPONSE

Touto zprávou nody odpovídají na výzvu k adresaci. Právě zde se využijí neuzemněné analogové vstupy. Pokud by byl k subsystému připojen více jak právě jeden nový nod, nebylo by možné identifikovat který z nich má data s odpovědí na adresaci odeslat. Všechny by odeslali data ve stejný okamžik od přijetí zprávy MESSAGE_DISCOVERY. Proto nody vezmou šum, který se indukuje na analogových vstupech AN0 a AN1 (hodnota o velikosti 0 – 1023), vynásobí hodnoty mezi sebou a použijí jej jakou proměnnou *random*. Po tuto dobu v milisekundách pak čekají než odešlou odpověď, která obsahuje právě toto náhodné číslo, typ nodu a počet kanálů nodu. I tak může

nastat kolize při adresaci velkého množství nodů najednou. Centrální server vytvoří adresy nodům, jejichž pakety nekolidovali a upozorní uživatele, že je nutné adresační proces spustit ještě jednou.

MESSAGE_DISCOVERY_ADDRESS

Poté co aplikace přijme od nových nodů zprávu MESSAGE_DISCOVERY_RESPONSE, odešle vyšší vrstvě seznam nových nodů. Protože nody nemají ještě přidělené adresy, odesílá se právě náhodné číslo generované nody. Vyšší vrstva pak každému nodu přidělí adresu a pošle zpět i s náhodným číslem, aby bylo možné nody zpětně identifikovat. Aplikace posílá nodům zpět zprávu MESSAGE_DISCOVERY_ADDRESS, která obsahuje náhodné číslo, adresu subsystému a jejich nové adresy. Nody si tyto adresy uloží.

MESSAGE_DISCOVERY_CHECK

Po přijetí zprávy MESSAGE_DISCOVERY_ADDRESS si nod uloží nové adresy pouze do operační paměti a odesílá zpět k aplikaci svou novou adresu, adresu subsystému a náhodné číslo.

MESSAGE_DISCOVERY_CHECK_OK

Pokud subsystém úspěšně přijme a zpracuje zprávu MESSAGE_DISCOVERY_CHECK, znamená to, že nod svou novou adresu úspěšně přijal a je připraven k normální funkci. Aplikace tedy odesílá poslední adresační zprávu MESSAGE_DISCOVERY_CHECK_OK, nyní už bez náhodného čísla nodu, pouze s jeho adresou a adresou subsystému. Nod po přijetí ukládá svou adresu a adresu subsystému do paměti EEPROM a přechází k normální funkci, kdy dle svého typu stavy vyčítá, nastavuje, případně obojí.

7.1.2. Zprávy pro obsluhu

MESSAGE_REQUEST

Tuto právu odesílá subsystém nodům, ze kterých je možné data číst. Jako odpověď nody odesílají své aktuální data. Aby nody neodesílali data zároveň, obsahuje zpráva adresu nodu i adresu subsystému. Nod tedy po ověření kontrolního součtu zjistí, zdali je zpráva směrována právě jemu a až poté svá data odešle.

MESSAGE_GET

Touto zprávou nody odpovídají po přijetí MESSAGE_REQUEST. Součástí zprávy je vlastní adresa nodu, adresa subsystému a data. Nody posílají data pro všechny své kanály současně.

MESSAGE_SET

Subsystém touto zprávou nastavuje data nodům, do kterých lze zapisovat. Kromě dat (stavu) je ve zprávě také adresa nodu i subsystému.

MESSAGE_MEASURE

Protože zpožděné nody potřebují čas k naměření stavu, vysílá subsystém těmto nodům pokyn k naměření nových stavů. Pokud by nody měřili svůj stav neustále, mohlo by dojít k situaci, kdy se subsystém ptá na stav a nod by nebyl schopný odpovědět, protože by měřil stav. Zpráva MESSAGE_MEASURE tento problém odstraňuje. Po jeho přijetí má nod dostatek času k měření a se zpožděním pošle subsystém stejně jako v případě okamžitých nodů zprávu MESSAGE_REQUEST a nody odpoví zprávou MESSAGE_GET.

MESSAGE_PING

Protože nody do kterých lze data zapisovat neodesílají zpět žádné stavy a není tak možné zaručit, že opravdu pracují, zpráva MESSAGE_PING ověřuje jejich funkčnost. Ve zprávě je zahrnuta adresa subsystému a nodu samotného.

MESSAGE_PING_RESPONSE

Pokud nod obdrží paket MESSAGE_PING, odesílá zpět zprávu MESSAGE_PING_RESPONSE, ve kterém pouze uvede opět svoji adresu a adresu subsystému.

MESSAGE_RESET

Pokud by uživatel domu zjistil, že nějaký nod připojený k subsystému nepoužívá a chtěl by jej využít u jiného subsystému, může se touto zprávou odstranit v nodu informace o jeho adrese a adrese subsystému.

7.2. Zprávy mezi aplikacemi v subsystému

V tabulce 6 jsou uvedeny JSON zprávy, které si aplikace v jazyce C++ vyměňuje s vyšší vrstvou v jazyce Java. Stejně jako v případě zpráv mezi nody a subsystémem má každá zpráva přidělený unikátní celočíselný identifikátor zobrazený v tabulce 6.

Zpráva	Celočíselný identifikátor
REQUEST_SUB_ADDRESS	1
RESPONSE_SUB_ADDRESS	2
REQUEST_TYPE_LIST	12
RESPONSE_TYPE_LIST	13
REQUEST_NODE_LIST	8
RESPONSE_NODE_LIST	9
DISCOVER_NEW_NODES	15
NEW_NODES_FOUND	3
NEW_ADDRESSES	4
DATA_GET	5
DATA_SET	6
ERROR	7
REGISTER_NEW_TYPE	10
NODE_REMOVED	11
PING_REQUEST	16
PING_RESPONSE	17

Tabulka 6 - Seznam zpráv a jejich identifikátorů [autor]

7.2.1. Spuštění aplikace

Tyto zprávy se odesílají při prvním spuštění aplikace (nezávisle na tom, zdali se jedná o první spuštění nebo o spuštění po výpadku energie. Zprávy se musí posílat pokaždé, protože aplikace neobsahuje žádné perzistentní úložiště. U všech zpráv je uvedena jejich ukázka.

REQUEST_SUB_ADDRESS

Touto zprávou požaduje aplikace zaslání adresy subsystému od vyšších vrstev. Zpráva neobsahuje žádné další parametry.

```
{
  "message_type" : 1 //identifikátor zprávy
}
```

RESPONSE_SUB_ADDRESS

Vyšší vrstva odpovídá na požadavek o zaslání adresy subsystému.

```
{
  "address" : 10 ,    //adresa subsystému
  "message_type" : 2 //identifikátor zprávy
}
```

REQUEST_TYPE_LIST

Aplikace potřebuje ke korektnímu běhu znát všechny typy nodů, které se u subsystému vyskytují.

```
{
  "message_type" : 12 //identifikátor zprávy
}
```

RESPONSE_TYPE_LIST

Vyšší vrstva odpovídá na požadavek o zaslání všech typů nodů. V případě prvního spuštění celého systému bude seznam obsahovat pouze výchozí typy.

```
{
  "types" :
    [ //začátek pole typů nodů
      {
        "read_write" : 0 ,  //čtení nebo zápis
        "latency" : 0 ,     //okamžitý ne zpožděný
        "data_range" : 1 ,  //šířka dat
        "type" : 1 ,        //identifikátor typu
        "data_count" : 1    //počet dat
      } , {
        "read_write" : 1 ,
        "latency" : 0 ,
        "data_range" : 1 ,
        "type" : 3 ,
        "data_count" : 1
      }
    ]
  "message_type" : 13 //identifikátor zprávy
}
```

REQUEST_NODE_LIST

Aplikace také potřebuje při spuštění dostat seznam všech nodů připojených k subsystému. Tato zpráva se odesílá až po přijmutí seznamu typů, jinak by aplikace nemohla k nodům korektně přiřadit jejich typy.

```
{
  "message_type" : 1 //identifikátor zprávy
}
```

RESPONSE_NODE_LIST

Vyšší vrstva posílá seznam nodů, připojených k subsystému. Pokud se jedná o první spuštění subsystému, bude tento seznam prázdný.

```
{
  "nodes" : [ ] , //pole je prázdné, protože subsystém zatím nemá
                  //připojené žádné nody
  "message_type" : 9 //identifikátor zprávy
}
```

V případě spuštění aplikace po výpadku subsystému bude zpráva obsahovat všechny nody připojené k aplikaci.

```
{
  "nodes" :
    [  //začátek pole
      {
        "address":424 ,
        "channel_count" : 8 ,
        "data":
          [  //pole s posledním stavem uloženým v databázi
            {
            "data" : [0],
            "channel":0
            }
          ] ,
        "type" : 1
      }
    ] ,
  "message_type" : 9 //identifikátor zprávy
}
```

7.2.2. Detekce nových nodů

Tyto zprávy jsou odesílány, pokud uživatel chce detekovat nově připojené nody a nechat vyšší vrstvy přidělit adresy.

DISCOVER_NEW_NODES

Pokud uživatel rozšíří subsystém o nové nody, musí na centrálním serveru zvolit funkci pro detekci nových nodů. Tato zpráva tedy přichází od vyšší vrstvy. Aplikace poté pozastaví plánování obsluhy nodů a začne vyhledávat nové nody.

```
{
   "message_type" : 15  //identifikátor zprávy
}
```

NEW_NODES_FOUND

Pokud jsou na subsystému nalezeny nové nody, jsou jejich typy a generovaná náhodná čísla odeslána vyšší vrstvě, které těmto nodům přidělí nové unikátní adresy.

```
{
  "nodes" :
    [
      {
        "random" : 44583 , //náhodné číslo generované nodem
        "channel_count" : 8 , //počet kanálů
        "type" : 1    //identifikátor typu
      }
    ] ,
  "message_type" : 3  //identifikátor zprávy
}
```

NEW_ADDRESSES

Tuto zprávu odesílá vyšší vrstva jako odpověď na NEW_DEVICES_FOUND. Zpráva obsahuje generovaná náhodná čísla nodů s přidělenými adresami.

```
{ "nodes" :
    [
      {
        "random" : 44583 ,
        "channel_count" : 8 ,
        "type" : 1 ,
        "address" : 36
      }
    ] ,
  "message_type" : 4  //identifikátor zprávy
 }
```

REGISTER_NEW_TYPE

Protože aplikace je navržena tak, aby v ideálních[8] podmínkách byla spuštěna neustále, může nastat situace, že uživatel chce subsystém rozšířit o nový nod takového typu, který aplikace nemá v paměti uložen. Vyšší vrstva tedy aplikaci touto zprávou odešle informace o novém typu a aplikace si jej uloží do paměti. Pokud by nastal výpadek a aplikace se musela spustit znovu, bude tento nový typ již poslán při spuštění ve zprávě RESPONSE_TYPE_LIST.

```
{
  "type" : 33 ,        //identifikátor typu
  "read_write" : 0 ,   //čtení nebo zápis
  "latency" : 0 ,      //okamžitý nebe zpožděný
  "data_range" : 1 ,   //šířka dat
  "data_count" : 1 ,   //počet dat
  "message_type" : 10  //identifikátor zprávy
}
```

[8] Ideálním stavem je myšleno, že v domě nikdy nenastane výpadek elektrické energie a hardware subsystému se fyzicky nepoškodí.

7.2.3. Komunikace s nody

DATA_GET

Pokud plánovač naměří nový stav nodu, je v této zprávě odeslán. Svůj stav mohou odesílat pouze nody, ze kterých lze stav vyčítat. Níže je uvedena zpráva, která ukazuje sepnutí spínače osmikanálového nodu s adresou 356.

```
{
  "address" : 356 ,
  "data" :
    [  //pole, které obsahuje data pro každý kanál
      {
        "data" : [1] , //data nodu jsou také pole z důvodu počtu
                       //dat
        "channel" : 0  //číslo kanálu
      } ,
      {
        "data" : [0] ,
        "channel" : 1
      } ,
      {
        "data" : [0] ,
        "channel" : 2
      } ,
      {
        "data" : [0] ,
        "channel" : 3
      } ,
      {
        "data" : [0] ,
        "channel" : 4
      } ,
      {
        "data" : [0] ,
        "channel" : 5
      } ,
      {
        "data" : [0] ,
        "channel" : 6
      } ,
      {
        "data" : [0] ,
        "channel" : 7
      }
    ] ,
  "message_type" : 5 //identifikátor zprávy
}
```

DATA_SET

Tuto zprávu přijímá aplikace od vyšší vrstvy, pokud dle uživatelských definovaných pravidel potřebuje změnit stav nebo data nodu typu pro zápis, nebo pro zápis a čtení.

```
{
  "address" : 124 ,
  "data" :
    [
      {
        "data" : [255 , 0 , 0] ,
        "channel" : 0
      } ,
      {
        "data" : [0 , 255 , 0] ,
        "channel" : 1
      } ,
      {
        "data" : [0 , 0 , 255] ,
        "channel" : 2
      }
    ] ,
  "message_type" : 6  //identifikátor zprávy
}
```

NODE_REMOVED

Mohou nastat dvě situace. Buď dojde k poruše nodu a ten přestane reagovat na pakety odesílané aplikací nebo jej uživatel může manuálně odebrat ze subsystému. V obou případech ale začne aplikace odesílat vyšší vrstvě zprávy o nefunkčním nodu. Vyšší vrstva tuto informaci zobrazí uživateli a ten potvrdí že nod je buď nefunkční, nebo že jej odebral. Vyšší vrstva pak odesílá zprávu NODE_REMOVED a aplikace jej odstraní ze svého seznamu a přestane jej v plánovači obsluhovat.

```
{
  "address" : 250 ,
  "message_type" : 11  //identifikátor zprávy
}
```

PING_REQUEST

Pokud by v konkrétním subsystému byly připojeny jenom nody, ze kterých lze stav vyčítat a jejich stav se dlouhodobě neměnil, aplikace by dlouhou dobu neposlala vyšší vrstvě žádné zprávy. PING_REQUEST tedy odesílá vyšší vrstva aplikaci pouze jako ověření, zdali aplikace stále korektně pracuje.

```
{ "message_type" : 16  //identifikátor zprávy
}
```

PING_RESPONSE

Aplikace odpovídá touto zprávou na PING_REQUEST. Neobsahuje žádná další data.

```
{
  "message_type" : 17  //identifikátor zprávy
}
```

7.2.4. Hlášení chyb

Cílem každé aplikace je samozřejmě pracovat zcela bez chyb. Ale v takto komplexně navrženém systému je možné, že se nějaké chyby v aplikaci, nodech nebo samotné sběrnici mohou vyskytnout. Pokud aplikace dokáže chybu identifikovat, vysílá zprávu ERROR, které obsahuje další atribut a to kód chyby.

```
{
   "message_type" : 7 ,  //identifikátor zprávy
   "error" : 1..7 //kód chyby
}
```

Celkem bylo stanoveno sedm chybových hlášek, které mohou teoreticky nastat.

ERROR_UART

Aplikaci využívá pro komunikaci hardwarový UART vyvedený na Raspberry Pi 2. Pokud by došlo vlivem stáří k jeho poškození a nebylo by možné korektně otevřít spojení, případně skrze něj odeslat paket, je odesláno toto chybové hlášení vyšší vrstvě. Ta uživatele upozorní, že je nutné subsystém vyměnit.

ERROR_COMMUNICATION

Tuto chybu odesílá aplikace v případě problémů s TCP spojením. Například pokud by data dorazila nekompletní.

ERROR_NODE_NOT_RESPONDING

V případě že nod neodpovídá na požadavek k poslání stavu případně na kontrolu dostupnosti, je tato chyba logována pouze interně v aplikaci spolu se systémovým časem, kdy nastala. Pokud se ale chyba opakuje znovu do deseti sekund, je vyšší vrstvě odeslána chyba ERROR_NODE_NOT_REPONDING. Jestli se chyba začne vyskytovat často, je uživateli zobrazeno upozornění a doporučena výměna nodu. Tuto chybu také může způsobit sám uživatel, když nod odpojí za běhu subsystému.

ERROR_PACKET_NONSENSE

Tato chyba může výjimečně nastat, pokud se na sběrnici RS-485 indukuje elektrické rušení. Například pokud bude datový kabel tažen souběžně se silovým kabelem na 230V a dojde k sepnutí spotřebiče s velkým nárazovým příkonem. Například mikrovlnná trouba nebo konvice.

ERROR_NO_NEW_NODES

Pokud uživatel spustí vyhledávání nových nodů a v subsystému není nalezen žádný nový nod, je odeslána vyšší vrstvě tato zpráva.

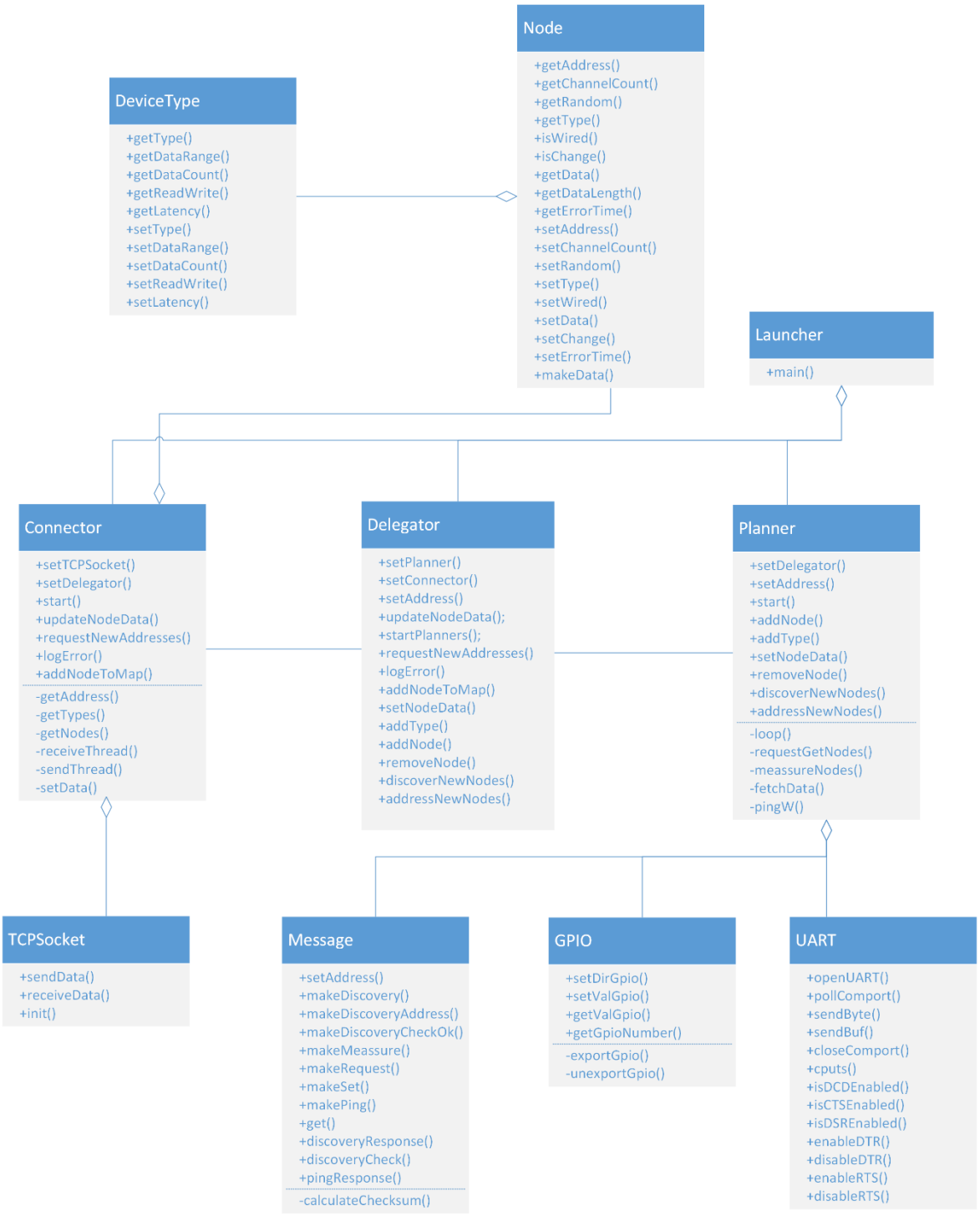
ERROR_FAILED_TO_ADDRESS_NODE

Tato chyba je odeslána pokud nový nod odpoví na detekci, ale neodpoví zpět, že přijal svou adresu. Takový stav by výjimečně mohl nastat, pokud by měl mikrokontrolér fyzicky poškozenou paměť.

ERROR_JSON_NOT_VALID

Tento kód chyby je odesílán v případě, že aplikace přijme JSON zprávu, ale je nekorektně formátována.

7.3. Diagram tříd aplikace



Obrázek 18 - Diagram tříd aplikace [autor]

Na obrázku 18 je znázorněn diagram tříd [**54**] aplikace. V rámci zachování přehlednosti jsou v diagramu uvedeny pouze privátní a veřejné metody tříd. Parametry metod, výčty a proměnné tříd v diagramu zahrnuty nejsou.

Třída *Connector* komunikuje s vyšší vrstvou zprávami ve formátu JSON. Pro jejich sestavování a parsování bylo zvolena knihovna JsonCpp [**55**], jejíž licence MIT dovoluje využití i komerčních projektech [**56**]. *Connector* má v sobě instanci třídy *TCPSocket*, která odesílá zprávy vyšší vrstvě a zprávy také přijímá pomocí TCP klient/server spojení. *Delegator* má za úkol v současném stavu pouze předávat volání metod mezi třídami *Connector* a *Planner*. *Planner* je zodpovědný za obsluhu nodů pomocí interního plánovacího algoritmu. Protože ke své funkci potřebuje UART rozhrání a GPIO pin (pro řízení směru komunikace) používá instance stejně pojmenovaných tříd *UART* a *GPIO*. *UART* otevírá sériové spojení. Třída *GPIO* pak využívá konkrétní pin Raspberry Pi 2. *Planner* také využívá třídu *Message* pro sestavování zpráv a dekódování zpráv ze sběrnice.

Třída *DeviceType* reprezentuje typ nodu. Obsahuje identifikátor typu, šířku a počet dat, typ přístupu a zpoždění. Typ přístupu je výčet *R, RW, W* (čtení, čtení i zápis, zápis). Zpoždění je také výčet, a to *NOW* nebo *DELAYED* (okamžitý nebo zpožděný). Třída *Node* pak představuje samotné nody. Agreguje typ nodu, obsahuje svoji adresu, počet kanálů a náhodné číslo.

Všechny aplikace se v C++ spouští metodou *main* a ta je obsažena v souboru Launcher.cpp.

7.4. Spuštění aplikace

Při spuštění binárního souboru s aplikací se jako první volá metoda *main* v souboru Launcher.cpp. Metoda *main* jako první otevře soubor hausy.conf[9], který obsahuje základní konfiguraci aplikace ve formátu INI [**57**]. Pro parsování souboru je použita knihovna simpleini [**58**], také šířená pod MIT licencí. Výchozí konfigurace aplikace vypadá následovně:

```
[settings]
ip=127.0.0.1
port=1007
uart=/dev/ttyAMA0
gpio=4
```

V konfiguračním souboru se tedy nachází IP adresy vyšší vrstvy, TCP portu na kterém se komunikuje, označení UART portu a číslo GPIO pinu. Nastavení je

[9] Slovo hausy pochází ze zkratky, jakou byla tato diplomový práce a diplomová práce bakaláře Martina Vancla pojmenována. Home AUtomation SYstem. Tedy HAUSY.

konfigurovatelné pro účely testování a možnosti zkompilovat a spustit aplikaci i na jiném počítači než Raspberry Pi 2.

Protože aplikace potřebuje neustále čekat na příchozí data od vyšší vrstvy a zároveň odesílat změny stavu nebo dat nodů, vyvstává zde problém. Jedná se o obousměrnou asynchronní komunikaci mezi třídami *Connector*, *Delegator* a *Planner*. Pokud *Planner* přes *Delegator* volá *Connector*, potřebuje mít přístup stále k jedné instanci obou tříd. *Connector* zase volá přes *Delegator Planner* a musí mít také přístup stále ke stejné instanci tříd. Zjednodušeně řečeno se jedná o stav kdy třída A volá metody třídy B, ale zároveň třída B potřebuje volat metody třídy A. Takový problém je ale C++ jednoduše řešitelný díky ukazatelům [**59**].

Metoda *main* v Launcher.cpp nejprve vytvoří instanci třídy *Connector*, pak instanci třídy *Planner* a v jejím konstruktoru předá název UART portu z konfigurace a číslo GPIO pinu. Následně vytvoří také instanci třídy *Delegator*. Poté se instanci třídy *Connector* předá ukazatel na instanci třídy *Delegator* metodou *setDelegator*. Ukazatel na instanci třídy *Delegator* se také předá metodou *setDelegator* předá instanci třídy *Planner*. Poté se instanci třídy *Delegator* metodami *setConnector* a *setPlanner* předají ukazatele na instance tříd *Connector* a *Planner*. Díky tomuto postupu mají všechny třídy ukazatel na jediné běžící instance ostatních tříd a mohou mezi sebou obousměrně volat metody.

```
Connector con;  //vytvoření instance třídy Connector
Planner pln(uart, gpio); //Vytvoření instance třídy Planner
Delegator dlg;  //Vytvoření instance třídy Delegator

con.setDelegator(&dlg); //Předání ukazatele na instanci třídy
                        //Delegator instanci třídy Connector
pln.setDelegator(&dlg); //Předání ukazatele na instanci třídy
                        //Delegator instanci třídy Planner
dlg.setConnector(&con); //Předání ukazatele na instanci třídy
                        //Connector instanci třídy Delegator
dlg.setPlanner(&pln);   //Předání ukazatele na instanci třídy
                        //Planner instanci třídy Delegator
```

Může se zdát, že třída *Delegator* je v aplikaci zbytečná, protože pouze předává (deleguje) volání dvou jiných tříd. *Delegator* slouží jako příprava pro rozšíření systému o bezdrátové nody. Ty budou potřebovat jiný plánovač pro jejich obsluhu. Stávající třída *Planner* se pouze přejmenuje na *WiredPlanner* (drátový plánovač) a vytvoří se nová třída *WirelessPlanner* (bezdrátový plánovač). *Delegator* pak bude od třídy *Connector* volat metody dle připojení nodu. Třída *Node* již obsahuje *bool* proměnnou *wired*, kde *true*

znamená drátový nod a *false* nod bezdrátový. Vyšší vrstva s současném stavu vždy posílá *true*.

Posledním krokem metody *main* je vytvoření instance třídy *TCPSocket*, kde se přes konstruktor třídy předá nastavení IP adresy a portu TCP socketu. Aplikace vyšší vrstvy musí být spuštěna vždy jako první, protože se chová jako TCP server a čeká na připojení klienta. Na instanci třídy *TCPSocket* zavolá metodu *init*, která vrací 1 v případě úspěšného navázání spojení, nebo záporné číslo s kódem chyby. V případě úspěchu je reference na instanci třídy *TCPSocket* předána metodou *setTCPSocket* instanci třídy *Connector*. Pak je v instanci třídy *Connector* zavolána metoda *start*. Případě neúspěchu je aplikace ukončena s návratovým kódem chyby, který si může vyšší vrstva přečíst a provést kroky k jeho odstranění.

Proces spouštění pokračuje v metodě *start* třídy *Connector*, kde je první krok získat vlastní adresu subsystému metodou *getAddress*. Aplikace odešle přes *TCPSocket* metodou *sendData* zprávu REQUEST_SUB_ADDRESS. Všechny JSON zprávy jsou sestavovány a také parsovány knihovnou JsonCpp.

```
//Poslání zprávy s požadavkem o přidělení adresy
messageSend["message type"] = REQUEST_SUB_ADDRESS;
Connector::socket.sendData(writer.write(messageSend));

//Čekání na příjem adresy
receive = socket.receiveData();

messageReceive.clear();  //Smazání předchozí přijaté zprávy
//Pokus o parsování zprávy
bool parsed = reader.parse(receive, messageReceive, false);

if (parsed) {
     address = messageReceive.get("address", "0").asInt();
     pDelegator->setAddress(address);
} else {
     logError(ERROR_JSON_NOT_VALID);
}
```

Metoda pak čeká na zprávu RESPONSE_SUB_ADDRESS a uloží si adresu subsystému a předá jej přes instanci třídy *Delegator* metodou *setAddress* plánovači. Program se vrací zpět do metody start a volá metodu *getTypes*. Ta odesílá zprávu REQUEST_TYPE_LIST a čeká na zprávu RESPONSE_TYPE_LIST, která obsahuje seznam typů v systému. Zprávu parsuje a seznam typů ukládá do neřazené mapy [**60**], kde ID typu je klíč a celý objekt *DeviceType* je hodnota. Každý přijaty typ je také delegován plánovači metodou *addType* na instanci třídy *Delegator*. Opět se program vrací do metody *start* a zde volá metodu *getNodes*, která získává od vyšší vrstvy zprávami

REQUEST_NODE_LIST a RESPONSE_NODE_LIST seznam nodů připojených k subsystému. Seznam nodů si ukládá také do neřazené mapy, kde adresa je klíč a objekt *Node* hodnota a také je delegován plánovači. Následně metoda *start* vytvoří nové vlákno, ve kterém se provádí metoda *sendThread*. Ta obsahuje nekonečný blokující cyklus, ve kterém se čeká na zaplnění fronty zpráv k odeslání vyšší vrstvě. Pokud je ve frontě zpráva, vyzvedává se z fronty a odesílá. Protože je fronta blokující, tak nekonečný cyklus nezatěžuje CPU. Vlákno s metodou *sendThread* je odpojené od hlavního vlákna a proto proces spouštění pokračuje dál.

```
void Connector::sendThread() {
      string msg;  //do stringu se ukládají složené JSON zprávy
      while (true) {
            msg = messageQueue.pop();  //vyzvednutí z fronty
            socket.sendData(msg); //odeslání přes inst. TCPSocketu
      }
}
```

Předposledním krokem metody *start* je zavolání metody *startPlanners* na instanci třídy *Delegator*. Posledním krokem je přechod do metody *receiveThread*, která obsahuje nekonečný cyklus a neustále čeká na příchozí data, které vrací metoda *receiveData* třídy *TCPSocket*.

7.5. Princip plánovače

Funkce plánovače začíná v momentě, kdy je *Delegator* zavolán instancí třídy *Planner* metodou *start*. Ta instancuje třídu *GPIO*, a metodami *exportGpio* a *setDirGpio* registruje pin definovaný v konfiguraci. Logická hodnota pinu se mění při vysílání a přijímaní dat ze sběrnice. Ta provádí otevření UART spojení na zařízení definovaném v konfiguraci zavoláním metody *openUART* v instanci třídy *UART*. Pokud se UART spojení nezdaří otevřít, je přes *Delegator* odeslána chyba a metoda start se opouští. Jinak proces pokračuje vytvořením odpojeného vlákna. Ve vlákně se volá metoda *loop*, která již obsahuje celou logiku obsluhování nodů. Protože vlákno je odpojené, metoda je ukončena a plánovač již vykonává periodickou obsluhu nodů. Zdrojový kód nekonečného cyklu (realizovaným jako *while (true)*) metody *loop* vypadá následovně:

```
if (!halted) {  //kontrola zámku
      confirmed = false;  //nastavení zámku
      pass++;

      lockR = true;
      requestGetNodes(nowNodesR);
      lockR = false;

      if (updateW == false) {
            lockW = true;
            setNodes(nowNodesW);
            lockW = false;
      }

      if (updateRW == false) {
            lockRW = true;
            setNodes(nowNodesRW);
            lockRW = false;
      }


      if (pass == 10 && updateRW == false) {
            lockRW = true;
            requestGetNodes(nowNodesRW);
            lockRW = false;
      }

      if (pass == 50) {
            lockDR = true;
            meassureNodes(delayedNodesR);
            lockDR = false;
            if (updateW == false) {
                  lockW = true;
                  pingW();
                  lockW = false;
            }
      }

      if (pass == 100) {
            lockDR = true;
            requestGetNodes(delayedNodesR);
            lockDR = false;
            pass = 0;
      }
      confirmed = true;  //odemknutí zámku
}

usleep(1000);
```

V každém průchodu smyčkou je získáván stav z nodů pouze pro čtení metodou *requestGetNodes*. Před procházením je *bool* proměnná *lockR* nastavena jako *true*. Znamená to, že s množinou nodů pouze pro čtení se právě manipuluje a jedná se tedy o zámek. Po dokončení metody *requestGetNodes* je *lockR* nastaven zpět na *false*. Metoda prochází

neřazenou mapu *nowNodesR*. Pokud je mapa prázdná, tak kód pokračuje dál. Ve třídě plánovače jsou celkem čtyři neřazené mapy, dle možných typů nodů. Okamžité pro čtení *nowNodesR*, okamžité pro čtení i zápis *nowNodesRW*, okamžité jen pro zápis *nowNodesW* a zpožděné jen pro čtení *delayedNodesR*. Tyto množiny jsou naplněny při startu aplikace metodou, kterou volá *Delegator,* a to *addNode*. Ta dle typu nodu nod vloží do příslušné mapy.

Metoda *requestGetNodes* přijímá jako parametr ukazatel na párovou mapu nodů, které má obsloužit. Pro každý nod v mapě pošle zprávu MESSAGE_REQUEST, kterou pro ni složí instance třídy *Message* metodou *makeRequest*. Pak metoda čeká na přijetí MESSAGE_GET zprávy.

```
//potřebujeme příjmout vlastní data plus osm bajtů dalších dat
int rxpos = fetchData(node, node.getDataLength() + 8);
//metoda get vrací 0 když jsou data stejná jako předchozí
//1 pokud jsou odlišná
//-1 pokud jsou chybná
int change = message.get(rxbuf, node, rxpos);
if (change == 1) {
          pDelegator->updateNodeData(node);
}
```

Každý nod má metodu *getDataLength*, která vrací počet bajtů, které nod musí při dotazu na stav vrátit. Zpráva MESSAGE_GET navíc obsahuje 8 další bajtů. Informace o nodu, který se obsluhuje a délkou dat, které se mají přijmout, jsou předány jako parametry metody *fetchData*. Ta vrací počet skutečně přijatých dat a upravuje globální pole přijatých dat *rxbuf*. Pro kontrolu přijatých dat je pole *rxbuf*, počet přijatých dat a nod předán metodě třídy Message *get*. Ta vrací tři možná čísla. -1 značí chybu odpovědi nodu. Buď je paket krátký, nebo nesouhlasí kontrolní součet CRC8. Je změřen čas, kdy chyba nastala a porovnán s časem poslední chyby nodu (přes metodu nodu *getErrorTime*). Pokud by se jednalo o druhou chybu během deseti sekund, je tato chyba okamžitě odeslána vyšší vrstvě. Když metoda *get* vrátí nulu, nod vrátil stejná data nebo stav jako dříve a není třeba upozornit vyšší vrstvy. Pokud je návratová hodnota 1, tak se nová data nebo stav předávají instanci třídy *Delegator* metodou *updateNodeData* a jako parametr nod. Ta volá stejnou pojmenovanou metodu na instanci třídy *Connector*, která sestaví zprávu DATA_GET a pošle nový stav nebo data vyšší vrstvě.

V každém průchodu je také znovu poslán stav nodům s typem pouze pro zápis nebo pro zápis a čtení, tedy pokud u některého nodu z této množiny nejsou právě nastavovány

nová data nebo stav. Pokud *Connector* dostane stav nebo data od vyšší vrstvy, zavolá přes *Delegator* metodu *setNodeData*. Ta nastaví *bool* proměnnou *updateW* resp. *updateRW* jako *true*, tedy příznak toho že se některému nodu v mapách právě mění data. Pokud jsou nová data přijata v momentě, kdy se s mapou pracuje a zámky *lockW* nebo *lockRW* jsou nastaveny jako *true*, čeká metoda *setNodeData* než jsou zámky uvolněny, nastaví *updateW* nebo *updateRW* jako *true*, upraví data nebo stavy, a nastaví zpět *updateW* nebo *updateRW* jako *false*.

Důležitou proměnnou v plánovacím algoritmu je číslo *pass*, které je počátečně nastavené na nulu a slouží jako počítadlo průchodů smyčky. V každém průchodu je iterováno. Pokud je *pass* roven 10 a neprobíhá aktualizace stavu nebo dat nodů pro čtení i zápis (viz algoritmus výše uvedený), je vyčten z těchto nodů stav metodou *requestGetNodes*, jinak je vyčítání přeskočeno.

Pokud je *pass* rovno 50, tak je nastaven zámek na množinu nodů typu zpožděných. Jejich stav ani data sice není možné měnit, takže by se zámek mohl zdát jako zbytečný, ale je možné je ze subsystému odebrat stejně jako ostatní nody. Odebrání nodu iniciuje uživatel na centrálním serveru. Ten přes vyšší vrstvy pošle zprávu NODE_REMOVED třídě *Connector*, která předá přes *Delegator* metodou *removeNode* adresu nodu, který se má odebrat. Pokud by se s množinou nodů právě pracovalo (je nastaven *lock* na *true*), tak metoda čeká, až je možné nod z konkrétní množiny odstranit.

Metoda *meassureNodes* iteruje skrze mapu a pro všechny zpožděné nody pouze pro čtení sestaví pomocí instance třídy *Message* metodou *makeMeassur*, která přijímá jako parametr nod, zprávu MESSAGE_MEASSURE a pošle pokyn nodům pro naměření svého stavu nebo dat. Nastavuje zámek *lockDR*. Pří padesátém průchodu je také odeslána zpráva MESSAGE_PING metodou *pingW* všem nodům s typem pouze pro čtení. Předtím je nastaven zámek *lockW* a poté opět odemknut. Metoda *pingW* také používá instanci třídy *Message* pro sestavení zprávy MESSAGE_PING a pak metodu *fetchData* pro získání a validaci odpovědi. Metoda instance třídy *Message*, *pingResponse* pak vrací buď -1 při nevalidní zprávě a 1 při korektní zprávě MESSAGE_PING_RESONSE. Logování chyby je řešeno stejně jako u metody *requestGetNodes*.

Při stém průchodu je pak pro párovou mapu *delayedNodesR*, s nody zpožděného typu zavolána metoda *requestGetNodes*. Platí stejná logika pro proměnné *updateDR* a *lockDR*

jako pro ostatní zámky. Při stém průchodu je proměnná *pass* nastavena zpátky na nulu a plánovač čeká jednu milisekundu.

Třída *Message* musí u metod *set* a *get* řešit jeden problém. Vyšší vrstva posílá nový stav jako datový typ *unsigned int*, tedy 32 bitů dlouhé neznamínkové číslo. Každý nod ale očekává přesný počet bajtů, dle šířky dat typu, počtu dat typu a počtu kanálů. Metoda *set* tedy enkóduje tyto čísla do posloupnosti bajtů, a vloží je do paketu se zprávou MESSAGE_SET. Metoda *get* zase posloupnost bajtů dekóduje a převádí na unsigned int. Oba dva algoritmy jsou uvedeny v příloze.

7.5.1. Detekce nových nodů

Pokud je třeba detekovat nové nody, musí se pozastavit normální průběh plánování. Metoda *loop* při každém průchodu smyčky nastaví na začátku *bool confirmed* jako *false* a na konci smyčky jako true a pak se čeká milisekundu.

Když uživatel na centrálním serveru spustí detekci nových nodů, tak *Connector* přijme zprávu DISCOVER_NEW_NODES a přes *Delegator* volá metodu *discoverNewNodes*. Ta kontroluje stav proměnné *confirmed*, dokud není *false*. Pak nastaví proměnnou *halted* jako *true*. Poté plánovač přestává obsluhovat nody a pouze čeká na nastavení *halted* zpět na *false*.

Metoda *discoverNewNodes* pak odesílá nodům zprávu MESSAGE_DISCOVERY a čeká na zprávu MESSAGE_DISCOVERY_RESPONSE od neadresovaných nodů. Pokud žádná zprávy neobdrží, loguje přes *Delegator* chybu ERROR_NO_NEW_NODES. V případě že obdrží data, předá je jako parametr metodě *discoveryResponse* v instanci třídy *Message*. Ta přes ukazatel naplňuje neseřazenou mapu nových nodů, které se zdařilo identifikovat. Pokud byl nalezen alespoň jeden nod, je přes *Delegator* instanci třídy *Connector* odeslán seznam nových nodů zprávou NEW_NODES_FOUND. *Connector* pak čeká na zprávu NEW_ADRESSES a volá přes *Delegator* metodu plánovače *addressNewNodes* a předává jako parametr neseřazenou mapu nových nodů s přidělenou adresou. Pokud jsou detekovány jiná nesmyslná data na sběrnici, znamená to, že nebyly zachyceny všechny zprávy MESSAGE_DISCOVERY_RESPONSE a v subsystému je alespoň jeden nedetekovaný nod. Spolu s žádostí o adresu je logována chyba ERROR_PACKET_NONSENSE.

Metoda *addressNewNodes* pak odesílá nodům zprávu MESSAGE_DISCOVERY_ADDRESS s přidělenou adresou a čeká na zprávu MESSAGE_DISCOVERY_ADDRESS_CHECK. Tu dekóduje přes metodu instance třídy Message, a to *discoveryCheck*. Při úspěchu je zpět nodu odeslána zpráva pomocí instanční metody Message *makeDiscoveryCheckOk* a nod je úspěšně adresován. Celá metoda se opakuje pro všechny nové nody. Pokud některý z nich na novou adresu nereaguje, je vyšší vrstvě odeslána chyba ERROR_FAILED_TO_ADDRESS_NODES.

Po dokončení adresace je proměnná *halted* nastavena zpět na *false* a plánovač se vrací zpět k obsluze nodů.

7.6. Program nodů

Aby mohly navržené nody plnit svou činnost, musí být pro mikrokontrolér PIC16F690 implementován program, který jej ovládá. Program je psán v zjednodušené variantě jazyka C [**61**] ve vývojovém prostředí společnosti Mikroelektronika [**62**], MikroC for PIC [**63**]. MikroC používá svůj interní kompilátor.

Zdrojový kód každého nodu je odlišný, ale sdílí velkou řadu společných funkcí a parametrů. Mikrokontrolér po zapnutí začne vykonávat kód funkce *main*. V něm je nejprve nastaveno využití externího oscilátoru. Poté jsou konfigurovány piny a je nastaven jejich směr (vstupní pin nebo výstupní pin) dle typu nodu a počtu jeho kanálů. Pokud se využívají analogové vstupy, je nutné nakonfigurovat [**64**], které konkrétní se využívají. Dále je inicializován UART na rychlost 115200 kbaud/s Ukázky kódu jsou pro osmikanálový nod se spínači na vstupu. Jiný nod by měl piny konfigurované odlišně.

```
OSTS_bit = 0; //Externí oscilátor
TRISC = 255;  //PORT C je vstupní
TRISB.F6 = 0; //pin RB6 je výstupní
TRISB.F4 = 0; //pin RB4 je výstupní
TRISA = 0b11111011; //PORT A je vstupní, kromě pinu RA2
ANSEL  = 3; // Analogové vstupy AN0 a AN1
ANSELH = 0;

UART1_Init(115200);
```

Protože každý nod využívá jiný pin k řízení směru komunikace RS-485, je tento pin definován preprocesorovým makrem *#define RX_TX_BIT RB6_bit*. Dále jsou vynulovány globální proměnné *start_char* a *timeout*. Následně je zavolána funkce *init*. Ta čte z paměti EEPROM adresu nodu. Pokud nod adresu nemá (na obou bajtech paměti je hodnota 255) nastaví se globální proměnná *has_address* na nulu. Program tak ví, že nod není naadresován.

```
void init() {
     //předpokládáme, že nod nemá adresu
     has_address = 0;

     address_hi = EEPROM_Read(0x00); // čtení nulté adresy EEPROM
     address_lo = EEPROM_Read(0x01);
     address_subsystem = EEPROM_Read(0x02);
     if (address_hi != 0xFF && address_lo != 0xFF) {
        has_address = 1;
     }
}
```

Po ukončení kódu funkce *init* tedy nod ví, jestli již má adresu a podle toho se program větví. Dalším krokem v metodě *main* je povolení globálního přerušení, nastavení 16bitového časovače [**65**], LED diod a vynulování počítadla *count*.

```
GIE_bit = 1;  //povolení globálního přerušení
PEIE_bit = 1;  //povolení interního periferálního přerušení

T1CKPS1_bit = 1;  //nastavení flagů časovače
T1CKPS0_bit = 1;
T1OSCEN_bit = 1;
TMR1CS_bit = 0;
TMR1ON_bit = 1;
T1IE_bit = 1;
TMR1H = 0xD8;  //nastavení horní poloviny počítadla
TMR1L = 0xF0;  //nastavení dolní poloviny počítadla

COMM_LED = 0; //COMM_LED v tomto případě RB4_bit
STATUS_LED = 1; // RA2_bit

count = 0;
```

Tím končí úvodní inicializace mikrokontroléru a pokračuje se na cyklus *while (has_address == 0)*. Pokud má nod adresu, je tento cyklus přeskočen a pokračuje se do nekonečného cyklu *while(1)*, který slouží již pro běžnou obsluhu nodu. V adresačním cyklu se v každém průchodu inkrementuje *count* a při dosažení hodnoty 50000 je otočen stav STATUS_LED a *count* vynulován. Stavová LED dioda pak bliká přibližně dvakrát za sekundu, a tím indikuje, že nod čeká na přidělení adresy. Dále se kontroluje HW zásobník UART a vyhodnocují data.

```
if (UART1_Data_Ready() == 1) {  //jsou k dispozici bajty?
     TMR1H = 0xD8;
     TMR1L = 0xF0;
     TMR1ON_bit = 1;  //zapnutí časovače

     received_char = UART1_Read(); //čtení znaku

     //detekce zaváděcího znaku
     if ((received_char == '#') && start_char == 0)  {
          start_char = 1;
     }

     //vyhodnocení minimální délky dat pro validní paket
     if (start_char == 1 && received_index < 40) {
          received[received_index] = received_char;
          if (received_index > 3 && received_char == '$') {
               check_packet_no_address();
          }
          received_index++;
     }
}
```

Na začátku jsou proměnné *start_char* a *received_index* nulové. Zabudovaná funkce *UART1_Datat_Ready* vrací informaci o dostupném počtu bajtů. Jakmile je alespoň jeden k dispozici, je zapnut 16bitový časovač, který vyprší za 5 milisekund. Pokud do té doby není přijat validní paket, jsou počitadla i data v UART vynulovány a začíná se od začátku. Přijatý bajt se uloží do proměnné *received_char* metodou *UART1_Read*. Poté se kontroluje, zdali se jedná o zaváděcí znak paketu. Pokud je zachycen, tak je *start_char* nastaven na 1. Po zachycení je každý bajt uložen do pole *received* a je posunut index pole *received_index*. Když je splněna podmínka přijetí alespoň 5 bajtů (což je minimální možná délka paketu) a je zároveň zachycen ukončovací znak paketu, je volána funkce *check_packet_no_address*.

Funkce *check_packet_no_address* prochází pole přijatých bajtů a počítá kontrolu CRC8 funkcí *calculate_checksum*. Když vychází s číslem uvedeným v poli, je nalezen validní paket a vypnut časovač, aby pole při přerušení nepřemazal. Pak je kontrolováno, jestli zpráva v paketu patří právě tomuto nodu. Může akceptovat pouze tři zprávy. MESSAGE_DISCOVERY, MESSAGE_DISCOVERY_ADDRESS a MESSAGE_DISCOVERY_CHECK_OK. První MESSAGE_DISCOVERY zprávu odpovídá, pokud ještě nezačal adresační proces funkcí *discovery_response* odesílající paket se zprávou MESSAGE_DISCOVERY_RESPONSE. Ve funkci generuje náhodné číslo z analogových vstupů AN0 a AN1. Pro odeslání odpovědi se využívá funkce *send_packet_random*.

```
void send_packet_random (unsigned char * tx, short length,
unsigned int ran) {
     int index = 0;

     Delay_ms(10);
     for (index = 0; index < ran*5; index ++) {
         Delay_us(1);
     }
     RX_TX_BIT = 1;
     Delay_us(100);

     UART1_Write('#');
     UART1_Write(length);
     for (index = 0; index < length; index++) {
         UART1_Write(tx[index]);
     }
     UART1_Write(create_checksum(tx, length));
     UART1_Write('$');

     delay_time = length+4;
     us_delay(delay_time);
     RX_TX_BIT = 0;
}
```

Funkce *send_packet_random* přijímá tři parametry. Ukazatel na pole se zprávou, které se má odeslat. Počet bajtů zprávy a náhodné číslo pro délku čekání. Funkce pak vždy čeká 10 milisekund, pak v cyklu čeká tolik mikrosekund, kolik je velikost proměnné *ran**5. Následně odešle přes UART zaváděcí znak, délku vlastní dat, kontrolní součet, a ukončovací znak. Výsledek kontrolního součtu vrací jako návratovou hodnotu metoda *create_checksum*. Jako parametry potřebuje data a délku zprávy.

```
unsigned char create_checksum(unsigned char * t, short length){
       short index;

     unsigned char crc_value=0;
     crc_value = crc_tab[ (crc_value ^ length) ];

     for(index = 0; index < length; index++) {
          // provední XOR
     crc_value = crc_tab[(crc_value ^ t[index])];
}
```

Kontrolní součet se spočte iterací skrze pole s odesílanou zprávou. Pro každý bajt dat se provádí exkluzivní disjunkce [**66**] předchozí hodnoty s hodnotou uloženou v před počítaném poli *crc_tab*, které obsahuje 256 hodnot (jednu hodnotu pro každou hodnotu bajtu).

Pro všechny ostatní zprávy se na odesílání využívá funkce *send_packet*, která je téměř identická s funkcí *send_packet_random,* pouze nepřijímá náhodné číslo a data odesílá na sběrnici okamžitě bez čekání.

Po tomto procesu již má program náhodné číslo vygenerované a funkce *check_packet_no_address* čeká na zprávu MESSAGE_ADDRESS. Kontroluje, zdali odpovídá odeslané náhodné číslo, a pokud ano, tak volá funkci *discovery_address*. Ta si přijatou adresu uloží do operační paměti. Pak se volá funkce *discovery_check*, která sestaví odpověď MESSAGE_DISCOVERY_CHECK. Poté se čeká na přijetí zprávy DISCOVERY_CHECK_OK. Když náhodná čísla i adresy odpovídají předchozím hodnotám, je zavolána funkce *discovery_check_ok*. Ta ukládá adresu nodu a subsystému do EEPROM paměti a nastavuje *has_address* na 1. Proto je adresační smyčka ukončena a přechází se do nekonečné smyčky vyčítání stavu.

V nekonečné smyčce kontroluje program přijetí paketu stejným způsobem, ale při zachycení možného paketu nevolá funkci *check_packet_no_address* ale *check_packet_adddress*.

```
void check_packet_address() {
     if ((received[0] == '#') &&
         (received[received_index - 1] ==
         calculate_checksum(received, 1, received_index - 2)) &&
         (received[received_index] == '$')) {

         TMR1ON_bit = 0;
         TMR1H = 0xD8;
         TMR1L = 0xF0;
         start_char = 0;

         if ( received[2] == TYPE_REQUEST && //požadavek data
              received[3] == address_hi &&
              received[4] == address_lo &&
              received[5] == address_subsystem) {
                 get();
         }

         timeout = 1;
     }
}
```

Funkce *check_packet_address* také po validaci paketu vypíná časovač a kontroluje, komu zpráva patří. Pokud je to zpráva MESSAGE_REQUEST, což je jediná na kterou může nod se spínači v naadresovaném stavu odpovídat (jelikož je typu pouze pro čtení), volá se

funkce *get*. Ta funkcí *send_packet* odešle stav spínačů, tedy stav celého portu C. Příznak *timeout* způsobí nastavením na 1 vynulování indexů a UART v hlavní smyčce.

7.7. Optimalizace distribuce Raspbian

Subsystém využívá na Raspberry Pi 2 linuxovou distribuci Raspbian ve verzi wheezy. Protože se jedná o nejrozšířenější distribuci pro Raspberry Pi, obsahuje velké množství aplikací, programů a knihoven, které pro běh subsystému nejsou potřeba [**67**] a mohou zbytečně zvyšovat velikost systému a zabranou operační paměť. Příkladem budiž počítačová hra Minecraft, simulační prostředí Wolfram Alpha a NFS server pro sdílení dat[10].

Raspbian se neinstaluje jako operační systémy pro platformu x86. Stačí stáhnout ze stránek distribuce obraz systému ve formátu img, který se nahraje na paměťovou kartu. Je vyžadována microSD karta o velikosti alespoň 4GB. Z toho je po konfiguraci systému zabráno 3,2 GB. Po bootování je zabráno 120MB operační paměti. Po spuštění grafického desktopového prostředí LXDE je zabráno 640 MB operační paměti. Protože subsystém nebude pro uživatele přístupný, není desktopové prostředí ani X Window System potřeba, včetně veškerých grafických aplikací. Pro provoz subsystému je třeba pouze běhové prostředí jazyka Java, databázový server MySQL, ovladače síťové karty a periférií Raspberry Pi 2. Pro vývoj je ještě potřeba vývojové prostředí Java a také kompilátor GCC [**68**] ve verzi alespoň 4.8.

Po odstranění všech nepotřebných aplikací, programů a ovladačů (zvuková karta a ovladače bezdrátových karet) příkazy *apt-get remove* a *apt-get purge*, se obsazené místo microSD kartě snížilo o 2,2GB na 1GB. Zabraná operační paměť se odebráním nepotřebných ovladačů (zvukový systém, podpora WiFi technologid apod.) po nabootování snížila na 70 MB. K dispozici pro aplikace subsystému je tak zanecháno naprosto dostatečných 904 MB. Čas bootování se snížil z 50 sekund na 33 sekund. Všechny optimalizace tedy zlepšují parametry subsystému.

7.8. Testování systému

Protože při testování [**69**] subsystému nebyla k dispozici aplikace vyšších vrstev, byl napsán jednoduchý program v jazyce Java, který chování vyšších vrstev simuluje několika scénáři. Dokáže aplikaci přidělit statickou adresu subsystému a předpokládá několik

[10] Taková služba může dokonce představovat potencionální bezpečnostní hrozbu.

připojených nodů. Jmenovitě jeden s tlačítky, jeden se spínači, dva světelné nody a jeden nod s regulátory. Tyto nody jsou všechny osmikanálové, ale u nodu s regulátory byl využit pouze jeden kanál. Dalším nodem připojeným k subsystému je jednokanálový ovladač krokového motoru. Testovací program poté vykonává přesně definovaná pravidla. Stisk tlačítek nodu vyvolá obrácení změnu stavu prvního světelného nodu. Světla na druhém nodu svítí podle stavu spínačů a motor se otočí podle otáčení regulátoru. Testovací program a aplikace byly na Raspbery Pi 2 spuštěny po dobu dvou týdnů. Během této doby bylo monitorováno okamžité procentuální využití procesoru a zabraná operační paměť aplikace. S tlačítky, spínači a regulátorem se náhodně manipulovalo. Hodnoty byly zaznamenávány každých 5 sekund a ukládány do souboru *logfile* následujícím bash [**70**] skriptem:

```
while true //nekonečná smyčka
do
    echo "$(date '+TIME:%H:%M:%S')
          $(ps -C Hausy -o %cpu,%mem)" | tee -a logfile
    sleep 5
done
```

Data byla analyzována a ukázalo se, že aplikace nikdy nevyužívala více jak 9 procent procesorového času a zabraná operační paměť nepřekročí 0,5%, tedy přibližně 40 MB operační paměti. Pro aplikaci vyšší vrstvy zůstane k dispozici přibližně 864 MB volné operační paměti.

7.8.1. Testování adresace nodů

Protože adresace využívá generování náhodného čísla, může nastat situace, kdy dva a více nodů vygenerují stejné náhodné číslo a vyšlou data na sběrnici současně. Subsystém je pak vyhodnotí jako nesmyslný paket a nodům adresu nepřidělí. Proces adresace se pak musí opakovat ještě jednou.

Testovací program obsahuje druhý scénář, kdy je k subsystému připojeno deset nodů, s typem světlo[11]. V programu byly implementovány nezbytné funkce pro adresaci nodů a byl proveden následující test. V programu byla spuštěna funkce pro adresaci nodů a bylo zjišťováno, kolik nodů se podaří v jednom kroku úspěšně adresovat. Byl zaznamenán počet kroků a všem nodům byla odeslána resetovací zpráva MESSAGE_RESET. Celý proces byl zopakován 50x.

[11] U tohoto testu na typu nodu nezáleží, protože adresační algoritmus je ve všech implementovaných prototypech stejný.

Výsledkem bylo, že v případě 47 testů se podařilo všechny nody úspěšně adresovat v jediném kroku a ve zbývajících 3 testech bylo nutné adresaci opakovat. V prvním případě se neadresovali dva nody, ve druhém také dva a ve třetím pouze jeden. Protože adresace nových nodů není běžná činnost a probíhá pouze při prvním zapojení celého domu a poté při rozšiřování domu o nové nody, lze takový výsledek považovat za velmi uspokojivý. Celý proces adresace trvá pouze několik sekund.

8. ZÁVĚR

Cílem práce bylo implementovat část nově navrženého systému pro řízení inteligentního domu. Byla navrhnuta a naprogramována aplikace nižší vrstvy subsystému v programovacím jazyce C++. Náročnost aplikace na hardwarové prostředky je velice nízká. Testováním se ověřilo, že aplikace na počítači Raspberry Pi 2 nikdy nealokuje více jak 40 MB operační paměti a celkem nevytíží procesor více jak na 9 procent. Díky tomu je přenechána značná část hardwarových prostředků pro vyšší vrstvy subsystému.

Prototypy vyrobených nodů je možné pouze výměnou programu jejich mikrokontrolérů využít ke změně typu nodu a připojit k nim odlišné periférie. Programy jednotlivých typů nodů mají většinu funkcí společných a proto je možné implementovat program pro nový typ velice rychle. Prototyp vstupního nodu může dle programu obsluhovat tlačítka, spínače, pohybové senzory, dveřní kontakty, kouřové senzory, senzory zaplavení a desítky dalších periférií. Výstupním nodem lze spínat světelné obvody, spotřebiče v domě, spouštět zalévání zahrady a podobně. Analogovým node lze číst stavy nástěnných regulátorů, informace o teplotě, vlhkosti, kvalitě ovzduší a tak dále. Motorovým nodem lze řídit krokové motory s ty mohou například otáčet roletami v domě.

Celý systém je v budoucnu možné rozšiřovat v několika směrech. Protože nody komunikují se subsystémem přes RS-485 relativně nízkou rychlostí 115200 kbit/s, nejsou vhodné pro přenos audio a video záznamů. Dalším krokem tedy může být návrh a implementace nodů využívajících rychlejší komunikační standart. Konkrétními typu nodů pak mohou být kamery, přehrávače zvuku nebo například obrazovky. Může se také směřovat k vývoji bezdrátových variant nodů, které budou napájeny z baterie a tím zjednoduší instalaci v domě. U stávajících nodů je možné se zaměřit na implementaci SMD variant plošných spojů a tím je dostatečně zmenšit jejich velikost a umožnit zabudování přímo v domě. S dostatečnými finančními prostředky je také možné navrhnout nový počítač na platformě ARM, který by nahradil stávající počítač Raspberry Pi 2 doplněný o shield za specifické řešení přímo pro subsystém.

9. REFERENCE

1. DAE-MAN, H. a L. JAE-HYUN. Design and implementation of smart home energy management systems based on zigbee. *Consumer Electronics* [online]. IEEE, 2010, č. 56, s. 1417 - 1425 [cit. 2014-Leden-28]. ISSN 0098-3063. Dostupné z: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5606278&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5606278

2. ZIGBEE ALLIANCE. Smart Homes. *The ZigBee Alliance* [online]. 2014 [cit. 2014-Březen-05]. Dostupné z: http://www.zigbee.org/what-is-zigbee/494-2/

3. RASHIDI, P. a C. D.J. Keeping the Resident in the Loop: Adapting the Smart Home to the User. *Systems, Man and Cybernetics, Part A: Systems and Humans* [online]. IEEE, 2009, č. 39, s. 949 - 959 [cit. 26-Leden-2014]. ISSN 1083-4427. Dostupné z: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5196706&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5196706

4. FOROUGHI, H. B. S. ASKI a H. POURREZA. Intelligent video surveillance for monitoring fall detection of elderly in home environments. In: IEEE. *11th International Conference on Computer and Information Technology* [online]. 1st. Khulna: IEEE, 2008, s. 219 - 224 [cit. 2014-Leden-18]. ISBN 978-1-4244-2136-7. Dostupné z: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4803020&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D4803020

5. Fibaro. *Systém Fibaro* [online]. 2014 [cit. 2014-Únor-13]. Dostupné z: http://www.fibaro.com/cz/Fibaro-syst%C3%A9m

6. ČESKO. Vyhl. č. 50/1978 Sb. Vyhláška Českého úřadu bezpečnosti práce a Českého báňského úřadu o odborné způsobilosti v …. 1978.

7. ABB S.R.O. *Ego-n* [online]. 2014 [cit. 2014-Leden-13]. Dostupné z: http://www117.abb.com/index.asp?thema=8920

8. ELKO EP. iNELS Bus system. *iNELS* [online]. 2015 [cit. 2015-Únor-15]. Dostupné z: http://www.inels.cz/produkty/inels-bus-system

9. LOXONE S.R.O. Loxone. *Loxone* [online]. 2014 [cit. 2015-Leden-11]. Dostupné z: http://www.loxone.com/cscz/start.html

10. ORACLE. *Java SE* [online]. 2014 [cit. 2014-Duben-17]. Dostupné z: http://www.oracle.com/technetwork/java/javase/overview/index.html

11. PIVOTAL SOFTWARE. *Spring* [online]. 2014 [cit. 2014-Duben-30]. Dostupné z: http://spring.io/

12. ORACLE. *MySQL* [online]. 2014 [cit. 2014-Únor-17]. Dostupné z: https://www.mysql.com/

13. RICHARDSON, L. M. AMUNDSEN a S. RUBY. *RESTful Web APIs*. O'Reilly Media, 2013. ISBN 978-1-4493-5805-1.

14. ANTOŠOVÁ, M. a V. DAVÍDEK. *Číslicová technika*. České Budějovice: Kopp, 2007. ISBN 978-80-7232-314-2.

15. ECMA INTERNATIONAL. ECMA-404. In: *The JSON Data Interchange Format* [online]. 2013 [cit. 2014-Červen-12]. Dostupné z: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

16. LIBERTY, J. *Naučte se C++ za 21 dní*. Praha: Computer Press, 2002. ISBN 80-7226-774-4.

17. W. KERNIGHAN, B. a D. M. RITCHIE. *Programovací jazyk C*. Bron: Computer Press, 2013. ISBN 9788025108970.

[18] FREE SOFTWARE FOUNDATION. *GNU/Linux* [online]. 2014 [cit. 2014-Leden-18]. Dostupné z: https://www.gnu.org/

[19] IEEE. IEEE 802.3 Ethernet. *IEEE Standarts Association* [online]. 2013, verze 802.3bk [cit. 2014-Březen-08]. Dostupné z: https://standards.ieee.org/about/get/802/802.3.html

[20] CAN IN AUTOMATION E.V. *Controller Area Network* [online]. 2014 [cit. 2014-Květen-04]. Dostupné z: http://www.can-cia.org/index.php?id=can

[21] THE RS485 COMMUNICATIONS AUTHORITY. The RS485. In: *QUICK REFERENCE FOR RS485, RS422, RS232 AND RS423* [online]. 2013 [cit. 2014-Srpen-11]. Dostupné z: http://www.rs485.com/rs485spec.html

[22] STŘELEC, J. a M. LÍŠKA. *Architektury procesorů RISC*. Praha: Grada Publishing, 1992. ISBN 80-85424-75-4.

[23] ARM The Architecture for the Digital World. *Procesor Licensees* [online]. 2014 [cit. 2014-Květen-25]. Dostupné z: http://www.arm.com/products/processors/licensees.php

[24] Microchip. *All 8-bit Microcontrollers Products* [online]. 2015 [cit. 2015-Duben-18]. Dostupné z: http://www.microchip.com/ParamChartSearch/Chart.aspx?branchID=1012

[25] 8-bit PIC Microcontrollers. *PIC10F200* [online]. 2014 [cit. 2014-Duben-29]. Dostupné z: http://www.microchip.com/wwwproducts/Devices.aspx?product=PIC10F200

[26] 8-bit PIC Microcontrollers. *PIC12F617* [online]. 2014 [cit. 2014-Duben-29]. Dostupné z: http://www.microchip.com/wwwproducts/Devices.aspx?product=PIC12F617

[27] 8-bit PIC Microcontrollers. *PIC16F690* [online]. 2014 [cit. 2014-Duben-29]. Dostupné z: http://www.microchip.com/wwwproducts/Devices.aspx?product=PIC16F690

[28] 8-bit PIC Microcontrollers. *PIC16F767* [online]. 2014 [cit. 2014-Duben-29]. Dostupné z: http://www.microchip.com/wwwproducts/Devices.aspx?product=PIC16F767

[29] 8-bit PIC Microcontrollers. *PIC18F86J50* [online]. 2014 [cit. 2014-Duben-29]. Dostupné z: http://www.microchip.com/wwwproducts/Devices.aspx?product=PIC18F86J50

[30] GOFTON W. P. *Sériová komunikace*. Praha: Grada Publishing, 1995. ISBN 80-7169-131-3.

[31] MAXIM INTEGRATED. Maxim Integrated. In: *MAX481/MAX483/MAX485/MAX487-MAX491/MAX1487* [online]. 2014 [cit. 2014-Říjen-11]. Dostupné z: http://datasheets.maximintegrated.com/en/ds/MAX1487-MAX491.pdf

[32] HRBÁČEK, J. *Komunikace mikrokontroléru s okolím 1.* Praha: BEN, 1999. ISBN 80-86056-42-2.

[33] HRBÁČEK, J. *Komunikace mikrokontroléru s okolím 2.* Praha: BEN, 2000. ISBN 80-86056-73-2.

[34] Electronics Calculator. *Microchip PIC Baud Rate Generator calculator* [online]. 2012 [cit. 2014-Listopad-18]. Dostupné z: Microchip PIC Baud Rate Generator calculator

[35] Raspberry Pi. *RASPBERRY PI 2 MODEL B* [online]. 2015 [cit. 2015-Březen-02]. Dostupné z: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

[36] Raspberry Pi. *RASPBERRY PI 1 MODEL B+* [online]. 2014 [cit. 2014-Květen-25]. Dostupné z: https://www.raspberrypi.org/products/model-b-plus/

[37] DEBIAN PROJECT. *Raspbian* [online]. 2014 [cit. 2014-Březen-10]. Dostupné z: http://www.raspbian.org/

[38] SOFTWARE IN THE PUBLIC INTEREST, INC. *Debian* [online]. 2014 [cit. 2014-Březen-06]. Dostupné z: https://www.debian.org/

[39] BeagleBone. *BeagleBone Black* [online]. 2013 [cit. 2014-Květen-25]. Dostupné z: http://beagleboard.org/BLACK

[40] ODROID. *ODROID-U3* [online]. 2013 [cit. 2014-Květen-25]. Dostupné z: http://www.hardkernel.com/main/products/prdt_info.php?g_code=g138745696275

[41] KŘIŠŤAN, L. a V. VACHALA. *Příručka pro navrhování elektronických obvodů*. Praha: SNTL, 1982.

[42] MATOUŠEK, D. *C pro mikrokontroléry PIC*. Praha: BEN, 2011. ISBN 978-80-7300-413-2.

[43] DUNCAN, M. *Learning to fly with EAGLE V6*. Elektor Publishing, 2013. ISBN 978-1907920202.

[44] PLIVA, Z. *EAGLE prakticky*. Praha: BEN, 2007. ISBN 978-80-7300-227-5.

[45] TEXAS INSTRUMENTS. Farnell. In: *LM2576* [online]. 2013 [cit. 2014-Listopad-15]. Dostupné z: http://www.farnell.com/datasheets/1849802.pdf

[46] VOBECKÝ, J. a V. ZÁHLAVA. *ELEKTRONIKA součástky o obvody*. Praha: Grada Publishing, 2000. ISBN 80-247-9062-9.

[47] HAVLÍČEK, V. M. POKORNÝ a I. ZEMÁNEK. *Elektrické obvody 1*. Praha: Nakladatelstvá ČVUT, 2005. 80-01-03299-X.

[48] FAIRCHILD. Fairchild. In: *BC546/BC547/BC548/BC549/BC550 NPN Epitaxial Silicon Transistor* [online]. 2002 [cit. 2014-Květen-15]. Dostupné z: https://www.fairchildsemi.com/datasheets/BC/BC547.pdf

[49] MICROCHIP. PICkit 3 In-Circuit Debugger. *Microchip* [online]. 2011 [cit. 2014-Duben-13]. Dostupné z: http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=PG164130

[50] FAIRCHILD. Fairchild. In: *BC556/BC557/BC558/BC559/BC560 PNP Epitaxial Silicon Tranzistor* [online]. 2002 [cit. 2014-Květen-15]. Dostupné z: https://www.fairchildsemi.com/datasheets/BC/BC557.pdf

[51] ON SEMICONDUCTOR. MC3749 Stepper Motor Driver. *ON Semiconductor* [online]. 2006, verze Rev. 7 [cit. 2014-Srpen-02]. Dostupné z: http://www.braude.ac.il/files/departments/electrical_electronic_engineering/labs/data_pages/MC3479.pdf

[52] MODBUS ORGANIZATION. *Modbus* [online]. 2014 [cit. 2014-Červenec-10]. Dostupné z: http://www.modbus.org/

[53] PROFIBUS & PROFINET INTERNATIONA. *Profibus* [online]. 2014 [cit. 2014-Červenec-10]. Dostupné z: http://www.profibus.com/

[54] ARLOW, J. a I. NEUSTADT. *UML2 a unifikovaný proces vývoje aplikací*. Brno: Computer Press, 2008. ISBN 978-80-251-1503-9.

[55] DEVELOPERS, O. JsonCpp. *GitHub* [online]. 2014 [cit. 2014-Červen-02]. Dostupné z: https://github.com/open-source-parsers/jsoncpp

[56] JANSA, L. a P. OTEVŘEL. *Softwarové právo*. Brno: Computer Press, 2011. ISBN 978-80-251-3458-0.

[57] WIKIMEDIA FOUNDATION INC. INI file. *Wikipedia: The Free Encyclopedia* [online]. 2014 [cit. 2014-Prosinec-13]. Dostupné z: http://en.wikipedia.org/wiki/INI_file

[58] DEVELOPERS, O. simpleINI. *GitHub* [online]. 2013 [cit. 2014-Červenec-03]. Dostupné z: https://github.com/brofield/simpleini

[59] PRATA, S. *Mistrovství v C++*. Brno: Computer Press, 2013. ISBN 9788025138281.

[60] CPLUSPLUS. std:unordered_map. *C++ Reference* [online]. 2013 [cit. 2014-Prosinec-11]. Dostupné z: http://www.cplusplus.com/reference/unordered_map/unordered_map/

[61] MANN, B. *C pro mikrokontroléry*. Praha: BEN, 2003. ISBN 80-7300-077-6.

[62] MIKROELEKTRONIKA. *MikroElektronika Embedded Solutions* [online]. 2014 [cit. 2014-Duben-25]. Dostupné z: http://www.mikroe.com/

[63] MIKROELEKTRONIKA. MikroC PRO for PIC. *MikroElektronika Embedded Solutions* [online]. 2014 [cit. 2014-Duben-25]. Dostupné z: http://www.mikroe.com/mikroc/pic/

[64] HRBÁČEK, J. *Moderní učebnice programování PIC 1. díl*. Praha: BEN, 2004. ISBN 80-7300-136-5.

[65] HRBÁČEK, J. *Moderní učebnice programování PIC 2. díl*. Praha: BEN, 2007. ISBN 978-80-7300-137-7.

[66] WIKIMEDIA FOUNDATION INC. Exkluzivní disjunkce. *Wikipedia: The Free Encyclopedia* [online]. 2015 [cit. 2015-Březen-01]. Dostupné z: http://cs.wikipedia.org/wiki/Exkluzivn%C3%AD_disjunkce

[67] NEGUS, C. a C. BRESNAHAN. *Linux Bible*. 8th edition. San Francisco: John Wiley & Sons, 2012. ISBN 978-1-118-21854-9. Dostupné také z: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-111821854X.html

[68] FREE SOFTWARE FOUNDATION. *GCC, the GNU Compiler Collection* [online]. 2015 [cit. 2015-Březen-14]. Dostupné z: https://gcc.gnu.org/

[69] SELECKÝ, M. *Penetrační testy a exploitace*. Brno: Computer Press, 2012. ISBN 978-80-251-3752-9.

[70] FREE SOFTWARE FOUNDATION. *GNU Bash* [online]. 2015 [cit. 2015-Duben-11]. Dostupné z: https://www.gnu.org/software/bash/bash.html

Příloha A – Obsah přiloženého CD

Přiložené CD obsahuje zkomprimovaný soubor prilohy.zip s následujícími složkami:

- bin
- dp
- plosne_spoje
- schemata
- src

Složka bin obsahuje podsložky nody a subsystem. Ve složce nody jsou zkompilované programy pro vybrané nody ve formátu HEX. Složka subsystem obsahuje zkompilovanou aplikaci C++ pro Rasbperry Pi 2 (platforma armhf). Ve složce src jsou také podložky nody a subsystem. Složka nody obsahuje projekty vývojového prostředí MikroC for PIC pro vybrané nody. Složka src obsahuje knihovny, hlavičkové soubory a soubory tříd aplikace. Je možné ji zkompilovat na libovolné Linuxové distribuci s kompilátorem C++ s podporou specifikace C++11 a nainstalovaným nástrojem make.

Složka plosne_spoje obsahuje výkresy plošných spojů. Ve složce schemata jsou pak umístěna schématická zapojení pro jednotlivé obvody.

Diplomová práce ve formátu pdf je ve složce dp.

Příloha B – Algoritmy pro enkódování a dekódování dat

Enkódování dat

Tento algoritmus je uvnitř metody makeSet, která vrací délku paketu k odeslání a přijímá parametry ukazatel na pole odesílaných dat txbuf a instanci třídy Node node, pro kterou je paket sestavován.

```
//Vyšší bajt adresy nodu
unsigned char addressHigher = (unsigned char)
                    (node.getAddress() >> 8);

//Nižší bajt adresy nodu
unsigned char addressLower = (unsigned char)
                     (node.getAddress() & 0xff);

txbuf[0] = START_CHAR; //zaváděcí znak paketu
txbuf[1] = 4 + node.getDataLength(); //délka dat
txbuf[2] = MESSAGE_SET; //identifikátor zprávy
txbuf[3] = addressHigher; //vyšší bajt adresy nodu
txbuf[4] = addressLower; //nižší bajt adresy nodu
txbuf[5] = address; //bajt s adresou subsystému

//data nodu, tedy pole polí
unsigned int **data = node.getData();

//informace o počtu kanálů nodu
int channelCount = node.getChannelCount();

//informace o počtu dat
int dataCount = node.getType().getDataCount();

//informace o šířce dat
int dataRange = node.getType().getDataRange();

//aktuální hodnota nového bajtu dat
unsigned char byteValue = 0;

//index bajtů
unsigned char byteIndex = 0;

//hodnota bitu
unsigned char bitValue = 0;

//index bitů v bajtu
unsigned char bitIndex = 0;

//data pro aktuální kanál nodu
unsigned int currentData = 0;
```

```
//iterace skrze všechny kanály
for (unsigned char channel = 0; channel < channelCount; channel++)
{
      //iterace přes všechy data
      for (unsigned char count = 0; count < dataCount; count++) {

            currentData = data[channel][count];

            //iterace jednotlivých bitů v datech
            for (int range = 0; range < dataRange; range++) {

                  //získání hodnoty bitu
                  bitValue = ((currentData >> range) & 0x01);

                  //vložení bitu do aktuálního bajtu
                  if (bitValue == 0) {
                        byteValue &= ~(0x01 << bitIndex);
                  } else {
                        byteValue |= 0x01 << bitIndex;
                  }

                  //přiřazení aktuální hodnoty do odesílaného pole
                  txbuf[byteIndex + 6] = byteValue;

                  bitIndex++;

                  //kontrola přetečení
                  if (bitIndex == 8) {
                        bitIndex = 0;
                        byteIndex++;
                        byteValue = 0;
                  }
            }

      }

}

//výpočet CRC8
txbuf[7 + byteIndex] =
      calculateChecksum(txbuf, 1, (6 + byteIndex));

//vložení ukočovacího znaku
txbuf[8 + byteIndex] = END_CHAR;

//vrácení délky paketu
return 9 + byteIndex;
```

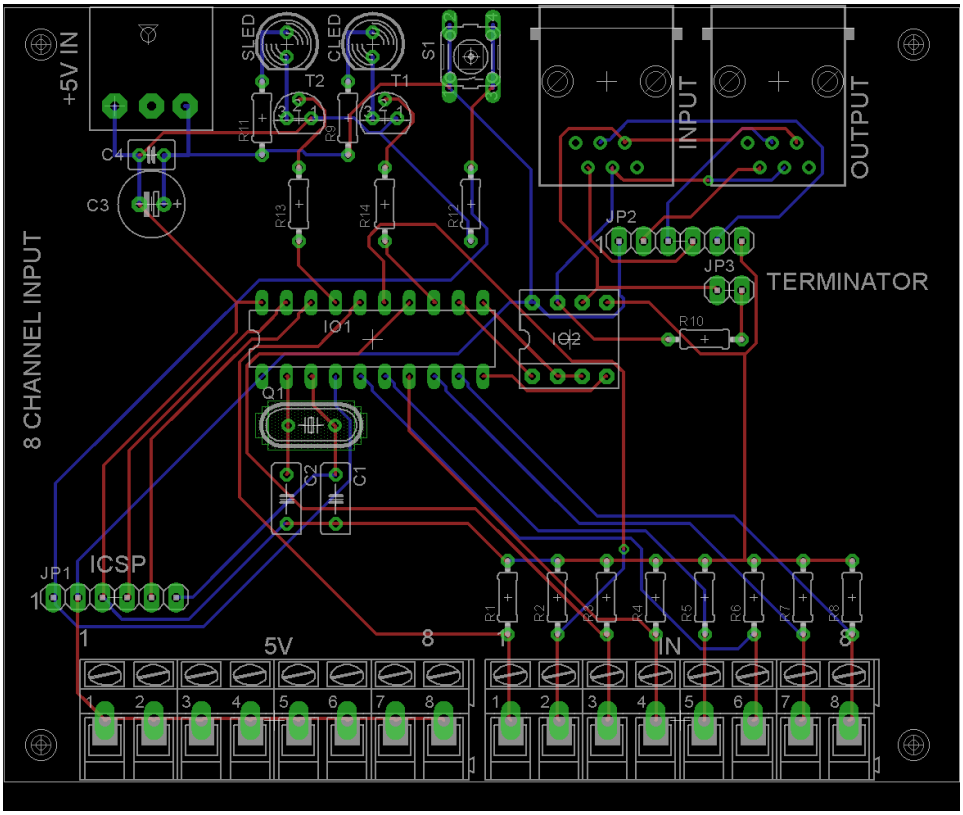
Dekódování dat

Algoritmus pro dekódování dat vrací celočíselnou proměnnou change, kde hodnota -1 znační chybný, nebo nekopletní paket. Hodnota 0 pak ukazuje, že nedošlo ke změně stavu oproti předchozímu vyčítaní, hodnota 1 pak ukazuje na změnu dat nebo stavu nodu. Metoda požaduje předat ukazatel na přijaté pole dat, délku přijatých dat a nod.

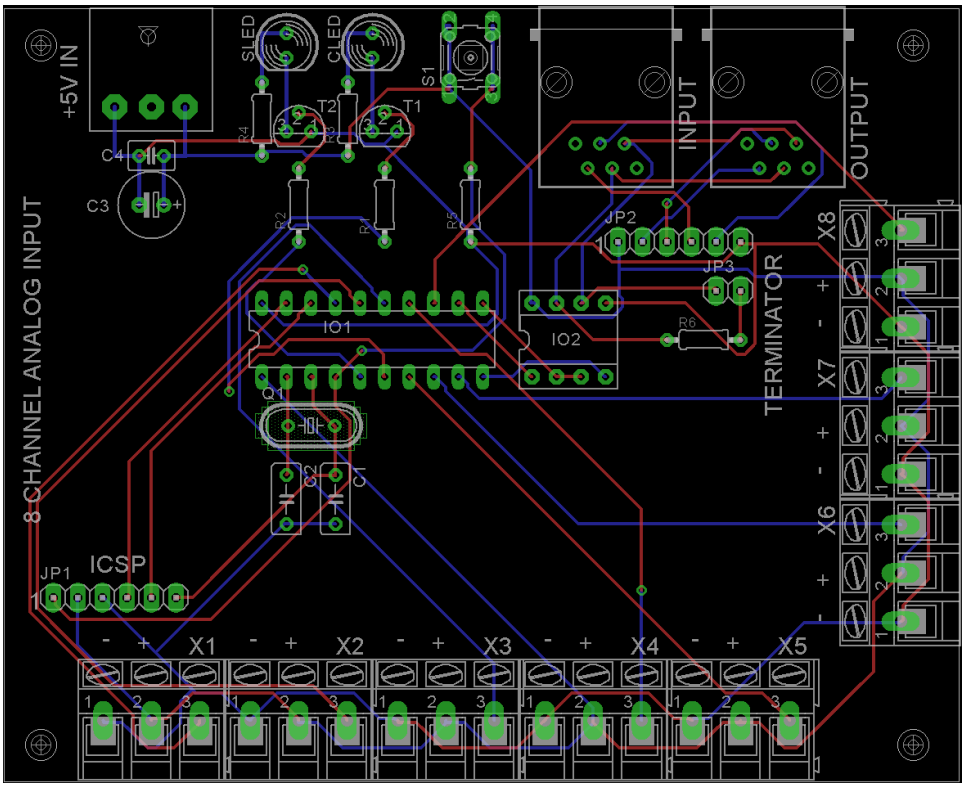
```
int change = -1; //výchozím stavem je chyba
unsigned char crc8 = 0; //vynulování crc
unsigned char length = node.getDataLength(); //délka dat
unsigned int ** data; //nová data nodu

//iterace všech přijatých bajtů
for (int i = 0; i < n; i++) {

      //výpočet crc
      crc8 = calculateChecksum(rxbuf, i + 1, i + length + 5);

      //hledání validního paketu
      if (rxbuf[i] == START_CHAR && rxbuf[i + 1] == (4+length)
                  && rxbuf[i + 2] == MESSAGE_GET
                  && rxbuf[i + length + 7] == END_CHAR
                  && rxbuf[i + length + 6] == crc8) {

            //předpokládáme stejná data
            change = 0;

            //získání počtu kanálů, počtu dat a šířky dat
            unsigned char channelCount = node.getChannelCount();
            unsigned char dataCount = node.getType().getDataCount();
            unsigned char dataRange = node.getType().getDataRange();

            //vynulování nové hodnoty dat
            unsigned int intValue = 0;

            //initializace indexu hodnoty, hodnoty bitu, indexu a
            //dat bajtu
            unsigned char intIndex = 0;
            unsigned char bitValue = 0; //hodnota aktualniho bitu
            unsigned char bitIndex = 0;
            unsigned char byteIndex = 0;
            unsigned char byteValue;

            //předchozí hodnota dat
            data = node.getData();

            //první bajt přijatých dat
            byteValue = rxbuf[i+6];

            //iterace všech kanálů
            for (unsigned char channel = 0; channel < channelCount;
                                          channel++) {
                  //iterace všech dat
```

```
                for (unsigned char count = 0; count < dataCount;
                                                    count++) {
                    //nulování hodnoty a indexu
                    intValue = 0;
                    intIndex = 0;

                    //iterace bitů dat
                    for (unsigned char range = 0; range <
                                            dataRange; range++) {

                        //získání bitu z paketu
                        bitValue = (byteValue >> bitIndex) & 0x01;

                        //přesun bitu z paketu do dat nodu
                        if (bitValue == 0) {
                            intValue &= ~(0x01 << intIndex);
                        } else {
                            intValue |= 0x01 << intIndex;
                        }
                        intIndex++;

                        //kontrola přetečení
                        bitIndex++;
                        if (bitIndex == 8) {
                            bitIndex = 0;
                            byteIndex++;
                            byteValue = rxbuf[i + 6 + byteIndex];
                        }
                    }

                    //pokud se jakákoliv data odlišují, nastavení
                    //změny
                    if (data[channel][count] != intValue) {
                        change = 1;
                    }

                    data[channel][count] = intValue;
                }
            }
        }
}

//při změně dat se mění data přímo v nodu
if (change == 1) {
     node.setData(data);
}

//vrácení chyby -1, žádných změn 0 nebo změny 1
return change;
```

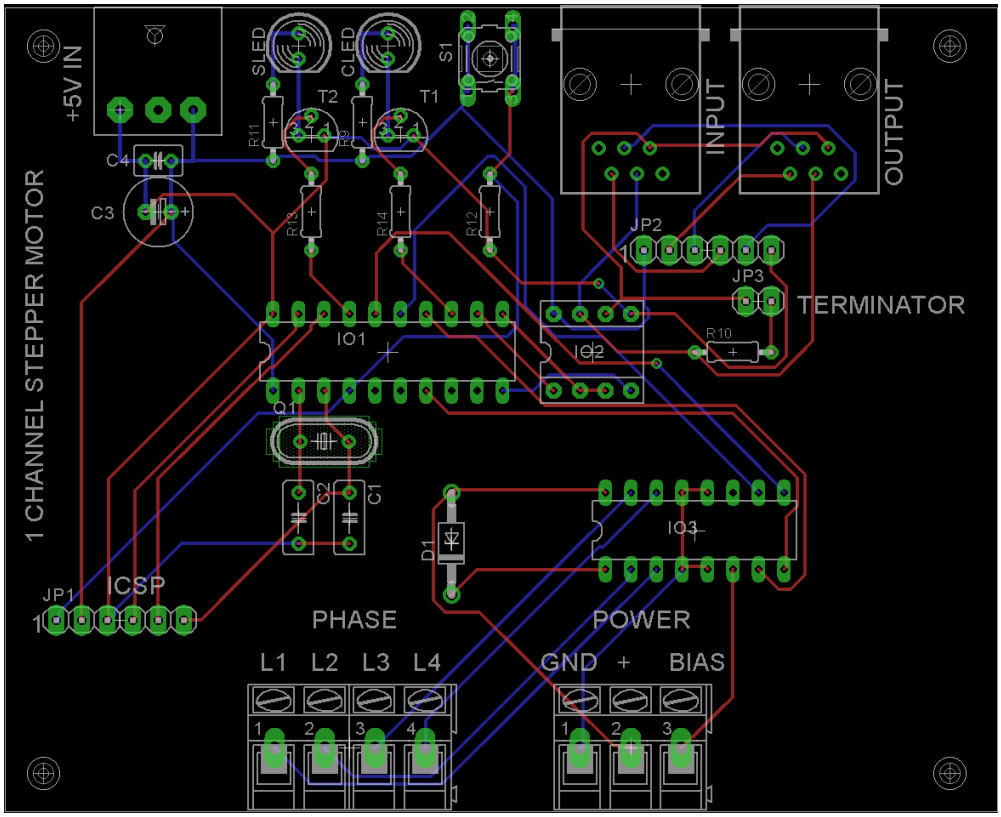
Příloha C – Výkresy plošných spojů



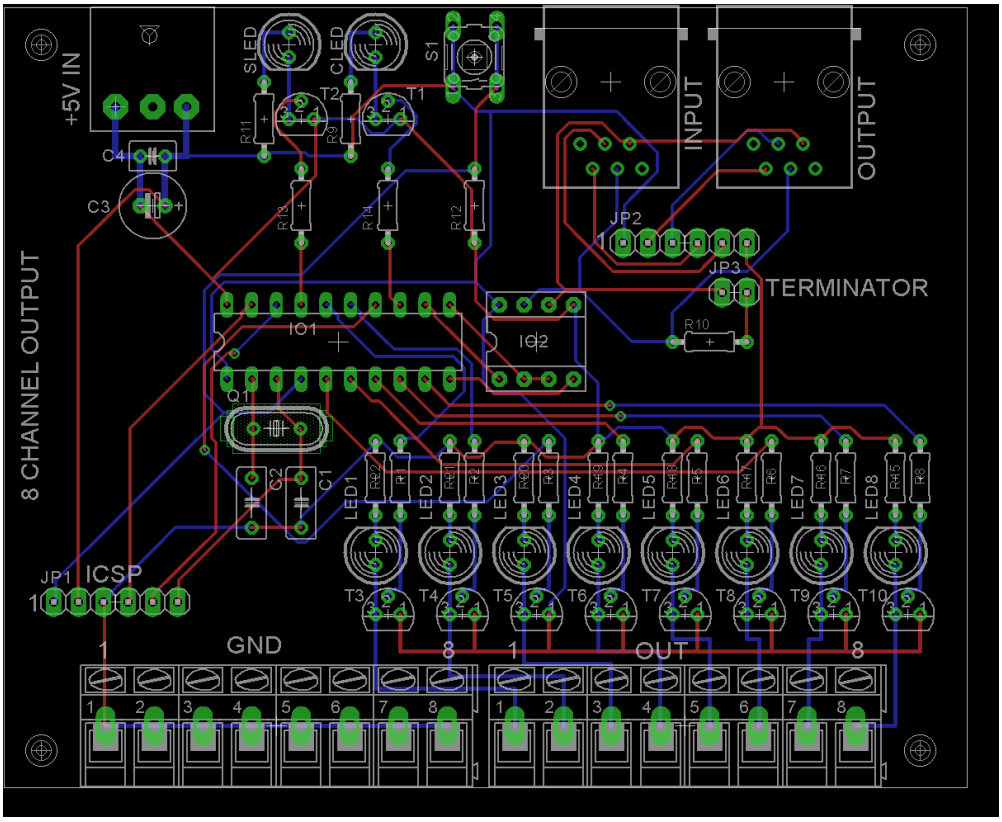
Obrázek 19 - Plošný spoj vstupního nodu [autor]



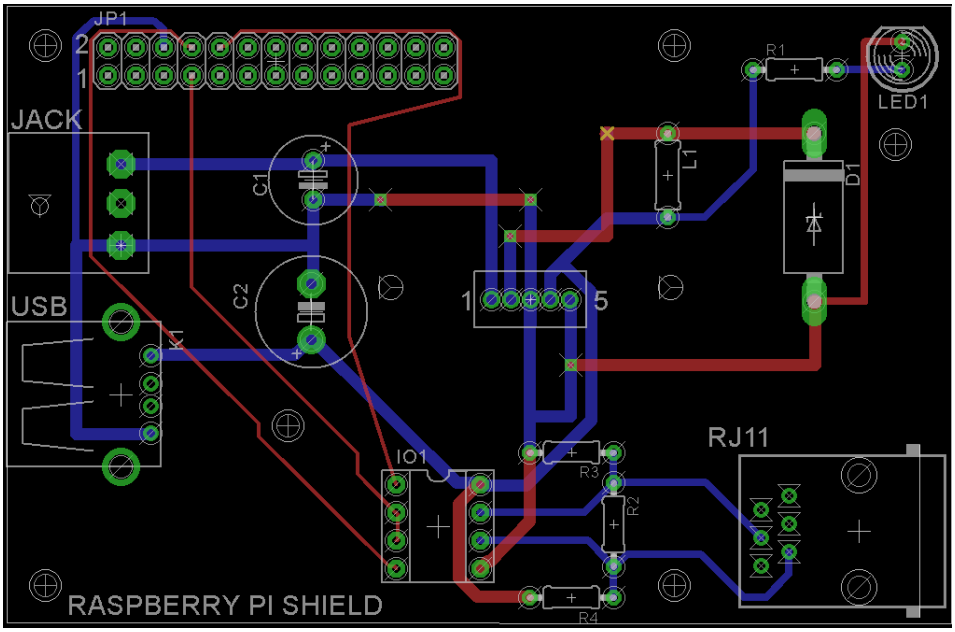
Obrázek 20 - Plošný spoj analogového nodu [autor]



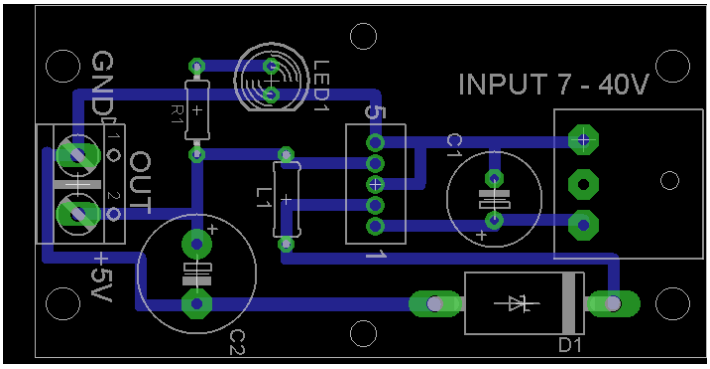
Obrázek 21 - Plošný spoj motorového nodu [autor]



Obrázek 22 - Plošný spoj výstupního nodu [autor]



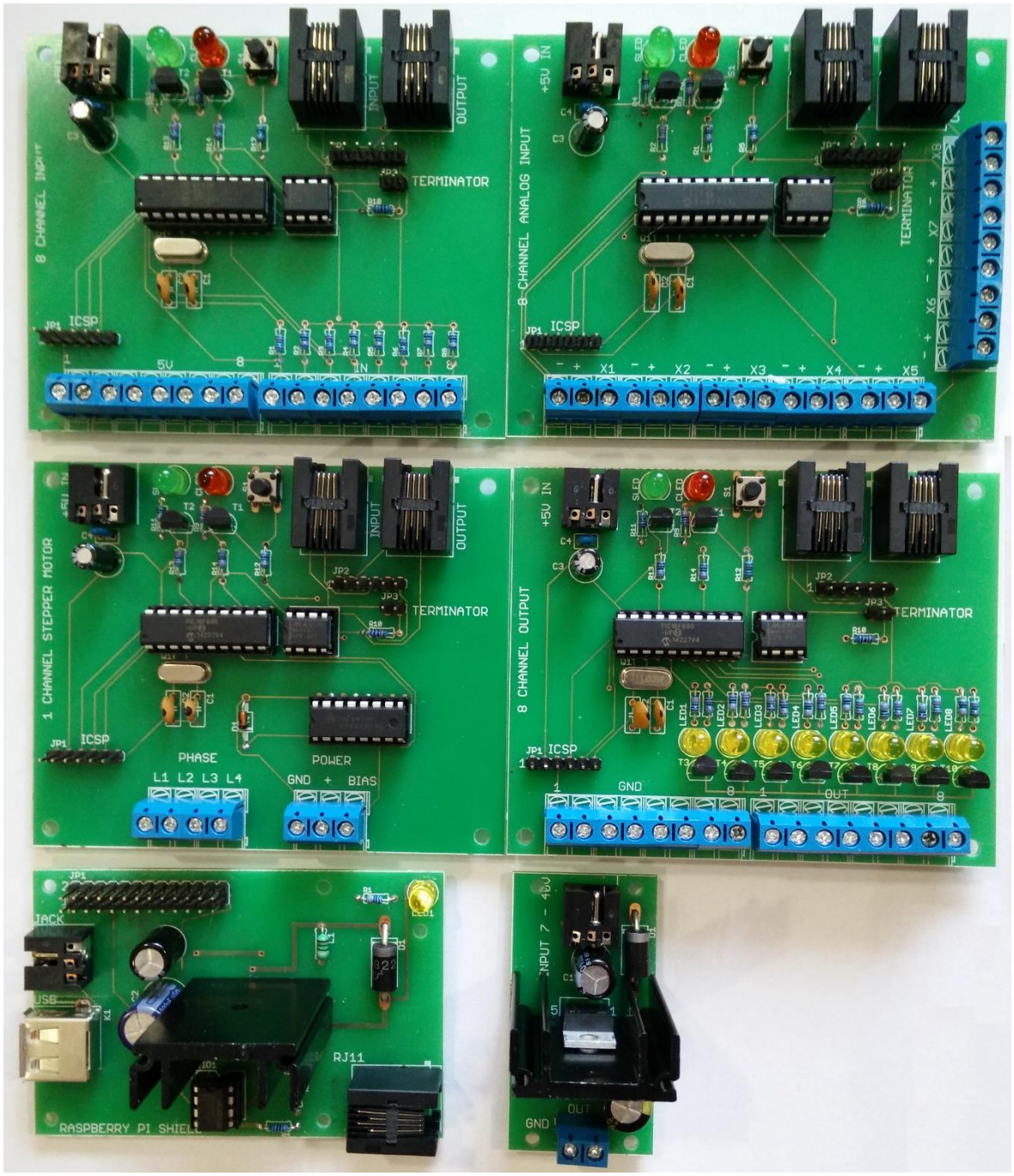
Obrázek 23 - Plošný spoj pro Raspberry Pi shield [autor]



Obrázek 24 - Plošný spoj napájecího zdroje [autor]

Všechny nody mají rozměr 100 mm x 80 mm. Shield pro Raspberry Pi má rozměry 86 mm x 56 mm. Rozměry napájecího zdroje jsou 57 mm x 30 mm.

Příloha D – Fotografie osazených plošných spojů



Obrázek 25 - Fotografie osazených plošných spojů [autor]

FIM UHK
UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Jan Štěpán**

Obor studia: Aplikovaná informatika (2)
Jméno a příjmení vedoucího práce: **Josef Horálek**

Název práce:
Návrh a implementace HW a SW smart zařízení pro komunikaci v inteligentním domě

Název práce v AJ:
Design and implementation of HW and SW for smart devices communication in an intelligent house

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Cílem práce je navrhnout a realizovat část smart zařízení pro komunikaci v inteligentním domě na urovni komunikující s hardwarovou platformou v rámci jednotlivých nodů a subsystémů.

Osnova práce:
Úvod
Rešerše konkurenčních přístupů
Představení komplexního modelu systému
Logický návrh nodů
Logický návrh subsystémů
Výběr vhodných HW platforem
Představení realizovaného řešení
Závěr

Projednáno dne:

Podpis studenta

Podpis vedoucího práce