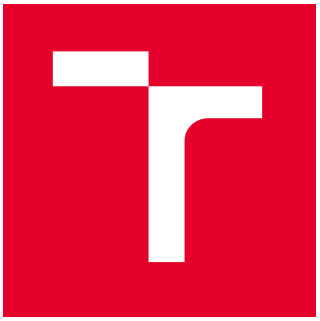


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

KNIHOVNA PRO PRÁCI SE SENZORY UMOŽŇUJÍCÍMI DISTRIBUOVANÉ VÝPOČTY V JAZYCE FORTH

THE LIBRARY FOR WORKING WITH SENSORS ALLOWING DISTRIBUTED PROCESSING IN FORTH LANGUAGE

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Kryštof Kudela**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Miroslav Jirgl, Ph.D.**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Kryštof Kudela ***ID:*** 206809

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Knihovna pro práci se senzory umožňujícími distribuované výpočty v jazyce FORTH

POKYNY PRO VYPRACOVÁNÍ:

Cílem této bakalářské práce je vytvořit knihovnu slov pro snadnou implementaci chytrého senzoru umožňující provádět distribuované zpracování dat. Funčnost knihovny bude demonstrována na několika vzájemně komunikujících jednotkách.

1. Seznamte se s jazykem FORTH.
2.Seznamte se s různými variantami FORTHu pro MCU a srovnejte jejich vlastnosti.
3.Definujte chytré senzory, distribuované systémy a výpočty.
4. Vytvořte knihovnu pro práci se senzory umožňujícími distribuované výpočty v jazyce FORTH.
5. S využitím vámi vytvořené knihovny realizujte nekolik funkčních jednotek, které spolu budou komunikovat.

DOPORUČENÁ LITERATURA:

[1] BRODIE, Leo. Starting FORTH: an introduction to the FORTH language and operating system for beginners and professionals [online]. Englewood Cliffs, N.J.: Prentice-Hall, c1981. ISBN 01-384-2930-8. Dostupné z: https://www.forth.com/wp-content/uploads/2018/01/Starting-FORTH.pdf

[2] GAGLIO, Salvatore, et al. Use of Forth to Enable Distributed Processing on Wireless Sensor Networks. Dostupné z: http://www.complang.tuwien.ac.at/anton/euroforth/ef15/genproceedings/papers/peri.pdf

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 22.5.2023

Vedoucí práce: Ing. Miroslav Jirgl, Ph.D.

Konzultant: Ing. Vilém Kárský

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

ABSTRAKT

Tato bakalářská práce se věnuje programovacímu jazyku Forth, variantám operačních systémů Forth, rozboru mikrokontrolérů, definici chytrých senzorů a distribuovaným systémům a výpočtům. Cílem je čtenáře uvést do celé vývojové platformy Forth, kde se nejprve naučí ve Forthu programovat, poté se dozví o operačních systémech FlashForth a AmForth. V kapitole MCU je popsán jednočipový počítač, jeho rozdělení a použití. Následuje kapitola s definicí chytrých senzorů. Poté teoretický úvod do distribuovaných systémů a výpočtů, na něž navazuje řešení multitaskingu ve Forthu. V jazyce Forth je vytvořena síť, která umožňuje vzdálené ovládání za pomocí bezdrátového modulu.

KLÍČOVÁ SLOVA

Forth, distribuované systémy, distribuované výpočty, chytré senzory, chytrý senzor, MCU, mikrokontrolér, jednočipový mikropočítač, jednočipový počítač, FlashForth, AmForth, NRF24L01+, ATmega328P, síť, vzdálený příkazový řádek

ABSTRACT

This semestral work is about programming language Forth, variation of operating systems, analysis of microcontroller unit, definition of intelligent sensors and distributed systems and distributed computing. The purpose of this work is introduce the reader to entire Forth development platform, where first chapter is learning how to program in Forth, then there are informations about the FlashForth and AmForth operating systems. The MCU chapter describes the microcontroller unit, where they are used and its division. Next chapters are about definition of smart sensors and theoretical introduction to distributed systems and distributed computing, which contain also a multitasking solution in Forth. A network is created in Forth that allows remote control using a wireless module.

KEYWORDS

Forth, distributed systems, distributed computing, intelligent sensor, MCU, microcontroller unit, FlashForth, AmForth, NRF24L01+, ATmega328P, network, remote shell

Vysázeno pomocí balíčku `thesis` verze 4.07; `http://latex.feec.vutbr.cz`

KUDELA, Kryštof. *Knihovna pro práci se senzory umožňujícími distribuované výpočty v jazyce FORTH*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2023, 55 s. Bakalářská práce. Vedoucí práce: Ing. Miroslav Jirgl, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Kryštof Kudela
VUT ID autora:	206809
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Knihovna pro práci se senzory umožňujícími distribuované výpočty v jazyce FORTH

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucím bakalářské práce panu Ing. Miroslav Jirglovi, Ph.D. a Ing. Vilému Kárskému za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Úvod

Tato práce se zabývá vývojovou platformou Forth v implementaci na MCU s chytrými senzory s pomocí distribuovaných systémů a výpočtů. Forth je především programovací jazyk, ale i operační systém a programovací prostředí. Existují různé verze jak OS tak i programovacích prostředí (např. GForth, SwiftForth).

V této semestrální práci se budeme zabývat programovacím jazykem, variantami OS a komunikací chytrých senzorů pomocí distribuovaných systémů a výpočtů. Věnuje se vytvoření sítě senzorů, které dokážou komunikovat mezi sebou a umožňují vzdálené ovládání.

Nejprve si ukážeme jaké má tento jazyk výhody a nevýhody, v čem se liší od ostatních jazyků a jak v něm programovat, tzn. budou ukázány syntaxe příkazů a jejich vysvětlení. Druhá kapitola je o variantách operačních systémů a jejich rozdílech. Ve třetí kapitole bude definice jednočipového mikropočítače a embedded systému, budou popsány jeho části a rozdělení. Čtvrtá kapitola obsahuje definici chytrých senzorů a poté bude vysvětleno, jakým způsobem spolupracuje chytrý senzor s CPU. V páté kapitole je definice distribuovaných systémů a výpočtů a jaké konkrétní oblasti v tomto oboru se budeme věnovat my. Dále v šesté kapitole je popis bezdrátového modulu, a jakým způsobem se ovládá. V poslední části BP je popsán kód, který vytváří síť senzorů a umožňuje jejich vzdálené ovládání.

1 Forth

První kapitola pojednává o programovacím jazyku Forth, který vznikl na konci 60. let 20. století. Byl vyvinut pouze jedním člověkem – Charlesem H. Moorem, původně pouze jako pomůcka při jeho vlastní práci. Později se ukázal jako velmi vhodný jazyk především pro vývoj mikrokontrolérů díky svému minimalistickému provedení a rychlosti. Informace obsažené v této kapitole vycházejí z oficiálních stránek [7].

1.1 Úvod do Forthu

Jazyk Forth se výrazně liší od tradičních programovacích jazyků a je mezi nimi jedinečný tím, že jeho vývoj nebyl podporován žádnými významnými firemními ani akademickými sponzory. Tehdejší programátoři byli nuceni vytvářet programy co nejrychlejší, a s co nejmenšími nároky na kapacitu operační paměti, protože výpočetní čas byl nákladný. Moore navrhl jazyk tak, aby dovoloval psát kompaktní a snadno odladitelné programy, díky čemuž se stal populární. Jelikož jej sám v začátcích využíval pro pracovní matematické výpočty v oblasti astronomie, je Forth optimalizovaný na složitější výpočty s rozhodováním (větvení programu) a velké množství podprogramů a funkcí, které jsou označovány jako *slova*. Samotný jazyk je v základu velmi minimalistický (obsahuje jen minimum nutných slov), což mimo jiné umožňuje snadné portování vývojového prostředí pro různé instrukční sady nových mikroprocesorů. Důvodem je Moorovo přesvědčení, že programátor by si měl vytvářet všechny pomocné prostředky (funkce a procedury) až v případě potřeby a to téměř od nuly, aby musel dopodrobna pochopit, jak cílová platforma vlastně funguje a výsledek vytvořil co nejefektivněji.

Jedná se o jazyk interpretovaný[1] a současně interaktivní (do běhu programu lze zasahovat přes přístupový terminál), proto lze vyvíjené programy velmi jednoduše testovat v porovnání s kompilovanými jazyky. Ve své syntaxi se však výrazně liší od ostatních dnes běžně používaných jazyků postfixovou notací zápisu matematických operací a řídicích příkazů. Z pohledu architektury jej můžeme zařadit jako:

- **strukturovaný** – jde o speciální typ procedurálního programovaní, které používá pouze 3 řídicí struktury[6]:
  - **sekvence**: provádění příkazů postupně za sebou.
  - **výběr**: příkazy jsou vykonávány v závislosti na vstupní podmínce řídícího příkazu, patří sem např. příkazy if ... else, case.
  - **opakování**: do této řídící struktury patří cykly: while, do ... until, for.

[1] Ke spuštění programu je nutný zdrojový kód, který se interpretuje (provádí) až za běhu – viz. `https://cs.wikipedia.org/wiki/Interpretovaný_jazyk`.

Zásadní výhodou je od první oficiální verze jazyka také dostupnost stejnojmenné vývojové sady nástrojů pro Forth, která zahrnuje překladač, editor i interaktivní interpretr. Díky tomu je například možné přímo ve vlastním programu využít celý integrovaný editor nebo část překladače pro překlad či okamžité vyhodnocení uživatelem zadaného kódu (výrazu) za běhu aplikace. Vývojové prostředí je navíc více uživatelské a podporuje také multitasking[2].

Kompletní prostředí jazyka Forth lze při programování ve své podstatě díky výše popsaným možnostem použít i jako náhrada operačního systému, jelikož je schopno zpřístupnit operační paměť i případné blokové paměťové zařízení (disky, pásky). Pro tyto účely má vyvinutý vlastní blokový souborový systém – k datům se přistupuje po zarovnaných blocích, který pro některé aplikace (zejména databázové a zpracovávající obraz) značně urychlil práci se soubory.

Zásobníkově orientované programování

Forth pracuje se dvěma zásobníky: Parameter stack a Return stack. Parameter stack je zásobník určený pro ukládání aktuálních hodnot, je ten hlavní, se kterým programátor pracuje. Všechny základní příkazy jako například DUP, SWAP, ROT, atd. pracují právě s tímto zásobníkem. Je tedy určen pro operandy. Druhý zásobník Return stack se používá pro návratové adresy, aby se zajistil jednoduchý mechanismus pro vnořování rutin[3] a lze do něj také vložit námi určené hodnoty pro dočasné použití, musí však být odstraněny před dokončením dané funkce/*slova*.

Obrácená polská notace

Zápis příkazů je v tzv. *obrácené polské notaci*, neboli postfixová notace (anglicky Reverse Polish notation [4]). Znamená, že se nejprve zadají operandy a poté operátory. Příklad takového zadávání si můžeme ukázat na sčítání dvou čísel:

`1 + 2 = vysl`
ve Forth zápis vypadá následovně:

```
1 2 + .
```

```
>> 3
```

[2]Více úloh běžících současně – viz. `https://cs.wikipedia.org/wiki/Multitasking`.

[3]Stack-oriented programming – `https://en.wikipedia.org/wiki/Stack-oriented_programming`

[4]Reverse Polich notation – `https://en.wikipedia.org/wiki/Reverse_Polish_notation`

Nejprve se na Parameter stack zapíšou tyto dvě čísla, poté se příkazem + čísla vezmou ze zásobníku, sečtou se a výsledek se vrátí na zásobník. Tečkou na konci se vypíše výsledek na terminál[5].

Všechny příkazy – operátory, které Forth definuje jsou *slova*. Seznam slov se ukládá do tzv. *slovníku*, kde se nacházejí i *slova* definovaná Forthem.

Výhody Forthu:

- je velmi rychlý oproti jiným jazykům
- kód je snadno přenositelný na jiné procesory

Všechny zápisy kódu zde budou mít tuto strukturu:

```
kód, vstup na terminál
```

```
>> výpis na terminál, výstup
```

1.2 Základy jazyka FORTH

1.2.1 Vysvětlení a definice *Slova*

Hlavní výhodou Forthu je, že můžeme definovat již zmíněné *slova (word)*, které určují, co se bude dít. Můžeme je připodobnit funkcím v jazyce C, kde začínající znak je *":"*. Ukázku definice *slova* můžeme vidět níže:

```
: HELLO ." Ahoj svete" ;
HELLO
```

```
>> Ahoj svete
```

první slovo za dvojtečkou je název *slova*, následují příkazy a středník ukončuje definici *slova*. Nesmíme zapomenout, že příkazy se oddělují mezerou.

Slova můžeme definovat z již vytvořených slov:

```
: HELLO2 HELLO ." , programuju ve Forth!" ;
HELLO2
```

```
>> Ahoj svete, programuju ve Forth!
```

[5]Terminál – `https://cs.wikipedia.org/wiki/Terminál_(informatika)`

Problém s redefinicí *slova*

Pokud *slovo 1* redefinujeme (vytvoříme nové *slovo* s již existujícím názvem), původní definice zůstává v *knihovně slov* (ve *slovníku*). To nám způsobuje potíže ve *slově 2*, které je definováno mimo jiné i původním *slovem 1*. Děje se tak, protože v definici *slova 2* je pouze adresa na původní *slovo 1*. Tím, že se ukládá jen adresa, se ovšem program značně zrychlí, protože nemusí po každém volání prohledávat *slovník*, ale jde přímo k žádanému *slovu*. Problém s redefinicí se řeší následujícím způsobem:

- Program s mnoha příkazy se samozřejmě píše do souboru, který poté nahrajeme do MCU. Chybu v souboru napravíme a poté program nahrajeme znova do MCU. S tím souvisí další problém – resetování slovníku lze jen znovunahráním OS do MCU a to trvá poměrně dlouho (řádově desítky sekund). Řešením je použití příkazu `MARKER název_značky` na začátku programu, případně kdekoliv v kódu, který nám do slovníku zapíše *značku* a zavoláním příkazu `název_značky`, který zadáme do terminálu za běhu programu, se smažou všechny definované *slova* včetně *značky*. Více o práci se zdrojovým souborem v podkapitole 1.2.4. Pokud chceme ladit kód přímo přes terminál, můžeme tak za pomocí již zmíněného slova `MARKER` nebo `FORGET název_slova_k_vymazání`, které odstraní všechna slova definovaná po něm a včetně něj a v případě, že jsou ve slovníku již dvě slova stejného názvu, příkaz odstraní jen ten poslední definovaný, tudíž musíme příkaz použít, kolikrát bude potřeba. Po smazání se musí znovu definovat slova, která jsme odstranili, proto se z valné většiny tento způsob nepoužívá.

1.2.2 Výpis znaků

Výpis znaků na terminál se provádí pomocí *slova/příkazu EMIT* nebo pomocí tečky a uvozovek `." znaky "`, kde příkaz `."` je výpis následujících znaků na terminál. Výpis ukončíme opět uvozovkou. Při použití příkazu *EMIT* buďto napíšeme číslo z ASCII tabulky, které odpovídá danému znaku nebo před konkrétní znak napíšeme *[CHAR]*, to ovšem funguje jen u definice nového *slova*.

```
64 EMIT
```

```
>> @
```

```
: ZNAK [CHAR] @ EMIT ;
ZNAK
```

```
>> @
```

```
." znaky "
```

```
>> znaky
```

```
: ZNAKY ." vypis znaku" ;
ZNAKY
```

```
>> vypis znaku
```

1.2.3 Aritmetické výpočty

Ve Forth se nepoužívají závorky pro určení priority u matematických výpočtů. Díky postfixové notaci nejsou potřeba. Příklady složitějších výpočtů (inspirováno z [12]):

zápis ve standartním tvaru:

1. `12(10+11)`
2. `(3*10 - 11) / 4 + 12`

zápis v postfixové notaci:

1.

```
10 11 + 12 *
```

nebo:

```
12 10 11 + *
```

2.

```
3 10 * 11 - 4 / 12 +
```

Příklad převodu metrů na milimetry:
$milimetry = metry * 1000$

```
: m>mm 1000 * . ." mm" ;
1 m>mm
```

```
>> 1000 mm
```

Tento příklad zde uvádím kvůli ukázce spojení aritmetického výpočtu se *slovem*.

1.2.4 Zdrojový soubor

Chceme-li psát kód do zdrojového souboru, stačí nám k jeho nahrání do MCU použít příkaz `INCLUDE název_souboru`. Název musí být včetně cesty, pokud je soubor mimo zdrojovou složku terminálu. Jak jsem zmiňoval, redefinicí se nové *slovo* zapíše do slovníku a stará verze zůstává, tím hrozí zaplnění paměti. Proto musíme před začátkem kódování použít příkaz `MARKER název_značky` provedeným do terminálu. Následně do zdrojového kódu na začátek napíšeme příkaz `název_značky`. Protože se tím při includování odstraní všechny slova včetně *značky*, napíšeme jako druhý příkaz opět `MARKER název_značky`. Každým `INCLUDE název_souboru` tedy dosáhneme automatického odstranění starých *slov* ze slovníku a nahrání nového kódu. [5]

Vytvoření *značky* na začátku programování a *INCLUDE* (příkazy do terminálu):

```
MARKER work
INCLUDE mysourcecode.f
```

Začátek zdrojového souboru:

```
work
MARKER work
. . . kód . . .
```

1.2.5 IF .. ELSE .. THAN řídící struktura

Forth neobsahuje čistě boolean datový typ, proto pokud je na zásobníku 0, je to bráno jako FALSE, pokud je tam jakékoliv jiné číslo –> TRUE. Standardně je TRUE -1, což znamená v signed aritmetice všechny bity rovno 1. Operátory porovnání po vyhodnocení uloží na zásobník číslo FALSE/TRUE, tomuto číslu se říká flag, který je pak vyhodnocen IFem. *ELSE* je nepovinné slovo.

```
        desítkově   binárně (jeden byte)
TRUE    -1          1111 1111
FALSE   0           0000 0000
```

```
podmínka IF xxx ELSE yyy THEN zzz
```

Pokud podmínka je true, vykonej xxx, jinak yyy, poté pokračuj na zzz.

Příklad definice slova s IF .. THEN:

```
: ?DAY  32 < IF  ." Looks good " ELSE  ." no way " THEN ;
30 ?DAY
35 ?DAY
```

```
>> Looks good
>> no way
```

Příklad použití IF .. ELSE .. THEN zakomponovaný do sebe [12]:

```
: EGGSIZE
   DUP 18 < IF  ." reject "      ELSE
   DUP 21 < IF  ." small "       ELSE
   DUP 24 < IF  ." medium "      ELSE
   DUP 27 < IF  ." large "       ELSE
   DUP 30 < IF  ." extra large " ELSE
      ." error "
   THEN THEN THEN THEN THEN DROP ;

23 EGGSIZE
29 EGGSIZE
40 EGGSIZE
```

```
>> medium
>> extra large
>> error
```

1.2.6 Operace na zásobníku

SWAP	Prohodí dvě horní čísla
DUP	Zkopíruje poslední číslo
OVER	Zkopíruje 2. číslo a vloží ho na vrchol
ROT	Vezme 3. číslo a vloží ho na vrchol
DROP	Smaže číslo na vrcholu
2SWAP	Prohodí dva poslední páry čísel
2DUP	Zkopíruje dvě poslední čísla a vloží je na vrchol
2OVER	Zkopíruje 2. pár čísel a vloží je na vrchol
2DROP	Smaže dvě poslední čísla

1.2.7 Matematické operace

```
+       sčítání
-       odčítání
*       násobení
/       dělení
/MOD    vrátí výsledek dělení a zbytek po dělení
MOD     vrátí zbytek po dělení
NEGATE  negace čísla na vrcholu zásobníku
ABS     absolutní hodnota z čísla
MIN     vrátí minimum ze dvou čísel
MAX     vrátí maximum ze dvou čísel
```

1.2.8 Matematická logika

Porovná dva operandy na zásobníku a vrátí TRUE nebo FALSE.

Zápis operátorů porovnání:

```
standardní tvar     postfixová notace
2 < 5               2 5 <
20 > -50            20 -50 >
```

Operátory porovnání:

```
=   je rovno
<>  není rovno
<   menší než
>   větší než
0=  je rovno nule / logická negace
0<  je menší než nula
0>  je větší než nula
<=  je menší nebo rovno
U<  unsigned menší než
U>  unsigned větší než
```

Bitové operátory

```
AND         vrátí logický bitový AND
OR          vrátí logický bitový OR
INVERT      vrátí logický bitový NOT
0=          porovná číslo s nulou a vrátí TRUE nebo FALSE,
            dá se použít jako NOT, ale není bitový
```

1.2.9 Některé další užitečné příkazy

```
?DUP        zkopíruje poslední číslo na zásobníku když je různé od 0
ABORT" xx"  jestliže je flag TRUE, vypíše na terminál zprávu xx,
            pokud je flag FALSE, neprovede nic
```

1.2.10 Cykly

Příkazem `LEAVE` ukončíme cykly okamžitě.

DO ... LOOP

Syntaxe DO ... LOOP cyklu, odpovídá cyklu FOR v jazyce C:

```
limit index DO ... LOOP
```

Příklad použití [12]:

```
: TEST   10 0 DO  CR ." Hello "  LOOP ;
TEST
```

```
>> Hello
   Hello
   Hello
   Hello
   Hello
   Hello
   Hello
   Hello
   Hello
   Hello
```

Varianta +LOOP zajistí přičítání indexu o uvedenou hodnotu. Proměnná I je index [12].

```
: PENTAJUMPS  50 0 DO  I .  5 +LOOP ;
PENTAJUMPS
```

```
>> 0 5 10 15 20 25 30 35 40 45
```

BEGIN UNTIL | BEGIN AGAIN | BEGIN WHILE REPEAT

```
\ opakuje příkaz aaa dokud je f FALSE
BEGIN aaa f UNTIL

\ opakuje příkaz aaa do nekonečna)
BEGIN bbb AGAIN

\ příkaz ccc se provede alespoň jednou a opakuje se pokud je f
\ TRUE, po kontrole f se také provede ddd a cyklus se opakuje )
BEGIN ccc f WHILE ddd REPEAT
```

1.2.11 Proměnné, konstanty, pole

Definice a použití proměnné:

```
\ definice názvu proměnné
\ definice dvojnásobné délky proměnné pomocí příkazu 2VARIABLE
VARIABLE DATE

\ přidělení hodnoty - nejdříve zapíšeme do zásobníku číslo,
\ příkazem ! ho zapíšeme do proměnné
12 DATE !

\ příkaz @ je načtení hodnoty proměnné do zásobníku, tečkou
\ vypíšeme na terminál ze zásobníku
DATE @ .

\ namísto @ . můžeme psát otazník ?
DATE ?
```

```
>> 12
>> 12
```

Konstanty

```
\ definice dvojnásobné délky konstanty pomocí příkazu 2CONSTANT
\ definice názvu konstanty ve tvaru:
\ hodnota CONSTANT nazev
10 CONSTANT LIMIT

\ hodnotu vypíše bez příkazu @. nebo ?
LIMIT
```

```
>> 10
```

Pole Definice pole s 5 položky:

```
VARIABLE nazev 5 CELLS ALLOT
```

2 Varianty Forthu pro mikrokontroléry

Pokud bychom se zaměřili na implementace Forthu zejména pro mikrokontroléry firmy Atmel, které jsou v dnešní době hojně rozšířené, nalezli bychom například varianty `FlashForth` a `AmForth`. V obou případech se jedná o open-source projekty s otevřenými zdrojovými kódy a rozsáhlými online dokumentacemi, odkud byly čerpány informace pro tuto kapitolu.

2.1 FlashForth

Na oficiálních stránkách[1] je projekt označen jako Forth systém implementovaný pro 8bitové mikročipy `PIC18F` a 16bitové mikročipy `PIC24`, `PIC30`, `PIC33`[2] a také mikrokontroléry rodiny Atmega [3] firmy Atmel. Zároveň FlashForth běží také na populárních deskách Arduino UNO či MEGA[4] a umožňuje vytváření a ladění komplexních aplikací v reálném čase. FlashForth implementace se snaží být kompatibilní se standardem Forthu ANS z roku 1994 a je uveřejněna pod licencí GNU General Public Licence v3[5].

Tato implementace Forthu je provozována kompletně na mikrokontroléru. Přestože se program na MCU nahrává přes přístupový terminál komunikující skrze sériový port (většinou z PC), kompletní vývojové prostředí s kompilerem a interpretem běží na procesoru mikrokontroléru. Vývoj aplikace tak nevyžaduje žádný dodatečný software ani hardware a zároveň nijak neomezuje funkčnost samotného hardwaru, na kterém je vyvíjena. Můžeme k němu tedy bez omezení připojovat a pracovat s libovolnými dalšími periferiemi.

FlashForth bere ohled na velikost písmen a to znamená, že všechna vestavěná slova lze používat pouze s malými písmeny. Hexadecimální číslice musejí být také zapsány ve tvaru s malými písmeny. Existuje i podpora pro 32 bitovou matematiku s dvojnásobnou přesností (tzv. double-precision) výpočtů, pro kterou slouží slova `UM/MOD`, `M+` a `UM*`, a dále také podpora pro 48 a 64 bitovou matematiku. Maximální délka slova je omezena na 15 znaků.

Přestože podporované procesory jsou postaveny na Harvardské architektuře, a tudíž mají oddělené paměťové prostory pro RAM, EEPROM a flash paměti,

[1] Oficiální stránky projektu FlashForth – viz. `https://flashforth.com/`.

[2] Mikročipy PIC – viz. `https://en.wikipedia.org/wiki/PIC_microcontrollers#PIC18`.

[3] Mikrokontroléry Atmel Atmega – viz. `https://en.wikipedia.org/wiki/AVR_microcontrollers`.

[4] Arduino desky – viz. `https://en.wikipedia.org/wiki/Arduino`.

[5] Svobodná licence GNU/GPLv3 – `https://www.gnu.org/licenses/gpl-3.0.en.html`.

FlashForth všechny tyto druhy mapuje do jednoho adresového prostoru, díky čemuž je možné při práci přistupovat stejnými slovy do všech pamětí. Každá paměť má svůj alokační ukazatel a lze ji alokovat pomocí sekvence patřičných slov `CREATE ALLOT VARIABLE 2VARIABLE , C, VALUE DEFER`. Slova pro přístup k paměti `@ !`, `C@` a `C!` lze používat pro všechny typy bez omezení. Pouze sada slov – `MSET MCLR MTST BSET BCLR BTST` – může přistupovat pouze k paměti typu RAM, protože se typicky používají pro správu bitů v registrech.

2.2 AmForth

Dle dokumentace projektu[6] se jedná o jednoduše rozšiřitelnou implementaci interpretru jazyka Forth pro rodinu mikrokontrolérů `Atmel AVR8 Atmega`[7] a nějaké další varianty `Texas Instruments MSP430`[8]. Aktuálně se pracuje na podpoře instrukční sady RISC-V[9] pro 32 bitové procesory, nicméně jedná se stále pouze o experimentální verzi a spousta vysokoúrovňové implementace je sdílena z 16 bitové varianty. AmForth je šířen pod svobodnou licencí GNU Public License v3[10].

I pro tuto variantu Forthu platí výše zmíněné, že celé vývojové prostředí běží na procesoru mikrokontroléru, tudíž pro práci je nabízeno jednoduché uživatelské rozhraní dostupné po sériové lince. AmForth se snaží o implementaci slov téměř kompatibilní se specifikací standardu `Forth 2012`[11], přičemž nepodporuje složitě proveditelné slovo `FORGET`, nýbrž místo něj doporučuje používat `MARKER`. Dále přidává následující nová slova specifická pro cílovou platformu:

- **COLD** – Slovo slouží k vyvolání inicializačního kódu uloženého v souboru `cold.asm`, který je automaticky zaveden po vyvolání přerušení restartu MCU. Zavádí základní běhové prostředí, inicializuje zásobník a následně spouští interpretr ve virtuálním systému Forthu.
- **WARM** – Zavoláním slova `WARM` z prostředí jazyka Forth je zahájena akce restartu, jež spustí slovo `PAUSE` k pozastavení běhu, aplikací definovanou akci `TURNKEY` a následně skončí slovem `QUIT`.
- **TURNKEY** – Jedná se o aplikací definovaný ukazatel v paměti EEPROM na zvolené slovo. V praxi se používá k zahájení sériové komunikace či inicializaci vstupně/výstupních portů, aby bylo například možné interagovat s

[6] Oficiální stránky projektu AmForth – viz. `http://amforth.sourceforge.net/`.

[7] AVR8 jsou 8 bitové mikrokontroléry s Harvardskou architekturou a instrukční sadou RISC od firmy Atmel – viz. `https://en.wikipedia.org/wiki/AVR_microcontrollers`.

[8] MSP430 je 16bitový mikrokontrolér pro zpracování signálů od Texas Instruments – viz. `https://en.wikipedia.org/wiki/TI_MSP430`.

[9] Otevřená instrukční sada z rodiny RISC – viz. `https://cs.wikipedia.org/wiki/RISC-V`.

[10] Svobodná licence GNU/GPLv3 – `https://www.gnu.org/licenses/gpl-3.0.en.html`.

[11] Standard Forth 2012 – viz. `https://forth-standard.org/standard/words/`.

interpretrem.

- **QUIT** – Inicializuje oba zásobníky do prázdného stavu a restartuje interpretr.

Mimo zmíněná slova výše AmForth přidává také podporu pro konfiguraci instrukcí assembleru `sleep` a `wdr` (watch dog reset). Samotný assembler vytvořil pro platformu AVR8 Luboš Pěkný, přičemž se do jeho kontextu lze v interpretru přepnout slovem `CODE` a vrátit zpět slovem `END-CODE`.

Atmega mikrokontrolér oplývá 3 různými typy pamětí – RAM, EEPROM a flash paměť. Nároky pro bezproblémový běh jádra AmForthu na 8 bitových mikrokontrolérech Atmel z rodiny AVR jsou 8-12KB paměti flash, 80 bytů EEPROM a 200 bytů RAM paměti. Slova `@` a `!` pracují v adresovém prostoru paměti RAM, kde jsou zahrnuty vstupně/výstupní porty i CPU registry. Pomocí slov `@E` a `!E` můžeme přistupovat k paměti EEPROM a slovy `@I` či `!I` lze pracovat s flash pamětí. Vždy dochází k přenosu 2 bytů mezi pamětí a datovým zásobníkem. Existují také speciální slova `C@` a `C!`, která umožňují přistupovat ke spodní polovině paměti RAM. Řetězce je pak možné ukládat do pamětí RAM a flash.

K obsloužení vstupů a výstupů se používají standardní slova `EMIT/EMIT?` a `KEY/KEY?`, která jsou namapována na sériovou komunikaci. AmForth má kromě přístupového terminálu podporu také pro další typy komunikačních sběrnic, mezi které patří například CAN, USB či IIC. Výjimky jazyka jsou podporovány naplno. V rámci vývojového prostředí však není dostupný editor.

3 MCU – Microcontroller Unit

MCU jsou jednočipové mikropočítače, které mají nízký výpočetní výkon, řádově jednotky až stovky MHz, a jsou používány v embedded systémech.

Embedded systémy

V překladu znamená *vestavěné zařízení*. Jsou to systémy pro předem daný účel a většinou specializované použití. Do této skupiny patří mj. i zařízení do ruky jako jsou staré mobilní telefony (např. všem známá Nokia 3310), digitální hodinky, kalkulačky nebo třeba ovladače televizorů. V dnešní době však mobilní telefony i chytré hodinky mají vysoký výkon a jsou víceúčelové čímž ze skupiny vestavěných zařízení vypadávají. Použití je také v průmyslu, kde mohou nahrazovat jednoduché PLC automaty pro ovládání strojů. Další uplatnění je ve všech jednoduchých zařízeních, které nepotřebují vysoký výpočetní výkon, ku příkladu pračky, myčky, automatické osvětlení.

Integrovaný obvod obsahuje následující části pro samostatné fungování [8]:

- RAM paměť
- paměť pro uložení programu (flash, ROM, EEPROM)
- procesor
- oscilátor
- vstupně/výstupní obvody jako jsou A/D a D/A převodníky nebo digitální vstupy/výstupy a komunikační porty.

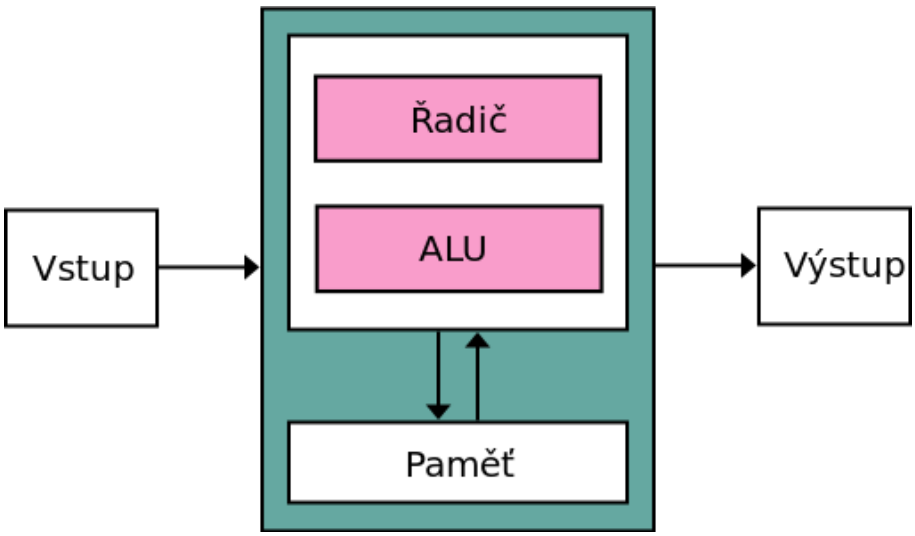
Procesor se skládá z řadiče a ALU – aritmeticko-logické jednotky. Řadič slouží pro zpracování instrukcí (načtení z paměti, příprava pro ALU, ...) a řídí součinnost všech ostatních bloků. ALU se stará o provedení (výpočet) instrukcí.

Kromě základních bloků mohou MCU obsahovat i další periferie [8]:

- řadič přerušení
- řadič displeje
- řadič klávesnice
- časovače
- čítače
- programovatelné hradlové pole
- watch dog - časovač, který neustále čítá během chodu programu a pokud se program zacyklí v nějaké funkci na určitou dobu, watch dog se nevynuluje a celý systém se zresetuje.

MCU můžeme rozdělit podle architektury obvodu do dvou kategorií – Harvardská a von Neumannova:

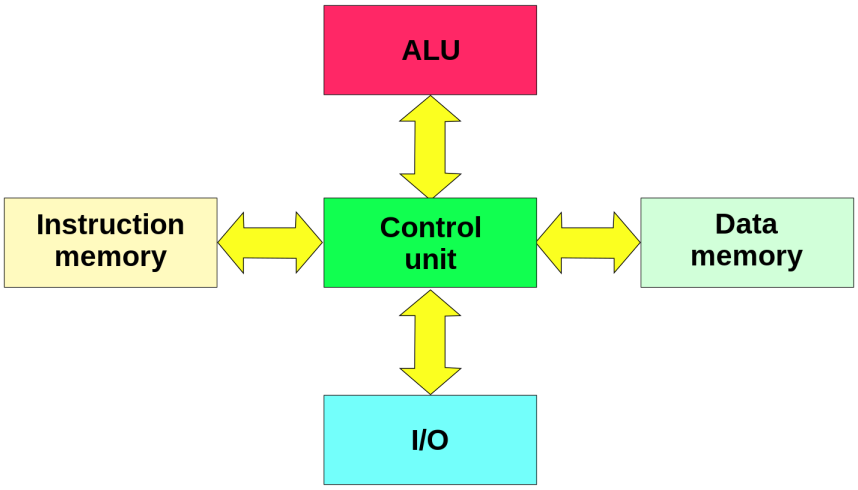
- **von Neumannova architektura**
  Tento způsob návrhu MCU má společnou paměť pro data i program, tudíž i jednu sběrnici. Zpracování programu je tedy sekvenční – nejprve se musí načíst instrukce a poté data. Z toho vyplývá nevýhoda v pomalejší rychlosti. Některé provedení mají na jedné sběrnici i vstupy a výstupy společně s pamětí a procesorem. [9]



Obr. 3.1: Schéma von Neumannovi architektury [9]

- **Harvardská architektura**
  Vyznačuje se oddělenou pamětí pro data a pro programy. Musí mít proto i dvě sběrnice. To znamená, že můžeme mít rozdílnou šířku obou sběrnic, rozdílné časování, způsob adresování, což se hojně využívá. Velkou výhodou je také možnost načítat data a program (instrukce) zároveň, je tedy rychlejší než von Neumannova architektura. Nevýhodou je složitější návrh kvůli zmíněným dvěma sběrnicím.

  Existuje ještě **modifikovaná Harvardská architektura**, jejíž princip je ve spojení von Neumannovi a Harvardské, kde načítáme data i program z jedné paměti, ovšem procesory mají své cache paměti uvnitř čipu pro program i data zvlášť a procesor má tak samostatný přístup, jak pro data, tak pro program. Této metody využívají dnešní moderní PC. [10]



Obr. 3.2: Schéma Harvardské architektury [10]

Další rozdělení je podle instrukční sady procesoru:

- **RISC**
  Anglicky Reduced Instruction Set Computer v překladu znamená *redukovaná instrukční sada procesoru*. Obsahuje jen nejzákladnější velmi jednoduché příkazy, které jsou dokonale optimalizovány. Tím mohou být RISC procesory rychlejší než CISC ve stejné úloze. V dnešní době je tento způsob častější.
- **CISC**
  Anglicky Complex Instruction Set Computing v překladu *komplexní výpočetní instrukční sada*, jsou instrukce s komplexnějšími příkazy, které mají větší hustotu. Hustota instrukce je dána tím, že obsahuje více jednodušších instrukcí. Existuje taky spousta procesorů kombinující oba způsoby instrukčních sad.

4 Chytré senzory

Aby mohl vestavěný systém interagovat nějakým způsobem s okolím, potřebuje mít zpětnou vazbu. K tomu potřebné informace mohou systému zprostředkovat právě chytré senzory.

4.1 Definice chytrého senzoru

Pod inteligentním snímačem si můžeme představit zařízení, které je schopno změřit patřičnou fyzikální veličinu v podobě analogového signálu, dále tento vstupní signál převést do digitální podoby a vyhodnotit jej v závislosti na krajních mezích, kalibraci a konfiguraci. Vyhodnocení probíhá pomocí přiloženého mikrokontroléru, který také poskytuje vnější komunikační rozhraní k získávání naměřených dat. Toto rozhraní bývá ve většině případů standardizované a podporuje některou z existujících přenosových sběrnic/rozhraní – IIC, Ethernet, USB, SPI, RS232, RS485, Bluetooth, WiFi atp.

4.2 Příklady chytrých senzorů

Typické a často používané chytré senzory v praxi mohou sloužit například k měření teploty, světelnosti, vzdálenosti, vlhkosti, odběru elektrické energie, měření průtoku a spotřeby, nebo také k detekci pohybu a zvukových či jiných vln.

Níže se nachází příkladový seznam vybraných inteligentních snímačů s jejich popisem:

- **DS18B20**[1] – V praxi často používaný inteligentní teplotní senzor komunikující po 1-Wire rozhraní od firmy Maxim Integrated.
- **VL53L0X**[2] – Infračervený senzor vzdálenosti nejmenší na světě komunikující po IIC sběrnici.
- **MPU6050**[3] – Pohybový inteligentní senzor detekující 6 os pohybu a programovatelnou možností rozpoznávání pohybových gest.

[1] Datasheet DS18B20 – viz. `https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf`.

[2] Datasheet VL53L0X – viz. `https://www.st.com/resource/en/data_brief/vl53l0x.pdf`.

[3] Datasheet MPU6050 – viz. `https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf`.

5 Distribuované systémy a výpočty

Distribuované systémy jsou systémy, které sdílí své data mezi sebou. V těchto systémech pracují distribuované výpočty, jinak zvané komunikační protokoly. Pod systémem si můžeme představit software ale třeba i procesor. Systémy spolu komunikují pomocí sítě. Pod pojmem distribuovaný systém si můžeme představit i komunikaci lidí, kde systémem je člověk, distribuovaným výpočtem je společný jazyk, kterým se dorozumívají a síť jsou zvukové vlny šířící se ve vzduchu.

Pro účely komunikace chytrých senzorů a MCU lze využít výhod Forthu, mezi které patří podpora pro práci více uživatelů či více procesů současně na jednom zařízení. Kapitola vychází z práce [5] a skript [4] zabývajících se tímto tématem.

5.1 Distribuovaný systém

Jednotlivé distribuované systémy se nazývají uzly. Nemají společnou paměť ani společný časový mechanismus. Jednotlivé uzly na sebe nesmí být závislé a při výpadku některého z nich nesmí ohrozit celý systém. Stejně tak můžeme systém jednoduše rozšířit o další uzly bez nutnosti resetování nebo vypojení celého systému a všech uzlů.

Mezi jednu z nevýhod u distribuovaných systémů patří nutnost bránit se proti útokům vedoucím ke krádeži dat.

5.2 Distribuovaný výpočet

Jedná se o komunikační protokoly, které využíváme pro řízení distribuovaných systémů. Požadavky na distribuované výpočty je bezpečnost, životnost, férovost. Bezpečnost musí být zajištěna proti nečekávanému výpadku systému.

5.3 Multitasking ve Forthu

Podpora pro provádění více procesů současně je v jádře Forthu zakomponována na nejnižší úrovni. V praxi rozlišujeme mezi dvěma typy:

- **Kooperativní multitasking** – V případě kooperativního multitaskingu si běžící aplikace předávají mezi sebou řízení z vlastní iniciativy, tudíž je zodpovědnost na samotném programátorovi, aby včas předal kontext jiné aplikaci. Stejného principu se využívá i v jazyce Forth, v němž existuje vestavěné slovo `PAUSE` sloužící k pozastavení právě běžící aplikace a předání operačního kontextu jiné aplikaci, o které rozhodne plánovač.

- **Preemptivní multitasking** – Vyskytuje se většinou ve velkých operačních systémech a řízení přepínání mezi právě běžící aplikací probíhá na základě systémového přerušení. Jelikož je s přepínáním kontextu spojena vysoká režie, není tento způsob příliš vhodný pro vestavěné systémy.

Podle dokumentace není ve standardu ANS Forth z roku 1994 specifikována přesná forma multitaskingu, proto popis v práci [5] vychází spíše z příkladů a zkušeností již napsaných funkčních aplikací. Více procesové aplikace mají však celou řadu efektivních uplatnění. Mezi nejčastější použití patří obsluha vstupně-výstupních operací během jiné výpočetní činnosti aplikace (například slovo `KEY?` má paralelismus slovem `PAUSE` přímo implementovaný, aby při čekání na vstup nezablokovalo pokračování programu). Tento princip je možné využít i v rámci distribuovaných systémů, kdy v řídicí jednotce běží několik jednodušších úloh shromažďujících informace ze svého okolí. Samotná jednotka pak může provádět v rámci jiné hlavní úlohy zpracování těchto podnětů. Komunikace s okolím tedy nezpůsobuje nečinnost hlavní úlohy a vše se může dít plynule paralelně. Jiným příkladem může být obsluha přerušení, pro kterou existuje úloha běžící v pozadí a hlídající datový zásobník, zdali se na něm nevyskytla chyba způsobená nějakou nepovolenou operací.

Aby bylo možné multitasking provozovat, musí se nejprve jednou na začátku práce aktivovat pomocí slova `INIT_MULTI`, které způsobí inicializaci datových struktur.

5.3.1 Ukázka jednoduché úlohy

Uvedená ukázka níže vychází z kapitoly o multithreadingu [5].

```
#1000 VALUE DELAY       \ -- n ; časová prodleva mezi #
: ACTION1               \ -- ; úloha vypisující #
    [CHAR] $ EMIT       \ Vypíše dolar $
    BEGIN               \ Začátek smyčky
        [CHAR] # EMIT   \ Vypíše mřížku #
        DELAY MS        \ Znovu nastavit plánovač
        ?DO PAUSE LOOP
    AGAIN               \ Zpět na začátek smyčky
;
```

Úloha lze následně aktivovat slovy `TASK TASK1 ‘ ACTION1 TASK1 INITIATE` a přerušit slovy `TASK1 TERMINATE`. Její spuštění způsobí vypsání znaku $ při aktivaci a následné opakované vypisování znaku # s nastavenou časovou prodlevou pro vykonávání ostatních úloh 1 sekunda, jelikož se tato definovaná časová prodleva `DELAY`

v kombinaci se slovem `MS` převede na údaj v milisekundách.

5.3.2 Víceuživatelské prostředí

V rámci víceúlohového systému je možné definovat také pro každou úlohu zvlášť uživatelské proměnné pomocí slova `USER`. Každý uživatel má pak vlastní paměťový prostor pro proměnné, avšak některé systémové proměnné jako například `BASE` mohou být použity nezávisle na uživatelském kontextu jednotlivých úloh. Obdobně pak každá úloha může mít inicializované i svoje komunikační kanály na základě vlastního kontextu ve vyrovnávací paměti.

6 Bezdrátový modul nRF24L01+

Ke komunikaci mezi senzory byl vybrán bezdrátový modul *nRF24L01+*, který vysílá vlny na frekvenci 2,4 GHz. Dají se pořídit moduly s integrovanou anténou i s externí a se zesilovačem, který má dosah až 1 km ve volném prostoru. Přenosová rychlost je nastavitelná na 250kbps, 1Mbps nebo 2Mbps. Jeho použití je poměrně jednoduché, navíc na webu AmForthu (konkrétně na gitu AmForthu v sekci pro komunitu) je volně dostupná knihovna pro práci s modulem, která byla použita a pro naše účely mírně upravena [1]. Celá tato kapitola čerpá informace z technické dokumentace k *nRF24L01+* [2].

6.1 módy nRF24L01

nRF24L01+ má několik módů ve kterých funguje: *power down, standby, RX, TX.*

6.1.1 Power Down Mode

Power Down Mode je režim, ve kterém má čip nejmenší spotřebu. Všechny registry se dají přečíst a sběrnice SPI je taktéž dostupná. Jde změnit konfigurace. Do tohoto módu se dostaneme, pokud je napájení vyšší nebo rovno 1,9V a nastavením bitu *PWR_UP* na nulu v *CONFIG* registru.

```
PWR_UP = 0
```

6.1.2 Standby-I mode

Do tohoto režimu se dostaneme nastavením bitu *PWR_UP* na jedna v *CONFIG* registru. Spustí se oscilátor a čeká, dokud nepřijde příkaz na změnu módu na *TX* nebo *RX*. Změna na tyto módy se uskuteční nastavením *CE* pinu na 1 a bitu *PRIM_RX* v *CONFIG* registru na jedna nebo nula podle toho, jestli chceme přijímat nebo odesílat. Jakmile *CE* je nastaveno na low, čip se vrátí do *Standby-I*.

```
PWR_UP = 1
```

6.1.3 Standby-II mode

Ve *Standby-II* módu jsou aktivní navíc ještě další registry. Má o něco vyšší spotřebu než *Standby-I*. Tento mód se aktivuje tím, že je nastaveno odesílání *(PRIM_RX=0)*, pin *CE* je nastaven na 1 a zásobník pro odesílání dat je prázdný – *TX FIFO stack* (First In First Out). Jakmile se do něj uloží data, přejde zařízení na *TX* mód.

```
PWR_UP = 1
PRIM_RX=0
TX FIFO zásobník prázdný
CE = 1
```

6.1.4 RX mode

Tento režim je použit pro příjímání zpráv. Po jeho aktivaci se očekává příchod přijímacích frekvencí. Poté co přijdou, signál se demoduluje a kontroluje se, zda paket dat odpovídá tomu, co se očekává. Pokud je přijatý paket validní, uloží se do volného místa v *RX FIFO* zásobníku. Jestliže je *RX FIFO* zásobník plný, přijatý paket je smazán. Validnost paketů se určuje podle adresy a *CRC* bajtů (1 až 2) v paketu [6.3].

Bity k aktivaci módu:

```
PWR_UP = 1
PRIM_RX=1
CE = 1
```

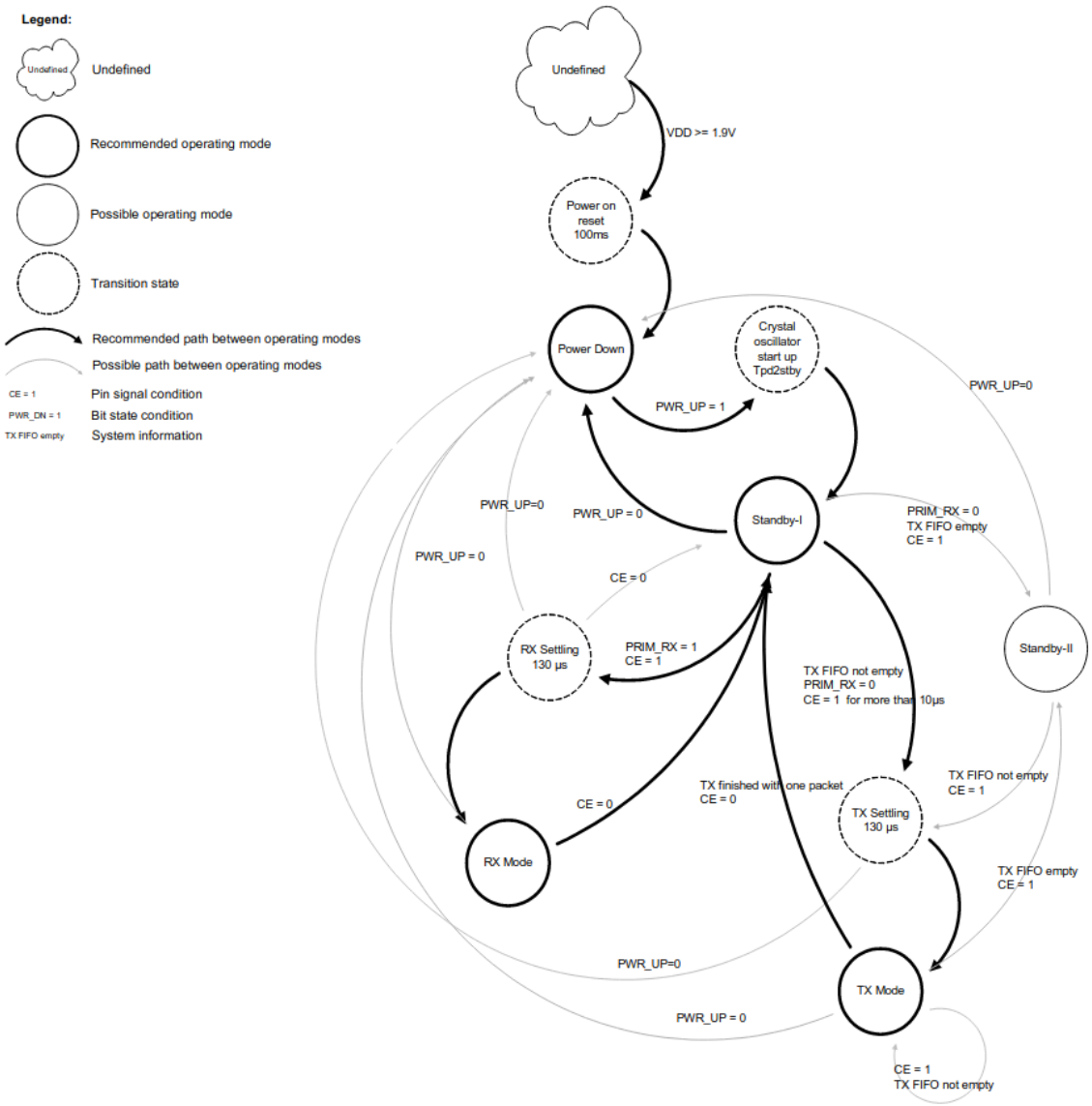
6.1.5 TX mode

V tomto módu se zůstává, dokud není odeslán celý zásobník *TX FIFO* po paketech. Poté se vrátí do módu *Standby-II* nebo do *Standby-I* pokud nastavíme pin *CE* na nulu. Je důležité nemít *TX* režim sepnutý déle než 4ms. Pokud ovšem používáme protokol vytvořen pro tento čip (Enhanced ShockBurst™), tak k tomu nikdy nedojde.

Bity k aktivaci módu:

```
PWR_UP = 1
PRIM_RX=0
TX FIFO zásobník naplněn daty
CE = 1
```

Stavový automat celého procesu je vidět na obrázku [6.1].



Obr. 6.1: Stavový automat procesu bezdrátového modulu *nRF24L01+* [2]

6.2 RF frekvenční kanál

nRF24L01+ dokáže pracovat na frekvencích od 2,4Ghz do 2,525 Ghz, přičemž tuto frekvenci můžeme měnit s rozlišením 1Mhz. Tím dokážeme pracovat na 125 různých frekvencích a na každé frekvenci můžeme určit 6 adres, tzn. každá frekvence může mít 6 zařízení. Hodnotu frekvence nastavujeme v registru *RF_CH* podle vzorce : $F_0 = 2400 + RF\_CH[MHz]$.

6.3 Zesilovač vysílače

V bitech *RF_PWR* v registru *RF_SETUP* lze nastavit úroveň zesílení vln. Úrovně a příslušné bity shrnuje tabulka [6.2].

SPI RF-SETUP (RF_PWR)	RF output power	DC current consumption
11	0dBm	11.3mA
10	-6dBm	9.0mA
01	-12dBm	7.5mA
00	-18dBm	7.0mA

Obr. 6.2: Tabulka výstupního zesílení vysílače *nRF24L01+* [2]

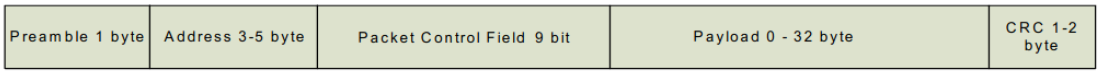
6.4 komunikační protokol Enhanced ShockBurst™

Protokol určující, co obsahuje paket dat vysílaný pomocí *nRF24L01+*. Zahrnuje také automatické přijímaní a odesílání paketů, 6 adresových příjemců a jiné. Zde si řekneme jen podstatné náležitosti, které budou potřebné k používání.

Proces odesílaní zpráv probíhá podle protokolu následovně. Vysílač (*PTX* = Primary Transmitter = primárně vysílač) pošle paket dat do přijímače (*PRX* = Primary Receiver = primárně přijímač). *PTX* poté přepne do módu *RX* a čeká na Acknowledgment paket (*ACK* paket), což je paket pro potvrzení přijetí zprávy. Pokud *PRX* zprávu bez chyb přijal, přepne se do módu *TX* a odešle *ACK* paket odesílateli. Pokud *ACK* paket není přijat u *PTX* v určitém časovém intervalu, *PTX* posílá zprávu znova dokud *ACK* paket nepřijde od *PRX*. Lze nastavit kolikrát se zpráva odešle znova. Spolu s *ACK* paketem je možné odesílat také uživatelská data. Je možné nastavit manuální odesílání *ACK* paketu.

Formát paketu je zobrazen na obrázku níže 6.3.

Obsahuje:



Obr. 6.3: Uspořádání paketu nRF24L01+ [2]

- úvodní bajt – Preamble byte, který je k synchronizaci demodulátoru s *PRX*,
- adresu příjemce,
- řídící paket s 9 bity určující délku posílaných uživatelských dat, zda je paket poslán po první nebo nikoliv a bit *NO_ACK* určující, zda má být *ACK* paket automaticky odesílán.
- Uživatelská data v rozmezí 0 až 32 bajtů. Lze nastavit, jestli bude délka dat statická nebo dynamická (proměnlivá). Statická délka je nastavena pomocí registru *RX_PW_P0* až *RX_PW_P5* (pro každou použitou adresu zvlášť) na straně *PRX* a délka dat u vysílače v *TX FIFO* musí být stejná jako nastavená v *RX_PW_Px*.
  V případě, že nastavíme dynamickou délku dat, musíme povolit v registru *FEATURE* bit *EN_DPL* a taky musí být povolen bit *DPL_Px* v registru *DYNPD* pro příslušnou adresu. Všechny tyto bity musí být nastaveny u *PTX* i *PRX*.
- *CRC* bajty pro validaci paketu u přijímače.

6.4.1 Auto Acknowledgement

Automatické odesílání potvrzovacího paketu se povoluje v registru *EN_AA* a je potřeba jej nastavit pro každou adresu zvlášť, defaultně je nastaveno na jedna, tzn. auto *ACK* je povoleno. V odeslaném paketu v řídících bitech je pak nutné mít nastaven bit *NO_ACK* na nulu, defaultně je tak nastaveno. Pokud chceme poslat uživatelská data s *ACK* paketem, musíme povolit bit *EN_ACK_PAY* v *FEATURE* registru. Příkazem *W_ACK_PAYLOAD* nahrajeme na straně *PRX* do *TX FIFO* zásobníku data, které chceme odeslat a poté, co se příchozí zpráva potvrdí, *ACK* paket s daty se odešle.

6.4.2 Auto Retransmission (ART)

Tato funkce je použita v případě, že do *PTX* nedojde *ACK* paket a je nutné paket odeslat znova. V registru *SETUP_RETR* lze nastavit, kolikrát se má paket odeslat znova. Pokud se paket přepošle maximálním počtem opakování, nastaví se bit *MAX_RT* v registru STATUS a je potřeba ho manuálně resetovat. Pokud je

ACK paket přijat, nastaví se *TX_DS* v registru STATUS na 1 a musí se resetovat zapsáním 1 do bitu. Informace o přeposlání paketu se může sledovat v registru *OBSERVE_TX* v bitech *PLOS_CNT*, který se inkrementuje po ztracení paketu a *ARC_CNT*, který počítá počet přeposlání paketu. Lze také přeposlat paket manuálně pomocí příkazu *REUSE_TX_PL*.

6.4.3 MultiCeiver ™

Tato funkce zajišťuje adresování *PRX* zařízení. Na jednom frekvenčním pásmu je možno adresovat 6 různých zařízení. V registru *EN_RXADDR* se povoluje datový proud (adresa) pro každé zařízení zvlášť, defaultně je povolena 0. a 1. adresa. Adresu definujeme v registrech *RX_ADDR_P0* až *RX_ADDR_P5* a musí být maximálně 5 bajtů dlouhá. Pokud chceme data posílat (*PTX mód)*, adresu zařízení napíšeme do registru *TX_ADDR*, přičemž musíme nastavit příslušnou frekvenci v registru *RF_CH*, stejně tak v případě, že na data čekáme, tedy u *PRX* modulu.

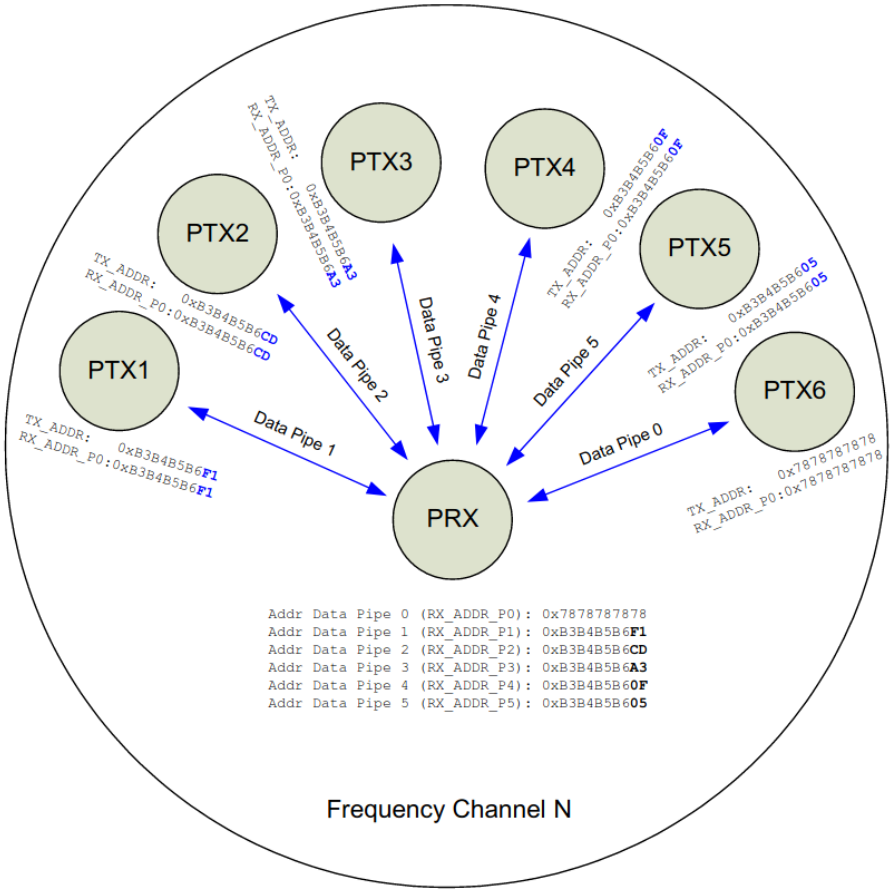
Schéma propojení uzlů (zařízení) je možno vidět na obrázku 6.4. U *PTX1* zapíšeme do registru *TX_ADDR* a do *RX_ADDR_P0* adresu shodnou s registrem *RX_ADDR_P1* v uzlu *PRX*. Takto postupujeme u všech dalších *PTX* uzlů s tím, že u PRX zapisujeme vždy do dalších datových proudů (*Data Pipe*) neboli adres, aby se *ACK* pakety odesílaly na správné zařízení.

V případě, že chceme napřímo propojit uzly PTX1 a PTX2, pojmenujeme PTX2 jako PRX2 a bude v tuto chvíli fungovat jako přijímač. Postup je takový, že u PTX1 nastavíme registry *TX_ADDR* a *RX_ADDR_P1* na stejnou hodnotu jako registr *RX_ADDR_P1* u PRX2. Zároveň musí být adresa jiná než již použité. Otevřeme tím další data pipe u obou uzlů. Adresy v registrech *RX_ADDR_P0* necháme stejné pro zachování původní komunikace. Tímto způsobem můžeme na stejné frekvenci rozšiřovat systém libovolně.

6.5 Řízení dat v nRF24L01+

K řízení nRF24L01+ se používá sériová sběrnice SPI, která je dostupná u mnoha MCU. Řízení probíhá pomocí pinů, vzhledem k MCU platí:

- IRQ – pin pro přerušení, přiřadí se libovolný digitální vstupní pin do MCU
- CE – pro aktivaci TX a RX módu, přiřadí se libovolný digitální výstupní pin z MCU
- CSN – SPI signál
- SCK – SPI signál
- MOSI – SPI signál



Obr. 6.4: Princip adresování uzlů v systému [2]

- MISO – SPI signál

Komunikace s nRF24L01+ po SPI je prostřednictvím příkazů. Po sběrnici pošleme danou hodnotu představující příkaz, nRF24L01+ toto číslo přečte, zjistí, zda se o příkaz jedná a vykoná ho. Například pro zapisování do registrů slouží příkaz W_REGISTER. Hodnotu tohoto příkazu upravíme podle adresy, do které chceme zapisovat a pošleme ji přes SPI spolu s hodnotou, kterou chceme uložit do registru. nRF24L01+ tuto zprávu přečte, zjistí, že se jedná o zapisování a hodnotu uloží na požadované místo. Tabulku s příkazy a všemi registry je možné vidět v technické dokumentaci k nRF24L01+ na stranách 51 a 57 [2].

6.6 Příklad komunikace mezi dvěma moduly nRF20L01+

Níže je ukázáno, co vše je potřeba nastavit pro komunikaci mezi dvěma nRF24L01+ a kód ve Forthu s příkazy z knihovny pro ovládání nRF24L01+ [1].

6.6.1 Příkazy pro vysílač

potřebné registry

```
sepnutí Standby-I režimu
PWR_UP = 1
PRIM_RX = 0
ENAA_P0 = 1
TX_ADDR = "00002"
RX_ADDR_P0 = "00002"
TX_PLD = " hello word"
```

```
\ PWR_UP = 1
radio-on

\ PRIM_RX = 0
radio-tx+

\ ENAA\_P0 = 1
0 auto-ack+

\ TX_ADDR = "00002"
s" 00002" tx-addr!

\ RX_ADDR_P0 = "00002"
s" 00002" 0 rx-addr!

: posilej 20 0 do
\ TX_PLD = " hello word"
s" hello word" 32 >tx-fifo

\ nastavíme pin CE na 1 a tím zahájíme komunikaci
startx

delay 1000 ms

loop;
```

6.6.2 Příkazy pro přijímač

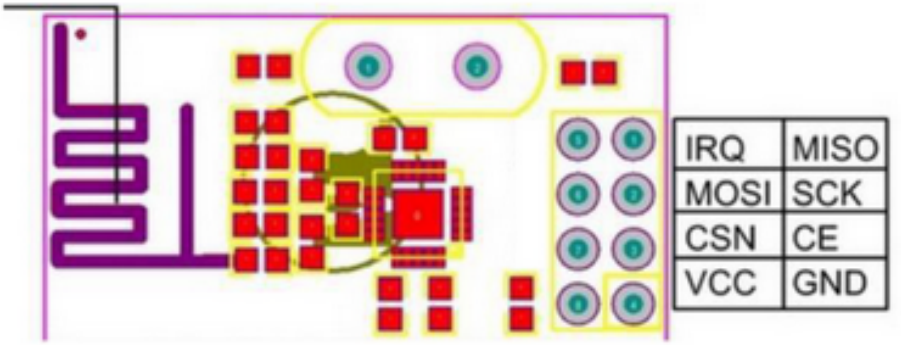
potřebné registry

```
sepnutí Standby-I režimu
PWR_UP = 1
PRIM_RX = 1
ERX_P0 = 1
ENAA_P0 = 1
RX_PW_P0 = 32 (bajtů)
RX_ADDR_P0 = "00002"
```

```
\ PWR_UP = 1
radio-on

\ PRIM_RX = 1
radio-rx+

\ ENAA_P0 = 1
0 auto-ack+

\ ERX_P0 = 1
0 pipe+

\ RX_ADDR_P0 = "00002"
s" 00002" 0 rx-addr!


create rx_data 32 allot

:  begin startx rx-data? if
rx_data 32 <rx-fifo
then

key?
until key drop ;
```

6.7 Zapojení modulu

Piny připojíme na SPI sběrnici MCU. CSN pin je často označován na MCU jako SS. Piny IRQ a CE připojíme na kterýkoliv digitální vstup MCU a poté v kódu upravíme podle toho. Zvýšenou pozornost musíme věnovat zapojení VCC, které je 3,3V, při vyšším napětí dojde ke zničení modulu. Logické signály mohou být 5V. Rozložení konektoru k modulu je na obr. 6.5.



Obr. 6.5: Rozložení pinů modulu [3]

7 Popis implementace

7.1 Představení sítě

Cílem bylo vytvořit síť senzorů, které budou moci jeden druhého ovládat. Snaha byla vytvořit vzdálený příkazový řádek, přes který by bylo možné posílat dotazy na výpis hodnot ze senzoru nebo i vytvářet nové funkce. Tento vzdálený přístup k MCU je omezen na několik věcí. Nedokážeme nahrávat ze souboru, ale musíme nové funkce ručně vypisovat do klíčové sekvence příkazů na poslání dat: `S" data to send " :tell`. Nelze do definic nových slov vepisovat string, jako například v tomto případě: `S" : secti + ." vysledek scitani je : " . CR ; " :tell`.

V síti standardně není žádný master a slave, každý může poslat data na kterýkoliv node (senzor) v síti. Pokud bychom chtěli jeden node, který bude sbírat informace od ostatních, musel by být jen jeden. V případě dvou a více takových nodů, by bylo nutné dodělat další funkce, které by zaručovaly, že v síti bude posílat dotazy vždy jen jeden master. Funkce, které děláme pro to, abychom pak jejich výsledky odesílali po síti, musí mít výsledek maximálně 95 bajtů. Tyto hodnoty se musí po vykonání objevit na zásobníku, odkud jsou pak automaticky odeslány. Na nultý index těchto dat pak ukládáme celkový počet znaků, dohromady tedy 96 bajtů dat. Dále může funkce obsahovat popis výsledku, ale je nutné ho v těle funkce zapsat do bufferu `s-buff` a to následujícím způsobem: `S" string to send back" s-buff swap st-str`. String musí mít maximálně 95 znaků, kde opět nultý index tohoto pole je celkový počet znaků. V případě, že nastala chyba při vykonávání příkazu na vzdáleném nodu, se zpět nepošle nic, neboť Forth interpreter vyprázdní návratové hodnoty a neví tedy, která další funkce je na řadě. MCU se tak nezasekne, ale už neví, že má odeslat odpověď v podobě zprávy, že se dotaz neprovedl. V tomto případě je nutné procesor restartovat a funkcemi, které se provedou po restartu, se nastaví potřebné záležitosti pro další příjem.

Síť může mít maximálně 31 senzorů a to z důvodu ladění kódu na MCU s malou pamětí. Jde ji ovšem po krátkém upravování předělat na větší.

Poté, co je síť vytvořena a senzor správně nastaven, zahájíme komunikaci následující sekvencí příkazů:

```
1 init-searching start-searching
S" data-to-send" :tell
```

7.1.1 Buffery k odesílání a hledání

Pakety dat, které posíláme na senzory jsou následující: CA, TA, HA, d-buff, s-buff.

CA buffer

Control Array

Pole CA má 4 bajty a jejich názvy jsou:

```
Control Array:
Index:  0             1      2         3
        searching?    TAP    +-cell    FRXN
```

Nultý bajt říká, zda chceme hledat finální node nebo už máme trasu nalezenou a jen data přeposíláme. TAP je pointer, který používáme, když trasa je nalezena. Podle +-cell se TAP upravuje a v FRXN je uložena aktuální adresa (jméno) nodu, na který posíláme data. FRXN se změní pokaždé, když finálního nodu dosáhneme, tzn. pokud inicializujeme vyhledávání, nastavíme FRXN na node, na který posíláme dotaz a ve chvíli nalezení tohoto nodu v síti, se změní FRXN na node, ze kterého jsme původně data poslali.

TA buffer

Trace Array

Obsahuje trasu finálního nodu. Adresy senzorů jsou do něj přidávány postupně během prohledávání sítě a ukládání nebo mazání probíhá podle toho, zda nejsme ve slepé větvi sítě. Jeho inicializace je pokud zahajujeme novou komunikaci funkcí INIT-searching. Pokud je trasa nalezena, nastavujeme adresy následujícího nodu pomocí tohoto pole a ukazatele TAP.

TAP se inicializuje ve chvíli, kdy vyhledáváme finální node a právě se nacházíme v nodu, který má finální node ve svém okolí. V tuto chvíli se nastaví TAP na celkový počet prvků v poli TA a tím ukazuje na FRXN. Tzn., pokud máme v TA uloženy prvky 10 12 8 5, tak TAP bude rovno 4, přičemž node 5 je FRXN a node 10 je ten původní, ze kterého data posíláme.

HA buffer

History Array

Slouží k zapamatování, na kterých senzorech jsem již hledal finální node. Pokaždé, když přijdou data a algoritmus je ve funkci hledání, se pole HA porovná s polem LNA a podle toho určí, na který node se vydá dál.

s-buff

String Buffer

Buffer o velikosti 96 bajtů je určen pro uložení stringu neboli popisu výsledku, který odesíláme společně s čísly z dotazovaného senzoru. O jeho uložení do bufferu se stará programátor, který vytváří konkrétní funkce pro odesílání dat po síti. Například pokud máme senzor teploty a vytváříme funkci pro měření a následné posílání hodnot po síti, musí být věta pro popis výsledku uložena v této funkci v tomto bufferu, způsobem již zmíněným výše.

d-buff

Data Buffer

Směrem k finálnímu nodu se do něj uloží string obsahující funkce, které se pak provedou a zpět se posílají čísla. Ty se pak jen přesunou na zásobník. Jeho velikost je 96 bajtů.

LNA

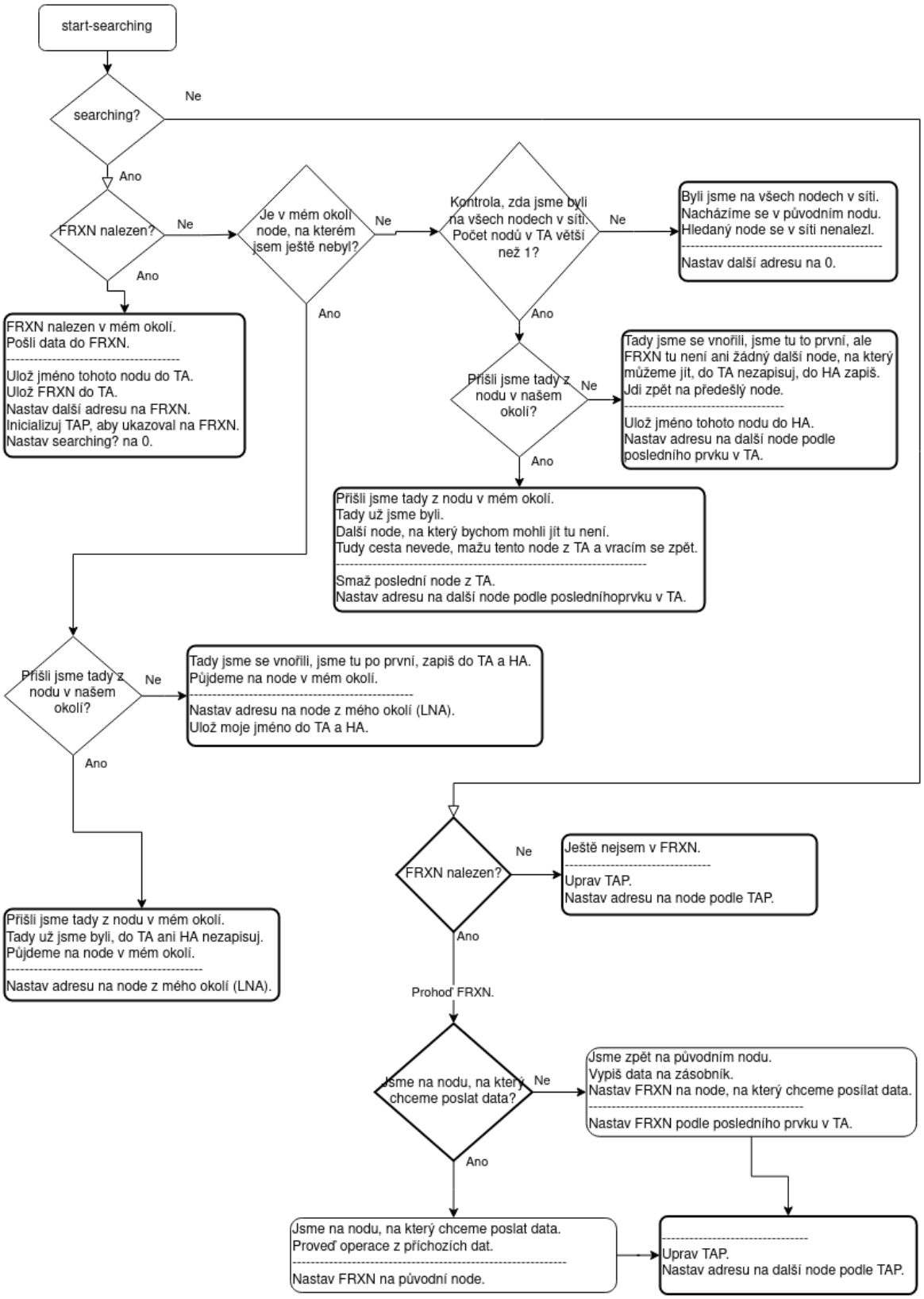
Local Node Array

Pole obsahující názvy (adresy) senzorů, které jsou v dosahu a můžeme na ně posílat data.

7.1.2 Stavový diagram *start-searching*

Funkce pro vyhledávání (start-searching) se vykonává poté, co máme připravené buffery CA, TA, HA a LNA. Ty jsou inicializovány (kromě LNA) ve chvíli, kdy odesíláme data po první, tzn. ve chvíli spuštění funkce INIT-searching, a nebo když dorazí nové data na senzor. Tím se přepíše původní obsah bufferů a může se podle nich nastavit následující adresa, kde se budou data posílat.

Princip vyhledávání finální trasy a následné přeposílání dat mezi nodem, ze kterého posílám data (původní node) a příjemcem je na obr. 7.1. Výsledná trasa je uložena v poli TA a to se nepřepíše, dokud není opětovně spuštěna funkce INIT-searching.



Obr. 7.1: Stavový diagram pro nalezení trasy k senzoru

7.1.3 Vytvoření sítě

Síť musíme vytvořit manuálně a musíme znát jak dosavadní síť vypadá v prostoru, tedy jaké mají senzory adresy a kde se nachází. Podle toho pojmenujeme nový senzor unikátním jménem (adresou). Zápis nového senzoru do sítě provádíme tak, že se připojíme ke kterémukoliv senzoru v síti a pomocí funkce wr-node zapisujeme do všech nodů, které jsou v okolí nově přidávaného senzoru.

```
node-in-area-of-new-node init-searching start-searching
S" new-node wr-node" :tell
```

Mazání senzoru ze sítě provedeme stejným způsobem s použitím funkce del-node.

```
node-in-area-of-deleting-node init-searching start-searching
S" node del-node" :tell
```

7.2 Popis zdrojového kódu

Níže budou vysvětleny všechny důležité funkce pracující se sítí. Kompletní zdrojový kód je možné vidět v přiloženém souboru distributed-library.frt.

Vysvětlivky:

```
: název-definice  \ krátký popis
                  ( vst -- výst ) in/out argumenty celé definice
    code          ( -- )   in/out argumenty slov v daném řádku
    code          (  )     výstupní argumenty slov v daném řádku
    ;
Pořadí čísel na zásobníku:
1 2 3   ( -- 1 2 3 )
swap    ( 1 2 3 -- 1 3 2 )
```

Funkce pro vypsání jména nodu, na kterém se nacházíme.

```
: my-name   \ for setting the name of this node
```

Vypíše na zásobník adresy jednotlivých proměnných z CA pole.

```
: searching?    \ flag if algorithm is searching for node
: TAP           \ Trace Array Pointer
: +-cell        \ variable for moving forward or backward on TA
: FRXN          \ Final RX Node
```

Funkce pro úpravu TAP pointeru. Pokud zavoláme některou z funkcí, do +-cell se uloží hodnota 1, -1 nebo se hodnota vynásobí -1. Díky tomu se pak při zavolání funkce TAP-adjust TAP o jedno přičte nebo odečte podle toho jaká hodnota se aktuálně nachází v +-cell.

```
: minus-cell
: plus-cell
: neg-cell      \ negate +-cell --> -1 * +-cell
: TAP-adjust    \ adjustment of TAP
```

Tato funkce porovnává jedno-bajtové hodnoty ze dvou polí. Nejprve vezme 1. prvek z pole1 a začne ho porovnávat s prvky pole2. Pokud nenajde shodu, nechá tento prvek z pole1 na zásobníku. Pokud najde shodu, přestane dál porovnávat prvky v poli2 a vezme další prvek z pole1. Pokud najde shodu každý prvek z pole1 v poli2, zanechá 0 na zásobníku. Používá se pro porovnávání polí LNA a HA. Vybere tak, na který node se vydají data.

```
: c-arr-arr \ compare arrays - compare elements of arrays
            \ only for one byte element array!
            \ ( addr1 addr2 -- nodeID-OR-0 )
```

Stejný princip jako c-arr-arr.

```
: c-arr-var \ compare array with variable
            \ only for one byte element array and variable!
            ( arr-addr var-addr -- nodeID )
```

Zapíše prvek na vrchol pole a upraví celkový počet prvků v poli.

```
: wr-arr   \ write value to array
           \ only for one byte element array and variable!
           ( val arr-addr10 -- )
```

Odstraní prvek z vrcholu pole a upraví celkový počet prvků v poli.

```
: d-e-arr \ delete element from array
\ only for one byte element array!
( arr-addr10 -- )
```

Přečte prvek z vrcholu pole.

```
r-arr    \ read last nodeID in array
         ( arr-addr10 -- nodeID )
```

Zapíše jméno nodu do LNA.

```
: wr-node   \ write node to LNA
            ( nodeID -- )
```

Smaže jméno nodu z LNA.

```
: del-node   \ delete node from LNA
             ( nodeID -- )
```

Přečte jméno nodu pomocí ukazatele TAP z pole TA.

```
: r-TA   \ read element from TA by TAP
         ( -- nodeID )
```

Přesune bajty z pole1 do pole2 a uloží celkový počet bajtů do nultého indexu pole2. Používá se pro ukládání stringů do bufferů s-buff a d-buff.

```
: st-str    \ store string to array
            \ 0. byte is length of string
            ( addr1 addr2 len -- )
```

Inicializace a zahájení komunikace. Nastaví se FRXN, searching? na 1, vynuluje pole HA a uloží se do něj jméno aktuálního nodu a vynuluje pole TA. Algoritmus start-searching tak začne hledat novou cestu k cíli.

```
: INIT-searching   \
                   ( FRXN-name -- )
```

Funkce je podrobně popsána v kapitole 7.1.2 a pomocí stavového diagramu na obr. 7.1.

```
: start-searching    \
                     ( -- )
```

Nastaví se bezdrátový modul k odesílání.

```
: init-transmit    \ Inicialialization of receiving
                   ( -- )
```

Nastavení bezdrátového modulu k přijímání dat. Funkce by se měla spustit po každém restartu. Volá se taky poté, co odešleme data, abychom pak následně mohli data přijímat.

```
: init-rec   \ inicialization of receive
             ( -- )
```

Funkce k zahájení odesílání. Pokud se dvakrát nepodaří pakety odeslat, vypíše chybovou hlášku a celý proces se ukončí.

```
: retran   \ retransmit until data sent with cca 1 sec timeout
           ( -- )
```

V případě, že je adresa nastavena na 0, nic neprovede, v opačném případě postupně nahrává a odesílá data na další node. Po odeslání všech paketů zavolá init-rec, která nastaví MCU na přijímání dat.

```
: transmit   \
             ( -- )
```

Kontroluje, zda nepřišly data s časovým omezením, který je nastaven přibližně na 1 sekundu. Po uplynutí se proces ukončí.

```
: dat-check   \ check for data
              ( -- )
```

Funkce na zpracování přijatých dat na FRXN. Nejprve zjistí, jestli se nachází na původním nodu nebo na příjemci a podle toho buďto data vykoná a nahraje do bufferu nebo je z bufferu přesune na zásobník.

```
: ex-data   \ execute data
            ( -- )
```

Příjem dat se spustí pomocí externího přerušení z bezdrátového modulu. Postupně načítá data do vnitřní paměti RAM. Poté, co je všechny přijme, zkontroluje zda je na FRXN a pokud ano, data jsou vyhodnocena. Pak zjistí, jestli se nacházíme na původním nodu a pokud ne, data odešle.

```
: receive   \
            ( -- )
```

Na zásobník se pomocí S""uloží adresa a počet bajtů. Funkce :tell tyto parametry vezme a uloží do d-buff, pak spustí transmit.

```
: :tell   \
          ( addr10 len -- )
```

Tato Funkce se provádí jen po restartu. Nastaví MCU na příjem dat.

```
: startup-process   \
                    ( -- )
```

7.3 Zprovoznění AmForthu na MCU

Celý kód se ladil na AmForth distribuci[1], procesoru ATmega328P a s použitím knihovny pro nRF24L01+ [1]. Nahrání bootloaderu do MCU se provádělo pomocí připraveného make-file, který dokáže přepisovat i pojistky MCU v případě potřeby. Nahrávání souborů do čipu pak probíhalo díky sériové lince, která po každé instrukci

[1] AmForth – `https://amforth.sourceforge.net`

čekala na zpětnou vazbu od MCU. Tím probíhalo nahrávání pomaleji, ale bez potíží. Konkrétní popis a použití obou programů je možné vidět na AmForth webových stránkách.

Závěr

Cílem této bakalářské práce bylo vytvořit knihovnu slov pro snadnou implementaci chytrého senzoru umožňující provádět distribuované zpracování dat. Následně pak funkčnost knihovny demonstrovat několika vzájemně komunikujícími jednotkami.

Nejprve jsem čtenáře seznámil s jazykem Forth. V kapitole popisuji přehled příkazů pro základní používání. Věnuji se vysvětlení a definici Slova, výpisu znaků, aritmetickým výpočtům, zdrojovému souboru, IF .. ELSE .. THAN řídící struktuře apod.

Dále jsem se věnoval variantám Forthu pro MCU, zvolil jsem si konkrétně FlashForth a AmForth. Jednotlivé varianty jsem se snažil popsat tak, abych charakterizoval jejich vlastnosti a rozdíly.

Dále popisuji jednočipový mikropočítač a jeho dělení. Tato kapitola slouží k vysvětlení pojmů jako MCU, RAM, flash a podobně a také je úvodem pro kapitolu Chytré senzory.

U definice chytrých senzorů se vycházelo ze senzorů pracujících s mikropočítačem, který byl spojen sběrnicemi např. A/D převodníku a který pak komunikuje s nadřazeným mikropočítačem pomocí distribuovaných výpočtů.

Také jsem popisoval distribuované systémy a výpočty. Přesto, že toto téma je velmi rozsáhlé popsal jsem ho pouze krátce a zaměřil se spíše na řešení multitaskingu. V této práci se pak věnuji distribuovaným systémům jen z hlediska přenosu dat a vzdáleného ovládání.

Dále jsem se věnoval bezdrátovému modulu nRF24L01+, způsobem jeho ovládání, principem fungování a příklady použití. Popsal jsem módy nRF24L01, Power Down Mode, Standby-l mode, Standby-ll mode, RX mode a TX mode. Věnoval jsem se také RF frekvenčnímu kanálu, zesilovači vysílače, komunikačnímu protokolu Enhanced ShockBurst, řízení dat v nRF24L01+ a zapojení modulu. Uvádím jednoduché příklady komunikace mezi dvěma moduly nRF24L01+.

V implementační části se věnuji vytvoření sítě a její prohledávání. Cílem bylo vytvořit plně funkční vzdálený přístup k senzoru (remote shell). Nakonec má však síť několik omezení v podobě malého maximálního počtu senzorů v síti, omezené vzdálené definování nových slov a nebo buffery pro data s malou velikostí. Ladění kódu probíhalo na čipu ATmega328P s distribucí AmForth, která nabízí spoustu připraveného kódu, make file pro bootování MCU nebo i sériovou linku pro nahrávání souborů. Celá implementace by se dala vylepšit automatickým přidáváním senzorů do sítě, zvětšením maximálního počtu senzorů v síti nebo například šifrováním dat.

Literatura

[1] Knihovna v jazyce FORTH pro nRF24L01+, 2006. In: *Amforth: Interpreter on Microcontrollers* [online]. 2014 [cit. 2022-05-13]. Dostupné z: https://sourceforge.net/p/amforth/community/HEAD/tree/nRF24L01+/

[2] *Technická dokumentace k nRF24L01+: Product Specification v1.0*, 2008. Revision 1.0. Trondheim, Norsko: Nordic Semiconductor. Dostupné také z: https://infocenter.nordicsemi.com/topic/struct_nrf24/struct/nrf24L01p_ps.html

[3] Bezdrátový modul NRF24L01, 2023. *LáskaKit* [online]. [cit. 2023-02-27]. Dostupné z: https://www.laskakit.cz/bezdratovy-modul-nrf24l01–2-4ghz/

[4] DOC. ING. KLIMEŠ, CSC., Cyril. *Distribuované systémy: texty pro distanční studium*. Ostravská univerzita, Ostrava. Dostupné také z: https://web.archive.org/web/20140714151456/http://www1.osu.cz/ prochazka/ds/SkriptaKlimes.pdf. Skriptum. Ostravská univerzita v Ostravě, Přírodovědecká fakulta Katedra informatiky a počítačů.

[5] *Programming Forth* [online], 2011. 4th. 133 Hill Lane, Southampton: MicroProcessor Engineering Limited [cit. 2022-01-02]. ISBN 978-0-9525310-5-0. Dostupné z: https://www.mpeforth.com/arena/ProgramForth.pdf

[6] Strukturované programování, 2020. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2022-01-01]. Dostupné z: https://cs.wikipedia.org/wiki/Strukturovan%C3%A9_programov%C3%A1n%C3%AD

[7] The Evolution of Forth, 2018. *FORTH, Inc.* [online]. [cit. 2022-01-01]. Dostupné z: https://www.forth.com/resources/forth-programming-language/

[8] *Rešerše jednočipových mikroprocesorů* [online], 2012. Plzeň [cit. 2021-12-30]. Dostupné z: https://dspace5.zcu.cz/bitstream/11025/2755/1/BP_SOUKUP_TOMAS_E09B0052K.pdf. Bakalářská práce. ZÁPADOČESKÁ UNIVERZITA V PLZNI FAKULTA ELEKTROTECHNICKÁ.

[9] Von Neumannova architektura, 2001-. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2021-12-30]. Dostupné z: https://cs.wikipedia.org/wiki/Von_Neumannova_architektura

[10] Harvard architecture, 2001-. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2021-12-30]. Dostupné z: https://en.wikipedia.org/wiki/Harvard_architecture

[11] Charles H. Moore, 2001-. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2021-12-27]. Dostupné z: https://en.wikipedia.org/wiki/Charles_H._Moore

[12] BRODIE, Leo. *Starting FORTH: an introduction to the FORTH language and operating system for beginnersand professionals* [online]. Englewood Cliffs, N.J.: Prentice-Hall, c1981 [cit. 2019-09-13]. ISBN 01-384-2930-8.Dostupné z: https://www.forth.com/wp-content/uploads/2018/01/Starting-FORTH.pdf

Seznam symbolů a zkratek

ALU aritmeticko-logická jednotka

OS operační systém

MCU microcontroller unit - jednočipový mikropočítač

PTX Primary TX - primárně vysílač

PRX Primary RX - primárně přijímač

ACK paket Acknowledgement - potvrzovací paket

ART Auto Re-Transmit- automatické přeposlání paketu

SPI Serial Peripheral Interface - Sériová sběrnice

Mbps Megabit per second - mega bitů za sekundu - přenosová rychlost

PWR_DWN Power Down - vypnutí

PWR_UP Power Up - zapnutí

RX Receive - přijmout

TX Transmit - vysílat

RX_DR Receive Data Ready - přijaté data připraveny

TX_DS Transmit Data Sent - vysílané data odeslány

CA Control Array

TA Trace Array

HA History Array

LNA Local Node Array

s-buff String Buffer

d-buff Data Buffer

A Obsah elektronické přílohy

- `/` ..... kořenový adresář přiloženého archivu
  - `xkudel11-BP.pdf` ..... Text bakalářské práce
  - `distributed-library.frt` ..... Knihovna pro práci se senzory
  - `nrfl.frt` ..... Knihovna pro obsluhu bezdrátového modulu nrf24l01+