VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Implementace konečného stavového automatu s použitím technologie NXP FlexIO

Implementation of a Finite State Machine using NXP FlexIO Technology

2018 Bc. Libor Chrástecký

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Zadání diplomové práce

Student: **Bc. Libor Chrástecký**

Studijní program: N2649 Elektrotechnika

Studijní obor: 2612T041 Řídicí a informační systémy

Téma: Implementace konečného stavového automatu s použitím technologie NXP FlexIO
Implementation of a Finite State Machine using NXP FlexIO Technology

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Seznámení se s funkcí vývojových nástrojů (vývojový kit MCU, vývojové prostředí Eclipse, kompilátor GCC, debugger).
2. Seznámení se s architekturou MCU NXP Kinetis L.
3. Seznámení se s technologií NXP FlexIO.
4. Seznámení se s metodami pro formátování dat přenášených bajtově orientovaným komunikačním kanálem (framing).
5. Návrh konečného stavového automatu (FSM) rozpoznávajícího framing.
6. Softwarová implementace FSM.
7. Posouzení možností implementace FSM s použitím FlexIO.
8. Návrh FSM s použitím FlexIO.
9. Implementace FSM s použitím FlexIO. Nalezení alternativního řešení, pokud vlastnosti FlexIO neumožní implementaci ve stanoveném rozsahu.
10. Testování algoritmu, porovnání efektivity softwarové a FlexIO implementace, diskuze technických omezení FlexIO. Zhodnocení výsledků práce.

Seznam doporučené odborné literatury:

[1] HEROUT, Pavel. *Učebnice jazyka C. 1. díl.* 4. přeprac. vyd. České Budějovice: Kopp, 2004. ISBN 80-7232-220-6.
[2] YIU, Joseph. *The Definitive Guide to ARM Cortex-M0 and Cortex-M0+ Processors.* 2nd Edition. Waltham: Newnes, 2015. ISBN 978-0128032770.
[3] YIU, Joseph. *The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors.* 3rd Edition. Waltham: Newnes, 2013. ISBN 978-0124080829.
[4] OSHANA, Robert. *Software Engineering for Embedded Systems.* 1st Edition. Waltham: Newnes, 2013. ISBN 978-0124159174.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Stankuš, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jiří Koziorek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení

„Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal."

V Ostravě dne 30. 4. 2018

Podpis autora: .....

Poděkování

Děkuji vedoucímu diplomové práce Ing. Martin Stankušovi, Ph.D za věcné rady, shovívavost a za odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Ostravě dne 30. 4. 2018

Podpis autora: .....

Abstrakt

Cílem této práce je návrh a implementace konečného stavového automatu prostřednictvím periférie FlexIO. Práce se zabývá teorií kolem funkcí vývojových nástrojů jako jsou kompilátor, vývojové prostřední na bázi prostředí Eclipse a nástroje pro ladění kódů. Popisuje architekturu mikrokontrolerů Kinetis L od společnosti NXP. Následně práce popisuje funkčnost a vlastnosti technologie NXP FlexIO, kde se zaměřuje na její jednotlivé funkční módy a nejvíce na použitý stavový mód. Poté práce nastíní některé možné protokoly používané pro rozpoznávání zpráv na sériových linkách. V práci se následně řeší návrh a implementace SW konečného stavového automatu, po návrhu a implementaci SW návrhu, práce nastíní možná úskalí, která přináší použití periférie FlexIO při návrhu stavového automatu. Poté se v práci popíše návrh a implementace stavového automatu prostřednictvím periférie FlexIO. Po návrhu obou konečných automatů nastíní testování jednotlivých algoritmů a následně vyhodnocení a porovnání výsledku testování obou návrhů.

Klíčová slova

Konečný stavový automat, FlexIO, UART, DMA, FRDM-KL28Z, mikrokontroler, paměť, debugger

Abstract

The object of this thesis is the design and implementation of the final state machine via the FlexIO periphery. The thesis deals with the theory of development tools, such as the compiler, the Eclipse-based development tool, and code debugging tools. Describes the architecture of Kinetis L microcontrollers from NXP. Subsequently, the thesis describes the functionality and properties of NXP FlexIO, where it focuses on its individual modes of operation and the most used state mode. Then the thesis outlines some of the possible protocols used for message framming on serial lines. The thesis was subsequently focused on design and implementation of software finite state machine, after the design and implementation of software design work outlines the possible problems that brings peripherals FlexIO use in the design of the state machine. Then, the thesis describes the design and implementation of the state machine via the FlexIO periphery. After designing both finite automata, it will outline testing of individual algorithms and then evaluate and compare the test result of both designs.

Keywords

Finete state machine, FlexIO, UART, DMA, FRDM-KL28Z, microcontroller, memory, debugger

Obsah

Seznam použitých symbolů a zkratek

ADC	Analog to Digital Convertor
CPU	Central Process Unit
DAC	Digital to Analog Convertor
DMA	Direct Memory Access
Embedded	Vystavěný systém
ETB	Embedded Trace Buffer
FlexIO	Flexible Input/Output
FRDM-KL28Z	Freeedom board Kinetis L 28Z
FSM	Finite State Machine
GCC	GNU Compiler Collection
GDB	GNU Debugger
GNU	bezplatné operační systémy
GPIO	General Purpose Input Output
I/O	Input/Output
ITM	Instruction Trace Macrocell
Kit	platforma
LCD	Liquid-crystal display
LUT	Look Up Table
MTB	Micro Trace Buffer
OpenSDA	Open-standard Serial and Debug Adapter
Pragmas	Specifikace pro kompilátor
RTOS	operační systém reálného času
SFR	Special function register
SPI	Serial Peripheral Interface
SWD	Seriál Wire Debug
SWO	Seriál Wire Output
SWV	Serial Wire Viewer
TCD	Transfer Control Descriptor
TRGMUX	Trigger Mux Conrol
UART	Universal Asynchronous Receiver transiver

Seznam použitých obrázků a tabulek

1. Úvod

Tato práce popisuje návrh a implementaci konečného stavového automatu pomocí SW řešení a řešení s periférií FlexIO. Motivací k řešení této práce bylo navrhnout algoritmus využívající HW prostředky MCU k realizací úloh, které obvykle bývají realizovány softwarově a tím zefektivnit celé řešení těchto úloh, neboť by CPU nemusel řešit samotný stavový automat, ale pouze by mohl vyhodnocovat data. Následně navržený a implementovaný stavový automat by šlo použít v řadě aplikacích, které buď vysoce zatěžují CPU, a tak přenést část řešení této aplikace z CPU, například operací, ve kterých se ve video signálu hledají vložené signalizační značky nebo v takových aplikacích, kde naopak, kde řešení stavového automatu je jednoduché, a tedy je zbytečné, aby CPU tyto operace řešil. Pro návrh a implementaci konečného stavového automatu bude zvolena jedna z metod pro rozpoznávání zpráv přicházejících na sériové lince, a to z důvodu, že tyto metody bývají nejčastěji implementovány stavovými automaty. K tomu, aby bylo schopné správně konečný stavový automat navrhnout a implementovat bude nutné nejprve se seznámit s problematikou vývojových nástrojů, které budou následně použity k realizaci a testování konečného stavového automatu. Mezi tyto vývojové nástroje patří vývojové kity, neboli platformy zde se zaměřilo na obecné možnosti použití v průmyslu pro jednotlivé série vývojových platforem od společnosti NXP, také se zde obecně seznámí použitý vývojový kit FRDM-KL28Z. Poté práce seznámí s vývojovými prostředí na bázi prostředí Eclipse, zde se zaměří na použité vývojové prostředí Atollic TrueSTUDIO IDE. Následně seznámí s funkčností a jednotlivými části kompilátoru GCC, určeného k překladu kódu a jeho možnostmi na překlad kódu určeného pro embedded aplikace. Poté seznámí s možnostmi pro ladění vytvořeného kódu a jeho hardwarovou podporou pro ladění procesorů Cortex-M se zaměřením na použité ladící prostředí Segger Ozone. Po seznámení se s možnostmi vývojových nástrojů se v další části práce zaměřuje na možnosti a funkčnost architektury Kinetis L, tedy architekturou použité vývojové platformy FRDM-KL28Z. Dále je nutné seznámit se i s vlastnostmi periférie FlexIO a jeho jednotlivými módy se zaměřením se na stavový mód. V další části práce nastíní některé možné protokoly pro rozpoznávání zpráv. Poté již se přejde k návrhu a implementaci SW stavového automatu rozpoznávající zprávy. Dále se posoudí možnost implantace stavového automatu s použitím periférie FlexIO a přejde se k jeho návrhu a následné implementaci. Po dokončení návrhu a implementace obou algoritmů se v závěru práce otestují oba algoritmy a vyhodnotí se efektivnost obou řešení a následně se zhodnotí práce.

2. Vývojové kity

Vývojové kity od společnosti NXP jsou takové kity, které spojují nízko výkonové procesory s jádry typu Arm® Cortex®-M0+/M4/M7 a řadu dalších různých obvodových periférií. Vývojové kity jsou malé, nízko výkonové a vysoce efektivní vývojové platformy vhodné pro rychlé aplikační prototypy nebo pro demonstraci funkčnosti procesorů řady Kinetis MCU a NXP senzorů. Tyto vývojové kity se snadno používají kvůli tomu, že obsahují velkého uložiště k programování zařízení, virtuální sériový port a klasické programovací a ovládací možnosti. [1]

Mezi některé jejich přednosti patří.

- Nízká cena
- Vytvořeno dle průmyslových standardů
- Snadný přístup k vstupně/výstupným pinům procesoru
- Integrovaný otevřený standard pro sériové a ladící rozhraní (OpenSDA)

2.1. Série vývojových kitů od společnosti NXP

Společnost NXP poskytuje řadu různých vývojových kitů, tyto kity jsou děleny dle oblasti použití do několika různých sérií, každá z těchto sérií je vhodná pro některou část průmyslu.

2.1.1. V série

Tato série je určena pro aplikace s kontrolou v reálném čase, tato série používá jádra typu Cortex-M0+/M4/M7. Je navrhnutá pro širokou škálu aplikací přes BLDC, PMSM and ACIM řízení motorů až po převod digitálních energií. [2]

2.1.2. K série

Tato série je určena pro aplikace založené na výkon, tato série používá jádra typu Cortex-M4. Série obsahuje výkonné procesory s až 2 MB velkou vnitřní flash pamětí a 1 MB SRAM pamětí, dále používá pokročilé zabezpečení a má možnost připojení prostřednictvím sběrnic Ethernet, USB a CAN.[2]

2.1.3. W série

Tato série je určena pro aplikace založené na bezdrátových technologií, tato série používá jádra typu Cortex-M0+/M4. Série je použita pro systémy pro rádiové přenosy s frekvenci 1GHz až 2,4GHz, má možnosti zabezpečení komunikace, a tudíž je vhodná pro nízko výkonové bezdrátové řešení.[2]

2.1.4. E série

Tato série je určena pro aplikace založené na robustních 5V řešení, tato série používá jádra typu Cortex-M0+/M4. Série je vhodná pro vysoce výkonné a robustní systémy a také je vhodná pro prostředí s velkým rušením.[2]

2.1.5. L série

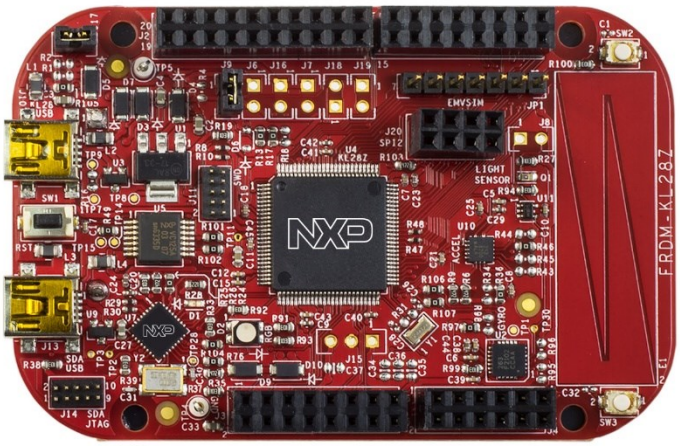
Tato série je určena pro nízko výkonové aplikace, tato série používají jádra typu Cortex-M0+. Série kombinuje nízko-výkonové operace a energetickou účinnost jádra Arm Cortex-M0+ a jejich výkonosti s řadou různých obvodových periférií, tato kombinace dělá tuto série vhodným řešení pro aplikace založené na internetu věcí. (IoT)[2]

2.1.6. EA série

Tato série je určena pro aplikace použité v automobilovém průmyslu, tato série používá jádra typu Cortex-M0+. Série je vhodná pro aplikace, použité pro automobilový průmysl, které vyžadují vysokou kvalitu a životnost.[2]

2.2. Vývojový kit FRDM-KL28Z

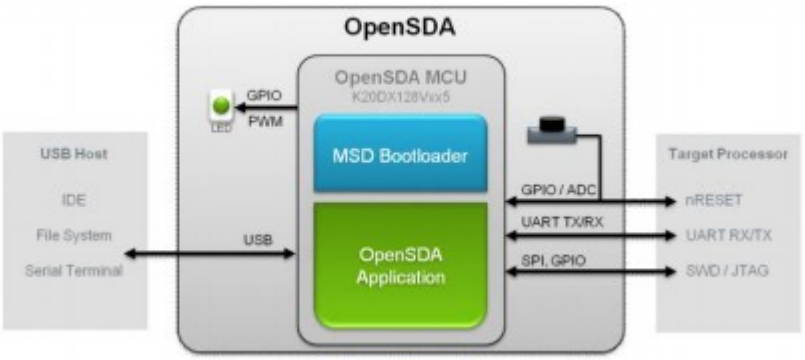
Jedná se o vývojový kit od společnosti NXP, který obsahuje procesor typu MCU MKL28Z512VLL7 na bázi jádra Arm®Cortex®-M0+ od firmy NXP spolu s řadou analogových a digitálních periférii připojených k MCU. Vývojový kit podporuje napájecí napětí v rozmezí 1,71V až 3,6V, maximální operační frekvence je 96MHz. Na desce je obsažena paměť typu flash o velikosti 512kB a paměť typu SRAM o velikosti 128kB. Mezi periférie obsažené na vývojovém kitu patří například RGB led dioda, 3-osy gyroskop, světelný senzor, různé komunikační sběrnice a dotykový kapacitní senzor.[3]



Obr. 1 Vývojový kit FRDM-KL28Z (převzat z [3])

2.2.1. Sériový a ladící adapter (OpenSDA)

OpenSDA je otevřený standard pro sériový a ladící adapter. Toto přemosťuje sériový a ladící komunikaci přes USB host konektor. Obvod pro OpenSDA obsahuje led diodu a resetovací tlačítko. Resetovací tlačítko může převést obvod OpenSDA do bootloader módu (mód zavaděče), když zapojujeme USB kabel do USB portu. [4]



Obr. 2 Schéma OpenSDA (převzat z[4])

3. Vývojová prostředí na bázi prostředí Eclipse

Prostředí Eclipse je open source vývojová platforma, která je pro většinu lidí známa jako vývojové prostředí (IDE) určené pro programování v jazyce Java. Flexibilní návrh této platformy dovoluje rozšířit seznam podporovaných programovacích jazyků za pomoci pluginů, například o C++, PHP nebo C. Právě pluginy umožňují toto vývojové prostředí rozšířit například o návrh UML, či zápis HTML nebo XML.[11]

3.1. Atollic TrueSTUDIO IDE

Toto vývojové prostředí je napsáno v programovacím jazyce Java na bázi prostředí Eclipse/CDT Framework. Obsahuje kompilátor GCC pro kompilaci kódu určeného pro procesory ARM a systémy x86. Ladění je vytvořeno na základě GDB. Je zde podpora pro řadu různých typů projektu (ARM a PC (x86) projekty, Aplikační a knihovní projekty, CDT moduly a makefile projekty). Dále obsahuje řadu možností pro editaci a navigaci v kódu. Obsahuje nástroje pro vytváření kvalitních embedded systémových softwarů, které mohou být buď spravovány prostřednictvím GUI, nebo spravovány manuálně prostřednictvím souborů. Prostředí využívá standard GNU a nabízí optimální kompilátor pro C/C++.[30]

V prostředí je také obsažen debugger pro C/C++ projekty, tento debugger je založen na základě GNU GBD ladění. Debugger pracuje s jakoukoliv sondou pro ladění, která je kompatibilní s GDB serverem. Podporované ladící sondy jsou P&E Micro, SEGGER J-Link/J-Trace, ST-Link, OpenOCD. Debugger je schopen ladit jedno jádrové, více jádrové a více procesorové systémy, mimo jiné i kód pro embedded mikrokontrolery a aplikace založené na Windows PC příkazové řádce. Debugger má i podporu pro mimo-systémový (cross-system) debugging a RTOS aplikace.[30]

4. Kompilátor GCC

GNU Compiler Collection (zkráceně GCC) je sada překladačů vytvořených v rámci projektu GNU. Původně se jednalo pouze o překladač programovacího jazyka C (a zkratka tehdy znamenala GNU C Compiler), později byly na stejném společném základě vytvořeny překladače jazyků C++, D, Fortran, Ada a dalších. Kompilátor slouží pro vygenerování souboru se spustitelným souborem. Primárně je však tento překladač určen pro překlad jazyků C, C++ a Assembleru. Tento překladač funguje v současné době na většině hlavních platforem (Unix, Windows, MacOS; Intel, PPC, Sparc, různé jednočipy, a jiné).[12]

4.1. Popis funkce

Kompilátor je program, který převede zdrojový text programu do strojového jazyka, tedy do jednotlivých instrukcí, kterým daný procesor rozumí. Výsledkem kompilace je řada objektových souborů, které obvykle neobsahují spustitelný kód, neboť na místech, ve kterých dochází k volání podprogramů z jiných modulů a připojování knihoven, nejsou konkrétní adresy těchto prvků, nýbrž pouze symboly určené pro pozdější připojení prvků prostřednictvím linkeru. Mimo jiné kompilátor má také za úkol kontrolu syntaktických chyb, tedy chyby v syntaxi daného programovacího jazyka, například chyby při použití nekompatibilních datových typů anebo použití chybných operátorů.[25]

4.2. Části kompilátoru GCC

Kompilátor GCC se obvykle skládá s různých částí. Každý kompilátor obsahuje sadu nástrojů, tzv. toolchain, které používá při překladu kódů, některé části z těchto nástrojů nejsou přímou součástí kompilátoru GCC, ale tyto části jsou při překladu použity.[29]

- cpp – Neboli preprocesor, jeho úkolem je zpracování maker, direktivy include, podmíněných překladů a odstranění komentářů. Vstupem je soubor se zdrojovým kódem v jazyce C a výstupem je modifikovaný soubor se zdrojovým kódem v jazyce C.[29]
- cc1 – Neboli jádro překladače, skládá se z několika částí, jako jsou frond end, middle end a back end. Front end převádí vstupní soubor do podoby programu, se kterým je kompilátor schopen pracovat. V middle end je následně prováděna optimalizace tohoto programu. V back end se provádí závěrečné úpravy, jako je optimalizace a převedení souboru do podoby assembleru cílového systému. Výstupem jádra překladače je soubor ve strojovém jazyku (v assebleru).[29]
- as – Neboli assembler, není přímo obsažen v kompilátoru GCC. Výstupem je binární soubor, který obsahuje strojový kód určený pro daný procesor. Tento kód není obvykle spustitelný, neboť může obsahovat odkazy na externí data, volání podprogramů z jiných modulů a odkazy na připojené knihovny.[29]
- ld – Neboli linker, není přímo obsažen v kompilátoru GCC. Úkolem linkeru je připojit k souboru, který je výstupem z assembleru, odkazy a vytvořit spustitelný soubor.[29]

4.3. GCC ARM Embedded Toolchain

Původně také GNU Tools for ARM Embedded Processors. Jedná se o sadu nástrojů (toolchain), pro sestavování aplikací pro cílovou platformu obsahující procesory ARM. Obsahuje integrované balíčky, které obsahují Arm Embedded GCC kompilátor, knihovny a ostatní GNU nástroje potřebné pro vývoj softwaru pro hardware. Tyto zařízení mohou být na základě Arm Cortex-M a Cortex-R procesory.[5]

4.4. Nástroje GNU pro embedded aplikace

Kompilátor C nebo C++, assembler a linker jsou základní části vývojářských nástrojů pro embedded aplikace. Stejně jako důležitý nástroj jako je debugger. Nástroje GNU nejsou pouze nástroje pro sestavení a linkování programu, ale obsahují řadu vysoce kvalitních příkazových nástrojů pro spravování, sestavení, debuggování a testování embedded aplikací pro ARM.[6]

- Vytváření utilit
- C/C++ kompilátor
- Assembler
- Linker
- C/C++ runtime a matematické knihovny
- Knihovník
- Ostatní binární utility
- Debugger

Řada dalších softwarů je použita v kombinací s nástroji GNU, jako je prostředí založené na prostředí Eclipse-IDE a gdb servery, které povolí GNU debugger (gdb) připojit se k JTAG ladící sondě za potřebí ladění kódu pro cílové zařízení.[6]

4.4.1. GNU vytváření utilit

Technicky není součástí sady nástrojů GNU C/C++ kompilátoru. Nejčastěji se používá k spuštění sestavovacího procesu. Čte obsah makefile, kde získá potřebné informace od vývojáře a následně sestaví vstupní soubory podle vhodného způsobu assembleru, kompilátoru a linkeru v závislosti na pravidlech obsažených v makefile.[6]

4.4.2. GNU C/C++ kompilátor (GCC a G++)

Kompilátor GNU C/C++ má řadu rozšíření, které jej dělají velice vhodné pro vývoj embedded zařízení. Obsahuje možnosti pro ovládání pomocí příkazů zadané v příkazové řádce, assembler v řádcích a kompilátor #pragmas, který pomáhá ovládat kompilátor detailněji.[6]

Kompilátory GNU mají širokou podporu cílových procesorů a standardů pro jazyky. Kompilátor GNU C a C++ jsou open-source, ale jejich licence (GPL) se nestahuje na aplikace, které sestavíme pomocí nich. Takže GNU kompilátory a knihovny s nimi spojené jsou open-source, ale aplikace, vytvořené za použití těchto nástrojů mohou být pouze vytvořené pro vlastní použití či uzavřené použití.[6]

Kompilátor nabízí například

- Optimalizovaný C/C++ kompilátor
- Podpora asembleru v řádcích
- Podpora obsluhy přerušení
- Podpora velkého množství generických a specifických #pragmas
- Generování optimálního kódu nebo debuggování kódu
- Generování informací pro optimalizaci linkeru (nepotřebný kód nebo nepotřebná data)
- Generování reportů (soubor položek disassembly)
- Rozšíření pro optimální ovládání operací prostřednictvím příkazové řádky

Kompilátor GNU C/C++ podporuje tyto standardy pro jazyk C

- C89 (originál ANSI-C, X3.159-1989)
- C90 (originál ANSI-C, ISO/IEC 9899:1990)
- C94/C95 (amendment 1, AMD1)
- C99 (ISO/IEC 9899:1999)
- C11 (ISO/IEC 9899:2011)
- GNU jazykové rozšíření

Kompilátor GNU C/C++ má také podporu pro ARM jádra jako jsou například

- ARM7
- ARM9
- Cortex-M0, Cortex-M0+, Cortex-M1, Cortex-M3, Cortex-M4, Cortex-M7
- Cortex-R4, Cortex-R5, Cortex-R7
- Cortex-A5, Cortex-A7, Cortex-A8, Cortex-A9, Cortex-A15, Cortex-A17, Cortex-A53, Cortex-A57, Cortex-A72

4.4.3. GNU makro assembler (GAS)

Je použit pro převedení assemblerského zdrojového kódu do objektových souborů (strojového kódu).

- Vytvoří specifický assemblerský zdrojový kód vhodném formátu pro linker
- C/C++ preprocesorová direktiva jako #define a #ifdef
- Numerické výrazy jsou vypočteny
- Makra s podporou smyček
- Rozšíření pro optimální ovládání operací prostřednictvím příkazové řádky

4.4.4. GNU linker (ld)

Spojí soubory objektů a před kompilované soubory knihoven do binárního aplikačního souboru.

- Spojí soubory objektů a před kompilované soubory knihoven do binárního ELF souboru
- Odstraní nepoužité funkce a sekce kódu (dead code removal)
- Odstraní nepoužité proměnné a datové sekce (dead data removal)
- Příkazy pro linker skript a nastavení jazyka
- Generuje report k použité paměti (map files)
- Rozšíření pro optimální ovládání operací prostřednictvím příkazové řádky

4.4.5. C/C++ runtime a matematické knihovny

Každý kompilátor potřebuje ke své správné činnosti runtime a matematické knihovny.

- C runtime knihovna
- C matematická knihovna
- C++ runtime knihovna

Nejvíce používané C runtime knihovna je pro embedded systémy newlib. Mimo knihovny newlib existuje také menší verze této knihovny s názvem newlib nano.

4.4.6. GNU knihovník (ar)

Nástroje GNU zahrnují GNU utilitu archiv, která je použita k vytváření a spravování před sestavené softwarové knihovny, k zjednodušení spravování a spojování softwarových komponentů.

- Vytváření nových knihoven ve vhodném formátu
- Přidávání, mazání, uspořádaní a vytahování souboru v knihovně
- Tisk obsahu knihovny
- Optimální skriptovací mód
- Rozšíření pro optimální ovládání operací prostřednictvím příkazové řádky

4.4.7. Další binární utility

Sada GNU dále obsahuje řadu utilit spouštěné v příkazové řádce.

- Převedení binárního souboru ELF do jiného formátu (Intel Hex nebo Motorola S-Record)
- List symbolů v objektových souborech
- Zobrazení informací v objektových souborech
- Zobrazení obsahu ELF souborů
- List velikosti souborových sekcí a celkové velikosti
- Převedení adresy na jméno souboru a číslo řádky

4.4.8. GNU debugger (GDB)

Je silný a flexibilní debugger. Příkazy gdb mohou být skriptovaný s použitím různých skriptovacích jazyků, toto přidá podporu pro výpočet matematických výrazů, smyček a podmíněného spuštění. Debugger sám o sobě neposkytuje dostatečnou podporu pro ladění aplikací pro ARM, nemá podporu pro DRAM a flash paměť, nebo integrovanou JTAG sondu. Nemá dostačující nástroje, které vývojáři požadující jako jsou grafické IDE, zobrazení obvodový registrů (SFR), sledování událostí v reálném čase a sledování dat s použitím technologie ARM Serial Wire Viewer (SWV), sledování instrukcí používající ETM/ETB technologie. Také nemá zahrnutou Cortex-M analýzu chyb nebo RTOS podporu.[6]

4.5. GCC parametry

Obsahuje řadu příkazů, které lze zadat do příkazové řádky pro každý zdrojový soubor .c lze tuto příkazovou řádku volat.

Mezi některé parametry určené pro kompilátor patří:

- -c = je určení správného vstupního zdrojového souboru typu .c, který je určen ke kompilaci
- .mthumb = použití Thumb(-2) instruční sady
- -mcpu = určení konkrétního typu CPU pro, který bude generován strojový kód
- -std = určení dialektu jazyka C
- -D = definice symbolických konstant ekvivalent #define
- -I = určení cest k header souborům
- -Ox = určení stupně optimalizace (O0-3, Os, Og)
- -ffunction-sections = každá funkce bude v objektovém souboru umístěna v samostatné podsekci sekce .text
- -fdata-sections = každý datový objekt buude v objektovém souboru umístěna v samostatné podsekci sekce .data/.bss
- Tyto parametry jsou nutné pro použití linkeru –gc-sections
- -g = výstupní objektový soubor bude obsahovat podporu ladění
- -fstack-usage = generování informace o využití zásobníku jednotlivými funkcemi (.su soubory)
- -Wall = všechny upozornění jsou aktivní a v případě výskytu budou vypsány na výstup
- -ffreestanding = freestanding implementace
- -o = určení jména výstupního objektu

Mezi některé parametry určené pro linker patří

- -mcpu = umožňuje linkeru najít správnou verzi knihoven
- -nostdlib = linkuje se pouze proti explicitně určeným knihovnám
- -Wl = parametry pro linker
- --entry = označneí symbolu, který představuje začátek programu
- --defsym = definice symbolu s určenou hodnotou. Usecase: obdoba #define ale pro linker
- -Map = zapnutí generování map souboru
- --cref = generování tabulky refenercí do map souboru
- --ge-sections = garbage collector, odstranění nepoužitých vstupních sekcí
- --script = výber linker skriptu
- --oformat = určení formátu výstupního objektu (typický elf, little endian)
- -o = určení jména výstupního objektu
- -I = určení knihoven, které mají být přilinkovány

5. Debugger

Je utilita či softwarový nástroj určen k hledání chyb při vývoji softwarové aplikace ve fázi ladění. Většina vývojových prostředí má debugger integrovaný nebo se připojuje na externí nezávislý debugger.

5.1. Hardwarová podpora Cortex-M pro debugging

Procesory Cortex-M od ARM mají dobrou podporu pro debuggování. Lze použít různé programy pro systémovou analýzu nebo sledování událostí v reálném čase. Existuje řada různých možností připojení těchto programů, jako je například Seriál Wire Viewer (SWV), Seriál Wire Debug (SWD) a Seriál Wire Output (SWO).[7]

- Serial Wire Debug (SWD) je port určený pro debuggování, který je podobný k JTAG a poskytuje stejné možnosti debuggování (ladění), jako jsou spuštění programu, zastavení programu na breakpointu nebo krokování programu s tím, že potřebuje méně pinů. Takže nahradí JTAG konektor za dvoupinové rozhraní (jeden hodinový pin a jeden obousměrný datový pin). Nevýhodou SWD je, že samostatně neposkytuje možnosti pro sledování procesů v reálném čase.[7]
- Serial Wire Output (SWO) je pin, který může být použit v kombinací s SWD k umožnění procesoru vysílat data pro sledování procesů v reálném čase. Tedy tento pin rozšiřuje dva piny, které používá SWD o další. Kombinací těchto pinů umožní Serial Wire Viewer (SWV) sledování v reálném čase v podporujících ARM procesorech.[7]
- Serial Wire Viewer (SWV) je technologie pro sledování v reálném čase, které používá SWD port a SWO pin. SWV poskytuje pokročilou systémovou analýzu a sledování v reálném čase bez nutnosti zastavení procesoru k získání potřebných informací k debuggování.[7]

Dále některé jádra Cortex-M a zařízení májí podporu pro sledování instrukcí, používající Instruction Trace Macrocell (ITM), Embedded Trace Buffer (ETB) nebo Micro Trace Buffer (MTB) technologii.

- Embedded Trace Macrocell (ETM) k sledování potřebuje, aby sonda pro debuggování podporovala sledování ETM, například sondu SEGGER J-Trace. Takové sondy obvykle mají velký sledovací zásobník k ukládání záznamů historie spuštění nebo dokonce používají zapisování v reálném čase do hostitelského PC, a tedy jsou často dražší než obyčejné sondy bez podpory sledování ETM. Sledovací zásobník použitý se sledováním ETM má typicky velikost v řádu MB nebo GB. Pro použitá sledování ETM, je použit 20-pinový JTAG konektor, kde 5 pinů je vyhrazeno pro sledování ETM (jeden hodinový pin a čtyři datové piny)[7]
- Embedded Trace Buffer (ETB) sledování nepotřebuje žádnou sondu, která umožňuje sledování, ale používá velmi malý sledování buffer, který je umístěn v RAM paměti na Cortex-M zařízení. ETB tedy může být použito bez drahých sond, které umožňují sledování s kompatibilním Cortex-M zařízením, ale čas provádění, které může být nahráno je podstatně kratší. Sledovací zásobník použitý s ETB je typicky velmi malý, pouze několik KB může být použity z RAM paměti mikrokontroleru, jako sledovací zásobník. K použíti sledování ETB, není nutné použití žádných dalších pinů z JTAG, jako sledovací zásobník je použita RAM paměť ze zařízení, stejně jako jiná pole bajtů.[7]

- Micro Trace Buffer (MTB) je podobný k ETB, ale je nízko koncové řešení pro jádra Cortex-M0+. Minimum ze sledovací zásobník je naplněno do malého paměťového zásobníku umístěného na čipu, který může být nahrán do vývojářského počítače pro analýzu.[7]

Nakonec má ARM přidanou podporu pro analýzu kritických chyb, která umožní dobrému ladícímu programu zjistit, proč tato chyba vznikla a přejít do stavu kritické chyby.

Mezi další možnosti debugger patří:

- Náhled do SFR registrů, tato možnost nám umožní získání informací z obvodových SFR registrů, tedy nám dovolí zjistit, jak je daný hardware nastaven, jeho status a je velmi důležitý při vývoji embedded aplikací.[7]
- Analýza Hard fault crash, toto umožní analyzovat, proč procesor šel do chybového stavu (hard fault crash state).[7]
- Serial Wire Viewer k sledování událostí v reálném čase a systémová analýza. Tato možnost patří k nejsilnějším nástrojům při debuggingu. SWV a kompatibilní debugger poskytují řadu informací, jako jsou statistické profilování, logování přístupu do paměti, sledování stavů proměnných, sledování průběhu dat na osciloskopickém grafu, upozornění na vstup a výstup z události a sledování softwarových prostředků. Vývojáři mohou nastavit SWV různými způsoby k získání různých kombinací informací. Čím více informací požadujeme, tím více dat musíme vyslat z procesoru, tedy v některých případech může nastat, že všechny data nebudou přijata před ukončením ladicího programu.[7]
- Sledování instrukcí. Slouží pro zjištění chyby při programu. K zjištění chyb se používá sledovací log, kde lze najít, co procesor dělal předtím, než nastala chyba. S použitím sledovače instrukcí, každý řádka kódu v jazyce C a instrukce v assembleru, může být vykonána až k místu, kde nastala chyba neboli lze krokovat v kódu. [7]
- Sledování stavu proměnných. S použitím ladící sondy může debugger zobrazit hodnotu proměnné, i když cílový systém pracuje při plné rychlosti. Lze získat i komplexní datové typ, jako jsou ukazatele na strukturu polí.[7]
- RTOS debuggování. Při použití RTOS jako je FreeRTOS, je potřebné mít i debugger, který je schopen s ním pracovat. Neboť je potřebné vidět i objekty RTOSu, jako jsou například tasky, vlákna, semafory, časovače, mutexy, fronty zpráv a další.[7]
- Analyzér paměti. Je použitý pro zjišťování chyb, které nastaly vlivem přepisu paměti, neboť tyto chyby nelze obvykle vidět ve zdrojovém kódu, a tedy se nadají normálně nalézt. Tyto chyby mohou být způsobeny systémem při sestavování.[7]

5.2. Debugger Segger Ozone

Je grafický debugger pro embedded aplikace. Za pomocí Ozone je možné ladit jakoukoliv embedded aplikaci s C/C++ kódem a assembler. Ozone může nahrát sestavené aplikace s jakýkoliv sadou nástrojů či IDE nebo ladit aplikaci uloženou v cíl bez zdrojového kódu. Obsahuje všechny možnosti pro kontrolu ladění a informační okna, a tudíž vytváří vhodnou podporu pro J-Link a J-Trace ladící sondy. Poskytuje řadu možností sledování, profilování kódu a analýzu kódu.[8]

- Samostatný grafický debugger
- Ladění výstupu jakéhokoliv sady nástrojů a IDE
- Ladění C/C++ zdrojových kódů a assemblerských instrukcí
- Informační okna (disassembly, paměť, globální a lokální proměnné, sledování stavu proměnných, procesorové a obvodové registry)
- Editor pro okamžitou úpravu malých chyb
- Rychle programování aplikace do cíle
- Přímé použití J-Link možností (Nekonečné flash breakpointy, stažení flash, terminál v reálném čase, sledování instrukcí)

5.2.1. Informační okna

Sledování kódu

Toto okno umožňuje sledovat zdrojový kód aplikace. Zobrazuje právě vykonávané instrukce a umožňuje body zastavení (breakpointy) pro schopnost krokování skrz kód. Jednotlivé řádky kódu lze rozšířit, a tak vidět i aktuální kód v podobě assemblerských instrukcí.

Disassembly

V tomto okně lze vidět kód v podobě assemblerských instrukcí, v tomto okně lze provádět krokování stejně jako v běžném okně pro sledování kódu, a tudíž lze sledovat vykonávané části kódu na úrovni strojových instrukcí.

Breakpoints

Umožňuje detailnější informace pro místa kódu, na kterých se nachází breakpoint. Existují různé druhy breakpointů, ve zdrojovém kódu, v instrukci a datový.

Variables

Toto okno umožňuje sledování a upravování proměnných a funkčních parametrů. Tyto proměnné mohou být upraveny periodicky, i když aplikace běží. Lokální proměnné ukazují data v právě vykonávaném bodě nebo funkci. Globální proměnné zobrazují data všech globálních proměnných a statických symbolů.

Registers

Toto okno ukazuje všechny registry, které se nachází v CPU. Dále kromě základních registrů lze zobrazit také mapované obvodové registry (SFR). Lze vidět aktuální hodnoty všech registrů a lze je také modifikovat.

5.3. SEGGER J-Link

J-Link podporuje připojení řady různých cílových rozhraní. Tyto ladící sondy mají podporu v řadě majoritních IDE prostředích založených na prostředí Eclipse tak i IDE založené na GDB. [9]

Seznam cílových rozhraní je.

- JTAG
- SWD
- FINE
- SPD
- ICSP

Zařízení typu ARM7/9 podporuji pouze JTAG rozhraní. Pozdější zařízení s ARM Cortex májí podporu ladění s rozhraní SWD, který je alternativou k JTAG. FINE je jedno drátové rozhraní, které používají zařízení od firmy Renesas. SPD je jedno drátové rozhraní od firmy Infineon, ICSP je dvoudrátové JTAG rozhraní od firmy Microchipu.[10]

5.3.1. SWD/JTAG konektor

JTAG byl klasický mechanismus pro ladění zařízení s ARM7/9 ale s příchodem řady Cortex-M, ARM představil i nové rozhraní Single Wire Debug (SWD), SWD je navrhnuto pro zredukování počtu pinů potřebných pro ladění z 5 použitých u JTAG na 3.[10]

Tab. 1 Rozdílné signály mezi konektory JTAG a SWD (převzato z [10])

JTAG mód	SWD mód	Signál	poznámka
TCK	SWCLK	Hodiny k jádru	Používá 10k-100k ohmové pull-down rezistory k GND
TDI	-	JTAG vstup testovacích dat	Používá 10k-100k ohmové pull-up rezistory k VCC
TDO	SWV	JTAG výstup testovacích dat/ SWV výstup	Používá 10k-100k ohmové pull-up rezistory k VCC
TMS	SWDIO	JTAG výběr testovacího módu / SWD vstup/výstup dat	Používá 10k-100k ohmové pull-up rezistory k VCC
GND	GND	-	-

6. MCU NXP Kinetis L

Tato série je určena pro nízko výkonové aplikace, tato série používají jádra typu Cortex-M0+. Série kombinuje nízko-výkonové operace a energetickou účinnost jádra Arm Cortex-M0+ a jejich výkonosti s řadou různých obvodových periférií, tato kombinace dělá tuto série vhodným řešení pro aplikace založené na internetu věcí. (IoT)[18]

Standard Key Features: Low-Power UART and Timers, PWM, SPI, I²C, RTC and Analog Comparator.

	CPU		Packages		Comms		HMI		Security	
Products	Arm Cortex-M0+	Memory	Type	Pin Count	I²S	USB Full-Speed	FlexIO*	Segment LCD	HW Encryption	Tamper Detection
KL8x › Security	72 MHz (Up to 96 MHz)	128 KB Flash 96 KB SRAM	LQFP, MAPBGA, WLCSP	64–121	✓	✓	✓	✓	✓	✓
KL4x › USB and Segment LCD	48 MHz	128–256 KB Flash 16–32 KB SRAM	LQFP, MAPBGA	64–121	✓	✓	✓	✓		
KL3x › Segment LCD	48 MHz	32–256 KB Flash 4–32 KB SRAM	LQFP, MAPBGA	64–121	✓		✓	✓		
KL2x › USB	48 MHz / 72MHz (Up to 96)	32–512 KB Flash 4–128 KB SRAM	LQFP, QFN, MAPBGA, XFBGA, WLCSP	32–121	✓	✓	✓		✓	
KL1x › Mainstream	48 MHz	32–256 KB Flash 4–32 KB SRAM	LQFP, QFN, MAPBGA, XFBGA, WLCSP	32–80	✓		✓			
KL0x › Entry-level	48 MHz	8–32 KB Flash 1–4 KB SRAM	LQFP, QFN WLCSP	16–48						

Obr. 3 Seznam řad série MCU NXP Kinetis L (převzato z [18])

6.1. Řada KL2x

Řada založená na požadavky aplikací na výkon.[22]

- Generace jádra 32-bitových ARM Cortex-M0+: 2x více CoreMark/mA která je blíž k 8/16-bitové architektuře
- Jedno cyklový rychlý přístup k I/O portům a emulace softwarových protokolů
- Větší flexibilita nízko výkonových modulů, zahrnuje novou volbu časování, která redukuje čas pro změnu mezi jednotlivými periferiemi v asynchronních stop módech
- Periférie LUART, SPI, I2C, FlexIO, ADC, DAC, LP časovač a DMA mají podporu práce v nízko výkonovém módu bez nutnosti probuzení jádra

Paměť

- Až 256kB flash paměť s 64 bajtovým flash cache, až 32kB RAM
- Zabezpečovací obvod k ochraně neautorizovaného přístupu k RAM paměti a obsahu flash paměti.
- 16kB ROM se zabudovaným zavaděčem

Výkonost

- ARM Cortex-M0+, s 48MHz frekvencí jádra pro celý rozsah napětí, teplotní rozsah -40 až 105
- Instrukční sada Thumb

Konverze

- Až 16 bitový ADC s nastavením rozlišení
- Integrovaný teplotní senzor
- Vysoko rychlostní komparátor s vnitřním 6 bitovým DAC
- 12 bitový DAC s DMA podporou

Časování a ovládání

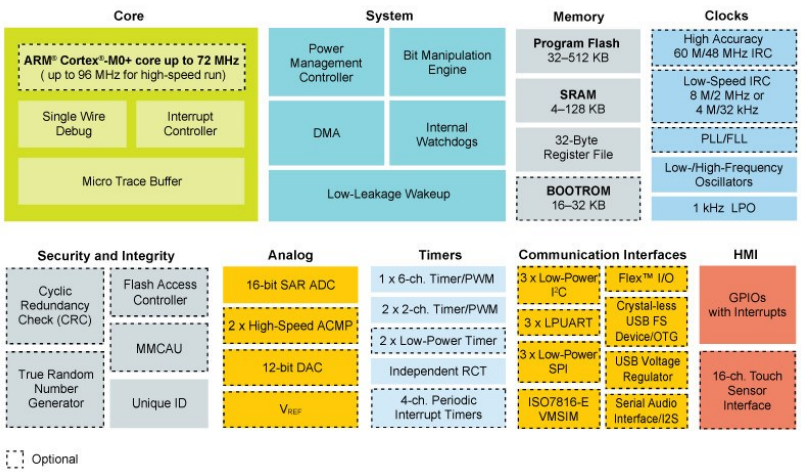
- Dva 6-ti kanálové a jeden 2 kanálový, 16-ti bitový nízko výkonový časovač s PWM moduly s podporou DMA
- 2 kanálový, 32 bitový periodický přerušovací čítač, který poskytuje časovou základnu pro RTOS plánovači úloh nebo spouštěcí zdroj pro ADC převod
- Časovač reálného času s kalendářem

Rozhraní

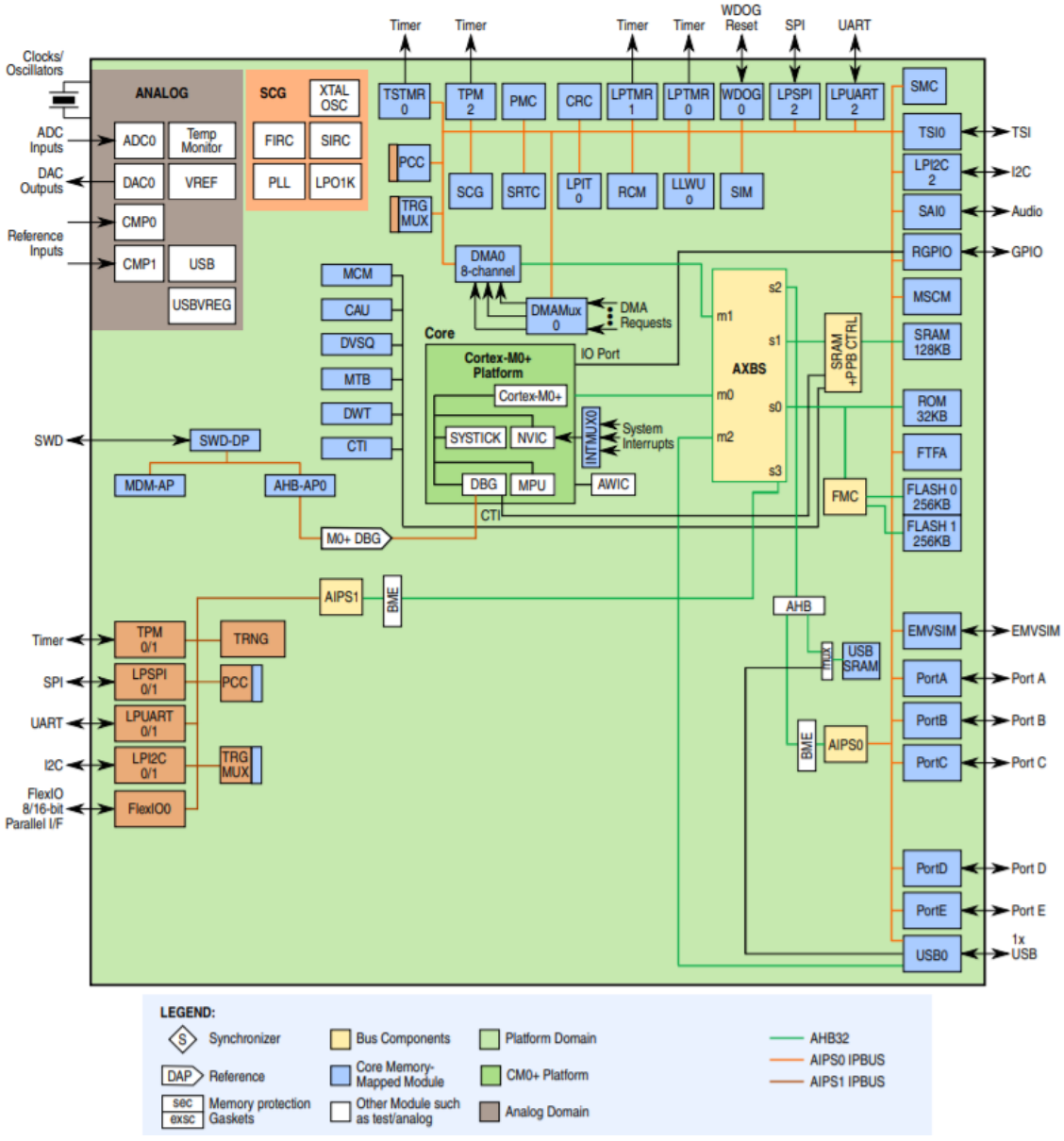
- Kapacitní dotykové rozhraní s podporou až 16 externími elektrodami a DMA datový přenos
- GPIO s podporou přerušení na daném pinu

Připojení a komunikace

- USB 2.0 On-The-Go (plná rychlost) s integrovaným USB nízko napěťovým regulátorem, USB 2.0 zařízení s hodinovým obnovením bez nutností externího krystalu
- FlexIO pro obecnou nebo specifickou emulaci sériových periférií
- Velmi přesné vnitřní referenční hodiny
- Dva I2C s podporou DMA
- Jeden LPUART a dva UART s podporou DMA
- Dva SPI s podporou DMA
- I2C modul pro aplikace se zvukem



Obr. 4 Periférie řady KL2x (převzato z [22])



Obr. 5 Blokový diagram periférií KL28z (převzato z[27])

7. NXP FlexIO

Tento modul byl poprvé představen v řadě Freescale Kinetis KL43. Je schopen emulovat různé protokoly pro sériovou či paralelní komunikaci zahrnující UART, SPI a I2C. Tento modul je velice flexibilní a lze jej nastavit podle potřeby dané komunikace. Hlavní části modulu FlexIO jsou posunovače (shiftery), časovače a piny. Data jsou nahrány do posunovače a časovač slouží ke generování hodinového signálu k posunu dat v posunovači a použitý pin slouží k výstupu dat, obsažených v posunovači. [28]

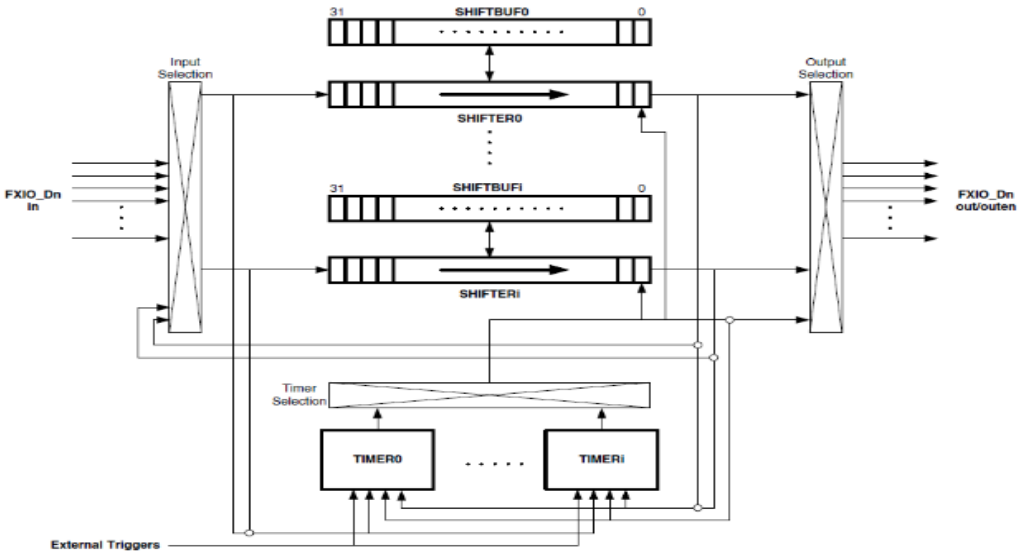
Podle verze tohoto modulu, může být použit například k emulaci sériové komunikačního rozhraní (UART, SPI, I2C, I2S, one wire a další), emulaci paralelního rozhraní (CMOS kamery, Motorola 68k Intel 8080 a další), generování uživatelsky definovaného komplexního časování grafů a spouštěcích signálů, vytvoření logické funkce na výstup pomocí logické náhledové tabulky (look-up tabulky), vytvoření hardwarového stavového automatu.[27]

7.1. Modul FlexIO obsažený v KL28Z

Jedná se o 16-bitový čítač s podporou pro spouštěcích signálů, resetu a podmínek pro spuštění a zastavení. Obsahuje programové logické bloky, které dovolují implementaci digitálních logických funkcí na čipu a nastavitelné interakce mezi vnitřními a vnějšími moduly. Programovatelný stavový automat dovoluje přenést základní systémovou kontrolní funkci z CPU.[27]

Poskytuje:

- Pole 32-bitový posunovacích registrů s vysílacím, přijímacím módem a módem srovnávací data.
- Dvojnásobnou vyrovnávací paměť pro posunovací operace v průběhu přenášení dat
- Posunovač s řetězením k podpoře přenášení velkého množství dat
- 1, 2, 4, 8, 16 nebo 32 vícebitový posunovací šířku pro podporu paralelních rozhraní
- Programovatelný stavový automat dovolující přenést základní systémovou kontrolní funkci z CPU s podporou až 8 stavů, 8 výstupů a 3 vstupů pro výběr stavů



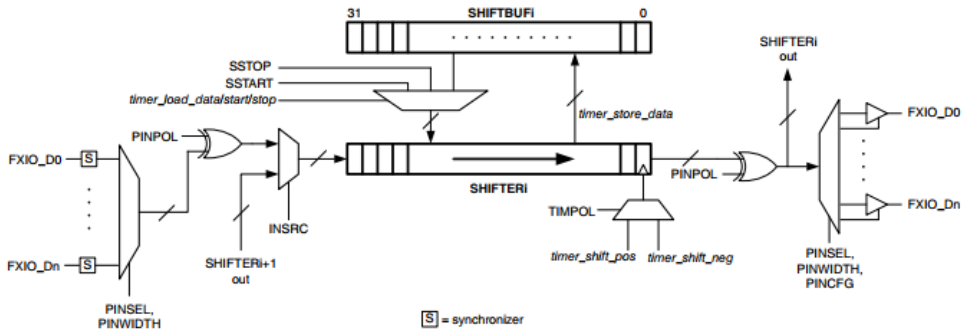
Obr. 6 Blokové schéma modulu FlexIO (převzato z [27])

7.2. Funkční módy modulu

FlexIO má řadu možných funkčních módu, které lze nastavit prostřednictvím jeho SFR (speciální funkčních registrů).

7.2.1. Posunovací operace

Posunovače jsou používány k ukládání dat do vyrovnávací paměti a posunu dat do nebo z FlexIO. Časování posunu, nahrání či uložení jsou kontrolovány prostřednictvím časovačů, které jsou přiřazeny k posunovači prostřednictvím jeho registrů. Posunovače jsou navrženy pro podporu DMA, přerušení a polled operace.[27]



Obr. 7 Blokové schéma modulu FlexIO pro posunovací operace (převzato z [27])

7.2.2. Vysílací mód

V tomto módu bude posunovač nahrávat data z registru SHUFTBUF a posunovat data na jeho výstup a to vždy, když je vyvolaná událost od přiřazeného časovače. Pomocí start/stop bitu lze automaticky nahrávat data. Status flag a jakékoliv povolené přerušení nebo požadavek od DMA budou nastaveny, když data budou nahranná z registru SHIFTBUF do posunovače nebo když posunovač je poprvé nastaven do vysílacího módu. Tento stav je zvolen, když SMOD v registru pro nastavení posunovače FLEXIO_SHIFTCTL[SMOD] je nastaven na hodnotu 0x2.[27]

7.2.3. Přijímací mód

V tomto módu bude posunovač posunovat data dovnitř a ukládat je do registru SHIFTBUF a to vždy, když je vyvolaná událost od přiřazeného časovače. Tento stav je zvolen, když SMOD v registru pro nastavení posunovače FLEXIO_SHIFTCTL[SMOD] je nastaven na hodnotu 0x1.[27]

7.2.4. Mód ukládání shodných dat (Match Store Mode)

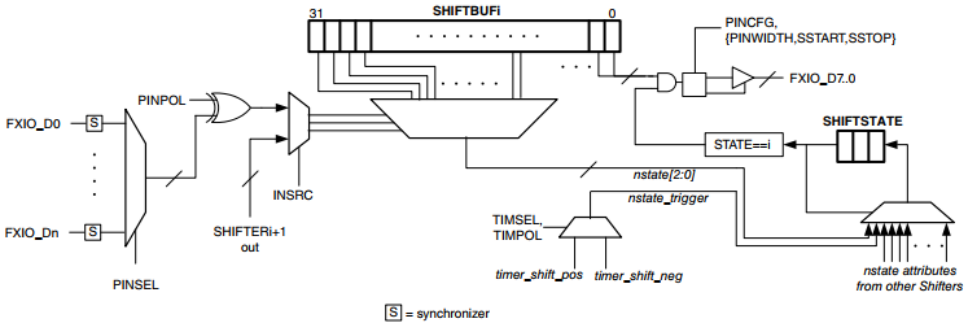
V tomto módu bude posunovač posunovat data dovnitř a kontrolovat výsledek shody a ukládat shodná data do registru SHIFTBUF a to vždy, když je vyvolaná událost od přiřazeného časovače. Tento stav je zvolen, když SMOD v registru pro nastavení posunovače FLEXIO_SHIFTCTL[SMOD] je nastaven na hodnotu 0x4.[27]

7.2.5. Mód trvalé kontroly shody (Match Continuous Mode)

V tomto módu bude posunovač posunovat data dovnitř a bude stále kontrolovat výsledek shody, kdykoliv bude vyvolána událost od přiřazeného časovače. Tento stav je zvolen, když SMOD v registru pro nastavení posunovače FLEXIO_SHIFTCTL[SMOD] je nastaven na hodnotu 0x5.[27]

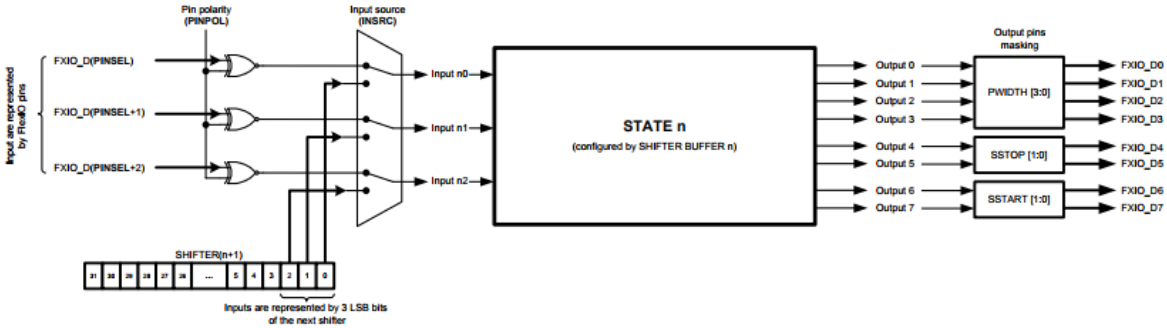
7.2.6. Stavový mód

Při použití tohoto módu lze uživatelsky implementovat jakýkoliv stavový automat s až 8 možnými stavy, 8 výstupy a 3 vstupní piny pro výběr stavu. Tímto lze přenést část funkce z CPU. Registr SHIFTBUF je použit k řízení výstupů a určení další hodnoty stavu, když posunovač byl vybrán aktivním stavovým ukazatelem. Tento stav je zvolen, když SMOD v registru pro nastavení posunovače FLEXIO_SHIFTCTL[SMOD] je nastaven na hodnotu 0x6.[27]



Obr. 8 Blokové schéma modulu FlexIO pro Stavový mód (převzato z [27])

Na následujícím obrázku jde vidět, že změna stavu může být provedena prostřednictvím, buď trojice FlexIO pinů od počátečního pinu, počáteční pin se nastavuje hodnotou v registru FLEXIO_SHIFTCTL[PINSEL] nebo trojicí LSB bitů v posunovači pro další stav (n +1). Volba zdroje pro změnu stavu je závislá na nastavené hodnotě bitu INSRC v nastavení posunovače FLEXIO_SHIFTCFG[INSRC]. Osm výstupů jsou pevně dané a jsou přiřazeny k FlexIO pinům FXIO_D0-FXIO_D7, tak že všechny stavy sdílí tyto výstupy.[26]

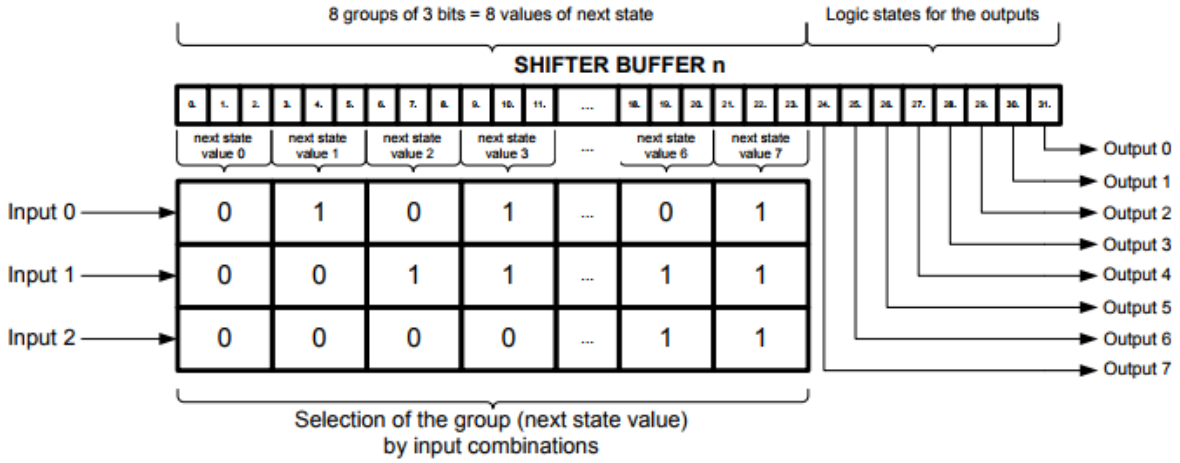


Obr. 9 Volba příštího stavu modulu FlexIO (převzato z [26])

Po výběru některého z posunovače prostřednictvím ukazatele aktuálního stavu se na výstupní piny FLEXIO_D [7:0] zapíše hodnota, která odpovídá hodnotě zapsané v registru SHIFTBUF [31:24]. K volbě dalšího stavu slouží tři vstupy, reprezentující buď vstupní piny nebo LSB bity v registru, prostřednictvím těchto tři vstupů se vybere adekvátní hodnota z SHIFTERBUF [23:0]. Pro změnu aktuální stavu periférie FlexIO na jiný stav, musí dojít k dopočtení čítače přiřazeného k aktuálnímu stavu.[26]

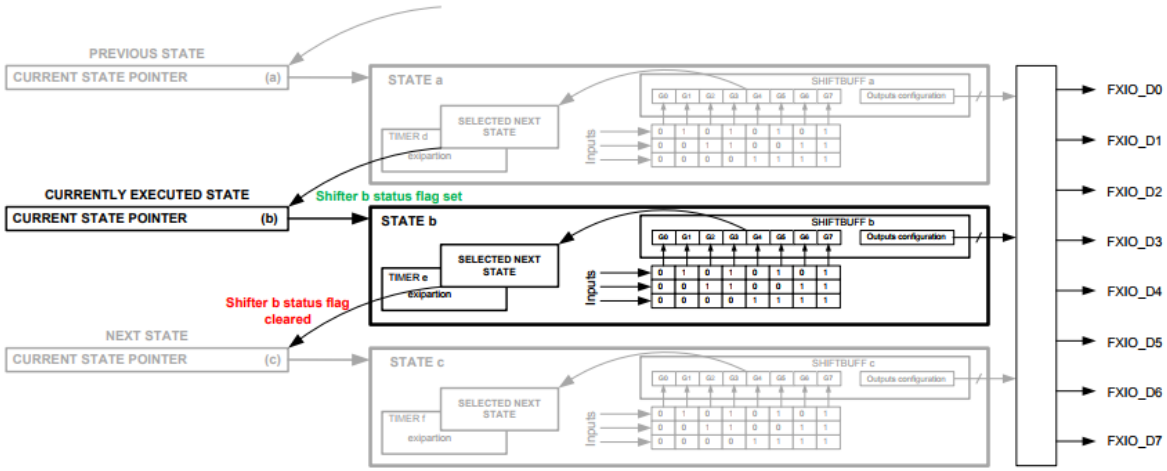
Po spuštění periférie se do registru FLEXIO_SHIFTSTATE zapíše hodnota stavu, která je nadefinována v registru posunovače FLEXIO_SHIFTERBUFF. Tento registr má 32 bitovou velikost a obsahuje nakonfigurovanou hodnotu výstupu stavu, které se nachází v registru FLEXIO_SHIFTBUFF[31:24] a definice pro volbu příštích stavů, které se nachází v registru FLEXIO_SHIFTBUFF[23:0].

Definice příštích stavů v registru FLEXIO_SHIFTBUFF [23:0] je tvořena osmi skupinami, každá tato skupina obsahuje trojici bitů. Tyto tři bity slouží k definici hodnoty příštího stavu periférie FlexIO, a to tak, že každá skupina reprezentuje číslo stavu neboli hodnoty 0-7. Následující stav je vybrán prostřednictvím kombinace vstupů.[26]



Obr. 10 Definice a volba všech stavů Stavového automatu (převzato z [26])

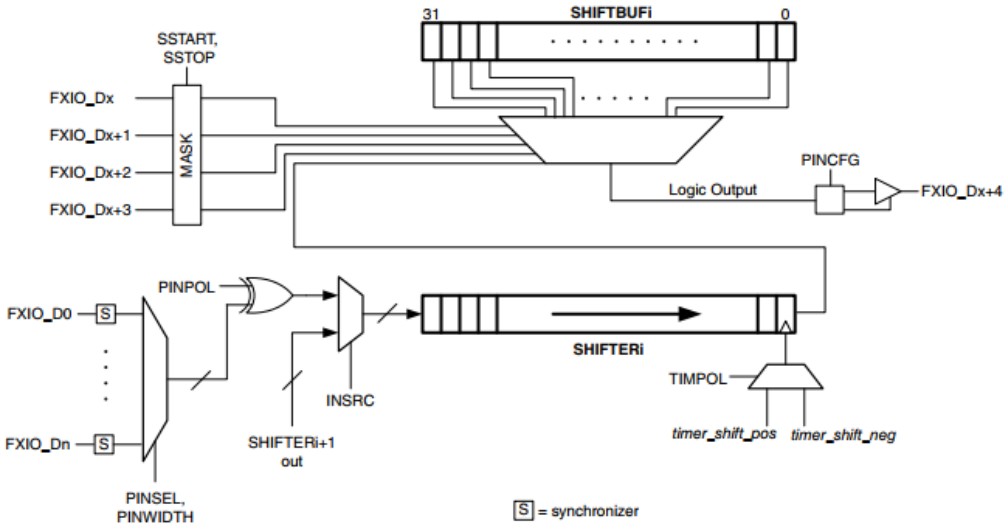
K přepnutí stavu dochází po dopočtení časovače, zvoleného právě vykonávaným stavem. Přiřazení časovačů se provádí prostřednictvím hodnoty TIMSEL v registru posunovače FLEXIO_SHIFTCTL[TMISEL]. Status flag se nastavuje při změně aktuálního stavu v registru FLEXIO_SHIFTSTATE, tento flag lze použít pro generování přerušení nebo generování požadavku pro DMA.[26]



Obr. 11 Logické schéma funkčnosti stavového automatu (převzato z [26])

7.2.7. Logický mód

Použitím tohoto módu umožní uživatelovy implementovat malé množství programovatelných digitálních logických operací prostřednictvím FlexIO posunovače. Toto řešení umožní řešení logických operací integrovat na čip s použitím periférie FlexIO. Tento mód implementuje 5 vstupů, 32-bitovou programovatelnou logickou náhledovou tabulku (look-up tabulku). Tento stav je zvolen, když SMOD v registru pro nastavení posunovače FLEXIO_SHIFTCTL[SMOD] je nastaven na hodnotu 0x7.[27]



Obr. 12 Blokové schéma modulu FlexIO pro Logický mód (převzato z [27])

Tabulka look-up je řízena prostřednictvím 4 vstupních pinů plus jeden vstup z vnitřního posunovače. Tabulka může být nastavená k tomu, aby poskytovala výstup na pin. Vstupní a výstupní piny jsou dané pro každou logiku v look-up tabulky a nelze je vybrat. [27]

7.2.8. Časové operace

Časovače modulu FlexIO jsou 16 bitové a jsou určeny pro kontrolu načítání, ukládání a posunování dat v registrech posunovače. Čítač načte obsah porovnávacího registru a sníží jeho hodnotu odkud nenabývá hodnoty nuly, a to v závislosti na hodinách periférie FlexIO. Tyto časovače mohou být použity ke generování hodin, výběru výstupu nebo generování PWM průběhu.[27]

7.2.9. Operace s piny

Konfigurace pinů pro každý časovač a posunovač může být nastaven pro použití FlexIO pin. Každý časovač a posunovač může být nastaven jako vstupní, výstupní pro data, povolení výstupu nebo obousměrný výstup. [27]

8. Konečné stavové automaty

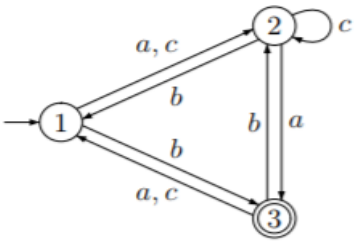
Konečný automat je systém či model systému, který může nabývat konečný počet stavů, obvykle počet stavů nebývá příliš vysoký. Aktuální stav je možno změnit v závislosti na vnějším podnětu s tím, že podnět v daném stavu, jednoznačně určují následující stav automatu. Konečné stavové automaty se nejčastěji zadávají v grafické formě, a to buď stavovým diagramem nebo grafem automatu. Konečný stavový automat je schopen rozpoznávat regulární jazyky. Regulárními jazyky se rozumí takové jazyky, které mohou být popsat regulárními gramatikami. Formální gramatiky mají definici prostřednictvím uspořádané čtveřice $(\Gamma,\ \Delta, R, S)$, kde: [35]

- Γ označuje konečnou množinu neterminálních symbolů
- Δ označuje konečnou množinu terminálních symbolů
- R označuje množinu generujících pravidel
- S označuje počáteční symbol, $S \in \Gamma$

Graf automatu se rozumí orientovaný a ohodnocený graf ve kterém [35]

- Vrcholy grafů odpovídají jednotlivým stavům automatu, tedy prvky množiny Q,
- Počáteční stav (q_0) se označuje příchozí šipkou
- Koncové stavy (F) se označují dvojitým kroužkem, popřípadě výchozí šipkou
- hrana či přechod z q do q'se označuje výpisem všech možných písmen abecedy, které stav q převede do stavu q', tedy prvky množiny $\{a \in \Sigma | \delta(q, a) = q'\}$.

Přechody se nezakreslují pro vrcholy, mezi kterými nejsou přechody pro žádné písmeno abecedy. V případě, že se vychází z vrcholu q a přechází zpět do stavu q, tak pro takový přechod se zakreslí smyčka na daném vrcholu.[35]



Obr. 13 Ukázka grafu konečného stavového automatu (převzato z [35])

Přechodová tabulka je tabulka, která má na místech řádků stavy automatu a ve sloupcích označené symboly abecedy, ve kterých buňka na řádku q a ve sloupci a udává stav $\delta(q, a)$. Počáteční stav q_0 je označen vstupní šipkou a koncový stav je označen koncovou šipkou, popřípadě kroužkem kolem čísla stavu. [35]

	a	b	c
→1	2	3	2
2	3	1	2
←3	1	2	1

Obr. 14 Ukázka tabulky konečného stavového automatu (převzato z [35])

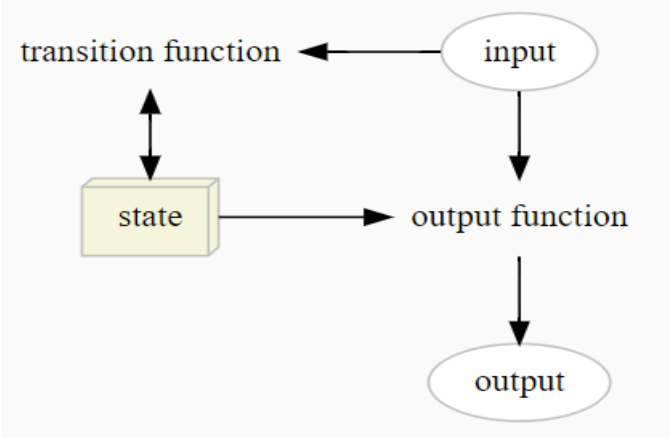
8.1. Mealyho stavový automat

Výstup tohoto automatu je generován na základě aktuálního stavu, ve kterém se automat nachází, a na hodnotě následujícího stavu. Mealyho stavový automat tedy disponují přiřazenou vstupní hodnotou a výstupní hodnotou pro každý přechod do stavu.

8.1.1. Definice mealyho stavový automat

Mealyho stavový automat je formálně definován uspořádanou šesticí $(S, S_0, \Sigma, \Delta, T, G)$, kde:

- S označuje neprázdnou konečnou množinu stavů
- S_0 označuje počáteční stav, $S_0 \in S$
- Σ označuje končenou vstupní abecedu
- Δ označuje konečnou výstupní abecedu
- T označuje přechodovou funkci ve formě zobrazení $S \times \Sigma \to S$
- G označuje výstupní funkci ve formě zobrazení $S \times \Sigma \to \Delta$



Obr. 15 Mealyho konečný stavový automat (převzato z [34])

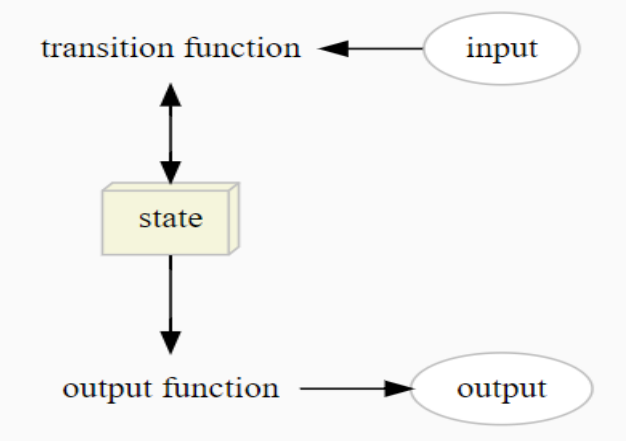
8.2. Mooreův automat

U těchto automatů se změna, kterou způsobí vstup projeví na výstupu až v následujícím vykonávaném stavu. Tyto automaty disponují možnostmi trvale poskytovat hodnotu svého vnitřního stavu, zpřístupněnou výstupem.

8.2.1. Definice Mooreova automatu

Mooreův stavový automat je formálně definován uspořádanou šesticí $(S, S_0, \Sigma, \Delta, T, F)$, kde:

- S označuje konečnou množinu stavů
- S_0 označuje počáteční stav, $S_0 \in S$
- Σ označuje vstupní abecedu
- Δ označuje výstupní abecedu
- T označuje přechodovou funkci ve formě zobrazení $S \times \Sigma \to S$
- F označuje výstupní funkci ve formě zobrazení $S \to \Delta$



Obr. 16 Mooreův konečný stavový automat (převzato z [33])

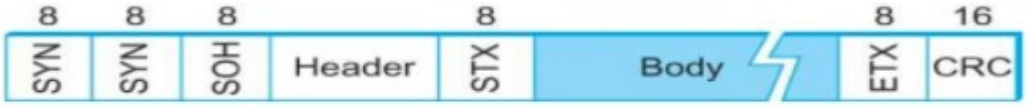
9. Metody pro rozpoznávání zpráv

Metody pro rozpoznávání zpráv se používají pro dekódování příchozí dat a určení informace ze zprávy. V základu existují dva druhy orientací, a tedy metody řešící znaky (byte) nebo samotné bity znaků.

- Znakově (byte) orientované protokoly (řeší obsah zpráv jako kolekci bajtů neboli kolekci znaků)
- Bitově orientované protokoly (řeší obsah zpráv jako kolekci bitů neboli kolekci jednotlivých logických stavů obsažený ve znaku (byte))

9.1. Binary synchronous communication (BISYNC, BSC)

Je protokol zaměřen na znakově (byte) orientovanou komunikaci s použitím poloviční duplexního spojovacího protokolu. Tato metoda nahradila starý protokol synchronous transmit-receive (STR). Tato metoda je odlišná než ostatní používané metody a to tím, že nemá stálý obsah zprávy, pro všechny zprávy poslané tímto protokolem. Bisync má pět různých formátu zpráv.[17]



Obr. 17 BYSYNC zpráva (převzato z [31])

- SYN – definuje začátek zprávy
- STX, ETX – uvozují porci dat, které mezi nimi obsaženy, kde STX (start of text) uvozuje začátek testu a ETX (end of text) uvozuje konec textu
- SOH (Start of Hreader) – uvozuje začátek hlavičky zprávy
- CRC (Cyclic Redundany Check) – určeno pro detekci chyby ve zprávy

Všechny formáty zpráv začínají dvěma SYN bajty, které slouží pro synchronizaci. Po úspěšné synchronizaci hledá přijímač znak, kterým uvozuje začátek dat ve zprávě. Znaky pro ukončení jsou obvykle (ETB nebo ETX). Obsah hlavičky není definovaný protokolem, ale závisí na použitém zařízení, pokud hlavička je obsažena ve zprávě, tak začíná znakem SOH a po hlavičce bezprostředně následuje znak STX uvozující začátek dat ve zprávě.[17]

9.1.1. Problémy při posílání zprávy

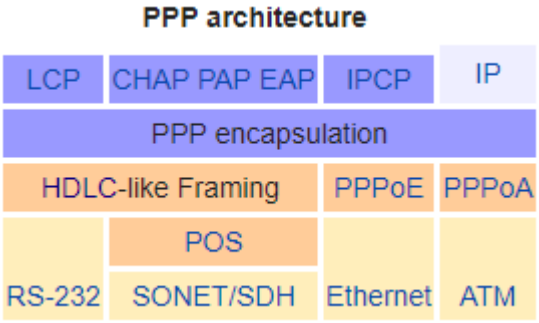
Z důvodu toho, že znak ETX může být obsažen v informacích, a to by způsobilo ukončení těchto dat a následně ke špatnému spočítání kontrolního součtu. Tento problém s předčasným ukončením je řešen tak, že objeví-li se v informaci zprávy znak ETX, tak dojde k záměně za znak DLE (data-link-escape). Z důvodu, že i znak DLE může být také obsažen v informaci zprávy, a tudíž by opět došlo k předčasnému ukončení zprávy nebo ke chybě, je tento znak ošetřen přidáním dalšího znaku DLE.[31]

9.2. Point-to-Point Protocol (PPP)

Je protokol používající datovou vrstvu a je použit ke přímou synchronní nebo asynchronní komunikaci mezi dvojici zařízení. Protokol obsahuje také definice pro komunikaci prostřednictvím Ethernetu (PPPoE) a ATM (PPPoA). Definice pro tento protokol je popsán ve standardu RFC 1661. Protokol byl vytvořen na základě specifikace HDLC.[14]

PPP je vrstvený a obsahuje 3 komponenty

- Zapouzdřenou komponentu, která je použita pro vysílání datagramů prostřednictvím zvolené psychické vrstvy
- Link Control Protocol (LCP) k vytvoření, nastavení a testování spojení
- Authentication Protocol (AP)
- Jeden nebo více Network Control protokol (NCP)



Obr. 18 Architektura protokolu PPP (převzato z [14])

9.2.1. Požadavky na fyzickou vrstvu

Tento protokol je schopen práce prostřednictvím většiny rozhraní DTE/DCE (jako jsou EIA RS-232-E, EIA RS-422, a CCITT V.35). Protokol požaduje od fyzické vrstvy, aby zařízení umožňovalo plný duplex, nebo zařízení, které mění duplex s tím, že musí umět operovat v asynchronně (pomocí start/stop), bitovou synchronizaci nebo oktet-synchronní mód.[13]

9.2.2. Definice obecné zprávy protokolu PPP

Tabulka níže popisuje obecný formát zpráv pro protokol PPP, definice zprávy se může lišit v závislosti o jakou komunikaci se jedná.

Tab. 2 Obecný protokol použitý v komunikaci PPP (převzato z [14])

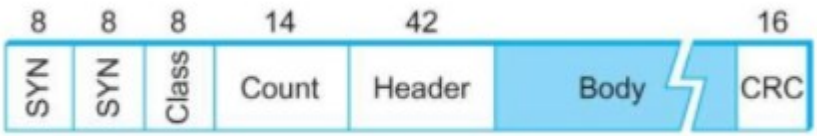
Název	Počet bajtů	Popis
Flag	1	Indikuje začátek a konec zprávy
Address	1	Broadcast adresa
Control	1	Kontrolní bajt
Protocol	1 nebo 2 nebo 3	Informační pole
Information	Volitelné	Datagram
Padding	Volitelné	Volitelný doplněk
FCS	2 (nebo 4)	Chybová kontrola

PPP definuje

- Hodnota Flag bajtu je nastavena na 0x7E
- Hodnota Escape bajtu je nastavena na 0x7D
- Pokud je v informaci obsažena hodnota odpovídající bajtu pro Flag nebo Escape, je tato informace ukončena prostřednictvím 0x7D a bajt informace je zakódován prostřednictvím logické funkce XOR a hodnoty 0x20. Tedy například pro 0x7E se stane 0x7D 0x5E a 0x7D se stane 0x7D 0x5D.

9.3. Digital Data Communication Message protocol (DDCMP)

Tento protokol je používán pro znakově orientovaný přenos, používá plný nebo poloviční duplex. Může být použít buď v synchronní či asynchronní komunikaci. Tento protokol používá položku Count, neboli počet, která určuje počet obsažených bajtů ve zprávě, tento počet je tedy nutné obdržet, aby došlo ke správnému konci zprávy. Pokud dojde k poškození obsahu položky určující počet bajtů, dojde k chybě. [16]



Obr. 19 DDCMP zpráva (převzato z [31])

- SYN – určuje začátek zprávy
- COUNT – počet bajtů ve zprávě

10. Návrh konečného stavového automatu (FSM) rozpoznávajícího zprávy

Pro návrh konečného stavového automatu byla zvolena metoda pro rozpoznání zprávy, dle protokolu PPP (Point to Point), přesněji jeho podobu popsanou dle protokolu RFC 1662, tento druh protokolu se nejčastěji využívá při sériové komunikaci. Při návrhu bylo nutné dbát v potaz, že začátek a konec paketu je vždy uvozen znakem (FLAG), tento znak nabývá hodnoty podle standardu 0x7E. Z důvodu toho, že tento znak může být obsažen také v samotné zprávě, je nutné, aby vysílač tento znak zakódoval a tím upozornil přijímač. Vysílač zakóduje znak FLAG tak, že nejprve vyšle znak ESC o hodnotě 0x7D a následně za ním 0x5E. Toto se opakuje, také pro znak ESC, který také může být obsažen v samotné zprávě, toto je provedeno zasláním znaku ESC a po něm hodnotu 0x5D. Obecná zpráva protokolu PPP definuje informace pro adresu, kontrol a protokol, ale při dvou bodovém spojení nejsou tyto informace potřebné, a tudíž základní definice zprávy je zobrazena na obrázku Obr. 20 níže.

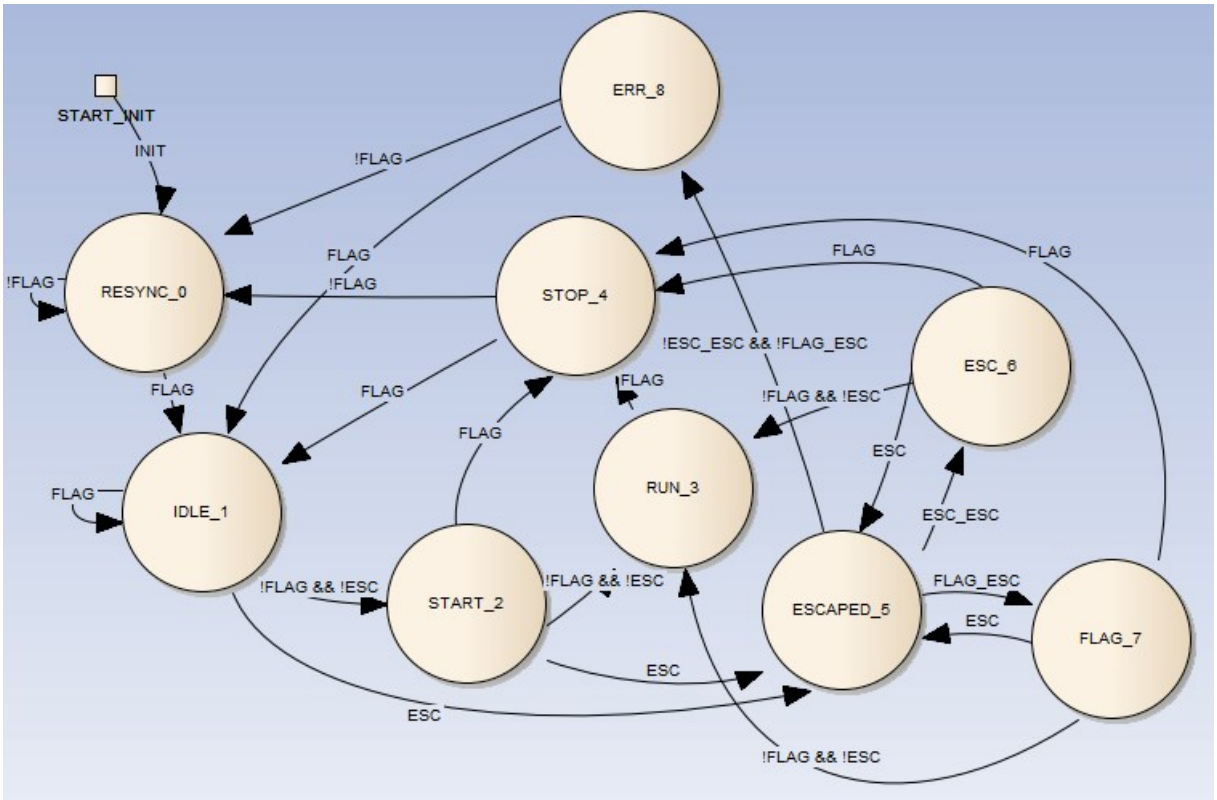
Obr. 20 PPP zpráva určena pro sériovou komunikaci (převzato z[15])

10.1. První verze návrhu FSM

Diagram navrhnutého FMS je zobrazen na obrázku Obr. 21, z diagramu je patrné, že existují 9 různých stavů, ve kterém se může automat nacházet. Přechod do jednotlivých stavů je závislý na bajtu, který je získán prostřednictvím sériového rozhraní UART.

Při návrhu stavového automatu jsem předpokládal, že při startu zařízení bude nastaven výchozí stav a tím bude stav RESYNC_0, v tomto stavu se bude očekávat příchod znaku o hodnotě FLAG, po obdržení tohoto znaku přejde automat do stavu IDLE_1. Ve stavu IDLE_1 pokud obdržíme znak o hodnotě FLAG, tak se stav nemění, neboť je možné, že znak o hodnotě FLAG, který se obdržel ve stavu RESYNC_0 byl ukončující, pokud neobdržíme znak jiný než FLAG, znamená to, že znak o hodnotě FLAG obdržený při stavu RESYNC_0 byl startovací a tudíž již můžeme obdržet platná data, tedy pokud ve stavu IDLE_1 obdržíme znak o hodnotě ESC, přejde automat do stavu ESCAPED_5, pokud obdržený znak bude nabývat jiných hodnot nežli ESC nebo FLAG, tak se přejde do stavu START_2. Ve stavu START_2 se již jedná o stav, kdy čteme data, a tudíž při obdržení hodnoty znaku FLAG, přejde automat do stavu STOP_4, pokud obdržíme znak ESC, tak přejde do stavu ESCAPED_5, pokud hodnota obdrženého znaku je rozdílná nežli hodnoty pro FLAG a ESC, přejde do stavu RUN_3. Ve stavu RUN_3 je automat ve stavu pracovním a přijímá data, tedy pokud obdržíme znak FLAG, tak přejdeme do stavu STOP_4, pokud ESC, tak přejdeme do ESCAPED_5, nebo zde zůstáváme, pokud obdržený znak bude nabývat jiných hodnot nežli FLAG nebo ESC. Stav ESCAPED_5 uvozuje to, že automat přešel do stavu, kde obdržel znak o hodnotě ESC, tudíž v tomto stavu se očekává příchod hodnoty znaku ESC_ESC, pokud ji obdrží, přejde automat do stavu ESC_6 nebo FLAG_ESC, pokud ji obdrží, přejde automat do stavu FLAG_7, které jsou v tomto stavu patřičně rozkódovány. Pokud však ve stavu ESCAPED_5 dostaneme hodnotu jinou nežli FLAG_ESC nebo ESC_ESC, jedná se o chybu, a tudíž přejde automat do stavu ERR_8. Ve stavu ESC_6, pokud obdržíme znak FLAG, přejdeme do stavu STOP_4, pokud obdržíme ESC, přejde se do stavu ESCAPED_5 a v případě jiného znaku přejde automat opět do stavu RUN_3. Ve stavu FLAG_7, pokud obdržíme znak FLAG, přejdeme do stavu STOP_4, pokud obdržíme ESC, přejde se do stavu ESCAPED_5 a v případě jiného znaku přejde automat opět do stavu RUN_3.

Stav STOP_4 je stav, kdy automat obdržel znak FLAG, při některých stavech, při kterých již automat přijímal zprávu tedy, START_2, RUN_3 nebo ESC_6, FLAG_7, v tomto stavu se čeká na příchod znaku FLAG, pokud jej obdrží, přejde automat do stavu IDLE_1 v jiném případě přejde do stavu RESYNC_0.



Obr. 21 Schéma stavového diagramu

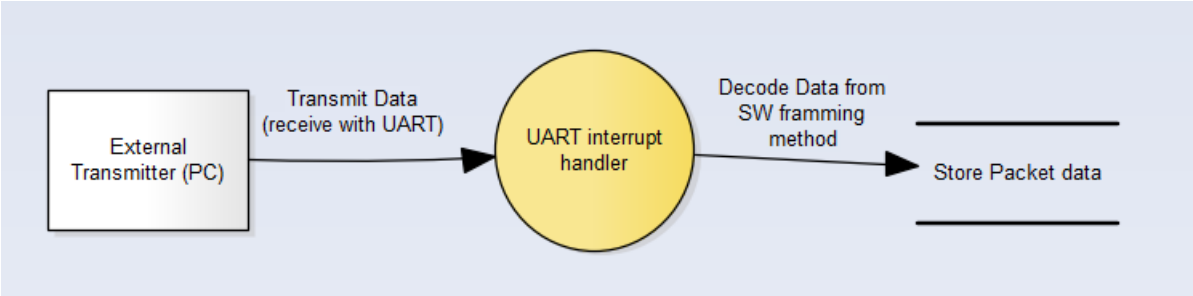
Tabulka Tab. 3 zobrazuje hodnoty, které jsou výstupy automatu při přechodu do jednotlivých stavů, s touto hodnotou lze následně pracovat, pouze za předpokladu, že je platná.

Tab. 3 Tabulka hodnot pro přechod do daného stavu

Přechod do stavu	Výstupní znak	Planost znaku
RESYNC_0	-	Neplatný
IDLE_1	-	Neplatný
START_2	Vstupní znak	Platný
RUN_3	Vstupní znak	Platný
STOP_4	-	Neplatný
ESCAPED_5	-	Neplatný
ESC_6	ESC	Platný
FLAG_7	FLAG	Platný
ERR_8	-	Neplatný

11. Softwarová implementace FSM

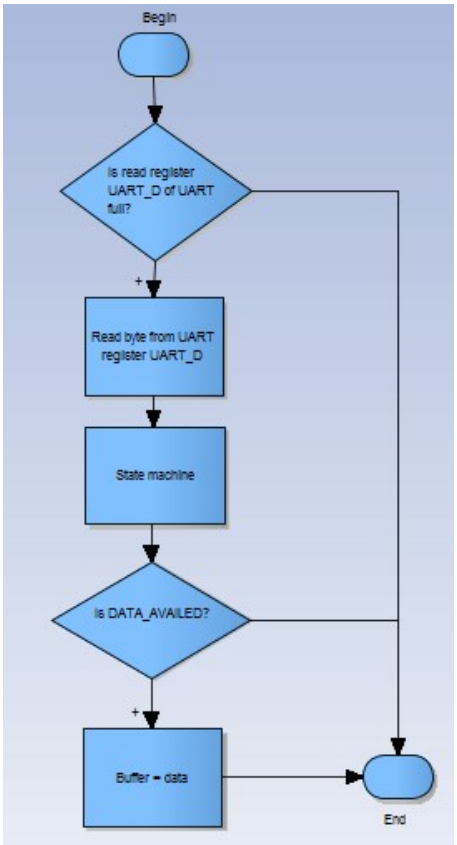
Softwarová implementace spočívá v tom, že řešení bude prováděno prostřednictvím CPU daného vývojového kitu. SW stavový automat bude přijímat znaky vyslané z počítače. Znaky budou přijímány prostřednictvím komunikační sběrnice UART, obsažené na vývojovém kitu FRDM-KL28Z. Připojení mezi periférii UART a počítačem bude řešeno prostřednictvím USB kabelu a konektoru Open-SDA. Po přijmutí znaku prostřednictvím periférie UART bude vyvolána obsluha přerušení, ve které se bude nacházet navržený SW stavový automat.



Obr. 22 Návrh SW implementace

11.1. Diagram obsluhy přerušení UART

Na Obr. 23 lze vidět diagram obsluhy přerušení od periférie UART. SW verze stavového automatu je navržena tak, že nastane-li přerušení od periférie UART, kontroluje se, zda toto přerušení bylo vyvoláno čtecím bufferem, tedy se kontroluje, zda byl nastaven daný příznak. Pokud je daný příznak nastaven, přečte se obsah přijímacího registru. Přečtený obsah, odpovídajícímu znaku následně je použit jako vstupní parametr SW stavového automatu.



Obr. 23 Diagram přerušení pro UART

Prototyp funkce pro sw implementaci je zobrazen níže:

```
uint8_t RFC1662_automatic_Final(state_machine_Final_t *state, uint8_t
*data_avail, uint8_t data_in)
```

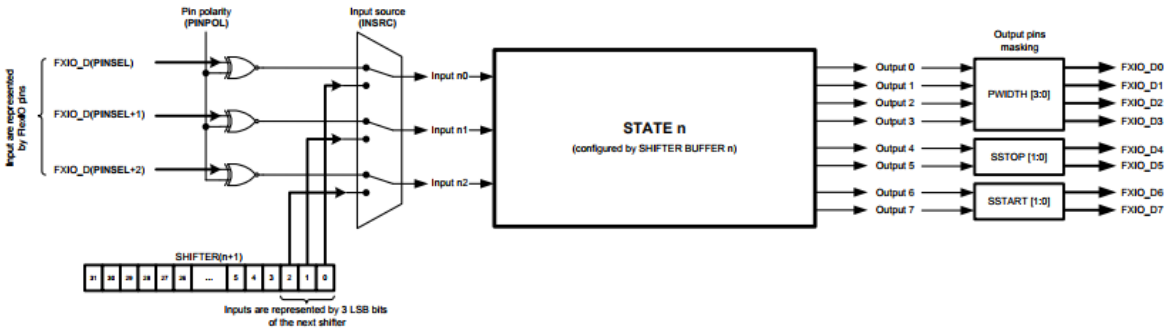
Znak, který byl obdržen periférii UART následně slouží jako jeden ze vstupních parametru funkce stavového automatu data_in. Spolu s přijatým znakem je vstupním parametrem této funkce pointer ukazující na aktuální stav *state, ve kterém se SW stavový automat nachází. Posledním vstupním parametrem funkce je pointer *data_avail, který určuje platnost návratové hodnoty funkce, která je závislá na aktuálním stavu.

12. Posouzení možností implementace FSM s použitím FlexIO

Při posuzování implementace FSM s použitím FlexIO je nutné se nejprve podívat na vlastnosti samotné periférie FlexIO, která je obsažena na vývojovém kitu FRDM-KL28Z. V kapitole NXP FlexIO, ve které se popisoval daný modul, bylo zjištěno, že periférie disponuje možností nastavení až 8 různých stavů, tedy navrhnutý SW stavový automat pro rozpoznávání zpráv s protokolem RFC1662, nebude lze použít a bude tedy nutné návrh stavového automatu poupravit.

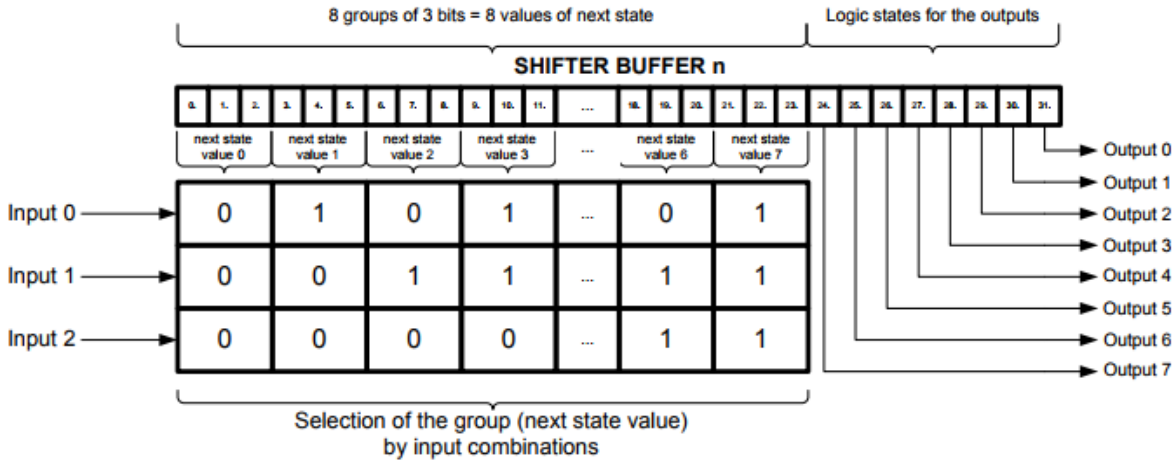
Další omezení plynoucí z dokumentace je problém volby příštího stavu periférie FlexIO a to z důvodu, že volba u periférie FlexIO je řešena prostřednictvím 3-vstupé hodnoty, reprezentované trojicí vstupních pinů nebo trojicí LSB bity v registru aktuálně zvoleného stavu. A tedy není možné přivést na vstup periférie FlexIO přijatý znak, reprezentující byte, tedy bude nutné vymyslet algoritmus, který přijatý znak zakóduje do 3-bitové hodnoty.

Zmíněný problém s volbou dalšího stavu lze vidět na obrázku Obr. 24, kde je patrné, že volba příštího stavu je závislá na kombinaci trojice vstupů.



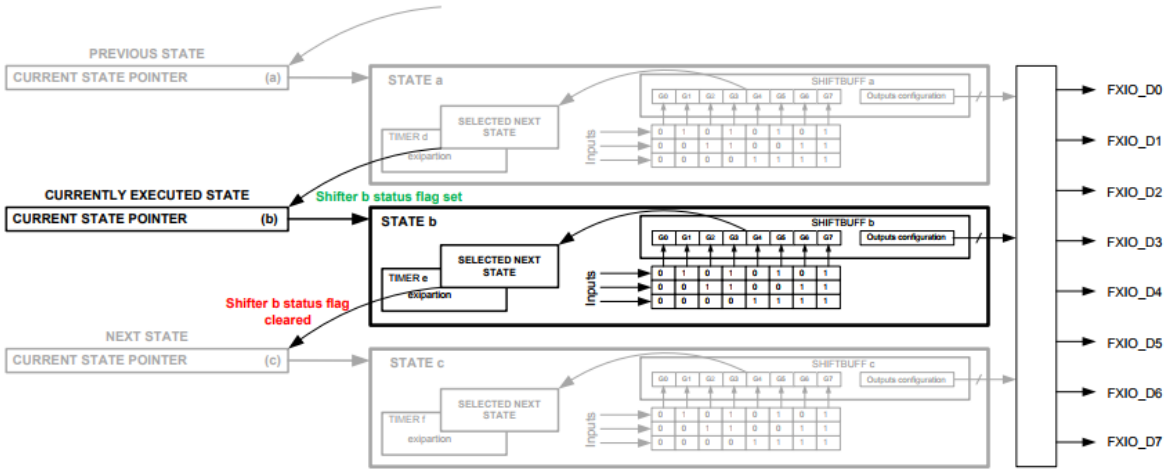
Obr. 24 Problém s volbou dalšího stavu (převzato z [26])

Při implementaci bude nutné vzít v potaz také to, že volba příštího stavu musí být přímo reprezentována hodnotou stavu, do kterého má nastat změna. Na obrázku Obr. 25 můžeme nalézt, požadovaný tvar a obsah registru.



Obr. 25 Nastavení registru shifter buffer(převzato z [26])

Následně objevený problém se týká časovače, protože aby došlo ke změně stavu je nutné, aby časovač, který je přiřazen k aktuálně vykonávaném stavu, dopočetl. Tento problém je velice závažný, a to z důvodů toho, že pro správnou funkčnost konečné automatu rozpoznávající zprávu dle protokolu RFC1662 musí automat měnit svůj stav v závislosti na přijatém znaku a aktuálním stavem ve kterém se periférie FlexIO nachází. Tento objevený problém lze nalézt na obrázku níže, kde je jasně patrné, že změna nastává až po expiraci časovače.



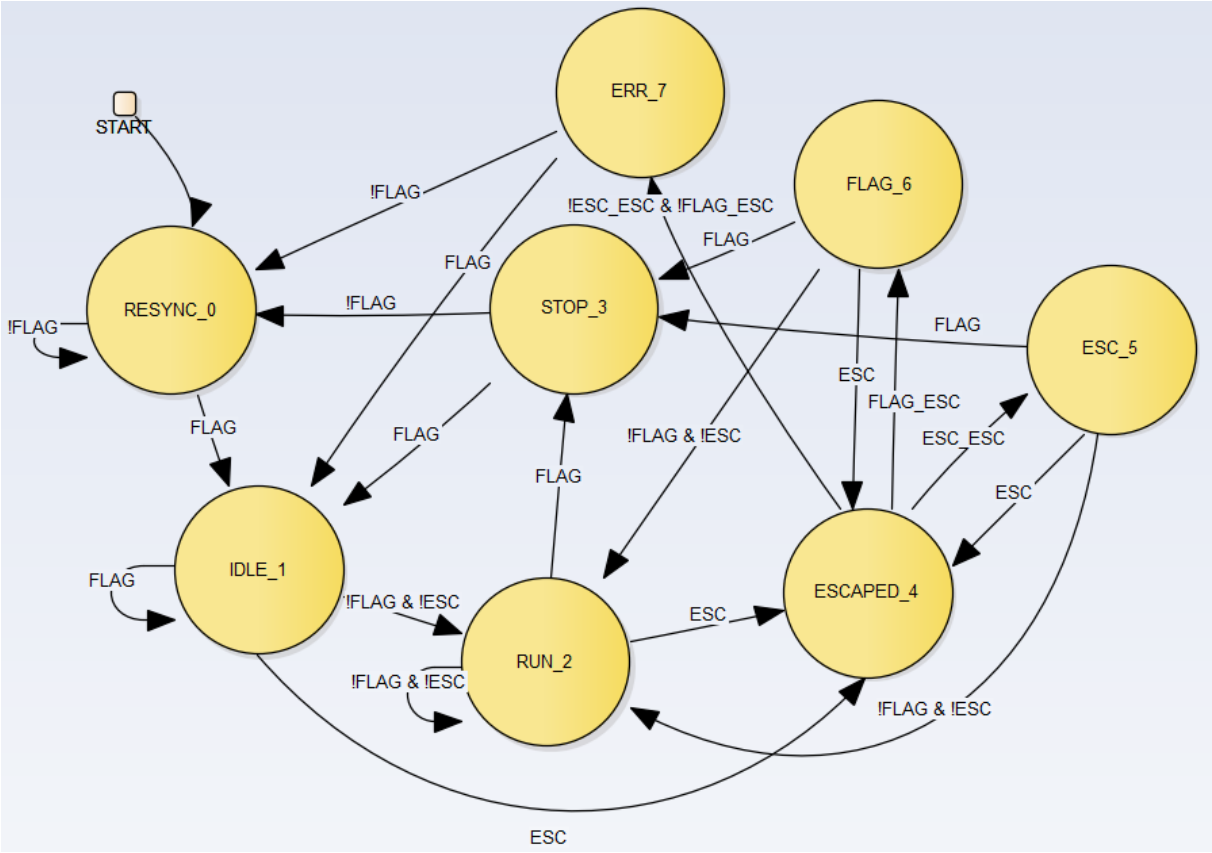
Obr. 26 Problém s přepínáním stavu v závislosti na časovači (převzato z [26])

Poslední objevený problém při implantaci stavového automatu do periférie FlexIO je fakt, že výstupní hodnota aktuálně zvoleného stavu by se měla nacházet v registru FLEXIO_SHIFTBUFF[31:24], odkud následně po zvolení stavu se zapíše na předdefinované výstupy periférie FlexIO. Toto představuje problém z důvodu toho, že výstup některých stavů má být závislý na přijatém znaku a tento znak by měl být následně zapsán do pole pro přijaté znaky neboli obsahy zprávy.

13. Návrh FSM s použitím FlexIO

Při návrhu FSM s použitím FlexIO je nutné brát v potaz problémy, které se ukázaly při vypracování rešerše této práce, a to tedy, že modul FlexIO obsažený na vývojovém kitu FRDM-KL28Z umožňuje implementaci pouze 8 různých stavů, takže první verze navrženého automatu v softwarové implementaci nelze použít.

Na Obr. 27 lze vidět navrhnutý stavový diagram automatu rozpoznávajícího zprávy dle protokolu RFC1662 a jeho jednotlivé stavy, které budou následně reprezentovat jednotlivé stavy periférie FlexIO. Z důvodů toho, že samotný modul pouze dovoluje možnost maximálně 8 různých stavů, muselo se k tomuto ohledu přihlédnout a pokusit se navrhnout adekvátní řešení, které bude používat pouze 8 stavů, oproti například různým SW implementacím, kde lze použít více stavů nežli pouze 8.



Obr. 27 Návrh FSM

Tabulka Tab. 4 ukazuje platnost a výstupní hodnotu každého jednotlivého stavu. V některých stavech, ve kterých nejsou platné data bude následně prováděny jiné možné operace.

Tab. 4 Tabulka hodnot pro přechod do daného stavu

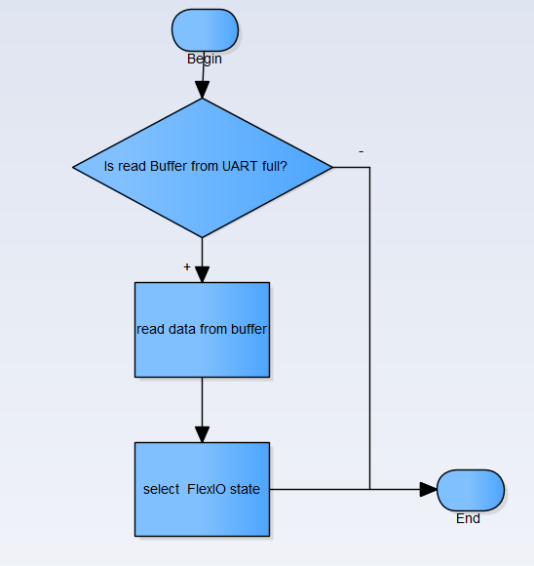
Přechod do stavu	Výstup	Planost znaku
RESYNC_0	-	Neplatný
IDLE_1	-	Neplatný
RUN_2	Vstupní znak	Platný
STOP_3	-	Neplatný
ESCAPED_4	-	Neplatný
ESC_5	ESC	Platný
FLAG_6	FLAG	Platný
ERR_7	-	Neplatný

13.1. První verze návrhu FSM

Tato verze sloužila k otestování správné funkčnosti návrhu a posouzení možnosti samotné implementace, také tato verze sloužila k lepšímu pochopení funkčnosti periférie FlexIO. Tato verze pracuje s 8 stavy viz. Obr. 27, podobně jako u softwarové implementace je tento FSM navrhnutý, tak aby byl schopen rozpoznávat zprávy podle protokolu RFC1662. Zde však bylo použito jiné hodnoty pro dané znaky, a to z důvodu, že modul FlexIO neumožňuje přepínání prostřednictvím 8 bitů, tedy znaku, ale prostřednictvím pouze 3 bitové hodnoty. V tomto návrhu se bude řešit změna stavu periférie FlexIO v obsluze přerušení periférie UART, obdobně jako u SW implementace. Jak již bylo zmíněno předtím, tato verze návrhu sloužila k otestování funkčnosti, a tudíž tento návrh není korektní.

13.1.1. Diagram přerušení UART

Na Obr. 28 lze vidět průběh obsluhy přerušení od periférie UART. Lze vidět, že průběh je podobný k SW implementaci, také se nejprve zjistí, zda je přijímací buffer plný. Poté se přečte přijatý znak, ale zde se vyhodnocuje vybírání stavu jinak a to tak, že se nejprve prostřednictvím zjištění aktuálně zvoleného stavu a obsahu registru SHIFTERBUF daného stavu se určí hodnota dalšího stavu.



Obr. 28 Diagram přerušení pro UART s FlexIO

13.1.2. Implementace FSM

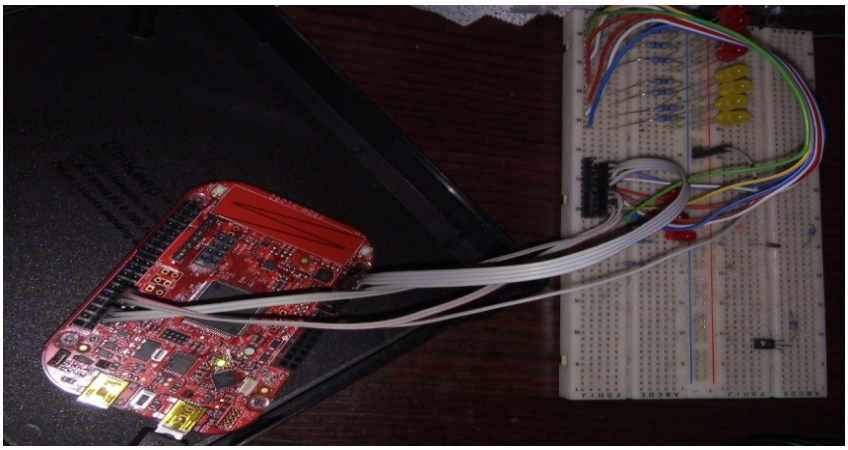
Jednotlivé hodnoty pro přechod budou obdrženy prostřednictvím komunikační sběrnice UART. Tabulka Tab. 5 zobrazuje definice pro jednotlivé přechody v jednotlivých stavech, tato tabulka byla vytvořena podle mnou definovaných předpokladů pro jednotlivé přechody, tedy pro hodnoty FLAG = 001, ESC = 010. ESC_ESC = 011 a FLAG_ESC = 100. Tyto hodnoty jsou pouze pro otestování funkčnosti daného stavového automatu a byly zapsány do registrů pro daný stav.

Tab. 5 Definice přechodů pro jednotlivé stavy napsané v registrů SHIFTERBUFn

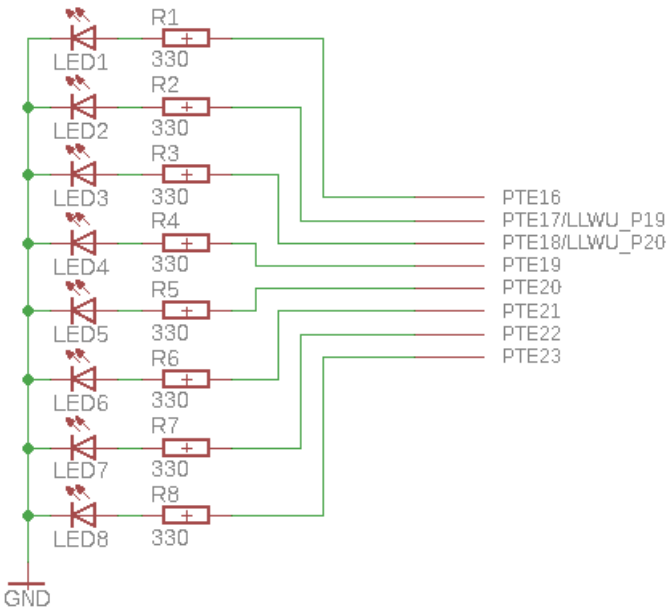
RESYNC_0								
Hodnota vstupů	0	1	2	3	4	5	6	7
Přechod	RESYNC_0	IDLE_1	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0
IDLE_1								
Hodnota vstupů	0	1	2	3	4	5	6	7
Přechod	RUN_2	IDLE_1	ESCAPED_4	RUN_2	RUN_2	RUN_2	RUN_2	RUN_2
RUN_2								
Hodnota vstupů	0	1	2	3	4	5	6	7
Přechod	RUN_2	STOP_3	ESCAPED_4	RUN_2	RUN_2	RUN_2	RUN_2	RUN_2
STOP_3								
Hodnota vstupů	0	1	2	3	4	5	6	7
Přechod	RESYNC_0	IDLE_1	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0
ESCAPED_4								
Hodnota vstupů	0	1	2	3	4	5	6	7
Přechod	ERR_7	ERR_7	ERR_7	ESC_5	FLAG_6	ERR_7	ERR_7	ERR_7
ESC_5								
Hodnota vstupů	0	1	2	3	4	5	6	7
Přechod	RUN_2	STOP_3	ESCAPED_4	RUN_2	RUN_2	RUN_2	RUN_2	RUN_2
FLAG_6								
Hodnota vstupů	0	1	2	3	4	5	6	7
Přechod	RUN_2	STOP_3	ESCAPED_4	RUN_2	RUN_2	RUN_2	RUN_2	RUN_2
ERR_7								
Hodnota vstupů	0	1	2	3	4	5	6	7
Přechod	RESYNC_0	IDLE_1	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0

13.1.3. Zapojení

Zapojení na Obr. 29 ukazuje připojení nepájivého pole s led diodami pro otestování, zda dochází ke správnému přepínání stavů. Tyto led diody byly připojeny na PORTE piny 16 až 23, které odpovídaly výstupním pinům modulu FlexIO. Přepínání jsem testoval prostřednictvím zasílání čísel 0 až 7 prostřednictvím terminálu Putty.



Obr. 29 Zapojení KL28z s nepájivým polem



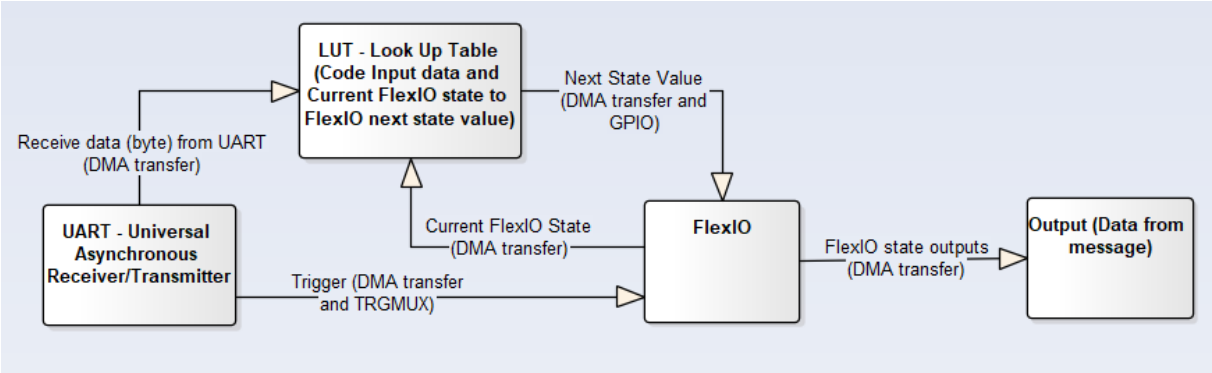
Obr. 30 Schéma zapojení k testu

13.2. Finální verze návrhu FSM

Po zjištění potřebných informací o chování periférie FlexIO, které byly obdrženy z první verze návrhu, bylo následně možné upravit samotný návrh stavového automatu. Při návrhu se bude vycházet z předpokládaného stavového diagramu, který lze nalézt v kapitole 13. Tedy navržený stavový automat bude obsahovat 8 různých stavů.

13.2.1. Obecný návrh

Návrh FSM s použitím periférie FlexIO bude vypadat následovně, nejprve se prostřednictvím periférie UART (Universal Asynchronous Receiver/Transiver) přijme znak, který se prostřednictvím jednoho z kanálu periférie DMA, přenese na LUT (look up table), kde společně s aktuálním stavem, ve kterém se nachází periférie FlexIO, zjistí hodnota dalšího stavu. Po zvolení dalšího stavu, musí dojít k spouštěcímu signálu pro časovač periférie FlexIO, aby mohlo dojít ke změně stavu, tento spouštěcí signál bude přiveden na časovač FlexIO prostřednictvím periférie TRGMUX a hodnota bude zapsána prostřednictvím kanálem DMA. Po změně stavu periférie FlexIO dojde přenos informací v závislosti na zvoleném stavu periférie FlexIO na výstup.



Obr. 31 Návrh k implementaci FSM

Tab. 6 Definice přechodů pro jednotlivé stavy napsané v registrů SHIFTERBUFn

RESYNC_0								
Hodnota vstupů	0	1	0	0	0	0	0	0
Přechod	RESYNC_0	IDLE_1	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0
IDLE_1								
Hodnota vstupů	2	1	4	2	2	2	2	2
Přechod	RUN_2	IDLE_1	ESCAPED_4	RUN_2	RUN_2	RUN_2	RUN_2	RUN_2
RUN_2								
Hodnota vstupů	2	3	4	2	2	2	2	2
Přechod	RUN_2	STOP_3	ESCAPED_4	RUN_2	RUN_2	RUN_2	RUN_2	RUN_2
STOP_3								
Hodnota vstupů	0	1	0	0	0	0	0	0
Přechod	RESYNC_0	IDLE_1	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0
ESCAPED_4								
Hodnota vstupů	7	7	7	5	6	7	7	7
Přechod	ERR_7	ERR_7	ERR_7	ESC_5	FLAG_6	ERR_7	ERR_7	ERR_7
ESC_5								
Hodnota vstupů	2	3	4	2	2	2	2	2
Přechod	RUN_2	STOP_3	ESCAPED_4	RUN_2	RUN_2	RUN_2	RUN_2	RUN_2
FLAG_6								
Hodnota vstupů	2	1	4	2	2	2	2	2
Přechod	RUN_2	STOP_3	ESCAPED_4	RUN_2	RUN_2	RUN_2	RUN_2	RUN_2
ERR_7								
Hodnota vstupů	0	1	0	0	0	0	0	0
Přechod	RESYNC_0	IDLE_1	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0	RESYNC_0

14. Implementace FSM s použitím FlexIO. Nalezení alternativního řešení, pokud vlastnosti FlexIO neumožní implementaci ve stanoveném rozsahu

14.1. Implementace návrhu FSM s použitím FlexIO

Před samotnou implementací nejprve bude vysvětleno, jak byly vyřešeny jednotlivé problémy s přepínáním stavu po dopočtení časovače, problém s volbou dalšího stavu a problém s výstupní hodnotou stavu.

14.1.1. Řešení problému z přepínání stavů po dopočtení časovače

Při řešení tohoto problému bylo objeveno, že vývojový kit FRDM-KL28Z obsahuje periférii Trigger MUX Control (TRGMUX), která umožňuje použít některé periférie jako zdroj spouštěcího signálu pro jiné periférie. Tedy prostudovaní dokumentace bylo zjištěno, že periférie TRGMUX umožňuje přivést spouštěcí signál k časovači FlexIO. Následně bylo nutné zvolit periférii, která bude sloužit jako zdroj spouštěcího signálu, k tomu byl následně zvolen pin nacházející se na portu, a to z důvodu, že ze všech možných zdrojů byl jediný, u kterého se dalo rozumně ovládat, kdy dojde k spouštěcímu signálu.

Při snaze zrealizovat spouštění časovače prostřednictvím pinu bylo zjištěno několik problémů, které plynuly z nedokonalé dokumentace k danému čipu a bylo tedy nutné zjistit chování časovače FlexIO při změně logické hodnoty z 0 na logickou hodnotu 1 na nastaveném pinu. A to, zda dojde ke změně hodnoty v časovači v reakci na hranu změny signálu nebo zda bude hodnota v časovači se měnit do té doby, nežli se opět změní hodnota na pinu. Dále bylo nutné zjistit, také zda pin může být nastaven ve výstupním režimu, a tedy zda lze zapsat na něj hodnotu přímo nebo musí být nastaven jako vstupní a muselo by se použít propojení prostřednictvím vodiče.

Z důvodu neznalosti těchto informací byla implementace z počátku složitá a nedařilo se správně spouštět časovač FlexIO bylo nutné přistoupit k sepsání testovací aplikace, kde bude nutné zjistit potřebné informace. K tomu byla vybrána periférie DAC. Po tomto testu bylo zjištěno, že zdroj spouštěcího signálu, tedy pin může být nastaven ve výstupním režimu a že spouštěcí signál se generuje v reakci na vzestupnou hranu přivedenou na daný pin.

14.1.2. Řešení problému s volbou dalšího stavu

Z důvodů, že volba příštího stavů je reprezentovaná 3-bitovou hodnotou, tedy hodnotami 0-7 a ne přijatým znakem, bylo nutné tento znak společně s aktuálně zvoleným stavem periférie FlexIO zakódovat do 3-bitové hodnoty pro volbu příštího stavu, který následně by měl být zapsán na výstup prostřednictvím kanálu z periférie DMA. Toto bylo vyřešeno prostřednictvím dvouúrovňové náhledové tabulky (look-up table, zkráceně LUT). Použití dvouúrovňové tabulky je z důvodu, že za předpokladu, že pro každý jednotlivý stav by se musela vytvořit tabulka o velikosti 256 prvků, by celková velikost všech tabulek byla 2048, tedy 8x256 s tím, že dle předpokladů by musel mít každý jednotlivý prvek z tabulky velikost odpovídající 4 byte, tak by bylo použito 8192 bytů paměti, a to by mohlo vést k jejímu přeplnění a mohlo by to vést k přepisu paměti či jiným problémům. Z tohoto důvodů se rozdělila jedna tabulka rozdělila na dvě, kde první o velikosti 2048 bude obsahovat bytové prvky, které budou reprezentovat posuny pro druhou tabulku, která bude obsahovat pouze 8 hodnot, každá o velikosti 4 byte, reprezentující výstupy na daný port.

První LUT je zobrazena na obrázku níže, tento obrázek zobrazuje, jak vypadají jednotlivé posuny pro první stav periférie FlexIO. Tedy lze vidět, že pouze pro znak odpovídajícímu FLAGU (126) má zde definovanou hodnotu jinou, nežli je hodnota 0, a tedy pouze znak FLAG způsobí změnu do jiného stavu periférie FlexIO. Na obrázku lze vidět, že u definice této tabulky je nastaven __atrribute__((aligned(65536))), toto je parametr pro kompilátor a říká, aby byla adresa tabulky zarovnaná na 2 byte, tedy aby poslední dva byte adresy měly nulové hodnoty a to z důvodu toho, aby bylo možné správně zvolit prvek z dané tabulky prostřednictvím DMA kanálů a to tak, že výběr bude realizován z adresy dané:

$$Základní\ adresa\ LUT + 1B\ odpovídající\ stavu\ FlexIO + 1B\ odpovídající\ přijatému\ znaku$$

Velikost této tabulky je 2048, což odpovídá velikosti 8x256, také mohlo být použito definice prostřednictvím dvourozměrného pole [8][256], protože v adresovém prostoru jsou adresy uspořádané stejně.

```
volatile __attribute__((aligned(65536))) uint8_t LUT_NextState[2048] =
{
    // next state for state 0
    //  (if receive FLAG (0x7E = 126), write to output 4)
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

Obr. 32 Ukázka první náhledové tabulky pro stav 0

Druhá LUT již obsahuje hodnoty, které se zapisují na výstup. Pro výběr správné hodnoty výstupu složí první LUT, kde jsou zapsány hodnoty offsetu od 0 do 28, tyto hodnoty korespondují s tím, že velikost jednotlivých prvků v tabulce je 32 bitová, tedy 4 byte, a tedy pro správné zvolení hodnoty je nutné pracovat s násobky 4 offsetu. Zde jde opět vidět parametr kompilátoru __attribute__((aligned(256))), zde je opět z důvodů, aby tabulka byla zarovnaná, ale zde stačí zarovnání na poslední byte dané adresy, tedy aby nabýval nulové hodnoty.

```
volatile const __attribute__((aligned(256))) uint32_t LUT_Output[8] =
{
    0x00000000, // offset 0 from LUT_NextState (value to write on PORTD pin 1, 2, 3 for set state 0)
    0x00000002, // offset 4 from LUT_NextState (value to write on PORTD pin 1, 2, 3 for set state 1)
    0x00000004, // offset 8 from LUT_NextState (value to write on PORTD pin 1, 2, 3 for set state 2)
    0x00000006, // offset 12 from LUT_NextState (value to write on PORTD pin 1, 2, 3 for set state 3)
    0x00000008, // offset 16 from LUT_NextState (value to write on PORTD pin 1, 2, 3 for set state 4)
    0x0000000A, // offset 20 from LUT_NextState (value to write on PORTD pin 1, 2, 3 for set state 5)
    0x0000000C, // offset 24 from LUT_NextState (value to write on PORTD pin 1, 2, 3 for set state 6)
    0x0000000E  // offset 28 from LUT_NextState (value to write on PORTD pin 1, 2, 3 for set state 7)
};
```

Obr. 33 Druhá náhledová tabulka

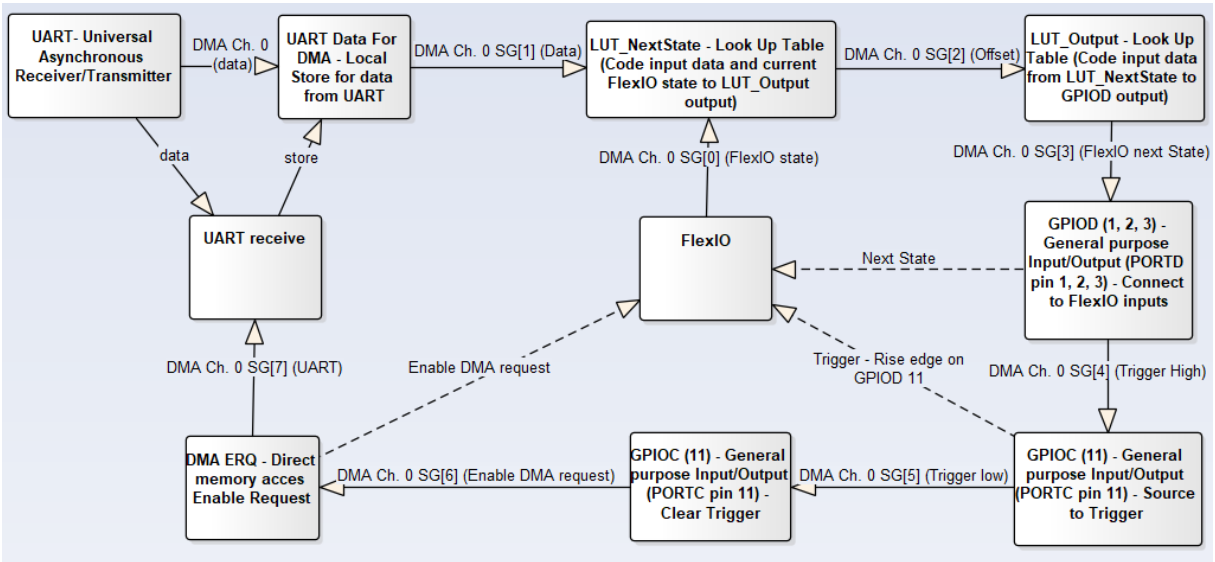
14.1.3. Implementace přijmutí znaku a jeho zpracování

V této části se ukáže řešení části od přijmutí znaku prostřednictvím periférie UART až po výběr příštího stavu a spouštění časovače periférie FlexIO.

Při návrhu bylo přihlédnuto na požadavek, že celkové řešení konečného stavové automatu prostřednictvím periférie FlexIO má pracovat bez zapojení CPU (control process unit), tedy bez procesoru. Takže pouze za použití různých periférii obsažených na vývojové platformě KL28Z.

Na obr níže můžeme vidět návrh prostřednictvím DMA kanálu 0 a operace Scather/Gather, která umožní, aby jeden kanál mohl zpracovávat více přenosů a to tak, že po skončení aktuálně zvolené TCD struktury si nahraje novou TCD strukturu.

Návrh, tedy funguje následovně, nejprve pro příjmu dat prostřednictvím periférie UART se vygeneruje DMA požadavek pro kanál 0, což způsobí jeho spuštění a ten přenese data z přijímacího registru LPUART0->DATA do globální proměnné UARTDataForDMA, to je z důvodu toho, že obsah registru LPUART0-DATA se automaticky vymaže po přečtení, a tak by bylo možné pouze přečíst jednou a následně by byly ztraceny. Po ukončení přenesu prostřednictvím DMA kanálu 0 se nahraje nová TCD struktura, která má za úkol přenést aktuální stav, ve kterém se nachází periférie FlexIO na adresu LUT_NextState, po ukončení přenosu se opět nahraje nová TCD struktura. Poté se přenese hodnota přijatého znaku z proměnné UartDataForDMA opět do adresy LUT_NextState, po ukončení přenosu se nahraje nová TCD struktura. Následně se z adresy dané základní adresou LUT_NextState, aktuálním stavem periférie FlexIO a přijetým znakem, přenese hodnota do adresy LUT_Output, po ukončení přenosu se opět nahraje nová TCD struktura. Následně se z adresy dané LUT_Output a offsetem z LUT_NextState přenese hodnota na výstup, potažmo vstup FlexIO periférie, po ukončení přenosu se nahraje nová TCD struktura. Po zapsaní dat se přistoupí k přenosu dat z paměti, které odpovídají logické hodnotě na pin 11, tedy 0x00000100, tato hodnota se zapíše na pin, který slouží jako zdroj spouštěcího signálu, toto způsobí spuštění časovače a následnou změnu stavu FlexIO, poté se nahraje nová TCD struktura. Aby bylo následně opět způsobit spouštěcí signál, musí se hodnota na daném pinu přivést do logické hodnoty, čemuž složí další TCD struktura, která z paměti zapíše nuly na pin, který slouží jako zdroj spouštěcího signálu, poté se opět nahraje nová TCD struktura. Po resetu pinu již lze přistoupit k povolení zpracování požadavku DMA pro FlexIO a poté již se nahraje výchozí TCD struktura pro příjem dat z periférie UART a celý proces se následně opakuje.



Obr. 34 Implementace DMA kanálu 0

Popis Scatter/Gather operace

Jak již bylo dříve zmíněno operace Scatter/Gather u periférie DMA slouží k možnosti nahrání nové TCD (Transfer Control Descriptor) struktury dat po ukončení aktuálně vykonané struktury. Tato operace byla nutná použít, neboť periférie DMA obsažená na vývojové platformě FRDM KL28Z disponuje pouze 8 kanály, a tudíž by nebylo možné správně navrhnout požadovanou funkci stavového automatu prostřednictvím periférie FlexIO bez probuzení CPU a jeho použití.

Návrh DMA kanálu 0 a jeho TCD struktur

V tabulce lze vidět základní konfiguraci pro kanál 0 periférie DMA, lze vidět, že bylo nutné použít 8 Scatter/Gather operačních TCD struktur pro realizování všech potřebných zásahů bez nutnosti použití CPU. V tabulce lze také vidět i počty přenášených bytů a z jaké adresy se data čtou a na kterou adresu se následně zapíší.

U operací Scatter/Gather 0 a 1 kanálu 0 si lze všimnout, že cílová adresa je SSADDR Scatter/Gather v této proměnné se nachází hodnota základní adresy první náhledové tabulky (look-up table, LUT) a tedy tyto operace následně zapíší hodnotu na poslední dva byte adresy. Tyto hodnoty odpovídají aktuálnímu stavu periférie FlexIO a přijatému znaku a jsou použity pro výběr dané položky. Stejně tomu je i u operace Scatter/Gather 2 kanálu 0, která následně z adresy přečte zvolený posun a zapíše je na SSADDR Scatter/Gather 3, kde se nachází základní adresa druhé náhledové tabulky, která obsahuje hodnoty výstupu.

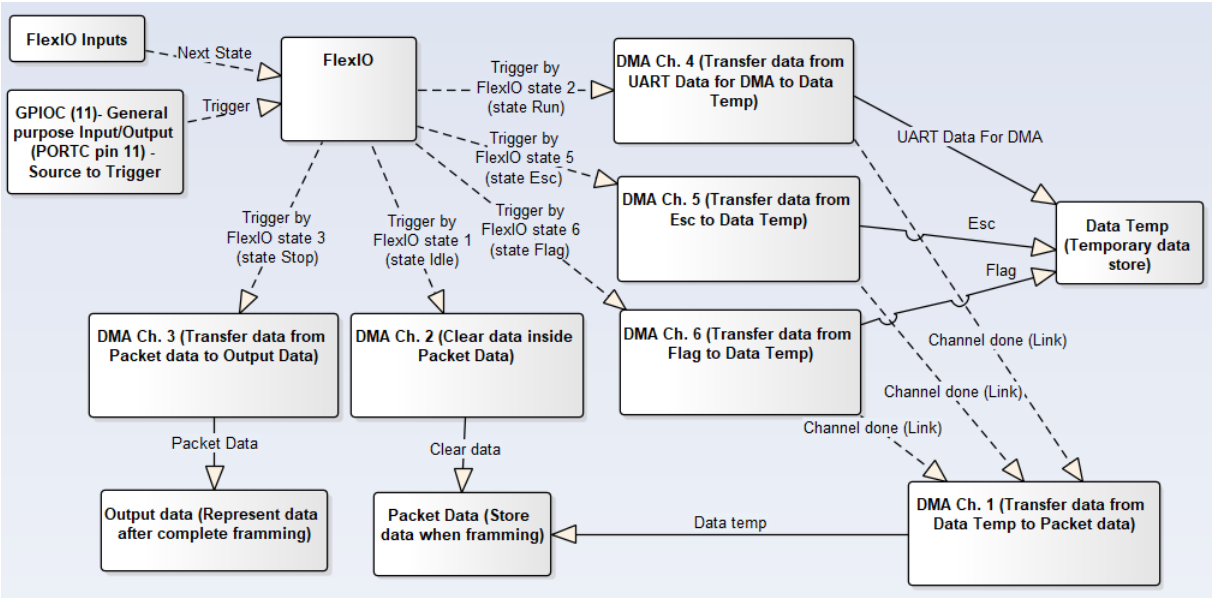
Tab. 7 Popis implementace DMA kanálu 0

Název operace	Z adresy	Na adresu	Počet bytů	Účel operace
DMA kanál 0	LPUART->DATA	UARTDataForDMA	1	Zápis dat do paměti
DMA kanál 0 (Scatter/Gather 0)	FLEXIO0->SHIFTSTATE	SSADDR (Scather/Gather 2) – základní adresa LUT_NextState	2	Volba prvku z tabulky
DMA kanál 0 (Scatter/Gather 1)	UARTDataForDMA	SSADDR (Scather/Gather 2) – základní adresa LUT_NextState	1	Volba prvku z tabulky
DMA kanál 0 (Scatter/Gather 2)	LUT_NextState	SSADDR (Scather/Gather 3) – základní adresa LUT_Output	1	Volba prvku z tabulky
DMA kanál 0 (Scatter/Gather 3)	LUT_Output	GPIOD->PDOR	4	Zápis dalšího stavu na výstup
DMA kanál 0 (Scatter/Gather 4)	TriggerPIN_11_H	GPIOC->PDOR	4	Spouštěcí signál

Název operace	Z adresy	Na adresu	Počet bytů	Účel operace
DMA kanál 0 (Scatter/Gather 5)	TriggerPIN_11_L	GPIOC->PDOR	4	Mazání spouštěcího signálu
DMA kanál 0 (Scatter/Gather 6)	DMA_Channel_0_ERQ Enabled_FlexIO_u	DMA0->ERQ	1	Povolení zpracování požadavku od DMA
DMA kanál 0 (Scatter/Gather 7)	LPUART->DATA	UARTDataForDMA	1	Zápis dat do paměti

14.1.4. Implementace zpracování periférií FlexIO

Implementace zpracování je opět obdobná k implementaci přijmu, a tedy většina přenosových operací je řešená prostřednictvím některého z kanálu periférie DMA. Při implementaci se muselo vzít v potaz, že hodnota pro volbu dalšího stavu musí být přivedena na vstupní piny FlexIO periférie a následně musí být vyvolán spouštěcí signál a přiveden na časovač periférie FlexIO daného stavu. Po změně aktuálního stavu se v závislosti, zda stav je použit jako požadavek některého z kanálu periférie DMA, se spustí vykonávání kanálu DMA.



Obr. 35 Implementace DMA a FlexIO

Implementace DMA Kanálů

V tabulce lze názorněji vidět použití jednotlivých kanálů DMA, které jsou použity při realizaci stavového automatu za pomocí periférie FlexIO. Z tabulky lze vidět, že kanály, které zapisují hodnoty do paměti DataTemp, mají následně propojení (link) s kanálem 1, toto je z důvodu, aby nedocházelo k přepisu dat v packetu, musí k danému paketu vždy přistupovat pouze jeden DMA kanál.

Tab. 8 Popis implementace ostatních DMA kanálu a FlexIO

Název operace	Trigger Source	Z adresy	Na adresu	Počet bytů	Link	Účel kanálu
DMA kanál 1	None (Link)	DataTemp	PacketData	1	-	Zápis přijatých dat do packetu
DMA kanál 2	FlexIO state 1	ZeroData	PacketData	1500	-	Mazání packetu pro možnost přijmu nových dat
DMA kanál 3	FlexIO state 3	PacketData	DataOutput	1500	-	Zápis hodnot z packetu do výstupních dat
DMA kanál 4	FlexIO state 2	UARTDataForDMA	DataTemp	1	1	Zápis přijatého znaku do paměti
DMA kanál 5	FlexIO state 5	Esc	DataTemp	1	1	Zápis znaku ESC do paměti
DMA kanál 6	FlexIO state 6	Flag	DataTemp	1	1	Zápis znaku FLAG do paměti

14.1.5. Konfigurace periférie FlexIO

Pro možnost pracovat s periférii FlexIO bylo nejprve nutné tuto periférii správně inicializovat a konfigurovat jednotlivé hodnoty pro příští stavy daných stavů. V této kapitole budu popisovat jednotlivé funkce, které slouží k nastavení a konfiguraci periférii FlexIO.

Inicializace periférie FlexIO

K tomuto účelu byla vytvořena funkce **`FlexIO_Init`**, jejich prototyp je:

```
void FlexIO_Init(void)
```

V této funkci se povolí hodiny do periférie FlexIO a také se nastaví jaké hodiny budou použity. Následně se zde nastavují jednotlivé piny použité pro periférii FlexIO jako vstupní piny byly nastaveny piny 0, 1, 2 nacházející se portu B a jako výstupní piny byly nastaveny piny 16-23 na portu E, výstupní piny nejsou při implementaci využity. Mimo nastavení pinů pro periférii FlexIO se zde nachází také nastavení pinů pro propojení na vstupy a nastavení pinu na zdroj spouštěcího signálu pro časovač periférie. Pro připojovací piny byly zvoleny piny 1, 2, 3 na portu D tyto piny budou připojeny prostřednictvím vodičů na vstupní piny periférie FlexIO a bude se na ně zapisovat hodnota dalšího stavu.

Jako pin, který slouží jako zdroj spouštějícího signálu byl zvolen pin 11 na portu C a musel být správně nastaven jeho registr, aby spouštěl signál v reakci na vzestupnou hranu neboli na změnu z logické 0 na logickou 1.

Konfigurace posunovačů (shiftbuffers) periférie FlexIO

K tomuto účelu byla vytvořena funkce **`FlexIO_Init_Shifters`**, jejich prototyp je.

```
void FlexIO_Init_Shifters(void)
```

V této funkci se nachází před inicializované pole, které obsahuje nastavení všech hodnot pro jednotlivé stavy včetně i nadefinovaných hodnot pro volbu následujících stavů. Tyto hodnoty se následně zapíší do registrů SHIFTCFG a SHIFTCTL periférie FlexIO. Součástí této funkce je také nastavení a zapsání hodnot dalších stavů do registrů SHFITBUF daného stavu a zapsání možnosti pro generování DMA požadavku do registru SHIFSDEN.

Tab. 9 Ukázka některých nastavovaných parametrů pro jednotlivé stavy FlexIO

Stav FlexIO	Časovač	Mód stavu	Volba stavů	Požadavek DMA
0	0	Stavový	Vstupní piny	Ne
1	0	Stavový	Vstupní piny	Ano
2	0	Stavový	Vstupní piny	Ano
3	0	Stavový	Vstupní piny	Ano
4	0	Stavový	Vstupní piny	Ne
5	0	Stavový	Vstupní piny	Ano
6	0	Stavový	Vstupní piny	Ano
7	0	Stavový	Vstupní piny	Ne

Konfigurace časovačů periférie FlexIO

K tomuto účelu byla vytvořena funkce **`FlexIO_Init_Timers`**, jejich prototyp je.

```
void FlexIO_Init_Timers(void)
```

V této funkci se provádí konfigurace časovače 0 periférie FlexIO, který je použit pro všechny stavy periférie FlexIO. Tento časovač musí být nastaven jako 16 bitový čítač a jeho hodnota musí se dekrementovat po přivedení spouštěcího signálu, a to tak aby po jeho přivedení došlo k dopočtení daného časovače, a tudíž nastane změna aktuálního stavu periférie FlexIO. Jako zdroj spouštěcího musí být nastaven externí signál přivedený prostřednictvím kanálu 0 periférie TRGMUX. Konfigurační hodnoty se následně zapíši do registrů TIMCGF, TIMCTL a TMCMP periférie FlexIO.

Tab. 10 Ukázka některých nastavovaných parametrů pro časovač 0 FlexIO

Položka nastavení časovače 0	Hodnota položky
Volba spouštěcího signálu	Výstup 0 periférie TRGMUX
Polarita spouštěcího signálu	Logická 1
Zdroj spouštěcího signálu	Externí zdroj
Mód	16 bitový čítač
Dekrementace časovače	Přivedením spouštěcího signálu
Vypnutí časovače	Po komparaci
Zapnutí časovače	Stále zapnutý
Komparační hodnota	0

Start FlexIO modulu

Po provedení inicializace a nastavení jednotlivých posunovačů a časovačů lze již spustit periférii k tomu byla vytvořena funkci FlexIO_Start, jejich prototyp je:

```
void FlexIO_Start(void)
```

V této funkci se provede pouze inicializační nastavení aktuálního stavu na hodnotu 0 a následně se prostřednictvím bitu FLEXEN nacházející se v registru CTRL periférie FlexIO zapne. Součástí této funkce je také povolení ladění periférie.

15. Testování algoritmu, porovnání efektivity softwarové a FlexIO implementace, diskuze technických omezení FlexIO. Zhodnocení výsledků práce

Tato kapitola se bude věnovat k testování navrhnutého stavového automatu implementovaného prostřednictvím periférie FlexIO, rovněž bude obsahovat i měření a porovnávání SW implementace a implantace s použitím FlexIO.

15.1. Testování algoritmu

Pro schopnost testování bylo nutné nejprve sepsání testovací aplikace na počítači, která umožní zasílání dat na vývojovou platformu FRDM-KL28Z.

Testování bude probíhat, tak že se bude testovat různá velikost zpráv, různý počet znaků ESC, obsažených ve zprávě a také i zda je automat rozpoznat chybovou zprávu.

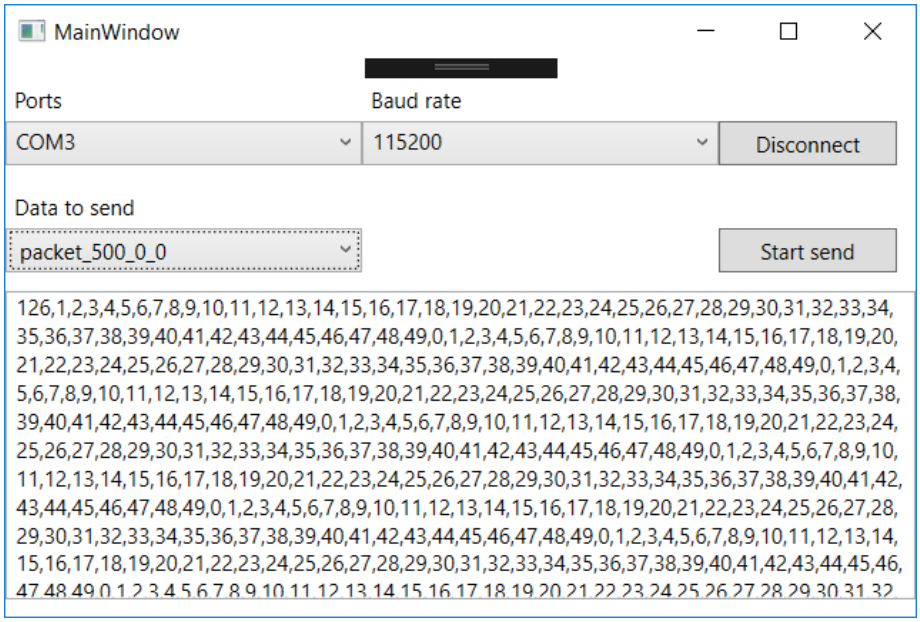
Z tabulky je tedy patrné, že v rámci testování bude vytvořeno 18 různých paketů. Vysílací rychlost bude nastavená na 115200 Baud. Maximální hodnota paketu 1500 je daná z definice protokolu PPP (Point to Point Protocol).

Tab. 11 Testovací parametry

Testování	Parametry
Velikost zprávy	500/1000/1500
Znaky ESC ve zprávě	0/10/20
Chybové zprávy	0/1

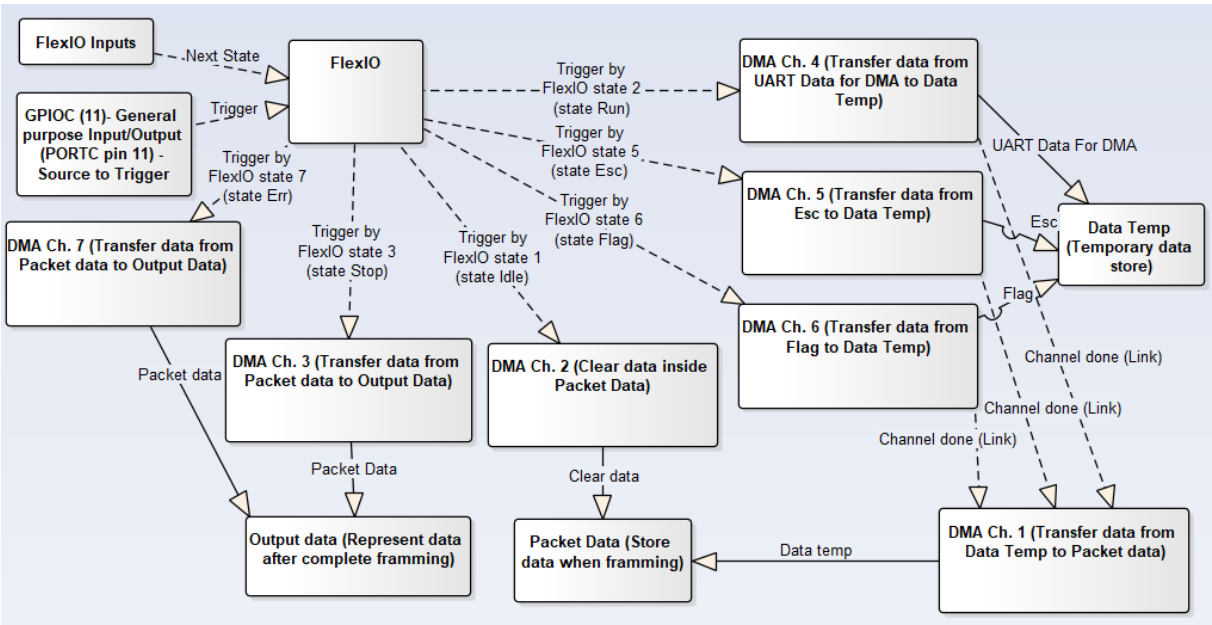
Testování bude probíhat tak, že se v testovací aplikaci vybere z předem definovaného obsahu zpráv, zprávu, kterou následně začneme posílat. Na straně příjemce se po přijetí začátku zprávy se vyčte hodnota z registru časovače SysTick a následně po obdržení ukončujícího znaku vyčte se opět hodnota z registru časovače SysTick. Poté se provede odečet těchto dvou hodnot a vypočte se výsledný čas na základě známé frekvence, která byla 72 MHz a maximální velikosti tohoto časovače, který je 24 bitový.

Souběžně s vyčítáním se bude i provádět měření prostřednictvím osciloskopu a to, tak že po přijetí začátku zprávy se nastaví hodnota na pinu a po přijetí se tato hodnota zneguje, což určí potřebný čas k zpracování zprávy. Toto bylo nutné využít z důvodu změření HW řešení, které bylo realizováno prostřednictvím periférie FlexIO a to protože periférie DMA nemá přímý přístup k registrům časovače SysTick protože se jedná o systémový časovač.

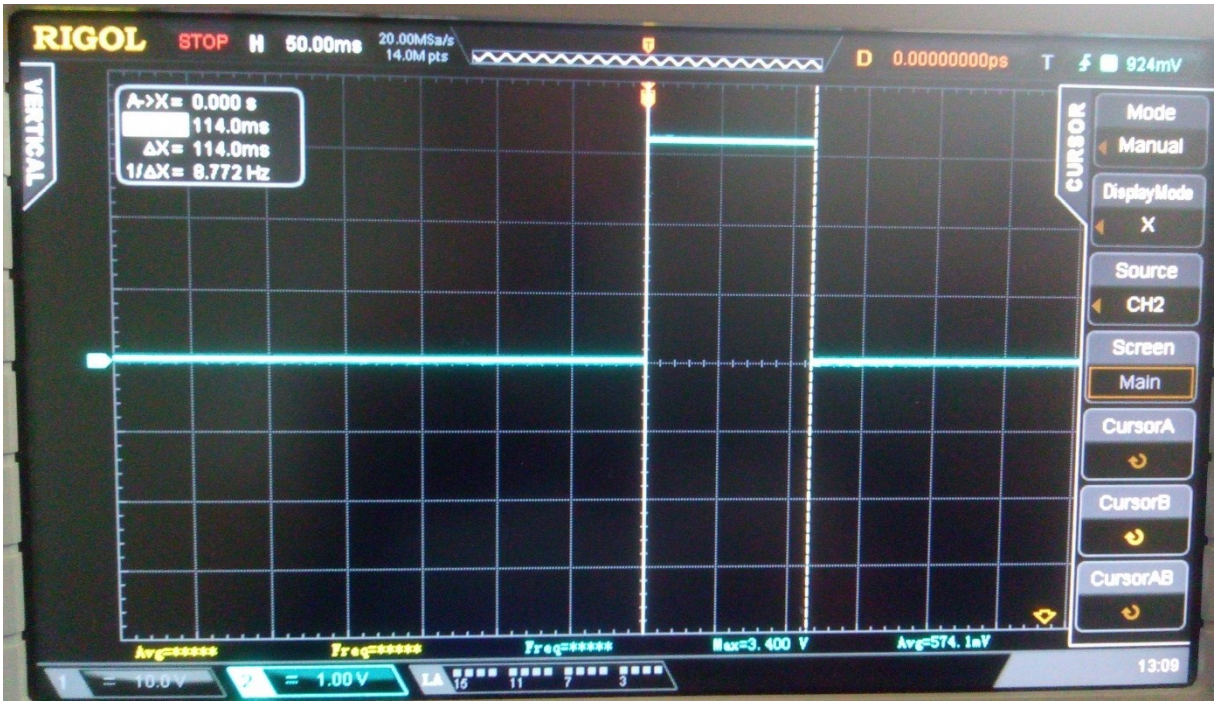


Obr. 36 Ukázka testovacího programu

V rámci testování bylo upraveno jak SW tak i HW řešení, aby bylo možné správně vyhodnotit výsledky testování. SW řešení bylo upraveno tak, že byl redukován jeden stav a tudíž, aby navrhnutý automat odpovídal návrhu HW a zároveň bylo přidáno nastavování pinu pro měření do obsluhy přerušení a HW automat byl upraven tak, že prostřednictvím DMA kanálů 1, který odpovídal stavu IDLE, neboli stavu 1 periférie FlexIO, bylo zapsána logická 1 pro spuštění měření a následně prostřednictvím obsluhy přerušení pro DMA kanály 3 a 7, které odpovídaly stavům stop a error, se zapíše logická 0 nula a tím se ukončí měření.



Obr. 37 Upravený HW návrh pro testování



Obr. 38 Ukázka naměřeného času (SW - 500_20_0)

Během měření bylo zjištěno, že vysílač realizovaný počítačem někdy ovlivňuje zasílání jednotlivých znaků zprávy a to, protože operační systém Windows, někdy zasílá znaky podle toho, jak moc vytížený je, ale naměřené hodnoty bylo ověřeny několika různými měřeními a tyto hodnoty se neměnily oproti chybným časům, které se značně lišily i při stejném odeslaném paketu.

15.2. Porovnání efektivnosti algoritmů

V tabulce níže se nachází naměřené hodnoty pro SW a HW řešení pro jednotlivé velikosti zpráv. Z tabulky lze vidět, že průměrný čas pro přenos pro SW řešení bylo 136,389 ms, kdežto pro HW řešení se jednalo 80,334 ms při průměrné velikosti zprávy 826,944 znaků, tedy v průměru bylo určeno, že HW řešení je o 57,72 ms (neboli o 45%) rychlejší nežli SW řešení a v průměru na znak je HW návrh o 0,067786362 ms rychlejší.

Tab. 12 Naměřené hodnoty pro SW a HW řešení

		SysTick	SW	HW	Rozdíl času	Rozdíl procent
packet	znaků	72000000	čas (OSC) [ms]	čas (OSC) [ms]	(HW – SW) [ms]	(HW – SW) []
500_0_0	498	0,1107985	110,8	49	-61,8	-56%
500_0_1	199	0,0738428	73,6	19	-54,6	-74%
500_10_0	488	0,1194238	119	48	-71	-60%
500_10_1	365	0,0848774	85,2	36	-49,2	-58%
500_20_0	478	0,1139876	114	48	-66	-58%
500_20_1	394	0,1044344	105,2	40	-65,2	-62%
1000_0_0	998	0,1449414	146	95	-51	-35%
1000_0_1	399	0,1057911	106	39	-67	-63%
1000_10_0	988	0,1433948	144	95	-49	-34%
1000_10_1	413	0,1116864	111,6	42	-69,6	-62%
1000_20_0	978	0,1478681	148	95	-83	-36%
1000_20_1	809	0,1292831	129,2	79	-50,2	-39%
1500_0_0	1498	0,1755468	176	143	-33	-19%
1500_0_1	649	0,1123678	112,4	62	-50,4	-45%
1500_10_0	1488	0,1907971	191	144	-47	-25%
1500_10_1	1311	0,1932991	193	128	-65	-34%
1500_20_0	1478	0,1800587	182	143	-39	-21%
1500_20_1	1454	0,2076995	208	141	-67	-32%
Průměr	826,94444	0,1361166	136,3888889	80,33333333	-57,72222222	-45%
Průměrně na znak		0,0001646	0,164931139	0,097144777	-0,067786362	-41%

15.3. Technické omezení FlexIO

Technické omezení již byly popsány v kapitole 12, tedy v kapitole Posouzení možností implementace FSM s použitím FlexIO, kde bylo zjištěno, že periférie FlexIO má omezení v počtu stavů, které může nabývat, toto omezení je docela podstatné, neboť vedlo k nutnosti změny již předem navrženého SW automatu. Dalším zjištěným omezením bylo to, že změna stavů je podmíněná na dopočtení časovače přiřazeného k danému stavu, toto omezení bylo vyřešeno prostřednictvím periférie TRGMUX. A posledním zjištěným problémem bylo, že volba příštího stavu je hodnotou stavu, do kterého chceme přejít, toto bylo vyřešeno prostřednictvím dvou úrovňové náhledové tabulky.

15.4. Zhodnocení výsledků práce

Z naměřených hodnot lze vidět, že navržený HW stavový automat prostřednictvím periférie FlexIO je efektivnější v průměru o 45%, to je dáno primárně tím, že na rozdíl od SW řešení, nezapojuje CPU (Central Process Unit), ale používá pouze obvodové periférie DMA (Direct Memory Access) a FlexIO, obsažené na vývojové platformě FRDM-KL28Z. Když vezmeme v potaz i fakt, že při HW návrhu CPU nevykonával prakticky žádný úkol, tak by celková efektivnost mohla ještě vzrůst, a to primárně z důvodu toho, že by CPU mohlo zpracovávat i jiné úlohy, například výpočty z obdržených dat. Ale při zhodnocení musíme brát v potaz kromě samotného času tak i to, že při HW řešení se využívá podstatně větší množství paměti nežli u SW řešení, toto je dáno důvodem nutnosti kódování přijatých znaků, prostřednictvím náhledových tabulek (look-up table). Dalším faktem, který musí být brán v potaz je to, že samotný HW návrh s použitím periférie FlexIO je několikanásobně složitější nežli při použití jednoduššího SW návrhu.

16. Závěr

V této práci jsem řešil využití periférie FlexIO v režimu stavového automatu, pro realizaci konečného automatu s cílem vyvinout algoritmus využívající HW prostředky MCU k realizaci úlohy, která obvykle bývá implementovaná softwarově. Pro zhotovení této práce jsem nejprve potřeboval zjistit informace o vývojových nástrojích, které jsem pro realizaci práce použil, mezi tyto nástroje patřily vývojové kity, přesněji mnou použitý FRDM-KL28Z, tento vývojový kit byl zvolen z důvodu, že obsahuje možnost použití periférii FlexIO v režimu stavového automatu. Dalším nástrojem, který jsem pro realizaci využil bylo vývojové prostředí na bázi prostředí Eclipse IDE, přesněji vývojové prostředí Atollic TrueSTUDIO for ARM 8.1.0, ve kterém jsem vytvářel algoritmy jednotlivých stavových automatů. Dalším použitým nástrojem, který jsem pro realizaci potřeboval byl kompilátor GCC, který je potřebný pro překlad napsaného kódu. Posledním z řady nástrojů, které jsem potřeboval pro realizaci práce byl debugger neboli ladící prostředí Ozone V2.54a, to bylo nutné pro možnost odladění obou návrhů konečných automatů. Jakmile jsem zjistil potřebné informace k mnou potřebných vývojových nástrojů pro realizaci stavových automatů, jsem následně potřeboval zjistit vlastnosti samotné periférie FlexIO, to bylo nutné pro možnost posouzení možného návrhu a implementace stavového automatu. Pro realizaci konečného stavového automatu jsem zvolil standardizované řešení pro rozpoznávání obsahu zpráv, které je využíváno mj. komunikačním protokolem PPP (Point to Point Protocol), tento protokol jsem zvolil z důvodu toho, že se nejčastěji využívá při komunikacích po sériových linkách a bývá implementován stavovým automatem. Poté jsem již mohl přejít k samotným návrhům a realizacím obou řešení konečného stavového automatu. Při návrhu HW konečného stavového automatu jsem musel vycházet ze zjištěných informací o periférii FlexIO a její omezení, tedy jsem musel vzít v potaz to, že periférie FlexIO disponuje možností implementace stavového automatu maximálně s 8 různými stavy, dále jsem musel přihlédnout k tomu, že výběr příštího stavu u periférie FlexIO je realizován kombinací trojice vstupních hodnot a tedy jsem nemohl pracovat přímo s přijatým znakem, ale musel jsem přijatý znak zakódovat prostřednictvím náhledových tabulek. Dalším Mezi další omezení, které má periférie FlexIO a kterou jsem musel přihlédnout při návrhu konečného automatu je to, že přechod do dalšího stavu je závislý na dopočtení časovače přiřazeného pro daný stav periférie FlexIO, toto bylo vyřešeno prostřednictvím periférie TRGMUX, kterou jsem použil k přivedení spouštěcí signálu z pinu na časovač a tím způsobil jeho dopočtení a přepnutí do jiného stavu. Po úspěšném návrhu a realizaci obou řešení konečného stavového automatu jsem mohl přejít k měření času pro zpracování různých typů paketů a vyhodnocení efektivnosti jednotlivých řešení stavových automatů. Při měření jsem zjistil, že návrh využívající periférii FlexIO je o 19 až 74% rychlejší nežli SW návrh v průměru byl HW návrh o 45% rychlejší. Tento výsledek je dán tím, že HW návrh využívá pouze obvodové periférie, oproti SW návrhu, který využívá ke své činnosti CPU. Při přihlédnutí k tomuto faktu je docela možné, že celková efektivnost HW návrhu může být ještě větší, protože lze následně využít i CPU pro zpracování přijatých dat. Nevýhodou HW návrhu je to, že oproti SW návrhu, využívá větší množství paměti, protože ke své správné činnosti potřebuje náhledové tabulky, ve kterých se nachází zakódované hodnoty stavů. Implementace stavového automatu prostřednictvím periférie FlexIO lze využít všude tam, kde lze přenést zpracování stavového automatu z CPU a nechat tak CPU vykonávat složitější matematické operace s využitím podpory stavového automatu, například lze jej využít k analýze vložených synchronizačních značek anebo rozlišování lichých a sudých rámců v případě prokládaného video signálu.

Doporučená literatura

[1] *FREEDOM DEVELOPMENT BOARDS. NXP [online]. The Netherlands: NXP, 2017 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/support/developer-resources/hardware-development-tools/freedom-development-boards:FREDEVPLA#products*

[2] *KINETIS® LOW-POWER 32-BIT MICROCONTROLLERS: MCUs Based on Arm® Cortex®-M Cores. NXP[online]. The Netherlands: NXP, 2017 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus:KINETIS#products*

[3] *FRDM-KL28Z: Freedom Development Platform for Kinetis KL28MCUs. NXP [online]. The Netherlands: NXP, 2017 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/l-seriesultra-low-powerm0-plus/freedom-development-platform-for-kinetis-kl28mcus:FRDM-KL28Z*

[4] *FRDM-KL28Z User's Guide. In: NXP [online]. UK: NXP, 2016 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/docs/en/user-guide/FRDMKL28ZUG.pdf*

[5] *GNU Arm Embedded Toolchain: Pre-built GNU toolchain for Arm Cortex-M and Cortex-R processors. ARM Developer [online]. United Kingdom: ARM Developer, 2017 [cit. 2017-12-19]. Dostupné z: https://developer.arm.com/open-source/gnu-toolchain/gnu-rm*

[6] *ARM GCC: The ARM compiler for embedded developers. Atollic [online]. United Kingdom: atollic, 2017 [cit. 2017-12-19]. Dostupné z: http://blog.atollic.com/arm-gcc-the-arm-compiler-for-embedded-developers*

[7] *8 debugging techniques every ARM developer should use. Atollic [online]. United Kingdom: atollic, 2017 [cit. 2017-12-19]. Dostupné z: http://blog.atollic.com/8-debugging-techniques-every-arm-developer-should-use*

[8] *Debugging Features of Ozone — The J-Link Debugger. Segger [online]. Germany: segger, 2017 [cit. 2017-12-19]. Dostupné z: https://www.segger.com/products/development-tools/ozone-j-link-debugger/technology/debugging-features/*

[9] *General Information about J-Link. Segger [online]. Germany: segger, 2017 [cit. 2017-12-19]. Dostupné z: https://www.segger.com/products/debug-probes/j-link/technology/general-information-about-j-link/*

[10] *SWD / JTAG Connectors and Pinout. Code Red [online]. Germany: Code Red, 2013 [cit. 2017-12-19]. Dostupné z: http://www.support.code-red-tech.com/CodeRedWiki/HardwareDebugConnections*

[11] *Eclipse (vývojové prostředí). In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-12-19]. Dostupné z: https://cs.wikipedia.org/wiki/Eclipse_(v%C3%BDvojov%C3%A9_prost%C5%99ed%C3%AD)*

[12] *GCC. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-12-19]. Dostupné z: https://cs.wikipedia.org/wiki/GCC*

[13] *PPP in HDLC-like Framing. IETF Tools [online]. IETF Tools, 1994 [cit. 2017-12-19]. Dostupné z: https://tools.ietf.org/html/rfc1662*

[14] *Point-to-Point Protocol. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-12-19]. Dostupné z: https://en.wikipedia.org/wiki/Point-to-Point_Protocol*

[15] Framing in serial communications. Eli Bendersky's website [online]. Eli Bendersky's website, 2009 [cit. 2017-12-19]. Dostupné z: https://eli.thegreenplace.net/2009/08/12/framing-in-serial-communications/

[16] Digital Data Communications Message Protocol. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-12-19]. Dostupné z: https://en.wikipedia.org/wiki/Digital_Data_Communications_Message_Protocol

[17] Binary Synchronous Communications. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-12-19]. Dostupné z: https://en.wikipedia.org/wiki/Binary_Synchronous_Communications

[18] Kinetis® L Series: Ultra-Low Power Microcontrollers (MCUs) based on Arm® Cortex®-M0+ Core. NXP[online]. UK: NXP, 2009 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/l-seriesultra-low-powerm0-plus:KINETIS_L_SERIES

[19] KL8x: Kinetis® KL8x-72/96 MHz Secure Ultra-Low Power Microcontrollers (MCUs) based on Arm® Cortex®-M0+ Core. NXP [online]. UK: NXP, 2009 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/l-seriesultra-low-powerm0-plus/kinetis-kl8x-72-96-mhz-secure-ultra-low-power-microcontrollers-mcus-based-on-arm-cortex-m0-plus-core:KL8x

[20] KL4x: Kinetis® KL4x-48 MHz, USB, Segment LCD, Ultra-Low-Power Microcontrollers (MCUs) based on Arm® Cortex®-M0+ Core. NXP [online]. UK: NXP, 2009 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/l-seriesultra-low-powerm0-plus/kinetis-kl4x-48-mhz-usb-segment-lcd-ultra-low-power-microcontrollers-mcus-based-on-arm-cortex-m0-plus-core:KL4x

[21] KL3x: Kinetis® KL3x-48 MHz, Segment LCD Ultra-Low-Power Microcontrollers (MCUs) based on Arm® Cortex®-M0+ Core. NXP [online]. UK: NXP, 2009 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/l-seriesultra-low-powerm0-plus/kinetis-kl3x-48-mhz-segment-lcd-ultra-low-power-microcontrollers-mcus-based-on-arm-cortex-m0-plus-core:KL3x

[22] KL2x: Kinetis® KL2x-48 MHz, USB Ultra-Low-Power Microcontrollers (MCUs) based on Arm® Cortex®-M0+ Core. NXP [online]. UK: NXP, 2009 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/l-seriesultra-low-powerm0-plus/kinetis-kl2x-48-mhz-usb-ultra-low-power-microcontrollers-mcus-based-on-arm-cortex-m0-plus-core:KL2x

[23] KL1x: Kinetis® KL1x-48 MHz, Mainstream Small Ultra-Low Power Microcontrollers (MCUs) based on Arm® Cortex®-M0+ Core. NXP [online]. UK: NXP, 2009 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/l-seriesultra-low-powerm0-plus/kinetis-kl1x-48-mhz-mainstream-small-ultra-low-power-microcontrollers-mcus-based-on-arm-cortex-m0-plus-core:KL1x

[24] KL0x: Kinetis® KL0x-48 MHz, Entry-Level Ultra-Low Power Microcontrollers (MCUs) based on Arm Cortex®-M0+ Core. NXP [online]. UK: NXP, 2009 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/l-seriesultra-low-powerm0-plus/kinetis-kl0x-48-mhz-entry-level-ultra-low-power-microcontrollers-mcus-based-on-arm-cortex-m0-plus-core:KL0x

[25] Překlad programu [online]. [cit. 2017-12-19]. Dostupné z: http://www.fit.vutbr.cz/~martinek/clang/gcc.html

[26] Emulating Hardware State Machine Using FlexIO Module. In: NXP [online]. UK: NXP, 2016 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/docs/en/application-note/AN5239.pdf?&fsrch=1&sr=4&pageNum=1

[27] KL28Z Reference Manual: MKL28Z512VDC7, MKL28Z512VLL7. In: NXP [online]. -: NXP, 2016 [cit. 2017-12-19]. Dostupné z: https://www.nxp.com/docs/en/reference-manual/MKL28ZRM.pdf

[28] Understanding FlexIO. NXP [online]. -: NXP, 2016 [cit. 2017-12-21]. Dostupné z: https://community.nxp.com/docs/DOC-105640

[29] Portace GCC pro procesor ADOP [online]. Praha, 2009 [cit. 2017-12-21]. Dostupné z: https://dip.felk.cvut.cz/browse/pdfcache/machzizka_2009dipl.pdf. Diplomová práce. České vysoké učení technické v Praze. Vedoucí práce Ing. Miloš Bečvář.

[30] Features. Atollic [online]. Sweden: atollic, 2016 [cit. 2017-12-21]. Dostupné z: https://atollic.com/truestudio/features/

[31] Framing Protocols. LinkedIn SlideShare [online]. -: LinkedIn SlideShare, 2015 [cit. 2017-12-21]. Dostupné z: https://www.slideshare.net/selvakumar_b1985/framing-43534946

[32] Digital Data Communications Message Protocol (DDCMP). Decnet [online]. -: -, 1977 [cit. 2017-12-28]. Dostupné z: http://decnet.ipv7.net/docs/dundas/aa-d599a-tc.pdf

[33] Konečný automat typu Moore. VOHO [online]. -: -, 2008 [cit. 2018-04-13]. Dostupné z: http://voho.eu/wiki/moore/

[34] Konečný automat typu Mealy. VOHO [online]. -: -, 2008 [cit. 2018-04-13]. Dostupné z: http://voho.eu/wiki/moore/

[35] JANČAR, Petr. Teoretická informatika [online]. Ostrava: Vysoká škola báňská - Technická univerzita, 2008 [cit. 2018-04-13]. ISBN ISBN978-80-248-1487-2. Dostupné z: http://www.cs.vsb.cz/sawa/uti/materialy/ti.pdf

Přílohy

Obsah přiloženého DVD

- Text diplomové práce ve formě PDF
- SW a HW implementace stavového automatu
- Testovací program