
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY
DEPARTMENT OF MICROELECTRONICS


# FUNKČNÍ BEZPEČNOST MIKROKONTROLÉRŮ

FUNCTIONAL SAFETY FOR MICROCONTROLLERS

SEMESTRÁLNÍ PRÁCE
SEMESTRAL THESIS


AUTOR PRÁCE — Bc. Jan Klecker
AUTHOR

VEDOUCÍ PRÁCE — Ing. Edita Hejátková
SUPERVISOR



BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

# Diplomová práce

magisterský navazující studijní program **Mikroelektronika**

Ústav mikroelektroniky

***Student:*** Bc. Jan Klecker ***ID:*** 203411
***Ročník:*** 2 ***Akademický rok:*** 2021/22

**NÁZEV TÉMATU:**

## Funkční bezpečnost mikrokontrolérů

**POKYNY PRO VYPRACOVÁNÍ:**

Prostudujte požadavky na funkční bezpečnost mikrokontrolérů podle mezinárodních norem IEC60730 a IEC60335 a seznamte se s principy realizace funkční bezpečnosti pro mikrokontroléry. Navrhněte řešení funkční bezpečnosti mikrokontroléru NXP splňující požadavky zmíněných norem. Práce bude zaměřena na návrh a implementaci autodiagnostických mechanismů pro daný mikrokontrolér zahrnující hardwarovou podporu ze strany mikrokontroléru a softwarové mechanismy. Proveďte teoretický rozbor autodiagnostických mechanismů, navrhněte vhodné algoritmy, implementujte navržené algoritmy a demonstrujte funkčnost řešení.

***Termín zadání:*** 7.2.2022 ***Termín odevzdání:*** 24.5.2022

***Vedoucí práce:*** Ing. Edita Hejátková
***Konzultant:*** Ing. Jaroslav Lepka, Ph.D.

**doc. Ing. Lukáš Fujcik, Ph.D.**
předseda rady studijního programu

# Abstrakt

Diplomová práce se zabývá principy a mechanismy funkční bezpečnosti pro mikrokontroléry. Jsou popsány normy IEC 60730 a IEC 60335, ze kterých plynou požadavky na testování dílčích funkčních bloků mikrokontroléru. Dále jsou rozebrány požadavky na samotný návrh bezpečného softwaru. Stěžejní kapitolou teoretické části je návrh a diskuse testů funkční bezpečnosti pro dané periferie. V praktické části je nejprve navržen demonstrační hardware plynového kotle s bezpečnostními prvky a diagnostickými mechanismy řízený dvěma mikrokontroléry NXP KV46, které vzájemně komunikují přes rozhraní UART. V kapitole zabývající se softwarem je popsán návrh aplikačního softwaru pro plynový kotel a následně jsou pro mikrokontroléry implementovány testy funkční bezpečnosti. Pro testovací a demonstrační účely je navrženo grafické uživatelské rozhraní v programu FreeMASTER.

# Klíčová slova

Funkční bezpečnost, mikrokontrolér, normy, IEC 60730, IEC 60335, EN 13611, Třída C, plynový kotel, Arm Cortex-M, samočinné testování, NXP KV46, Kinetis, UART, FreeMASTER

# Abstract

This Master’s thesis deals with the principles and mechanisms of functional safety for microcontrollers. The IEC 60730 and IEC 60335 standards are described, from which the requirements for testing of microcontroller functional blocks are derived. Furthermore, the requirements for designing safety software are discussed. The most important chapter of the theoretical part is the design and discussion of functional safety tests for the given peripherals. In the practical part, a demo hardware of a gas boiler with safety features and diagnostic mechanisms controlled by two NXP KV46 microcontrollers communicating via a UART interface is designed. In the software chapter, the design of the application software for the gas boiler is described and then functional safety tests are implemented for the microcontrollers. For testing and demonstration purposes, a graphical user interface is designed in the FreeMASTER tool.

# Keywords

Functional safety, microcontroller, standards, IEC 60730, IEC 60335, EN 13611, Class C, gas boiler, Arm Cortex-M, Self-testing, NXP KV46, Kinetis, UART, FreeMASTER

# Bibliografická citace

KLECKER, Jan. *Funkční bezpečnost mikrokontrolérů.* Brno, 2022. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/142445. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav mikroelektroniky. Vedoucí práce Edita Hejátková.

# Prohlášení autora o původnosti díla

| | |
|---|---|
| **Jméno a příjmení studenta:** | *Jan Klecker* |
| **VUT ID studenta:** | *203411* |
| **Typ práce:** | *Diplomová práce* |
| **Akademický rok:** | *2021/22* |
| **Téma závěrečné práce:** | *Funkční bezpečnost mikrokontrolérů* |

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

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dne: 24. května 2022 ----------

podpis autora

# Poděkování

Velice děkuji konzultantovi mé diplomové práce Ing. Jaroslavu Lepkovi, Ph.D. za dlouhodobou účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při laborování technologických postupů. Dále děkuji mým kolegům z firmy NXP za vstřícnost a za cenné technické rady. Děkuji také mé vedoucí diplomové práce Ing. Editě Hejátkové.


V Brně dne: 24. května 2022 ----------

podpis autora

# Obsah

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# Úvod

V současné době se mikrokontroléry využívají ve všech odvětvích průmyslu. Řídí a ovládají složité systémy v automobilech, letadlech, elektrárnách či v domácích spotřebičích. Struktury mikrokontrolérů a pamětí se neustále zmenšují jako důsledek rostoucí komplexnosti čipů. Zmenšené struktury činí mikrokontroléry náchylnější na přeslechy a na vnější vlivy. Mikrokontrolér je pomyslný „mozek“ celého zařízení, je tedy zřejmé, že jeho selhání může mít fatální důsledky jak z hlediska ohrožení zdraví, tak z hlediska škod na majetku. Normy byly vytvořeny proto, aby byla minimalizována rizika a bylo jim předcházeno.

Tato práce je zaměřena na funkční bezpečnost pro domácí spotřebiče. Na první pohled může vyvstat otázka, zdali je vůbec nutné věnovat úsilí například kontrole periferií mikrokontroléru, který řídí pračku. Špatné fungování určité periferie mikrokontroléru může u pračky způsobit v lepším případě poškození prádla, v horším případě únik vody mimo pračku a tím způsobit větší škody na majetku. Tím nejhorším scénářem je ohrožení lidského zdraví a života, například když dojde k nekontrolovanému zvýšení otáček bubnu, jehož následkem může dojít k mechanickému poškození bubnu a následné destrukci konstrukce pračky.

Teoretická část si klade za cíl přiblížit čtenáři problematiku funkční bezpečnosti, ačkoliv jde o velice komplexní obor. V této části je obecně rozebrána problematika funkční bezpečnosti a dále jsou rozebrány konkrétně normy IEC 60730 a IEC 60335. Pro lepší představu budou ukázány možné příčiny poruch v mikrokontroléru, které lze rozdělit na poruchy hardwaru (náhodné poruchy) a chyby softwaru (systematické chyby). Stěžejní kapitola teoretické části se zabývá samotnými funkčními testy mikrokontroléru, které jsou vyžadovány výše zmíněnými normami. Detailní informace o metodice a samotné implementaci testů odpovídající výše uvedeným normám zahrnující systémové řešení, jehož součástí je mikrokontrolér a softwarové vybavení, byly převážně získány ze zkušeností odborníků na danou problematiku, reálných průmyslových řešení a interakcí s certifikačními autoritami VDE a UL.

Mikrokontrolér jako součást systémového řešení vyžaduje důkladnou analýzu z hlediska implementace autodiagnostických mechanismů v závislosti na požadavcích konkrétní aplikace. Je vyžadována také důkladná analýza odpovídající normy definující požadavky na funkční bezpečnost. Pro demonstrační systémového řešení s prvky funkční bezpečnosti byl zvolen systém pro řízení plynového kotle na bázi mikrokontroléru NXP KV46. Jsou navrženy bezpečnostní mechanismy, které jsou aktivovány v případě vzniku poruchy systému a jejichž výsledkem je uvedení systému do bezpečného stavu, v tomto konkrétním případě jde o zastavení přívodu plynu. Navržená architektura bezpečnostního systému využívá obecně doporučované techniky založené na principu redundance a autodiagnostiky v reálném čase z důvodů minimalizace možného selhání a ztráty bezpečnostních funkcí systému.

V kapitole zabývající se softwarem je nejprve navržen aplikační software, který řídí logiku samotného plynového kotle a dále řídí komunikaci mezi oběma mikrokontroléry. V této kapitole jsou též navrženy další mechanismy, které přispívají celkové bezpečnosti tohoto systému. V další části jsou navrženy dílčí testy funkční bezpečnosti vyžadované výše uvedenými normami. Některé z nich jsou implementací testů rozebraných v teoretickém úvodu. V opodstatněných případech byly využity upravené knihovní funkce (testy jádra mikrokontroléru).

V poslední kapitole je navrženo grafické uživatelské rozhraní pomocí skriptovacího jazyka JavaScript. Toto rozhraní je součástí programu FreeMASTER. Softwarový nástroj FreeMASTER je určený k ladění programu běžícího v reálném čase a zároveň podporuje tvorbu grafického uživatelského rozhraní. Navržené grafické rozhraní zobrazuje měřené analogové signály, stav systému a detekované chyby. Pomocí tlačítek je možno do systému uměle injektovat chyby. Nakonec jsou provedena měření pro specifikované podmínky.

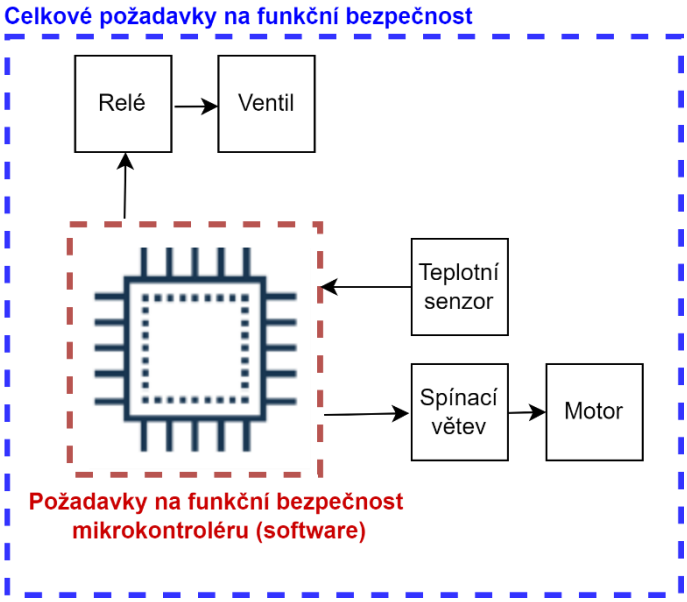
# 1. FUNKČNÍ BEZPEČNOST MIKROKONTROLERŮ

Funkční bezpečnost se zaměřuje na schopnost produktu správně reagovat na příkazy které obdrží a fungovat vždy bezpečně, spolehlivě a konzistentně. Funkční bezpečnost tedy spočívá v tom, že elektronické části systému fungují správně a nezpůsobují škodu nesprávným fungováním. Funkční bezpečnost zahrnuje bezpečnost řízeného zařízení (EUC). Úroveň funkční bezpečnosti závisí na opatřeních zavedených s cílem zmenšit riziko, a tudíž záleží i na správné činnosti těchto opatření. Nesprávné fungování opatření funkční bezpečnosti může vést k újmě na zdraví, škodě na majetku nebo poškození životního prostředí. Rozdíl mezi bezpečností a funkční bezpečností lze ukázat na následujícím příkladu. Teplotní senzor, který vynutí vypnutí systému v případě přehřátí, je opatření funkční bezpečnosti. Izolace k ochraně systému proti vysokým teplotám je bezpečností opatření, nikoliv však opatření funkční bezpečnosti. Obě opatření ale mohou zamezit požáru.

Při vývoji nového produktu je třeba zjistit, zdali jsou na systém kladeny požadavky na funkční bezpečnost, je tedy nutné provést analýzu rizik. Je nutné identifikovat všechny možné hrozby z hlediska funkční bezpečnosti již ve stádiu návrhu systému a zajistit, že jejich riziko je minimalizováno dobrým návrhem. V případě, že jim nejde zamezit, tak preventivními a nápravnými mechanismy zamezit škodám, nebo alespoň zmírnit jejich účinek.

V současné době většina automatických elektronických ovládacích prvků pro domácí spotřebiče, automobily, elektrárny, vlaky, letadla nebo průmyslové výrobky využívá k řízení mikrokontrolér. Pro řízení mikrokontrolérů je vyvíjen tzv. real-time software, výraz real-time v tomto kontextu znamená, že pro korektní fungování systému jsou zde požadavky na přesné časování [19]. Je proto zřejmé, že selhání mikrokontroléru může mít fatální následky. Například kritické zvýšení teploty nebo výboj statické elektřiny mohou mikrokontrolér, a tím celý produkt, dostat do nedefinovaného a potenciálně nebezpečného stavu [10]. Jelikož jsou často systémy velice komplexní, je prakticky nereálné odhalit každé možné selhání funkcionality a vyzkoušet všechna možná chování zařízení. Z tohoto důvodu jsou v řídícím systému implementovány bezpečnostní funkce, jejichž úkolem je zajistit nezměněné chování v případě poruchy.

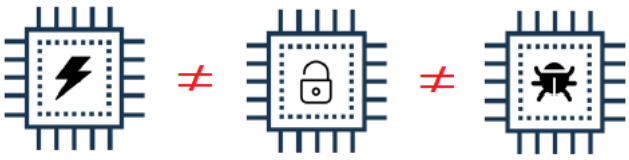
Funkční bezpečnost je vyžadována regulačními orgány, bez certifikace není možné uvést zařízení na evropský ani na americký trh. V rámci certifikace produktu probíhá důkladná analýza bezpečnostních rizik, navržených opatření a chování systému v případě poruchy. Systémové řešení se pak dělí na hardwarovou část (chování aktuátorů, kritických zpětných vazeb od senzorů atd.) a softwarové řešení, které zahrnuje stranu mikrokontroléru.




Obrázek 1: Funkční bezpečnost mikrokontroléru jako součást celkové funkční bezpečnosti

Jako příklad lze uvést domácí spotřebič z oblasti bíle techniky, automatickou pračku, která se řídí normou IEC 60730 a IEC 60335-2-7. Pro tento typ domácího spotřebiče je vyžadováno splnění požadavků normy pro kategorii třídy B, což zahrnuje hardwarové a softwarové řešení ve vzájemné spolupráci. Psaní samotné softwarové implementace se pak řídí specifickými požadavky normy pro mikrokontroléry a obecnými pravidly, která napomáhají vytvářet softwarové řešení funkční bezpečnosti, které eliminuje systematické chyby.

Funkční bezpečnost bývá často zaměňována za zabezpečení, jedná se však o dvě odlišné oblasti. Laicky řečeno, funkční bezpečnost chrání zařízení (uživatele) před ním samotným, zabezpečení chrání zařízení (uživatele) před okolím.



Obrázek 2: Funkční bezpečnost vs. zabezpečení vs. funkcionalita

- **Funkční bezpečnost (anglicky functional safety):** Funkční bezpečnost je část celkové bezpečnosti týkající se řízeného zařízení (anglicky Equipment Under Control), které závisí na správném fungování elektrických / elektronických / programovatelných elektronických bezpečnostních systémů, dalších technologických bezpečnostních systémů a externích zařízení pro snížení rizik.
- **Zabezpečení (anglicky security):** Zabezpečení v oblasti mikrokontrolerů zahrnuje více oblastí. Může jít například o zabezpečení intelektuálního vlastnictví

softwaru nebo o zabezpečení soukromých dat proti vnějším útokům (zařízení připojená k sítím).

- **Funkcionalita (anglicky functionality):** Zajišťuje požadované chování sytému, je definována výrobcem.

## 1.1 Certifikace softwaru

Při vývoji výrobku řízeného mikrokontrolérem je nutné splnit požadavky na funkční bezpečnost mikrokontroléru dle dané normy. IEC 61508 – „*Funkční bezpečnost elektrických/elektronických/programovatelných elektronických systémů*“ je považována za základní bezpečnostní normu pro průmyslové aplikace: ISO 26262 - automobilový průmysl, IEC 60730 / IEC 60335 - domácí spotřebiče. Je třeba podotknout, že norma samotná nedává přesně návod, jak daný test implementovat, ale definuje požadavky na diagnostiku a poskytuje doporučení, která vedou ke splnění jednotlivých požadavků normy. Například udává, že pro testování paměti RAM lze použít některý z March testů, neuvádí už ale přesný algoritmus či jeho přesnou implementaci. Lze použít i jiný způsob testování, který není uveden v tabulce. V takovém případě ale musí být spolehlivě prokázáno, že test korektně detekuje poruchu. Každý dílčí samočinný test pak musí být otestován, zdali skutečně odhalí danou poruchu. Všechny tyto kroky musí být detailně dokumentovány. Funkční bezpečnost mikrokontrolérů je poměrně rozsáhlá oblast, bez níž nelze uvést téměř žádný výrobek na trh. Výrobci mikrokontroléru proto často dodávají již hotové a certifikované knihovny, což zákazníkovi, který vyvíjí výsledný produkt s mikrokontrolérem, ušetří čas i peníze.

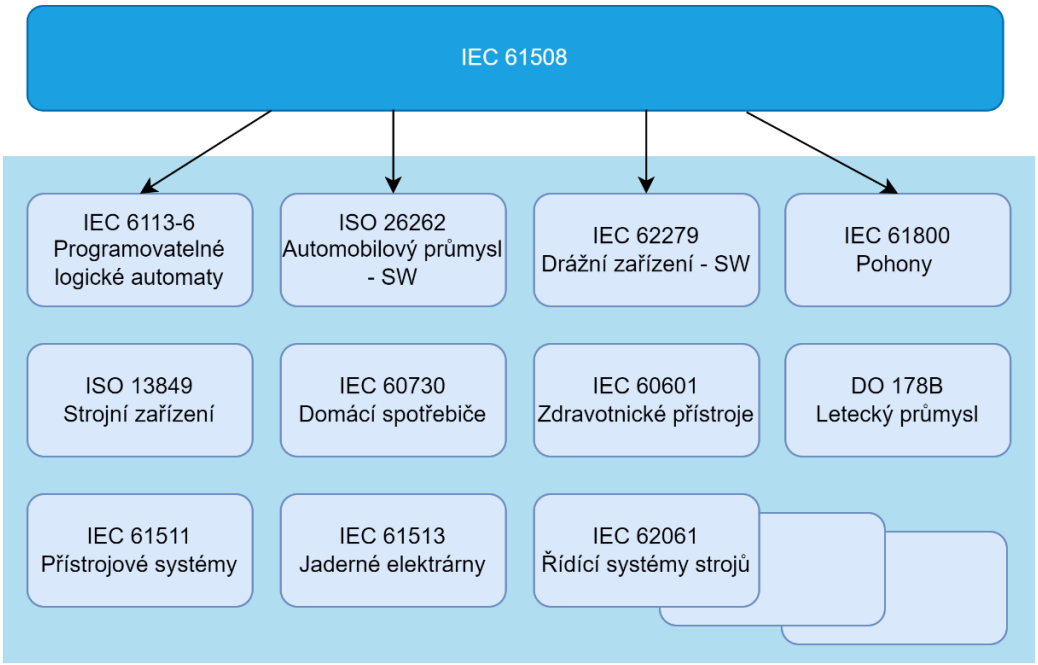
Certifikace produktů s ohledem na funkční bezpečnost probíhá u certifikačních autorit. Za nejuznávanější autority jsou považovány německý TÜV a VDE nebo americká certifikační autorita UL. Certifikační proces vyžaduje splnění celé řady požadavků, které lze rozdělit na požadavky procesní a implementační. Procesní požadavky vyžadují detailní dokumentaci celého vývoje systému s funkční bezpečností, kde základní požadavek je splnění V-modelu vývoje softwaru. Nedílnou součástí certifikačního procesu je dokazování korektního chování systému s funkční bezpečností, kdy certifikační autorita provádí důkladný rozbor celého systému a následně pak definuje množinu testů, které jsou prováděny a dokumentovány. Za tímto účelem je zapotřebí navrhnout systém, který umožňuje injektovat konkrétní chyby, a to ať chyby hardwaru nebo chyby na úrovni softwarové implementace a integrace do řídicího systému dané aplikace. Následně se pak monitorují detekční schopnosti systému s funkční bezpečností, výsledný stav systému, což je bezpečný stav a reakční doba. Certifikační autoritě se pak předkládá velmi detailní dokumentace, software opatřený elektronickým podpisem, který identifkuje verzi softwaru, procesní dokumenty organizace, FMEA analýza, případně další materiály v závislosti na dané řešení. V případě certifikace komponenty systému funkční bezpečnosti, což bývají knihovny algoritmů pro daný mikrokontrolér, certifikační

autorita klade důraz na správnou integraci jednotlivých testů a kontroluje jejich funkčnost na úrovni celého systému za pomocí injektování chyb.

## 1.2 Normy

Základní normou definující požadavky na funkční bezpečnost je norma IEC 61508 (*Funkční bezpečnost elektrických/elektronických/programovatelných elektronických systémů*). Celá řada norem specifikujících požadavky na funkční bezpečnost pro různá odvětví průmyslu jsou buď přímými deriváty této normy nebo jsou inspirované touto normou. Norma je založena na předpokladu, že systém musí fungovat správně a v případě selhání musí být jeho chování stále bezpečné. Obsahuje 4 normativní části a 3 informativní části. Vymezuje též míru četnosti výskytu nebezpečných poruch pomocí stupně integrity bezpečnosti systému SIL (anglicky Safety Integrity Levels), SIL1 až SIL4, kde SIL1 značí nejnižší stupeň integrity.

Selhání systémů může být způsobeno buď systematickými chybami (chyby v návrhu) nebo náhodnými chybami (selhání komponent). V průběhu návrhu by tedy mělo být postupováno tak, že výskytu systematických chyb se zamezí pomocí správných metod a postupů. Chybám náhodným se naopak zamezí správnou konfigurací systému. To vše by mělo vést k systému, který je ze své podstaty bezpečný nebo jehož funkce kritické z hlediska bezpečnosti (řídící systém pro plynový ventil, mikrokontrolér a další prvky) jsou zabezpečeny dle požadavků patřičné normy. Bezpečnostní prvky mohou být implementovány jak hardwarově (časovač watchdog, měření úrovně napětí), tak softwarově (test paměti RAM). Je nutné, aby tyto bezpečnostní prvky zajistily bezpečné vypnutí.




Obrázek 3: Rozdělení norem

### 1.2.1 Norma IEC 60730: Automatická elektrická řídicí zařízení pro domácnost a podobné účely.

Tato norma diskutuje mechanické vlastnosti, elektrické vlastnosti, elektronické vlastnosti, enviromentální aspekty, odolnost, EMC kompatibilitu a abnormální provoz AC spotřebičů. Definuje testovací a diagnostické metody, které zajišťují bezpečné fungování hardwaru a software pro řízení domácích spotřebičů. Nedefinuje však, jak mají být testy implementovány.

Mikrokontroléry se konkrétně řídí dle Přílohy H: Požadavky na elektronické řízení, která podrobně popisuje testovací a diagnostické metody pro zajištění bezpečného provozu vestavěného řídícího hardwaru a softwaru pro domácí spotřebiče. Důvod těchto požadavků je implementace mechanismů, které zamezí selhání nebo minimálně zajistí, že žádné selhání spotřebiče neohrozí bezpečnost uživatele. Jak již bylo zmíněno, implementované bezpečnostní mechanismy musí projít certifikačním procesem, který zajišťuje některá z certifikačních autorit, například TÜV, VDE nebo UL.

Při návrhu bezpečnostních mechanismů je nezbytné určit, jak mohou řídící funkce v případě poruchy ohrozit okolí. Pro tyto účely se spotřebiče dělí do tří tříd [3]:

- **Třída A (anglicky Class A):** Řídicí funkce, které nejsou určeny k tomu, aby se na ně spoléhalo z hlediska bezpečnosti.
    - **Příklad:** Termostaty, časovače, senzory vlhkosti atd.
- **Třída B (anglicky Class B):** Řídicí funkce, které mají zabránit nebezpečnému stavu spotřebiče. Selhání řídicí funkce nevede přímo k nebezpečné situaci.
    - **Příklad:** Mrazáky, pračky, sušičky, lednice atd.
- **Třída C (anglicky Class C):** Řídicí funkce, které mají zabránit speciálním nebezpečím, jako je například výbuch nebo jejíž selhání může přímo způsobit nebezpečnou nebezpečí ve spotřebiči.
    - **Příklad:** Řídicí systémy hořáků, tepelné pojistky pro uzavřené vodní systémy.

Z hlediska softwarového testování je nejdůležitější tabulka H.1 normy IEC 60730, ve které je uveden přehled softwarových testů vyžadovaných pro softwarovou třídu B a C. Jsou v ní rozepsány jednotlivé komponenty mikrokontroléru, které je nutné testovat. Ke každé komponentě je uvedena možná porucha/chyba. Ke každé poruše/chybě jsou uvedeny příklady přijatelných testů. Softwarová třída C je z hlediska funkční bezpečnosti přísnější než třída B, tudíž testy pro třídu C pokrývají třídu B. Lze použít i jiné metody testování než ty uvedené v normě. Musí však být prokázáno, že splňují požadavky kladené v tabulce H.1. Důležitým parametrem je reakční doba na poruchu (anglicky Fault Reaction Time), což je čas mezi výskytem poruchy a přechodem do bezpečného definovaného stavu. V Tabulka 1 jsou uvedeny požadované testy pro třídu B vycházející z již zmíněné tabulky H.1 [3].

Tabulka 1: Testy vyžadované normou IEC 60730 pro třídu B, převzato a upraveno z normy IEC 60730 [3]

| Testovaný blok | Komponenta | Porucha/chyba | Příklad přijatelného opatření |
|---|---|---|---|
| **CPU** | Registry | Stuck-at | Test statické paměti<br>Ochrana pomocí bitové redundance |
| | Programový čítač | Stuck-at | Monitorování časového slotu<br>Logické monitorování sekvence programu |
| **Obsluha a vykonávání přerušení** | - | Žádné přerušení<br>nebo příliš časté přerušení | Monitorování časového slotu |
| **Hodinový signál** | - | Nesprávná frekvence (Pro krystalový oscilátor harmonické/subharmonické frekvence) | Porovnání s frekvencí referenčního zdroje hodinového signálu |
| **Paměť** | Nevariabilní paměť (FLASH) | Všechny bitové poruchy | Kontrolní součet |
| | Variabilní paměť (RAM) | DC chyba | Periodický test statické paměti |
| | Adresování | Stuck-at | Ochrana slova pomocí bitové redundance zahrnující adresu |
| **Interní komunikační kanály** | Data | Stuck-at | Ochrana slova pomocí bitové redundance |
| | Adresování | Špatná adresa | Ochrana slova pomocí bitové redundance zahrnující adresu |
| **Externí komunikační kanály** | Data | Hammingova vzdálenost | Ochrana slova pomocí CRC |
| | Adresování | Špatná adresa | Ochrana slova pomocí CRC |
| | Časování | Špatné časování | Monitorování časového slotu |
| **Vstupní/výstupní periferie digitální** | Digitální vstup/výstup | Chybové podmínky specifikované v příloze H.27 | Test věrohodnosti |
| **Vstupní/výstupní periferie analogové** | AD převodník a DA převodník | Chybové podmínky specifikované v příloze H.27 | Test věrohodnosti |
| | Analogový multiplexor | Špatné adresování | Test věrohodnosti |

### 1.2.2 Norma IEC 60335: Elektrické spotřebiče pro domácnost a podobné účely

Tato norma pokrývá bezpečnost elektronických zařízení pro domácnost. Spolu s výše popsanou normou IEC 60730 tato norma popisuje požadavky na řídící funkce spojené se

softwarem pro automatické řízení spotřebičů pro domácnost a podobné účely. Do této normy taktéž spadají spotřebiče používané například v obchodech či na farmách.

Požadavky na software jsou uvedeny v Příloze R, konkrétně Tabulka R.1 a R.2 [4]. Tyto požadavky jsou převzaty z normy IEC 60730 Příloha H. Norma sestává ze dvou hlavních částí:

- **IEC 60335-1**: Zajišťuje obecné požadavky pro testování, klasifikaci, značení a pokyny pro spotřebiče pro domácnost a podobné účely.
- **IEC 60730-2**: Obsahuje detailní požadavky pro konkrétní spotřebiče. Je zde zahrnuto více než 100 různých typů spotřebičů.
    - IEC 60335-2-7 Pračky
    - IEC 60335-2-5 Myčky
    - IEC 60335-2-34 Kompresory
    - IEC 60335-2-11 Sušičky prádla

## 1.3 Vznik chyb

Obecně lze chyby rozdělit systematické chyby (anglicky systematic faults) a náhodné chyby (anglicky random faults).

- **Defekt (anglicky defect):** Nezamýšlený rozdíl mezi implementovaným hardwarem a jeho zamýšleným návrhem [26].
- **Porucha (anglicky fault):** Stav předmětu charakterizovaný jeho neschopností plnit požadovanou funkci. Porucha je stav, oproti tomu selhání je událost [3]. Jde o abstraktní vyjádření defektu.
- **Chyba (anglicky error):** Chyba je odchylka od požadované funkcionality systému nebo subsystému. Chyba je mechanismus, pomocí něhož se porucha stává viditelná. Chybou je například nesprávně generovaná hodnota signálu [26].
- **Selhání (anglicky failure):** Ukončení schopnosti systému plnit požadovanou funkci [3].

### 1.3.1 Systematické chyby

Systematické chyby jsou deterministicky spojené s určitou příčinou, kterou lze odstranit pouze změnou návrhu, úpravou výrobního procesu, provozních postupů, dokumentace nebo jiných relevantních faktorů. Do této kategorie patří také chyby plynoucí ze softwaru. Zde se řadí chyby způsobené špatnou specifikací, například použití nevhodného algoritmu nebo nesprávně implementovaná synchronizace událostí při použití více přerušení. Mezi další možné chyby plynoucí ze softwaru může být špatné přetypování proměnné. Norma IEC 60730 doporučuje vývoj softwaru pomocí metodického nástroje zvaného V-model. Tento nástroj napomáhá k zamezení systematických chyb. Je možné použít i jiné metody, ale musí obsahovat strukturované procesy zahrnující fáze návrhu a testování. Jednotlivé úrovně požadavků V-modelu jsou hierarchicky rozděleny do tří bloků:

### 1.3.1.1 V-model [3]

**Specifikace**

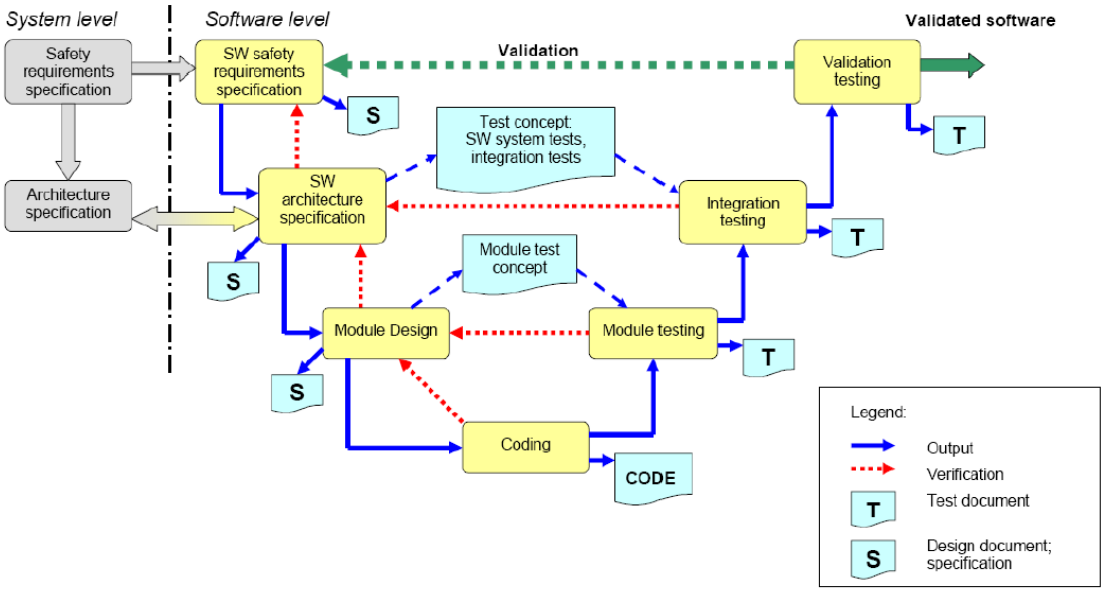
Jde o počáteční kroky při vývoji produktu. Definují se zde požadavky na bezpečnostní mechanismy pro software, výstupy z této části jsou zohledněny následně v softwarové specifikaci architektury. Výstupem softwarové specifikace architektury je specifikační dokument. Softwarová architektura je dále vstupem pro modulový design, jehož výstupem je opět specifikační dokument pro dílčí moduly.

**Implementace**

V této části je vytvářen samotný kód aplikace pro jednotlivé moduly. Kód je implementován dle specifikací daného modulu.

**Testování**

Za tuto část zodpovídá tester. Nejprve jsou testovány jednotlivé moduly, výstupem tohoto a následujících testování jsou testovací dokumenty. Následně je testováno seskupení dílčích bloků, a nakonec je testován celý systém.




Obrázek 4: Vývojová metodika pomocí V-modelu [3]

Jak je patrné z V-modelu, kód by měl být členěn na dílčí moduly. Tento přístup usnadňuje přehlednost a umožňuje snáze lokalizovat chyby při kontrole. Kromě dělení na moduly norma doporučuje nepoužívat některé konstrukce jazyka C, viz následující podkapitoly.

### 1.3.1.2 Omezení vybraných konstrukcí

**Omezené využívání přerušení**

Při použití velkého množství přerušení se systém stává náchylný k chybám a zároveň se obtížněji ladí a testuje. Z tohoto důvodu by mělo být přerušení použito jen v případech,

kde skutečně zjednoduší daný systém. Během kritických funkcí by mělo být zakázáno přerušení (změna dat, časově kritické funkce). Pro výpočty, které nelze přerušit, by měl být specifikován maximální čas, po který jsou přerušení blokována [15].

**Omezené využívání ukazatelů**
Ukazatele by měly být použity pouze v případech, kde je to nezbytné. Jejich použití zhoršuje přehlednost toku dat v programu a ztěžuje testování softwaru.

**Omezené využívání rekurze**
Funkce, která volá sebe sama, se nazývá rekurzivní funkce. Tato technika je pak nazvána rekurze. Musí existovat jasné kritérium, které umožní předpovědět hloubku rekurze. Cílem je zamezit netestovatelnému volání funkcí. Dalším problémem při velkém počtu vnoření je nebezpečí v podobě přetečení zásobníku.

**Nástroje a překladače**
Měly by být použity překladače, u kterých nejsou evidovány chyby v překladu a které byly otestovány ve více projektech.

**Dynamické objekty**
Dynamické proměnné a dynamické objekty jsou takové objekty, pro které je paměť alokována a adresy jsou určeny až za běhu programu. Hodnota alokované paměti závisí na stavu systému v okamžiku alokace, v důsledku čehož ji nelze kontrolovat kompilátorem či jiným nástrojem.

#### 1.3.1.3 Standardy a další doporučené konstrukce

Norma také doporučuje používání standardů pro návrh a psaní kódu. Normy pro kódování (anglicky coding standards) specifikují vhodný postup pro programování, zakazují nebezpečné konstrukce jazyka, popisují postupy pro dokumentaci zdrojového kódu a udávají konvence pro pojmenovávání datových struktur.

Mezi nejznámější standardy patří MISRA C [24], z anglické zkratky Motor Industry Software Reliability Association. Syntax jazyka C je taková, že je relativně jednoduché udělat typografickou chybu, i přes kterou je výsledný kód syntakticky správný. Je snadné zaměnit přiřazení „ = “ s porovnáním „ == “, což sice vede k validnímu kódu, který ale nebude plnit požadovanou funkcionalitu. Stejně tak nadbytečný středník na konci výrazu může kompletně pozměnit logiku kódu.

Filozofie jazyka C předpokládá, že programátor ví, co dělá. To v podstatě znamená, že chyby mohou projít bez povšimnutí. Příkladem je kontrolování datových typů, kdy je na straně programátora, aby pohlídal vzájemnou kompatibilitu mezi datovými typy.

Dalším častým problémem je nesprávné pochopení některých konstrukcí jazyka. Například přednost operátorů ve výrazech jsou sice velmi dobře definovány, ale jsou komplikované, tudíž může lehce dojít k nesprávné interpretaci výrazu. Některé konstrukce jazyka C jsou nejasně definovány a jejich implementace se může lišit

kompilátor od kompilátoru. Níže jsou uvedena některá vybraná pravidla doporučovaná standardem MISRA.

**Výrazy s různě přesnou aritmetikou**

Dílčí výrazy jsou vyhodnocovány s přesností odpovídající datovým typům daných operandů. Ta může být menší než přesnost výsledku. Za takového předpokladu můžou být dílčí výrazy vyhodnoceny se špatnou přesností, což může vést ke špatným výsledkům.

```
U_16 i = 1u;
U_16 j = 3u;
F_64 r0 = i / j;                          /* Spravny vysledek = 0.0 */
F_64 r1 = (F_64)    (i / j);              /* Spravny vysledek = 0.0 */
F_64 r2 = i / j;                          /* Spravny vysledek = 0.333*/
F_64 r3 = (F_64) i / F_64) j;             /* Spravny vysledek = 0.333*/

UI_16 i = 65535u;
UI_16 j = 10u;
UI_32 r0 = i + j;                        /* Nespravny vysledek = 9 */
UI_32 r0 = (UI_32) (i + j);              /* Nespravny vysledek = 9 */
UI_32 r0 = (UI_32) i + j;                /* Spravny vysledek = 65545 */
UI_32 r0 = (UI_32) i + (UI_32) j;        /* Spravny vysledek = 65545 */
```

**Testování čísel s plovoucí desetinnou čárkou a rovnost/nerovnost**

Porovnání čísel s plovoucí desetinnou čárkou se často nevyhodnotí jako pravda, i když to z logiky porovnání plyne. Chování se může lišit implementaci od implementace. Například výsledek následujícího kódu je nepředvídatelný.

```
F_32 x, y;
/*... Vypocty ... */
if (x == y)
{…}
```

**Závorky v příkazové části výrazů *if, else if, else, while, do ... while* nebo *for***

Příkazové části výše zmíněných výrazů by měly být vždy ohraničeny složenými závorkami, i když se jedná o samostatný příkaz. To zamezí nebezpečí, že kód, který má být součástí podmínkového bloku, není ve výsledné implementaci zahrnutý. V následujícím kódu vynechání závorek ve výrazu *else* způsobí, že se provede pouze přiřazení hodnoty do proměnné *x*.

```
if (test)
{
        x = 1;
}
else
        x = 3;
        y = 2;
```

**Závorky ve složitějších výrazech/makrech**

Je vhodné používat závorky v makrech a v rovnicích. Jazyk C má tabulku předností (anglicky precedence table), ve které jsou uvedeny vzájemné přednosti mezi operátory. Jelikož je těchto úrovní celkem 15, může dojít velice snadno k nesprávnému provedení

výrazu. Z tohoto důvodu je striktně doporučováno závorkovat dílčí části výrazů. Zjednodušuje to také zpětnou revizi kódu.

Následující konstrukce/doporučení sice přímo nesouvisí se standardem MISRA, jejich použití ale přispívá k celkové kvalitě softwaru.

**Kontrola rozsahu proměnné**

U mnohých proměnných v programu lze s jistotou říct, že jejich hodnota spadá do určitého intervalu. Například o proměnné ukládající úhel lze s jistotou tvrdit, že její hodnota bude ležet v intervalu od 0 do 360. Nejjednodušší ověření je pomocí podmínky *if*. Tento přístup však přidává mikrokontroléru další instrukce ke zpracování, což je v některých aplikacích nepřípustné [17].

**Pojmenování proměnných**

Využívání prefixů před názvem proměnných tak, aby při čtení kódu bylo zřejmé, jakou hodnotu (datový typ) tato proměnná ukládá.

- **Proměnné**
  - Předpona *ul* pro datový typ unsigned long: *ulVariable*
  - Předpona *fl* pro datový typ float: *flVoltage*
- **Ukazatele**
  - Předpona *pX*, kde X je datový typ: *pflVoltage*
- **Konstanty**
  - Konstanty by měly být psány velkými písmeny odděleny podtržítkem: *REFERENCE_HIGH*

**Nepoužívání číselných hodnot přímo v kódu**

Je vhodné se vyvarovat „napevno“ napsaných číselných hodnot ve výrazech (rovnicích) a namísto nich použit makro/konstantu. Takováto čísla poměrně zesložiťují pochopení daného výrazu při inspekci kódu. Další výhodou použití konstant/maker je úprava konstanty na jednom místě oproti přepisování čísla ve všech výrazech.

**Systémové parametry**

Důležité systémové parametry by měly být specifikovány pomocí maker. Například napěťové úrovně pro reference AD převodníku. Při změně těchto úrovní stačí přepsat makro na jediném místě. Dále je vhodné použití maker pro parametry související s hodinovou doménou. Hodinový signál je v mikrokontrolerech do různých periferií často distribuován přes děličky kmitočtu. Při změně frekvence oscilátoru by bylo nutné všechny parametry znovu přepočítat. Při použití maker se pak parametry přepočítají automaticky.

### 1.3.2 Náhodné chyby

Náhodné chyby vznikají v průběhu životního cyklu hardwaru a jejich výskyt se řídí pravděpodobnostní funkcí [22]. Z hlediska přetrvání chyby se náhodné chyby dělí dva druhy, přechodné (anglicky soft faults) a trvalé (anglicky hard faults).

#### 1.3.2.1 Přechodné chyby

Příčinou přechodných chyb je například rušení (elektromagnetické záření) nebo náraz částice (neutron). Dopad záření vede k tomu, že částice s vysokou energií procházejí přes hmotu integrovaného obvodu (přes křemíkový čip mikrokontroléru). Průchod takového částice může vytvořit krátký impulz (glitch), který může překlopit stav tranzistoru (klopného obvodu, paměťového prvku) [23]. To může způsobit například bitovou změnu v paměti programu, což může vést k vykonání jiné instrukce. Nebo může dojít k bitové změně v CPU registru, který ukládá mezivýsledek operace v ALU a může tak dojít ke změně výsledku operace. Zdrojem přechodných chyb je také například náhodný šum nebo přeslechy. Obvykle je porucha odstraněna resetem zařízení. Zmenšení odolnosti proti přechodným poruchám napomáhá trend snižování napájecího napětí a snižování velikosti tranzistorů, což sice přináší výhody v podobě delší výdrže baterie nebo menší plochy čipu, má to také jednu velkou nevýhodu, jelikož se tím zvyšuje náchylnost k přechodným chybám.

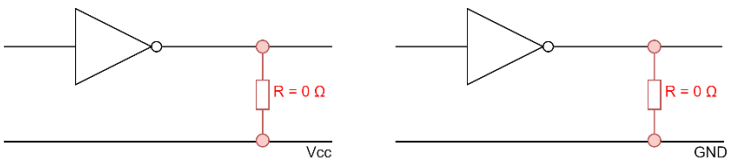
#### 1.3.2.2 Trvalé chyby

Pokud trvalé chyby přetrvávají i po restartu mikrokontroléru, jedná se o trvalé poškození. Tyto chyby mohou vznikat už při výrobě křemíku zanesením nečistot, nebo při neodborné manipulaci při osazování DPS či v průběhu životního cyklu zařízení, kdy při výboji statické elektřiny může dojít k zahřátí a následnému přepálení některých struktur (nebo částečné poškození a následná degradace v čase), což způsobí trvalou chybu, která již odstranit nelze [6].

V tabulce H.1 normy IEC 60730 se vyskytují dva chybové modely, DC coupling a stuck-at. Poruchový model je popis chování a předpokladů, jak se prvky v případě defektu chovají. Cílem poruchových modelů je pokrýt vysoké procento defektů, které se mohou v obvodu vyskytnout, na nejvyšší možné úrovni abstrakce.

**Stuck-at chyba**

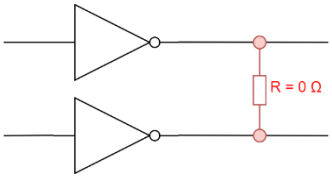
Stuck-at lze do češtiny přeložit jako „zaseknutý na“. Jde o chybový model, ve kterém se signálová cesta jeví jako konstantní logická hodnota, logická 1 nebo 0, tedy jako by byla zkratována k napájecímu napětí nebo k zemi (ve skutečnosti to nemusí přímo znamenat, že je signál zkratován přímo s napájecím napětím, ale spíše se jedná o defekt ve struktuře). V případě, že je signálová cesta trvale v úrovni logické 1, je chyba označována jako SA 1 (stuck-at 1). V případě, že je signálová cesta trvale v úrovni logické 0, je chyba označována jako SA 0 (stuck-at 0).




Obrázek 5: Princip stuck-at chyb, převzato z [6]

**DC coupling**

DC coupling lze do češtiny přeložit jako „spojení“. Jde o chybový model, ve kterém dochází ke zkratu (vodivému propojení) mezi signálovými cestami. Logická hodnota jednoho vodiče pak může ovlivnit logickou hodnotu vodiče druhého. Norma dodává, že vzhledem k počtu možných zkratů v testovaném zařízení se berou v potaz zkraty pouze souvisejících signálech.



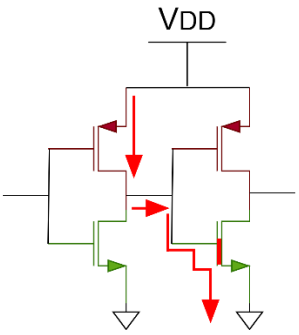

Obrázek 6: Princip DC coupling chyby, převzato z [6]

**Defekty**

Pro lepší představu jsou zde uvedeny vybrané fyzikální mechanismy, které mohou způsobit některou z výše uvedených chyb v mikrokontroléru. Defekty vznikají jak při výrobě, tak v průběhu životního cyklu integrovaného obvodu.

**Zkrat gate-oxid**

- Zkrat může být způsoben například ESD výbojem.
- Nedostatek kyslíku na rozhraní Si a SiO2.

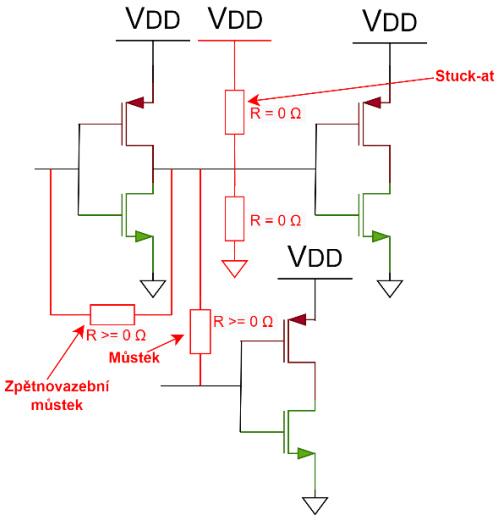



Obrázek 7: Vznik zkratu gate-oxid, upraveno z [8]

**Můstky**

Jde o nechtěné elektrické propojení mezi dvěma a více vodiči plynoucí z nadbytečného vodivého materiálu nebo naopak nedostatku izolačního materiálu.

- Například elektromigrace.

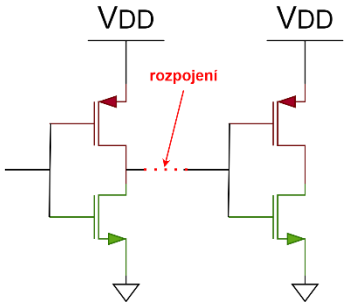



Obrázek 8: Vznik můstků, upraveno z [8]

**Rozpojení**

Jde o rozpojení vodičů v důsledku nedostatku vodivého materiálu nebo nadbytečného vodivého materiálu.

- Rozpojení může být způsobeno například ESD výbojem.




Obrázek 9: Vznik rozpojení

# 2. TESTOVÁNÍ MIKROKONTROLÉRŮ

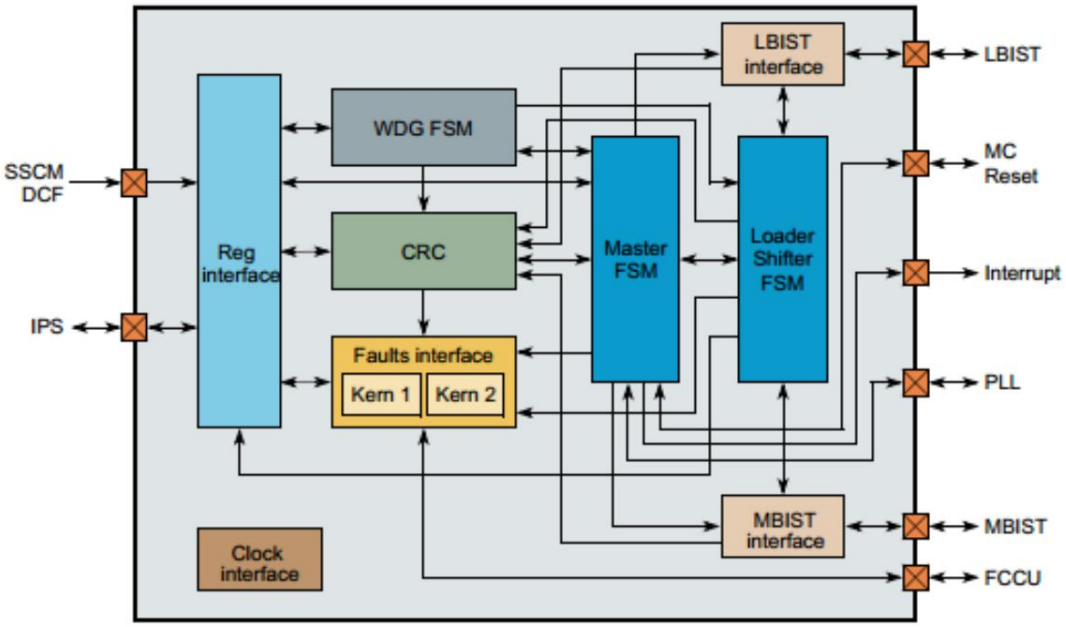
Testování lze rozdělit na dvě metody, online a offline. Online testování je prováděno souběžně s běžným provozem zařízení, aby bylo možné odhalit selhání co nejrychleji. Naopak offline testování vyžaduje, aby byl systém nebo jeho část vyřazena z provozu. V mnoha případech, když online testování odhalí poruchu, přijde na řadu technika offline testování. Ta se používá pro diagnostiku (lokalizaci a identifikaci) selhávající komponenty. Testování mikrokontroléru patří do online metod testování, jelikož probíhá za běžného provozu. Testování mikrokontroléru se dělí na hardwarové a softwarové [21].

## 2.1 Hardwarové testování

BIST (anglicky Built-In Self-Test) je hardware umístěný přímo na čipu, který umožňuje mikrokontroléru vykonávat periodické testy za účelem identifikace chyb. Na základě výsledků těchto testů může mikrokontrolér reagovat a zajistit tak bezpečný stav zařízení. Jako příklad může být uveden mikrokontrolér MPC5777M firmy NXP. Tento mikrokontrolér implementuje dva typy testování:

- MBIST (Memory Build-in-self-test): Testování pamětí (SRAM)
- LBIST (Logic Build-in-self-test): Testování digitální logiky (CPU)

U tohoto konkrétního mikrokontroléru zajišťuje testy jednotka STCU2. STCU (anglicky Self-Test Control Unit) je programovatelný hardwarový modul, který řídí samočinné testování mikrokontroléru. Nevýhodou této metody testování je zvýšení plochy čipu a vyšší spotřeba. [9]




Obrázek 10: Blokový diagram STCU [9]

## 2.2 Softwarové testování

SBST (anglicky Software-Based Self-Testing) je metoda testování, která k řízení testů využívá instrukce mikrokontroléru. Ve srovnání s hardwarovým testováním má následující nedostatky:

- Využití místa v paměti ROM
- Delší čas vykonávání
- Vytěžování výpočetního výkonu mikrokontroléru

Výhodou softwarového testování oproti hardwarovému je možnost lehce modifikovat logiku testu. V případě modifikace hardwarového testu je nutné fyzicky pozměnit dané komponenty a tím i návrh celého čipu, což je mnohonásobně komplikovanější a nákladnější.

Ve většině případů musí být testy prováděny periodicky na zařízení dle specifikací třídy B a C dle normy IEC 60730. Seznam komponent, které musí být otestovány, jsou specifikovány v odpovídajících normách, například IEC 60730 a IEC 60335. Jak již bylo zmíněno, ve většině aplikací musí být komponenty testovány za běhu programu. To vychází z požadavku, že chyba musí být detekována (test musí být vykonán) alespoň jednou za polovinu MTTF (anglicky Mean Time To Failure). Výjimkou jsou pouze zařízení, která jsou vždy vypnuta před uplynutím poloviny MTTF. Za těchto podmínek stačí provést test v inicializační fázi.

**Testy po resetu**

Ve standardních bezpečných aplikacích jsou všechny možné testy prováděny po resetu. Test hodinového signálu a test toku programu není běžně v této fázi prováděn. Také test přetečení/podtečení zásobníku může být vynechán. První fází je inicializace. Ta je společná pro testy po resetu a pro testy za běhu. Prvním prováděným testem je nejčastěji test CPU registrů, test programového čítače a následně test nevolatilní (ROM) a volatilní (RAM) paměti. Pořadí následujících testů je již libovolné.

**Testy za běhu programu**

Pří návrhu testů za běhu je nutné vzít v úvahu mnoho aspektů. K nejdůležitějším patří čas, testy nesmí příliš vytěžovat výpočetní výkon mikrokotroléru. Dobrým zvykem je řídit celou aplikaci (CPU, přerušení atd.) jediným zdrojem hodinového signálu. Naopak pro test hodinového signálu musí být k dispozici nezávislý zdroj hodinového signálu. Velká část bezpečných systému je založena na periodickém kontrolování, které je prováděno v rámci obsluhy přerušení. Samotné provádění přerušení musí být taktéž kontrolováno, viz následující podkapitoly.

V následujících podkapitolách budou rozebrány jednotlivé testy pro třídu B. Popsané metody vychází z aplikačních nót a dokumentů [5], [10], [11], [13], [16], z praktických zkušeností odborníků na funkční bezpečnost mikrokontrolérů nebo z modifikací funkčních řešení.

### 2.2.1 Volatilní paměť

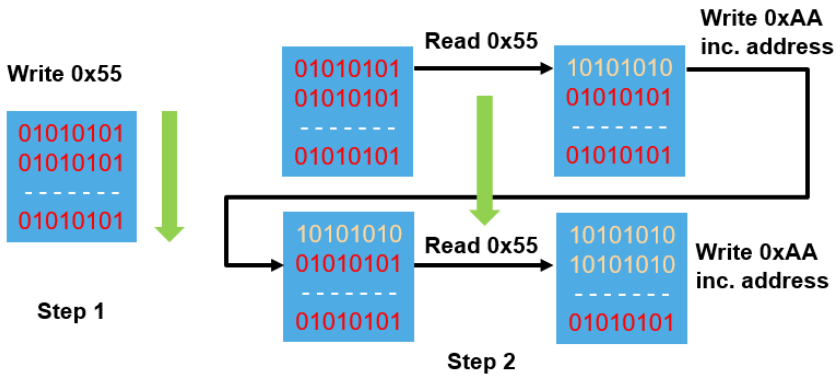
Ve volatilní paměti jsou uložena data, která se mění při vykonávání programu. Test volatilní paměti slouží k rozpoznání, zda testovaný bit obsahuje poruchu stuck-at 1 či 0 a také k rozpoznání, zdali nedochází k vzájemnému ovlivňování paměťových buněk. Test variabilní paměti může být destruktivní nebo nedestruktivní. Destruktivní testy „zničí“ obsah testované paměti přepsáním původních hodnot při vykonávání testu, zatímco při nedestruktivním testu zůstává obsah paměti zachován. Při destruktivním testu je nutné nejprve zálohovat část paměti před započetím testu a následně obnovit data po dokončení testu. Dle standardu IEC 60730 pro třídu B je nutné pro splnění požadavků normy použit minimálně jeden z mechanismů: Periodický test statické paměti nebo zabezpečení pomocí bitové redundance. Zabezpečení pomocí bitové redundance je však náročnější na paměťové prostředky, jelikož každému datovému slovu náleží paritní bit určující sudou/lichou paritu. Nejčastěji implementovanými periodickými testy bývají March testy.

Testování variabilní paměti patři mezi nejnáročnější testy ve třídě B. Samotný test nemůže být přerušen, jelikož by mohlo dojít k selhání programu. Test je také náročný z hlediska výpočetního času, proto v reálné aplikaci není možné kontinuálně otestovat celou paměť a je nutné otestovat paměť po částech. Níže uvedené March testy využívají bitové vzory 0x55 a 0xAA. Tyto vzory jsou zvoleny záměrně, jelikož si jsou vzájemně inverzní.

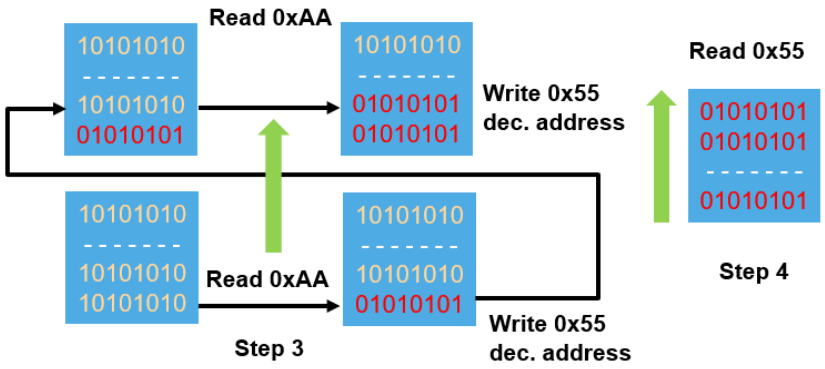
#### 2.2.1.1 March X

March X je průmyslově standardizovaný test pro kontrolu adresového dekodéru a stuck-at chyb v paměti RAM. Test se skládá z těchto kroků:

1. Zápis vzoru 0x55, inkrementace adresy.
2. Čtení vzoru 0x55 a zápis vzoru 0xAA, inkrementace adresy.
3. Čtení vzoru 0xAA a zápis vzoru 0x55, dekrementace adresy.
4. Čtení vzoru 0x55, dekrementace adresy.




Obrázek 11: Krok 1 a 2 March X testu [2]

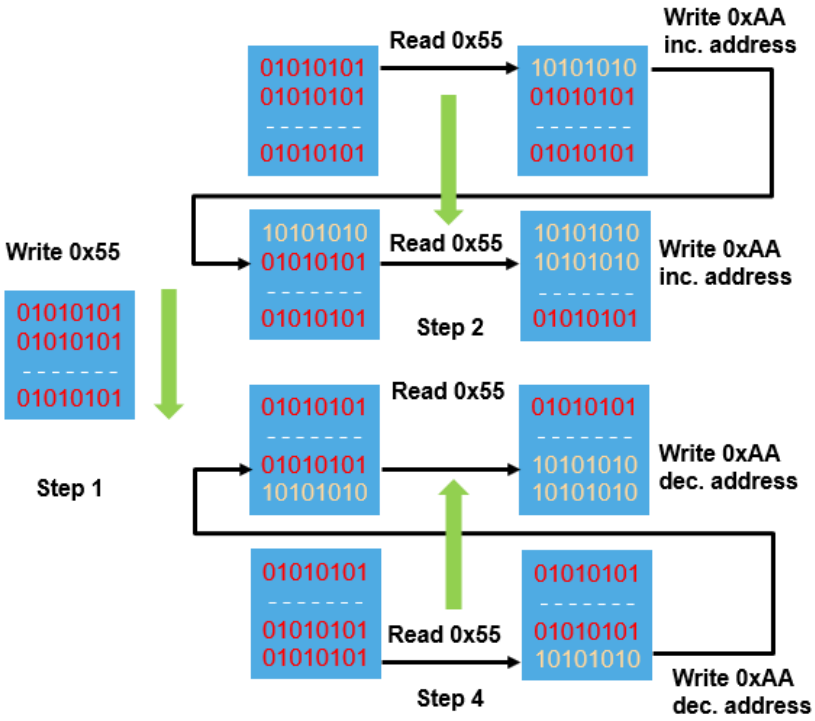



Obrázek 12: Krok 3 a 4 March X testu [2]

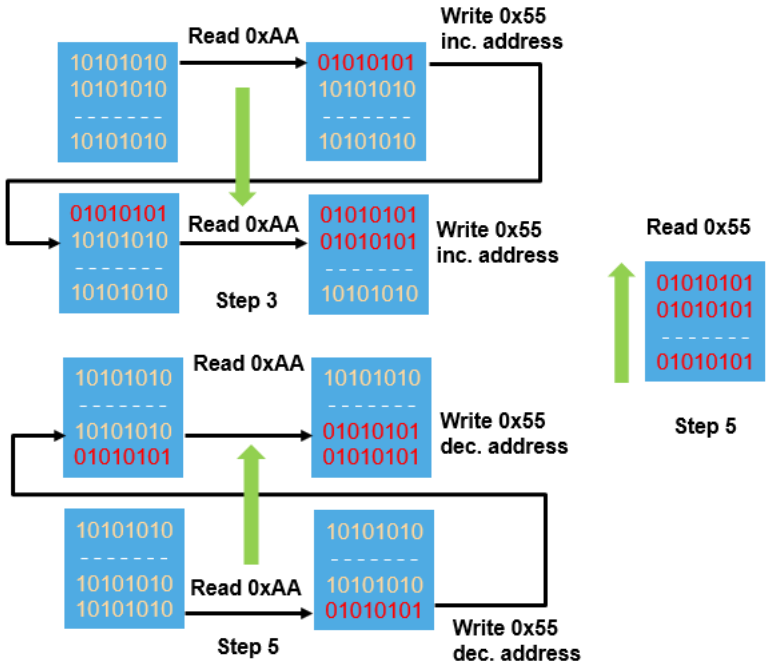
#### 2.2.1.2 March C

1. Zápis vzoru 0x55, inkrementování adresy.
2. Čtení vzoru 0x55 a zápis vzoru 0xAA, inkrementace adresy.
3. Čtení vzoru 0xAA a zápis vzoru 0x55, inkrementace adresy.
4. Čtení vzoru 0x55 a zápis vzoru 0xAA, dekrementace adresy.
5. Čtení vzoru 0xAA a zápis vzoru 0x55, dekrementace adresy.
6. Čtení vzoru 0x55, dekrementace adresy.

Testovací vzor March X je podmnožina testovacího vzoru March C. March X detekuje většinu chyb detekovanou testem March C. Jeho výhodou je rychlejší zpracování v čase.




Obrázek 13: Krok 1, 2, 3 a 4 March C testu [2]

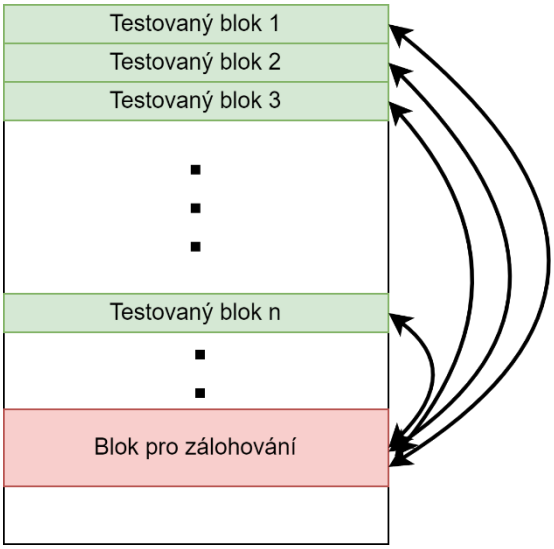



Obrázek 14: Krok 3, 4, 5 a 6 March C testu [2]

#### 2.2.1.3 Transparent March

Jelikož March C a March X testy jsou destruktivní, je nutné před započetím testu zálohovat data a po jeho dokončení data opět nahrát zpět. Paměť je rozdělena na bloky, které jsou podrobeny testu. V bloku pro zálohování jsou dočasně uložena data, dokud není dokončen March test. Blok pro zálohování je také nutné otestovat. Mechanismus se skládá z těchto kroků:

1. Kopírování testovaného bloku do bloku pro zálohování.
2. Ověření, zdali bylo kopírování úspěšné.
3. Otestování testovaného bloku pomocí March testu.
4. Kopírování blok pro zálohování do testovaného bloku.
5. Ověření, zdali bylo kopírování úspěšné.

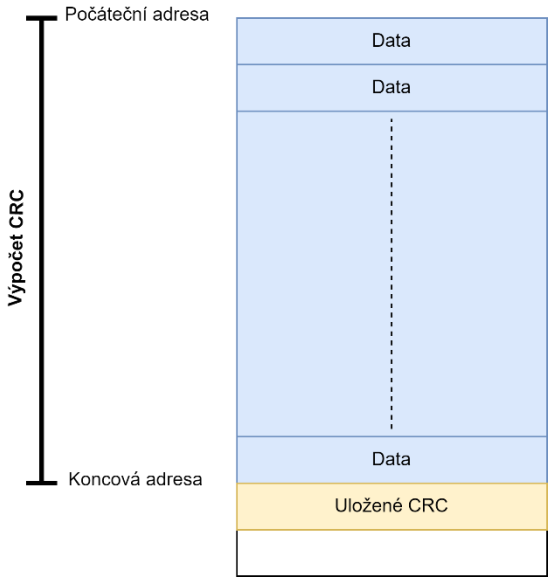



Obrázek 15: Segmentace paměti RAM pro destruktivní test

### 2.2.2 Nevolatilní paměť

V nevolatilní paměti je uložen vykonávaný kód a konstanty. Z paměti jsou následně data (program) postupně zpracovávána pomocí dekodéru instrukcí. Nevolatilní paměť znamená, že po odpojení napájecího napětí zůstávají data v paměti zachována. Chyba v jednom bitu může způsobit změnu strojové instrukce a tím fatálně změnit chod programu. Z tohoto důvodu je nutné tuto paměť testovat.

Test nevolatilní paměti je tedy kontrola, zdali nedošlo ke změně dat v paměti za běhu aplikace. K tomu může být využito více metod kontrolního součtu. Kontrolní součet je doplňková informace, která se předává společně s užitečnou informací. Kontrolní součet je výsledkem určité operace (CRC, Hammingův kód, parita). Originální hodnota kontrolního součtu je vypočítána ve fázi linkování na straně počítače a je uložena do nevolatilní paměti, ale do jiného segmentu, než pro který je kontrolní součet počítán. Kontrolní součet je následně periodicky počítán a porovnáván s originální hodnotou. Nejčastěji implementovaným testem je CRC (cyklický redundantní součet). Některé mikrokontroléry disponují hardwarovou podporou CRC. Využití hardwarové periferie pro výpočty šetří výpočetní výkon mikrokontroléru a je násobně rychlejší než softwarový výpočet.



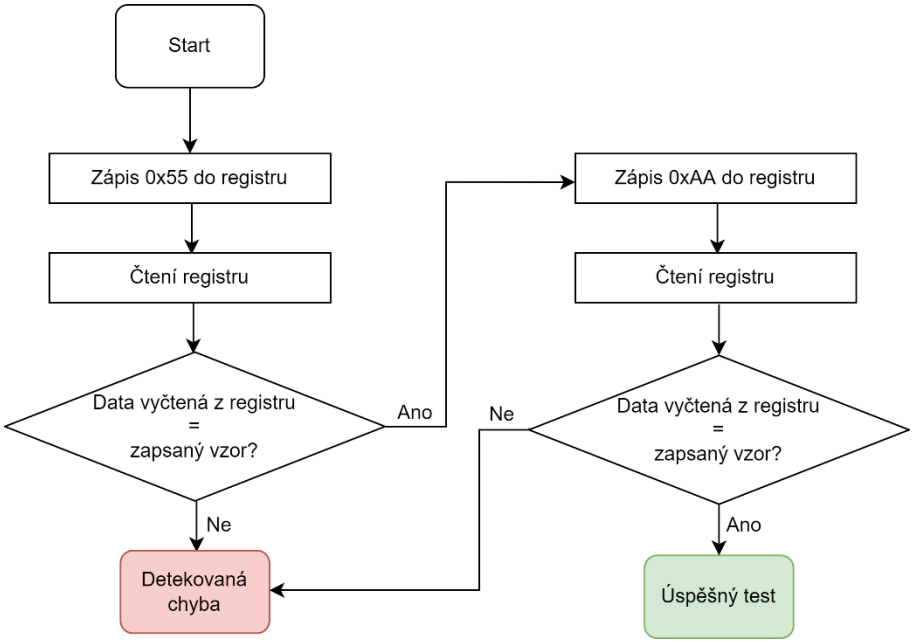

Obrázek 16: Rozložení nevolatilní paměti

### 2.2.3 CPU registry

CPU registry ukládají průběžné informace. Registry všeobecného použití ukládají výsledky aritmetických a logických operací. Speciální registry zase ukládají specifické informace, například nulový výsledek, přetečení atd. Aritmeticko-logická jednotka (ALU) nepracuje přímo s hodnotami v paměti, ale nejprve tyto hodnoty kopíruje do registrů a až následně provádí výpočty.

Cílem testů CPU registrů je odhalit stuck-at poruchu. Do registru je nahrán vzor, který je následně porovnán s konstantou nebo dalším registrem. Poté je do registru nahrán invertovaný vzor a opět porovnán. V případě, že se porovnávané hodnoty liší, byla odhalena porucha a je nutné na ni reagovat. K detekování stuck-at poruch je nejčastěji implementován algoritmus Checkerboard, který využívá specifické vzory 0xAA a 0x55. Tyto vzory jsou si vzájemnou negací. Algoritmus by měl taktéž rozpoznat vzájemné ovlivňování sousedních buněk. Popis algoritmu je vyobrazen na následujícím diagramu.

Důležité je poznamenat, že existují volatilní a nevolatilní typy registrů. Hodnota volatilního registru může být změněna v subrutině a není nutné hodnotu tohoto registru zálohovat. Oproti tomu hodnota nevolatilního registru musí být zálohována a po návratu ze subrutiny obnovena. V případě nezálohování hodnoty registru by došlo k jejímu přepsání po návratu ze subrutiny by funkce používala nesprávné hodnoty. Níže uvedený diagram je platný pro volatilní registry. Pro nevolatilní registry je nutné před zápisem do registru zálohovat jeho obsah a po ukončení testu jeho obsah opět obnovit.




Obrázek 17: Princip testu CPU volatilního registru algoritmem Checkerboard

### 2.2.4 Programový čítač

Programový čítač (anglicky Program Counter) je registr, který ukládá adresu instrukce, která má být následně vykonána. Po vykonání instrukce je programový čítač automaticky inkrementován. V případě skoků je do něj nahrána adresa, ze které má vykonávat další kód.

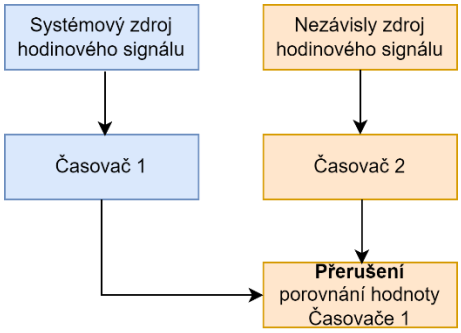
I když je programový čítač CPU registr, tak v oblasti funkční bezpečnosti se jedná o samostatnou komponentu. Je testován na „stuck-at“ poruchu. Programový čítač může být testován například pomocí skoku do volatilní paměti a následně skokem do nevolatilní paměti. Obě adresy musí být zvoleny tak, aby otestovaly všechny bity registru. Tento mechanismus testování programového čítače je uváděn v aplikačních nótách napříč všemi výrobci. Z praktického hlediska je tento test naprosto nevyhovující. Jestliže je chybový bit registru programového čítače součástí adresy zpracovávané před vykonáním testu, dojde ke zhroucení aplikace a test se tak ani nevykoná. V případě, že je vadný bit součástí testované adresy, nedojde ke správnému vykonání testu a test je tedy neprůkazný. Tento způsob testování může být vhodný pouze ve specifických případech. Například když program mikrokontroléru je vykonáván z první poloviny adresového prostoru (spodní bity), což znamená, že horní bity registru programového čítače nejsou využívány. Ve spodní části paměti (vrchní bity) může být uložena rutina kritická z hlediska bezpečnosti, která je volána v delších časových intervalech. Test tak v případě poruchy vykonává program z jiné adresy a mikrokontrolér detekuje poruchu některým z testovacích mechanismů (například časovačem watchdog) ještě dřív, než dojde k volání dané rutiny.

Nejspolehlivější metodou testování je například logické monitorování korektního vykonávání programu v návaznosti na časovač watchdog.

### 2.2.5 Hodinový signál

Test hodinového signálu kontroluje, zda hodinový signál, který řídí běh mikrokontroléru, je přesný. Dle normy musí být pro krystalový oscilátor detekovány harmonické a subharmonické frekvence. Ideální je pro test použít hardwarové prostředky mikrokontroléru tak, aby testování co nejméně vytěžovalo výpočetní výkon mikrokontroléru. Test neměří přesnou frekvenci systémového zdroje hodinového signálu, ale porovnává poměr dvou frekvencí. Test může být proveden pomocí dvou časovačů, Časovač 1 a Časovač 2, každý z nich musí být řízen nezávislým zdrojem hodinového signálu. V obslužné rutině přerušení pomalejšího časovače (Časovač 1) je čtena hodnota Časovače 2. Jestliže je hodnota Časovače 1 v mezích tolerance, test byl úspěšný.

Další možností provedení testu hodinového signálu je využití funkce zachycení (anglicky capture). Do časovače jsou přivedeny dva nezávislé zdroje hodinového signálu, primární a sekundární. Primárním zdrojem je systémový zdroj hodinového signálu. Sekundární vstup funguje jako spouštěč (anglicky trigger). S každou náběžnou hranou sekundárního signálu je zachycen určitý počet náběžných hran primárního signálu. V obslužné rutině přerušení je následně porovnán počet náběžných hran primárního signálu s mezemi tolerance.



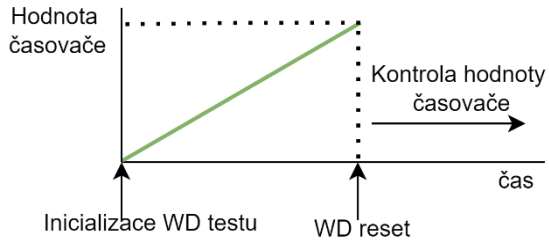

Obrázek 18: Test systémového zdroje hodinového signálu pomocí dvou časovačů

### 2.2.6 Časovač watchdog

Časovač watchdog je speciální druh časovače, který dokáže detekovat anomálie v softwaru (například uvíznutí programu ve smyčce) a následně resetovat mikrokontrolér, čímž jej uvede do definovaného stavu. V případě, že časovač watchdog dosáhne konce čítací periody, uvede mikrokontrolér do resetu. V kódu musí být watchdog obsluhován tak, že před koncem periody je časovač vyresetován, čímž se zabrání resetu. Některé umožňují pracovat v režimu „Window“, kdy je pomocí dvou hodnot nastaveno časové okno, ve kterém musí být časovač obsloužen (resetován). V tomto módu je možné rozpoznat, jestli se kód vykonává rychleji či naopak pomaleji. Nastavování watchdogu se liší v závislosti na mikrokontroléru, u některých je potřeba zapsat přesný vzor do speciálního registru pro odemknutí, což zajišťuje vyšší bezpečnost.

Test časovače watchdog se provádí pouze jednou, a to po Power-On Resetu (POR). Test je proveden za pomoci časovače. Na začátku testu je povolen časovač

watchdog a inicializována periferie časovače, který musí být řízen nezávislým zdrojem hodinového signálu. V nekonečné smyčce je pak periodicky vyčítána hodnota tohoto časovače a ukládána do paměti. Následně watchdog resetuje mikrokontrolér a ve funkci je kontrolována hodnota časovače. V případě, že je tato hodnota v nadefinovaných mezích, znamená to, že byl test úspěšný.




Obrázek 19: Princip testu watchdog časovače

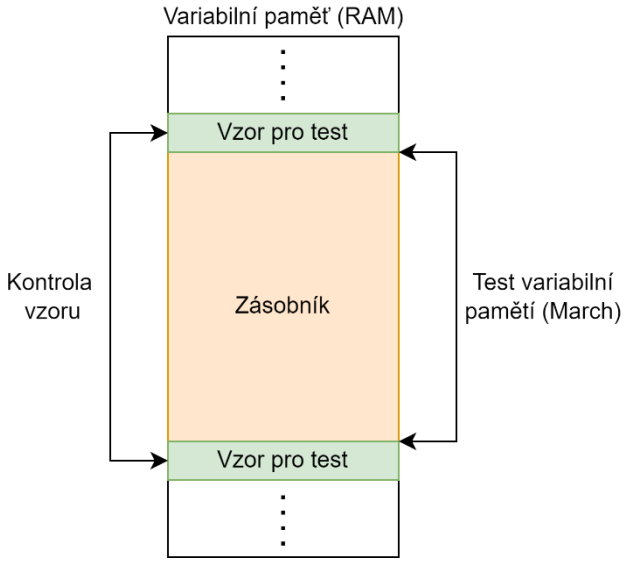
### 2.2.7 Zásobník

Zásobník je segment volatilní paměti, který slouží jako dočasné úložiště dat. Zásobník má strukturu LIFO (Last-In First Out), což znamená, že data, která byla uložena jako poslední, jsou čtena jako první. Zásobník je umístěn na předem definovanou adresu a při přidání objektu roste a při odebrání objektu se zmenšuje. Informaci o tom, kde se nachází aktuální vrchol zásobníku, nese registr ukazatel na zásobník (anglicky Stack pointer). Instrukce PUSH způsobí, že je objekt umístěn na vrchol zásobníku. Instrukce POP způsobí, že objekt je odebrán z vrcholu zásobníku. Slouží například k uložení obsahu registrů při volání funkce nebo při přerušení.

Zásobník je paměť typu RAM, takže je součástí testu požadovaného normou pro RAM paměť. Detekce podtečení a přetečení nejsou uvedeny v požadavcích normy, ale je dobrým zvykem tento test vykonávat. Špatná funkcionalita zásobníku může vést ke zhroucení celé aplikace.

Princip testu pro rozpoznání přetečení/podtečení je takový, že před oblast a za oblast, která je vymezena pro zásobník, jsou uložena definovaná data. Tato data jsou za běhu programu periodicky čtena. V případě, že došlo k podtečení nebo přetečení zásobníku, jsou tato data narušena. Častým důvodem přetečení/podtečení zásobníku je špatně navržený software (faktoriál velkého čísla pomocí rekurzivní funkce).

Princip testu pro odhalení stuck-at poruchy v paměťovém prostoru zásobníku je stejný jako u volatilní paměti, tedy například jeden z March testů.




Obrázek 20: Princip testování zásobníku

### 2.2.8 Tok programu

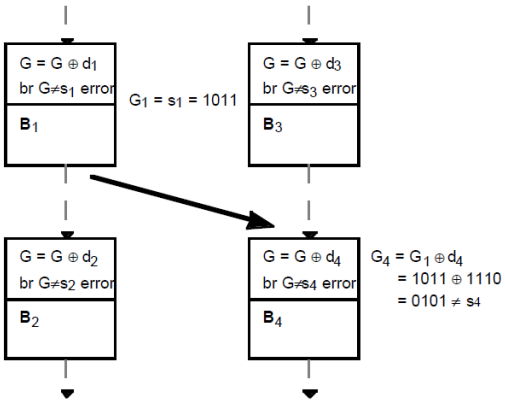
Tok programu vyjadřuje vykonávání jednotlivých bloků kódu v čase. Poruchy mohou způsobit vykonání nesprávné sekvence instrukcí a tím uvést zařízení do hazardního stavu. Test toku programu není normou přímo vyžadován, ale je důrazně doporučován. Často používanou metodou je CFCSS (anglicky Control Flow Checking by Software Signatures).

Program je rozdělen do základních bloků. Základním blokem je myšlena sekvence instrukcí bez instrukcí skoku, ty se nachází pouze na začátku a na konci bloku. Každému bloku je přiřazena určitá značka, která je vložena do programu při kompilaci přímo do konstantního pole instrukce. Tyto značky jsou následně porovnávány se značkami vypočítanými za běhu programu. Označme značku aktuálně vykonávaného bloku počítanou za běhu programu jako $G$. Každému bloku $v_i$ je přiřazena značka $s_i$ při kompilaci. $G_i$ je hodnotou $G$, když je vykonáván blok $v_i$. Při korektním běhu programu musí být $G_i$ rovno $s_i$. V případě, že se tyto značky nerovnají, došlo k chybě. Při přechodu do následujícího bloku je vypočtena nová hodnota $G$. Nová hodnota $G$ je vypočtena pomocí hodnoty $G$ předchozího bloku a hodnoty $G$ současného bloku [13].

Pro tento výpočet je použita funkce $f(G,d_i) = G \text{ XOR } d_i$, $s_s$ a $s_d$ jsou značky zdrojového bloku a cílového bloku. Rozdíl značek $d_d$ ($d_d = s_s \text{ XOR } s_d$) je vypočítán při kompilaci a uložen v cílovém bloku. Před tím, než dojde ke skoku mezi bloky, je do G uložena značka zdrojového bloku. Po skoku do cílového bloku je hodnota G aktualizována novou hodnotou.

$G_d = f(G_s, d_d)$, kde $G_s$ je značka zdrojového bloku a $d_d$ je rozdíl značek. Jestliže $G_d$ je rovno značce $s_d$ cílového bloku $v_d$, tok programu je správný. Operace XOR nebyla vybrána náhodně, je výhodnější než ostatní operace v ALU. Operace AND, OR a XOR využívají v ALU méně hradel než ostatní sčítání nebo násobení. Čím méně je pro danou

operaci použito hradel, tím je menší pravděpodobnost výskytu chyby. Navíc, operace AND a OR nemohou jednoznačně určit vstup daný dalším vstupem a výstupem.




Obrázek 21: Detekce nesprávného skoku [13]

### 2.2.9 Digitální vstupy/výstupy

Digitální vstupy/výstupy řídí vstupní a výstupní signály na pinech mikrokontroléru. Mohou být ovládány pomocí vstupně/výstupního registru nebo jim může být přidělena pomocí řídícího registru jiná funkcionalita, například SPI, I2C či výstup PWM modulu. Za normálních podmínek je velikost rezistivity mezi body pin-VCC a pin-GND velmi vysoká.

Pro detekování pin-GND zkratu je pin řízen pull-up rezistorem. Za předpokladu, že pin plní správně svou funkcionalitu, mikrokontrolér vyčte logickou 1 díky pull-up rezistoru. V případě, že je pin zkratován (velmi malá hodnota rezistivity pin-GND oproti hodnotě pull-up rezistoru), hodnota na pinu je čtena jako logická 0.

Pro detekování pin-VCC zkratu je pin konfigurován jako pull-down rezistorem. Za předpokladu, že pin plní správně svou funkcionalitu, mikrokontrolér vyčte logickou 0 díky pull-down rezistoru. V případě, že je pin zkratován (velmi malá hodnota rezistivity pin-VCC oproti pull-down rezistoru), hodnota na pinu je čtena jako logická 1.

### 2.2.10 AD převodník

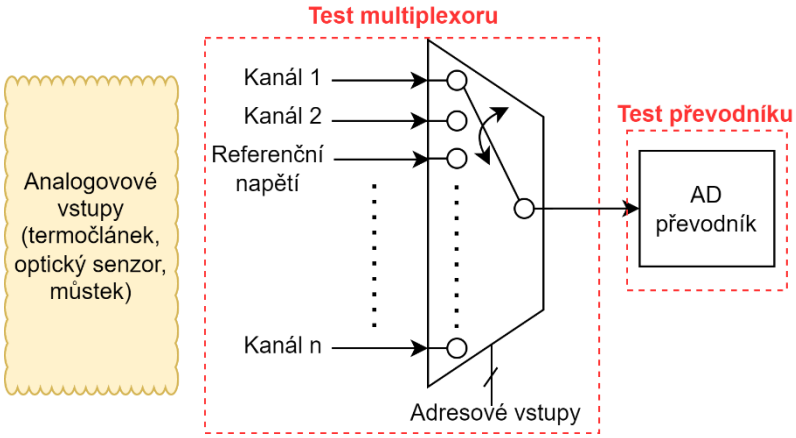
Analogové vstupy v mikrokontroléru jsou připojeny do AD převodníku. Tato periferie slouží k převodu vstupního analogového napětí do jeho digitální hodnoty. Vstupem může být například senzor teploty, tlaku nebo výstup z můstku.

#### 2.2.10.1 Převodník

Dle normy IEC 60730 pro třídu B je nutné testovat správné fungování samotného převodníku. To lze otestovat například pomocí převodu známé hodnoty analogového napětí, například referenční napětí, které je nezávislé na napájení převodníku a mikrokontroléru.

#### 2.2.10.2 Analogový multiplexor

Signál, který má být převáděn, je vybírán pomocí multiplexoru. Porucha v multiplexoru může znamenat převod jiného kanálu. Je tedy nutné otestovat správné adresování analogového multiplexoru. Způsob testování multiplexoru je úzce spjatý s konkrétní aplikací. Například u řízení pohonů jsou známy stavy při definovaných PWM cyklech a tím i očekávané hodnoty proudů.



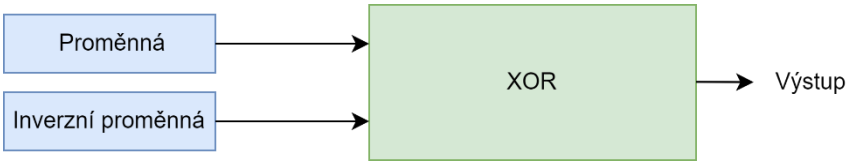

Obrázek 22: Testované části AD převodníku s vymezenými bloky

### 2.2.11 Duplikování paměti

Zatímco výše zmíněné testy variabilní paměti (March C a X) odhalí permanentní poruchy, metoda duplikování paměti odhalí přechodné poruchy. Ty mohou být způsobeny nechtěným přepsáním proměnné v přerušení s vyšší prioritou nebo překlopení bitu v pamětí vlivem ionizujícího záření.

Princip této metody je následující:

- Proměnná a její bitová inverzní hodnota jsou zapsány do rozdílných paměťových regionů
- Při zápisu do proměnné je aktualizována i její bitová inverzní hodnota
- Při čtení proměnné je provedena operace XOR mezi proměnnou a její inverzní hodnotou
- Je-li výsledek roven 0, test proběhl úspěšně




Obrázek 23: Princip duplikování paměti

# 3. NÁVRH DEMONSTRAČNÍHO HARDWARU

Jak již bylo zmíněno v teoretickém úvodu, mikrokontrolér je ve většině případů součástí většího celku, a proto je funkční bezpečnost mikrokontroléru úzce spjata s celkovou funkční bezpečností celého zařízení. Některé testy vyžadované normou lze splnit interakcí mikrokontroléru s okolními součástkami. Navíc mikrokontrolér hraje důležitou roli v diagnostice součástek kritických z hlediska bezpečnosti. Z tohoto důvodu je nutné některé testy mikrokontroléru navrhnout na míru pro konkrétní aplikaci i s ohledem na již zmíněnou diagnostiku.

Tato kapitola je zaměřena na návrh funkční bezpečnosti pro konkrétní aplikaci, a to plynový kotel. Budou též navrženy mechanismy pro diagnostiku součástek kritických z hlediska funkční bezpečnosti. Tyto principy, které je možné použít v reálné aplikaci, budou doplněny o podpůrné obvody tak, aby bylo možné simulovat funkci plynového kotle na demonstračním přípravku. Práce si tedy neklade za cíl navrhnout plynový kotel jako celek, ale pouze navrhnout bezpečné prvky pro ovládání plynového ventilu kotle a aplikovat testy pro funkční bezpečnost mikrokontroléru doplněné o diagnostické mechanismy.

V této kapitole tedy bude navržen demonstrační přípravek, který bude imitovat funkci plynového kotle. Bude na něm možné demonstrovat navržené testy, mechanismy funkční bezpečnosti a navržené diagnostické mechanismy.

## 3.1 Výběr mikrokontroléru

Pro návrh a implementaci funkční bezpečnosti byl zvolen mikrokontrolér KV46 výrobce NXP. Zadání práce vychází z norem IEC 60335 a IEC 60730 pro domácí spotřebiče. Mikrokontrolér KV46 byl zvolen z důvodu širokého uplatnění v tomto odvětví průmyslu.

### 3.1.1 Mikrokontrolér MKV46

Mikrokontrolér MKV46 patří do rodiny mikrokontrolérů Kinetis založené na jádře Arm® Cortex®-M4 s maximální frekvencí 168 MHz a s dedikovanou jednotkou pro výpočty s plovoucí desetinnou čárkou. Mezi nejvýkonnější periferie patří 12bitový ADC převodník a eFlexPWM periferie s rozlišením 312 ps. Díky zmíněným vlastnostem je mikrokontrolér určen převážně pro aplikace řízení motoru nebo aplikace pro řízení konverze výkonu [14].

**Vlastnosti mikrokontroléru:**

**Paměti:**

- Až 256 KB programové paměti flash
- Až 32 KB paměti RAM

**Systémové periferie:**

- 16kanálový DMA kontrolér
- Nezávislý časovač watchdog

**Hodinový signál:**

- Od 32 kHz do 40 kHz nebo od 3 MHz do 32 MHz krystalový oscilátor
- Multipurpose clock generator (MCG) s fázovým a frekvenčním závěsem odvozeného od interního nebo externímu zdroje hodinového signálu

**Provozní charakteristika:**

- Napěťový rozsah: 1,71 až 3,6 V
- Teplotní rozsah: -40 až 105 °C

**Vstupně-výstupní rozhraní:**

- General-purpose input/output (GPIO)

**Komunikační rozhraní:**

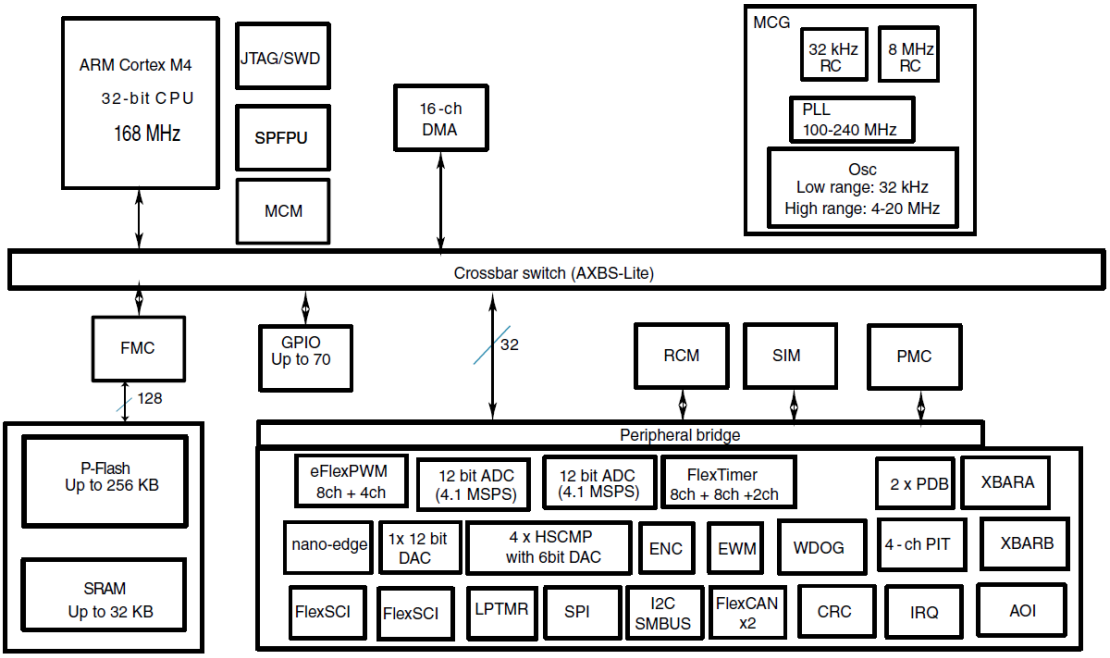
- 2x modul UART)/FlexSCI moduly s programovatelným 8 nebo 9bitovým datovým formátem
- 1x 16bitový SPI modul
- I2C modul
- 2x FlexCAN moduly

**Analogové moduly:**

- 2x 12bitový ADC modul
- 4x analogový komparátor (CMP) obsahující 6bitový DAC převodník a programovatelný referenční vstup
- 12bitový DAC převodník

**Časovače:**

- eFlexPWM s 4 submoduly, celkem umožňující generovat 12 PWM výstupů
- 2x 8kanálové FlexTimers (FTM0 a FTM3)
- 2kanálový FlexTimer (FTM1)
- 4x Periodic interrupt timers (PIT)
- 2x Programmable Delay Blocks (PDB)
- Quadrature Encoder/Decoder (ENC)
- Poměr mezi vstupní frekvencí časovače a frekvencí jádra je 1:2 jestliže frekvence jádra je 168 MHz
- Poměr mezi vstupní frekvencí časovače a frekvencí jádra je 1:1 jestliže frekvence jádra je menší rovna 100 MHz




Obrázek 24: Blokové schéma mikrokontroléru MKV46 [18]

## 3.2 Plynový kotel

Plynový kotel je nedílnou součástí každé domácnosti. Na trhu je mnoho různých typů kotlů s různými modifikacemi. K účelům této práce bude navrhováno ovládání pro plynový kotel vycházející ze zjednodušených předpokladů. Základem plynového kotle je spalovací komora, ve které dochází k samotnému procesu spalování. Přívod plynu je zajištěn pomocí plynového ventilu. Spalování je řízeno termostatem, plyn zažehnut ve spalovací komoře pomocí jiskry.

Správné fungování všech ovládacích prvků je z hlediska celkové bezpečnosti zařízení naprosto nezbytné. Teplota kotle je snímána pomocí teplotního senzoru. Výstup tohoto senzoru je přiveden na vstup AD převodníku. Při překročení kritické teploty musí spolehlivě dojít k vypnutí plynového ventilu. Poškození mikrokontroléru ale může způsobit, že AD převodník špatně převádí vstupní napětí, a i při překročení teploty nad kritickou mez je na jeho výstupu hodnota odpovídající teplotě pod kritickou hranicí. Může též dojít k situaci, kdy tranzistor (nebo jiný spínací prvek), který ovládá plynový ventil, je trvale zkratován, čímž je znemožněno vypnutí ventilu, což vede opět k nebezpečné situaci. K uzavření ventilu nemusí dojít kvůli špatně navrženým algoritmům pro ovládání kotle. Následný únik plynu může způsobit otravu člověka. Při větším úniku může dojít k výbuchu, což je potenciální nebezpečí pro větší skupinu lidí. Sekundárními důsledky jsou škody na majetku. Je zřejmé, že při návrhu musí být postupováno v souladu s normami, v tomto konkrétním případě v souladu s normou EN 13611 [27]. Před samotným návrhem je nutné klasifikovat třídu, do které bude zařízení (v tomto případě plynový kotel) spadat. Jelikož špatné fungování kotle může

způsobit smrt člověka nebo skupiny lidí, je řazen do třídy C: *Řídicí funkce, které mají zabránit speciálním nebezpečím, jako je například výbuch nebo jejíž selhání může přímo způsobit nebezpečnou nebezpečí ve spotřebiči*.

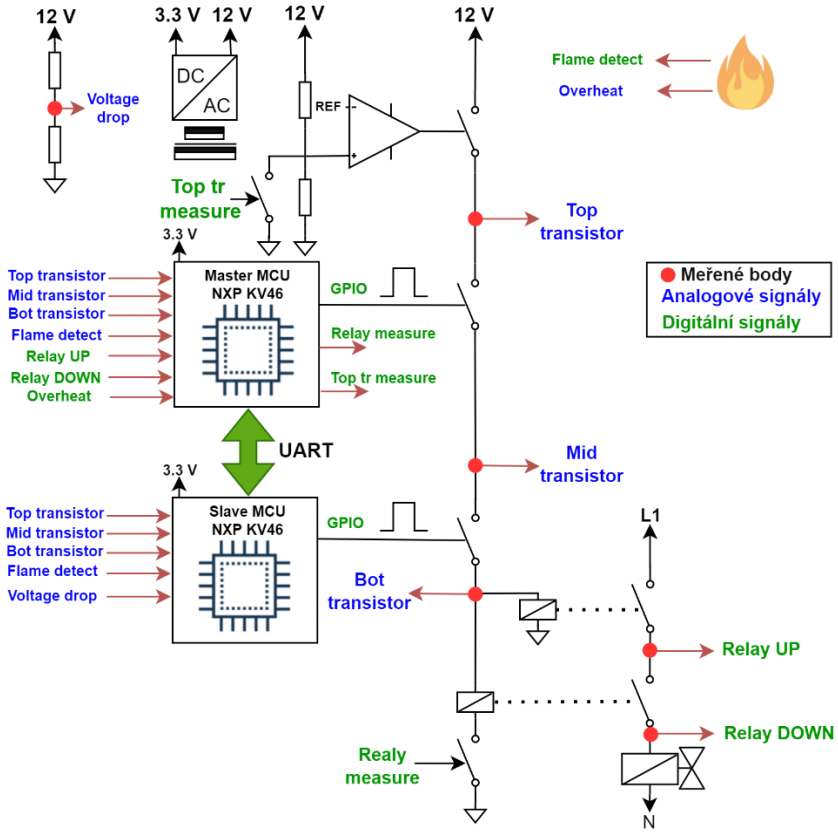
Řídicí funkce spadající do třídy C by měly být dle normy IEC 60730 navrženy podle jedné z následujících struktur:

- Jediný kanál s periodickými samočinnými testy a s monitorováním
- Duální kanál (homogenní) s porovnáním
- Duální kanál (heterogenní) s porovnáním

Teoreticky by bylo možné použít jeden mikrokontrolér pro řízení celého kotle, na kterém by probíhaly testy vyžadované třídou C. Tyto testy jsou mnohem složitější, časově a výpočetně náročnější a v mnoha případech obtížněji realizovatelné při konfiguraci uvažující pouze jeden kanál zabezpečující funkční bezpečnost. Jako příklad lze uvést test instrukcí. To znamená, že jednotlivé instrukce mikrokontroléru musí být periodicky testovány, zdali fungují správně. Navíc by toto řešení vyžadovalo externí součástky pro splnění požadavků na funkční bezpečnost. Z důvodu vyšší výpočetní náročnosti testů pro třídu C a také z hlediska celkové bezpečnosti bylo navrženo jiné řešení, a to pomocí dvou mikrokontrolérů s testy pro třídu B. Toto řešení má nejblíže ke struktuře Duální kanál (heterogenní) s porovnáním. Mikrokontroléry fungují v konfiguraci maste-slave. V dalších částech textu bude výrazem „master“ myšlen mikrokontrolér master, výrazem „slave“ mikrokontrolér slave. Oba mikrokontroléry, master i slave, monitorují celý systém v reálném čase a zabezpečují nezávisle odpojení přívodu plynu v případě detekované poruchy. Oba mikrokontroléry spolu komunikují a dochází k periodické křížové kontrole obou sub-systémů. Vzájemná interakce mikrokontrolérů probíhá přes rozhraní UART a bude podrobněji popsána v kapitole zabývající se návrhem softwaru.

Pro zvýšení celkové bezpečnosti by bylo vhodné využít mikrokontroléry, které jsou vyrobeny jinou technologií a jsou postaveny na jiném jádře, aby byl hardware diverzifikován. Tento přístup zesložiťuje proces vývoje [19]. Pro demonstrační účely je v této práci použit stejný typ mikrokontroléru.

Z hlediska bezpečnosti je nejkritičtější samotný plynový ventil. Ten musí být při detekované chybě vždy spolehlivě vypnut. Bezpečný stav zařízení je tedy takový, kdy je plynový ventil vypnutý a žádný plyn tak neproudí. Ventil musí mít takové provedení, že bez přítomnosti napětí je uzavřený. Návrh je tedy zaměřen na to, aby bylo možné bezpečně vypnout ventil při selhání jakékoliv součástky ve spínací větvi nebo při selhání mikrokontroléru. Selhání součástky může být způsobeno vadnou součástkou přímo z výroby, další chyby mohou být zaneseny v procesu osazování a pájení, kdy například některé součástky mohou být špatně zapájeny nebo zaměněny s jinými. To vše by vedlo k potenciálně nebezpečné situaci. Na obrázku níže je blokové schéma kotle s řešením pomocí dvou mikrokotrolérů KV46, vycházející z požadavků normy EN 13611 [27].




Obrázek 25: Blokové schéma řízení plynového kotle

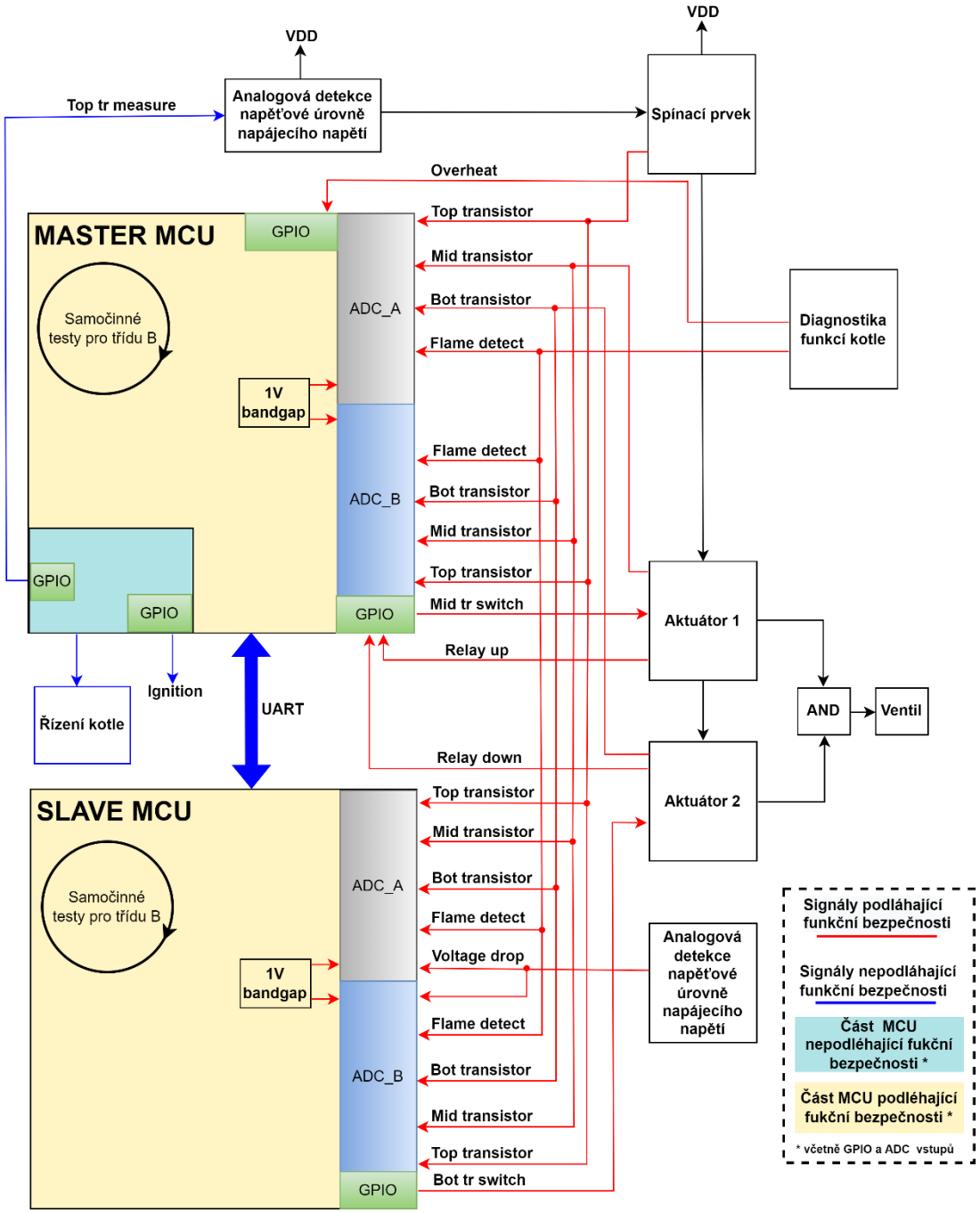
Z blokového schématu je patrné, že při sepnutých relátkách je na ventilu napětí o hodnotě L1. I když pro demonstrační účely této práce bude použito nižší napětí, navržené řešení je zamýšleno pro vyšší napětí.

## 3.3 Blokové schéma funkční bezpečnosti systému

Blokové schéma na Obrázku 26 ilustruje navržené řešení funkční bezpečnosti na úrovni celého systému.

Mikrokontroléry kooperují v konfiguraci master-slave, každý z nich je periodicky testován samočinnými testy odpovídajícími požadavkům třídy B. Červenou barvou jsou zvýrazněny signály, které podléhají funkční bezpečnosti. Každý analogový signál je přiveden zároveň do periferie převodníku *ADC_A* a *ADC_B*. Tyto signály jsou redundantně měřeny mikrokontrolérem slave (kromě měření úrovně napájecího napětí, to je měřeno pouze na straně mikrokontroléru slave). Naměřené hodnoty signálů mikrokontrolérem slave jsou posílány přes rozhraní UART a jsou vzájemně porovnávány na straně mikrokontroléru master, tedy každý signál je měřen čtyřmi nezávislými převodníky. Jako zdroj napětí pro referenční převod slouží interní bandgap reference mikrokontroléru. Vzájemná komunikace přes UART není považována za předmět funkční bezpečnosti, jedná se o „Black channel communication“. V případě selhání komunikace je systémem tento stav automaticky detekován a vyhodnocen jako chybový.

Funkční bezpečnosti nepodléhá část mikrokontroléru master (světlemodrá oblast), ze které vedou příslušné signály (modrá barva). Jde o signál řídicí zažehnutí plamene, diagnostický signál pro vrchní tranzistor a obecně o mechanismy řízení kotle (grafické uživatelské rozhraní, signální LED diody atd.). Detailněji budou jednotlivé mechanismy rozebrány v následujících podkapitolách.

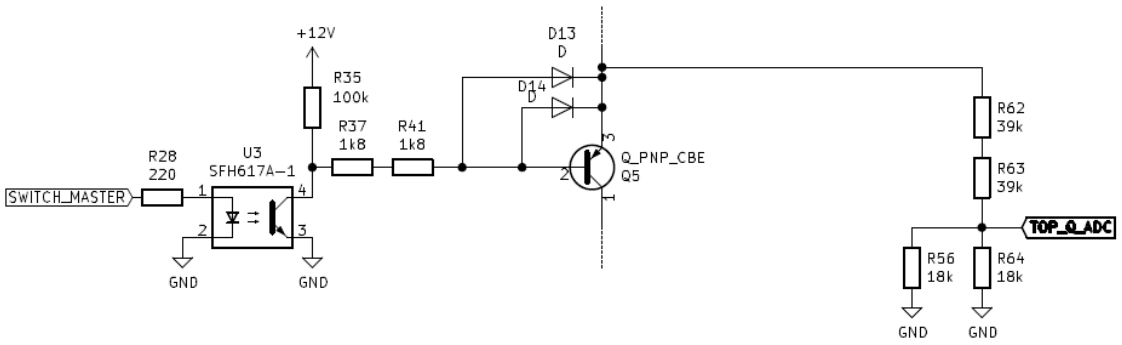



Obrázek 26: Vysokoúrovňový diagram navržených mechanismů funkční bezpečnosti

## 3.4 Spínací větev

Větev pro spínaní byla navržena s ohledem na maximální bezpečnost. Výše uvedené řešení přidává prvky redundance. Při použití jediného tranzistoru, kdyby došlo k trvalému zkratování spínacího tranzistoru, tak by zařízení již nemělo žádnou možnost, jak vypnout ventil a plyn by tak nebylo možné zastavit. Fungování této spínací větve lze připodobnit k logické funkci AND, kdy pro sepnutí ventilu musí být sepnuty všechny tři tranzistory, vrchní, prostřední a spodní, v textu na ně bude odkazováno jako na *Top*, *Mid* a *Bot*. Tranzistor s označením *Relay measure* slouží k diagnostice relátek, ale také přidává další prvek, který může v případě poruchy odpojit plynový ventil.

Oba mikrokontroléry (master i slave) mají možnost nezávisle jeden na druhém v případě poruchy odpojit plynový ventil od napětí a tím jej bezpečně uzavřít. Při použití této konfigurace nelze tranzistor sepnout přímo mikrokontrolérem. Navíc je vhodné oddělit napětí 12 V, které by mohlo poškodit vstup mikrokontroléru. Vhodným řešením je použití optočlenu, který vyřeší problém jak s oddělením, tak se sepnutím vyššího napětí. Na obrázku 27 je schéma spínací části mikrokontroléru. Uvedené schéma je spínací část jen pro mikrokontrolér master, ale pro mikrokontrolér slave i pro blok kontroly napětí jsou totožné.




Obrázek 27: Spínací část mikrokontroléru master

K sepnutí tranzistoru dojde v případě, že je na výstupu mikrokontroléru signál odpovídající úrovni logické 1 (pro mikrokontrolér KV46 3.3 V). Hodnota odporu relátka je 720 Ω. Protože jsou k sobě připojena paralelně, výsledná hodnota jejich odporu bude:

$$R_{load} = \frac{R_{REL1} \cdot R_{REL2}}{R_{REL1} + R_{REL2}} = \frac{720 \cdot 720}{720 + 720} = 360\ \Omega \qquad (1)$$

Při zanedbání saturačního napětí tranzistoru pro diagnostiku relátek bude celkový proud potřebný pro sepnutí relátek:

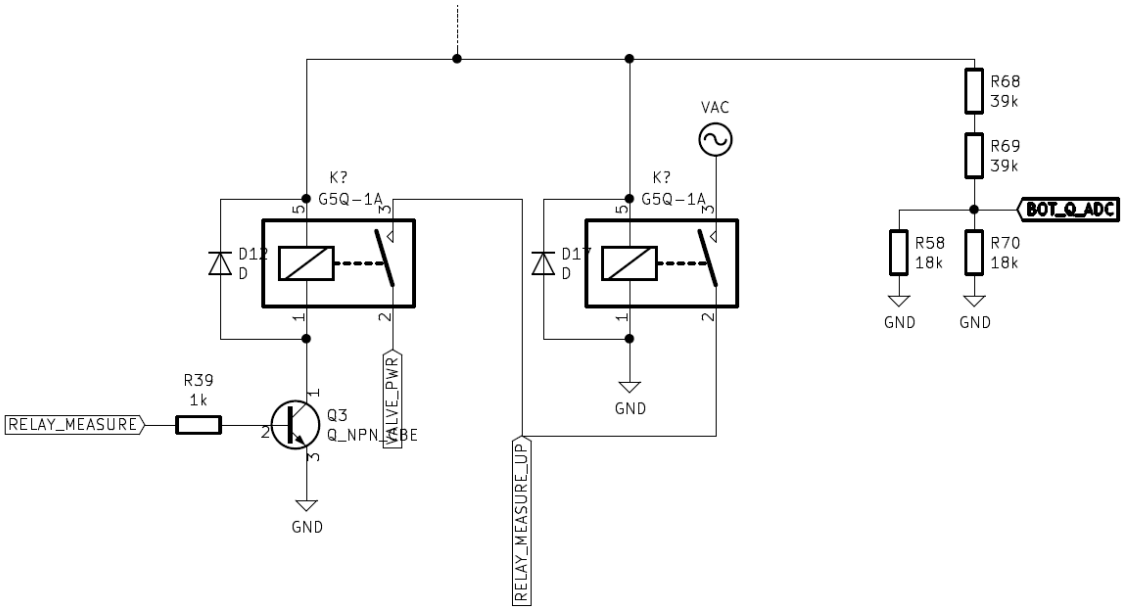
$$I_{load} = \frac{U_{load}}{R_{load}} = \frac{U_{CC} - 3 \cdot \mathrm{U}_{CE(sat)}}{R_{load}} = \frac{12 - 3 \cdot 0.2}{720} = 31.66\ mA \qquad (2)$$

Pro spolehlivé sepnutí PNP tranzistoru je pro výpočet použito pravidlo 1:10, což značí poměr bázového proudu ku proudu kolektorem:

$$I_B = 0.1 \cdot I_C = 0.1 \cdot 31.66 \cdot 10^{-3} = 3.17\ mA \qquad (3)$$

$$R_B = \frac{U_{CC} - U_{BE}}{I_B} = \frac{12 - 0.7}{3.17 \cdot 10^{-3}} = 3599\ \Omega \tag{4}$$

V bázi tranzistoru Q5 je vypočtená hodnota odporu realizována dvěma rezistory, i když z funkčního hlediska by byl dostatečný pouze jeden. Toto zapojení přidává prvek ochrany, kdy při jeho zkratování by hrozilo potenciálně vysoké napětí na přechodu báze-emitor. Dioda mezi bází a emitorem slouží k diagnostickým účelům, protože na dělič přivádí definované napětí. Pro zvýšení bezpečnosti jsou ve spínací větvi dvě paralelně připojená relátka. Jejich výstupy jsou spojeny do série tak, aby byl vždy spolehlivě odpojen plynový ventil od napětí. Je nutné použit typ relátek NO (normally open), což znamená, že jejich kontakt je bez přivedeného napětí rozpojen.



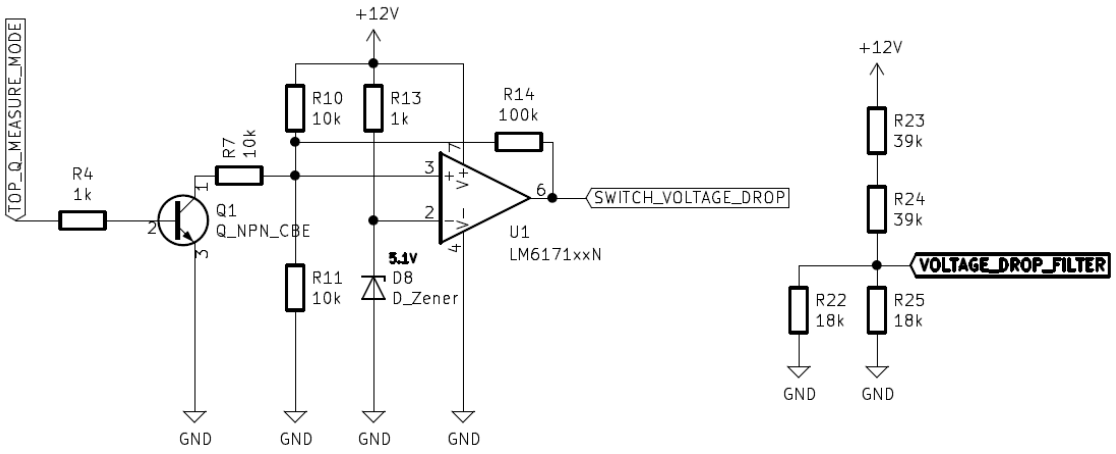

Obrázek 28: Zapojení relátek pro spínaní plynového ventilu

## 3.5 Detekce poklesu napětí

Při poklesu napájecího napětí by některé obvody mohly přestat správně fungovat. Vrchní tranzistor (*Top*) je rozpojen v případě, že napětí poklesne pod definovanou úroveň. Kontrola úrovně napájecího napětí je prováděna pomocí komparátoru. Na neinvertující vstup je přivedena podělená hodnota napájecího napětí. Na invertující vstup je připojena referenční hodnota napětí, v tomto případě Zenerova dioda o napětí 5.1 V. Proud diodou je napočten tak, že i při poklesu napětí udržuje stálou hodnotu referenčního napětí. Hodnoty součástek vychází ze simulace. Mezní přípustná hodnota napájecího napětí je nastavena na 9.4 V. Pomocí rezistoru R14 je zavedena hystereze. Tranzistor Q1 slouží k diagnostickým účelům. Jelikož za běžného provozu je na neinvertujícím vstupu operačního zesilovače vyšší napětí než na invertujícím vstupu, je na výstupu komparátoru hodnota přibližně napájecího napětí, a tedy je *Top* tranzistor sepnutý a není možné jen

rozepnout. Kvůli diagnostice je potřeba mít možnost tento tranzistor sepnout/rozepnout. Sepnutím tranzistoru Q1 je napětí na neinvertujícím vstupu sníženo a tranzistor *Top* ve spínací větvi je rozepnut.

Pro přídavnou kontrolu je napájecí napětí měřeno AD převodníkem mikrokontroléru slave na napěťovém děliči.



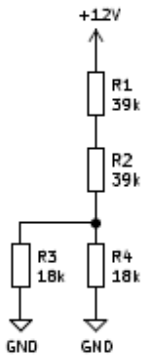

Obrázek 29: Měření poklesu napájecího napětí

Odporové děliče, jejichž odvozené napětí je vstupem pro AD převodník, jsou napájeny z napětí 12 V. Při použití děliče ze dvou rezistorů by se v případě zkratu/rozpojení rezistoru zpropagovalo napětí 12 V na vstup AD převodníku, což by jej zničilo. Dělič je napočítán tak, aby se na vstupu AD převodníku neobjevilo napětí vyšší než 3.3 V. Celková hodnota odporu děliče musí být také dostatečně velká. V případě měření napájecího napětí kvůli spotřebě, v případě měření tranzistorů kvůli minimálnímu zatížení spínací větve. Napětí na děliči je vypočteno dle:

$$V_{R4} = V_{CC} \cdot \frac{R3||R4}{(R_1 + R_2) + R3||R4} \tag{5}$$

$$V_{R4} = 12 \cdot \frac{0.5 \cdot 18 \cdot 10^3}{(39 \cdot 10^3 + 39 \cdot 10^3) + 0.5 \cdot 18 \cdot 10^3}$$

$$V_{R4} = 1.24\,V$$



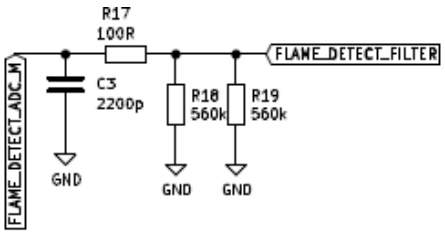

Obrázek 30: Provedení odporového děliče více rezistory

Při použití odporového děliče ze tvou rezistorů jsou nebezpečné dva stavy. Rozpojený spodní rezistor, čímž se do měřeného bodu dostane napájecí napětí (v tomto modifikovaném zapojení je v měřeném bodě napětí 2.25 V). Zkratovaný vrchní tranzistor způsobí propagaci napájecího napětí do měřeného bodu (v tomto modifikovaném zapojení je v měřeném bodě napětí 2.25 V).

## 3.6 Analogové senzory

Analogové senzory hrají důležitou roli z hlediska funkční bezpečnosti zařízení, dávají mikrokontroléru informaci o okolních podmínkách. Jako příklad lze uvést senzor teploty připojený do AD převodníku. Může nastat situace, že se signálový vodič senzoru odpojí od vstupu AD převodníku. Jelikož S/H kapacitor převodníku zůstane nabitý z předešlého převodu, tak je hodnota tohoto zbytkového náboje převedena AD převodníkem. Mikrokontrolér tedy obdrží data, která neodpovídají reálné hodnotě. Například teplota je nad kritickou hranicí, ale převáděná data odpovídají teplotě, která leží pod touto hranicí, nebezpečný stav tedy není detekován.

Před vstup AD převodníku jsou umístěny dva vzájemně paralelní rezistory proti zemi. Hodnota těchto rezistorů musí být dostatečně velká, aby neovlivňovala nabíjecí charakteristiku. V případě odpojení analogového senzoru se na vstupu AD převodníku přes rezistory objeví napětí země. Nulové napětí pak indikuje poruchu, což lze testovat pomocí softwaru. Rezistory jsou zdvojené opět kvůli bezpečnosti. Na Obrázku 31 je ukázka řešení pomocí rezistorů R18 a R19 pro detektor plamene na demonstračním přípravku. Vodič *FLAME_DETECT_ADC_M* je připojen do analogového vstupu mikrokontroléru.




Obrázek 31: Opatření pro detekování odpojení senzoru

## 3.7 Diagnostika

Diagnostika je důležitá ze dvou důvodů. Prvním je mít možnost identifikovat selhávající prvek. Například trvalé zkratování výstupu relátka v tomto zapojení nevede přímo k nebezpečnému stavu (v sérii se nachází další relátko), ale snižuje redundanci. Druhým důvodem je lokalizování vadného prvku kvůli následné opravě.

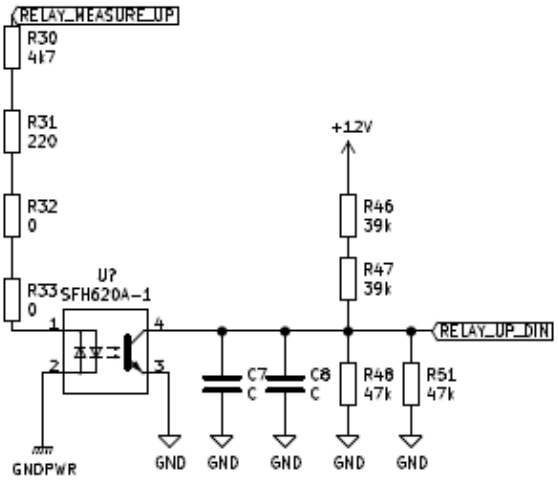
### 3.7.1 Diagnostika tranzistorů

Tato diagnostika slouží k rozpoznání, zdali tranzistor (*Top*, *Mid, Bot*) není trvale zkratován či rozpojen. Pro diagnostiku je v kolektoru všech tří tranzistorů vytvořen měřicí bod pomocí odporového děliče realizovaného pomocí čtyř rezistorů. Logika diagnostiky je taková, že je sepnut tranzistor a je měřeno napětí v jeho kolektoru. Poté je tranzistor rozepnut a opět se měří napětí v jeho kolektoru. V případě očekávaných hodnot napětí se testuje další (Mid) tranzistor. Pro test je nezbytné, aby na emitoru testovaného tranzistoru bylo vždy napětí 12 V. Pro testování prostředního tranzistoru je tedy nutné sepnout vrchní tranzistor. Stejnou logikou je testován i spodní tranzistor. Tímto mechanismem je možné lokalizovat poruchu konkrétního tranzistoru.

### 3.7.2 Diagnostika relátek

Tato diagnostika slouží k rozpoznání, zdali výstup relátka není trvale spojen či trvale rozpojen. Aby bylo možné proměřit obě relátka, je potřebné mít možnost sepnout pouze jedno relátko. Z tohoto důvodu je přidán tranzistor Q3 do série s relátkem K1. Je tedy možné pomocí tranzistoru sepnout jen jedno relátko. Body pro měření se nachází vždy za relátkem směrem k ventilu. Při testu vrchního relátka je nejprve rozepnuto spodní i vrchní relátko a je měřeno napětí. V případě, že je vrchní relátko není trvale spojené, tak v měřicím bodě nesmí být přítomno napětí. Poté je vrchní relátko sepnuto. V případě, že vrchní relátko je trvale rozpojené, není v měřicím bodě naměřeno napětí. Stejnou logikou je testováno i spodní relátko. Tímto mechanismem je možné lokalizovat poruchu konkrétního relátka.

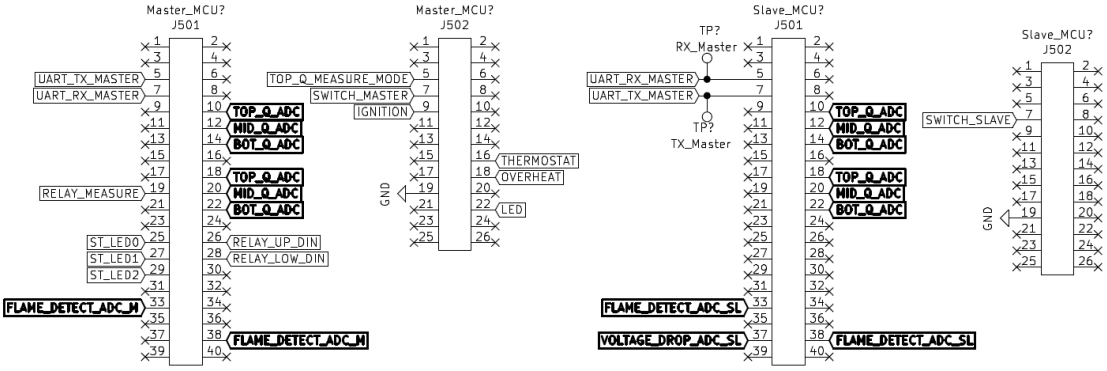
K měření přítomnosti napětí na relátkách slouží následující zapojení. Měření je problematičtější ze dvou důvodů. Jednak se jedná o střídavé napětí, a navíc je potřeba toto vysoké napětí galvanicky oddělit. Měřený bod je přes rezistory připojen ke vstupu optočlenu. Ten je uzpůsoben pro použití se střídavým napětím, jelikož má na vstupu dvě antiparalelní diody. Na výstupu optočlenu je napěťový dělič navržený stejnou logikou jako pro AD vstupy. K děliči jsou připojeny paralelně dva kondenzátory, které drží relativně stabilní hodnotu napětí na děliči a umožňují tak mikrokontroléru pomocí digitálního vstupu určit přítomnost napětí na vstupu optočlenu (plné napětí nebo žádné napětí). Čtyři rezistory v sérii mají význam z hlediska výkonového zatížení při napájení ventilu z vyššího napětí, pro demonstrační účely této práce jsou však nahrazeny nulovými rezistory.




Obrázek 32: Princip měření přítomnosti napětí na relátkách (schéma pro vrchní relátko)

## 3.8 Připojení mikrokontrolérů

Mikrokontroléry jsou umístěny na vývojové platformě tower a do obvodu jsou připojeny pomocí hřebenových konektorů J501 a J502 vývojové platformy. Připojení vstupů a výstupů mikrokontrolérů je na Obrázku 33. Tučným písmem jsou označeny analogové vstupy. V levé části obrázku je zapojení pro master, v pravé části pro slave.




Obrázek 33: Připojení mikrokontrolérů do obvodu

## 3.9 Demonstrační přípravek

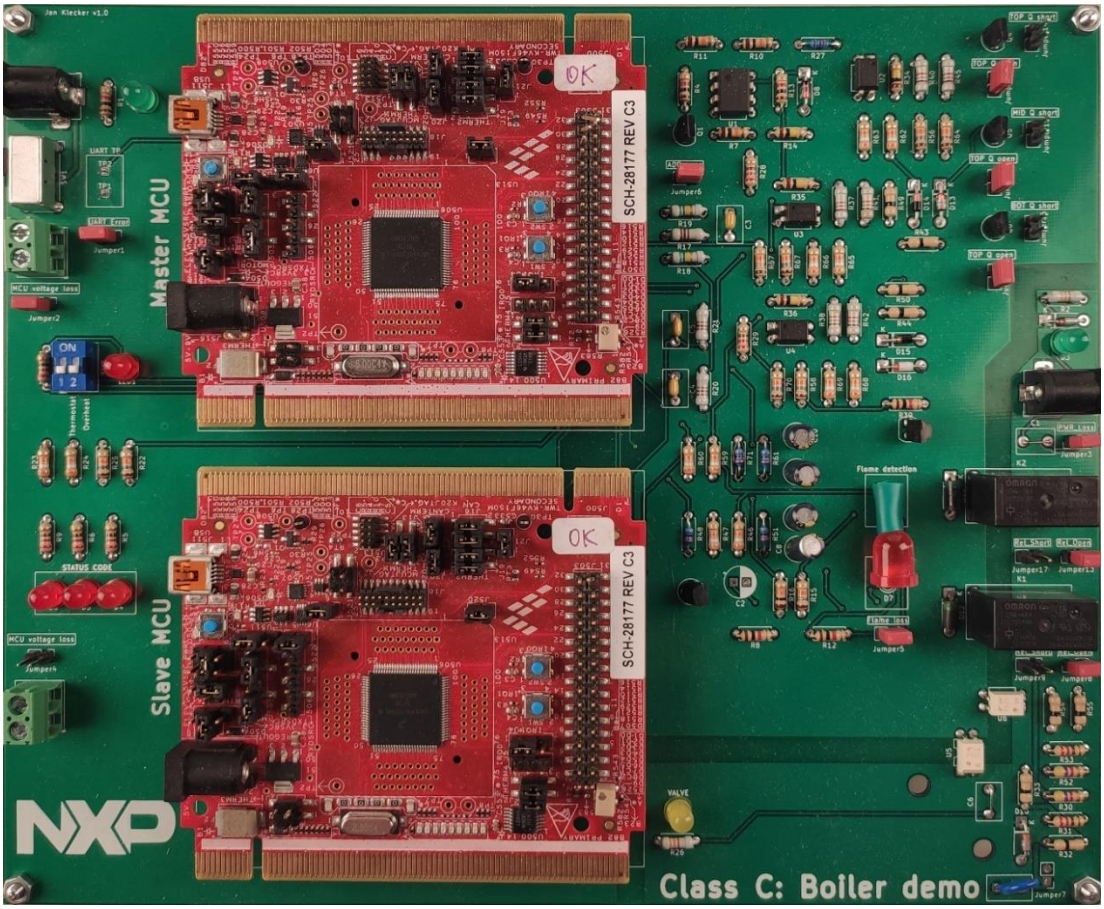
K výše navrženým mechanismům použitelných v reálném zapojení jsou přidány prvky pro imitaci funkcionality plynového kotle. Zažehnutí a následný plamen je simulován pomocí červené LED diody. Měření přítomnosti plamene je simulováno pomocí fotodiody. Pomocí přepínače *Thermostat* je simulován požadavek na zažehnutí plamene. Pomoci přepínače *Overheat* je simulováno přehřátí kotle. 12V část je napájena pomocí adaptéru připojeného do napájecího konektoru levé části desky. Napětí L1 je pro demonstrační účely nahrazeno napětím 24 V AC. Napětí pro ventil je přivedeno z 24V

AC adaptéru přes napájecí konektor v pravé části desky. Pravá část desky oddělená přerušením měděné plochy je oblast pro pomyslné vysoké napětí. Otvory ve spodní části desky slouží k připevnění ventilu. K demonstračním účelům byl vybrán 24V AC ventil pro malé svářečky. K signalizaci stavu ventilu slouží žlutá LED dioda. Deska plošných spojů byla navržena v programu KiCAD.


Obrázek 34: Plynový ventil

Součástí zapojení jsou zkratovací propojky, pomocí kterých lze simulovat různé poruchy:

- Ztráta napájecího napětí mikrokontroléru master nebo slave
- Zkrat/rozpojení tranzistorů ve spínací větvi
- Zkrat/rozpojení relátek
- Ztráta komunikace přes UART
- Zhasnutí plamene
- Odpojení senzoru plamene




Obrázek 35: Osazená DPS demonstračního přípravku

# 4. NÁVRH SOFTWARU

## 4.1 Softwarové nástroje

### 4.1.1 Vývojové prostředí

MCUXpresso je plně vybavené vývojové prostředí pro mikrokontroléry postavené na jádře ARM, obsahující množství podpůrných nástrojů pro vývoj softwaru (Clock Tool, Pin Tool, Peripheral Tool). Vývojové prostředí je založeno na platformě Eclipse a zahrnuje průmyslový standard ARM GNU. Podporuje ladění v jazyce C i v jazyce symbolických adres, k čemuž pomáhají nástroje jako „Memory Monitor“ (zobrazení dat v paměti RAM), „Variable Monitor“ (zobrazení hodnot proměnných) a „Register Monitor“ (zobrazení hodnot uložených v registrech).

**Podpůrné nástroje**
Tyto nástroje integrované do vývojového prostředí slouží k usnadnění vývoje a eliminaci chyb při vývoji softwaru.

**Pin Tool**
Přiřazuje interní signály externím pinům a nastavuje jejich elektrické vlastnosti. Na základě nastavení generuje zdrojový kód v jazyce C.

**Clock Tool**
Slouží pro grafickou reprezentaci distribuce hodinového signálu v mikrokontroléru. Na základě nastavení grafického konfiguračního nástroje je vygenerována funkce pro inicializaci hodinového signálu v mikrokontroléru.

**Peripheral Tool**
Umožňuje nastavit periferie a následně generovat funkce pro jejich inicializaci.

### 4.1.2 Program FreeMASTER

FreeMASTER je softwarový nástroj, který umožňuje ladit aplikaci v reálném čase, což zahrnuje monitorování proměnných, jejich změnu v reálném čase z uživatelského rozhraní běžícího na straně počítače a vizualizaci pomocí časových průběhů monitorovaných proměnných. Program umožňuje tři základní způsoby vizualizace proměnných:

**Variable Watch**
Vybrané proměnné jsou zobrazeny v tabulce v textovém formátu. Hodnoty proměnných lze modifikovat přímo v tomto rozhraní. U každé proměnné lze nastavit formát zobrazení (hexadecimálně, binárně atd.). Umožňuje též vkládání zadávaných hodnot do rovnic tak, že uživatel pouze zadá číslo a program za něj vypočítá hodnotu, která bude uložena do proměnné.

**Oscilloscope**

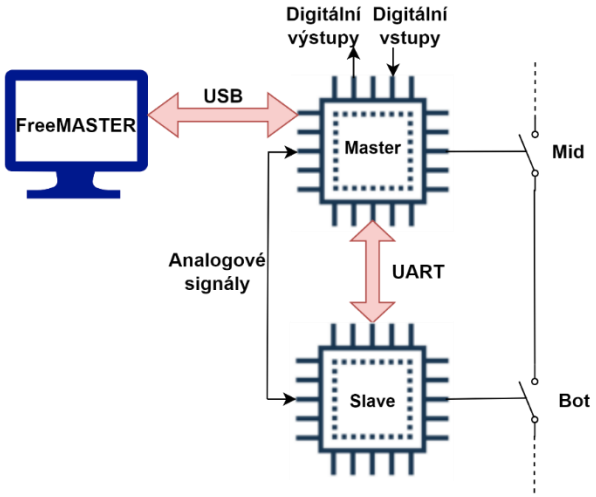
Funkce osciloskopu umožňuje monitorovat proměnné a zobrazovat průběhy v závislosti na čase. Vzorkovací frekvence není garantována a závisí na aktuálním vytížení mikrokontroléru a komunikační rychlosti rozhraní mezi mikrokontrolérem a uživatelským rozhraním běžícím na počítači straně. Tento typ monitorování proměnných je vhodný především pro pomalé děje.

**Recorder**

Tento typ monitorování proměnných odpovídá funkci digitálního osciloskopu, kde perioda vzorkování interních proměnných je přesně definována. Hodnoty proměnných jsou ukládány do interního paměti mikrokontroléru a následně pak přeneseny na stranu počítače.

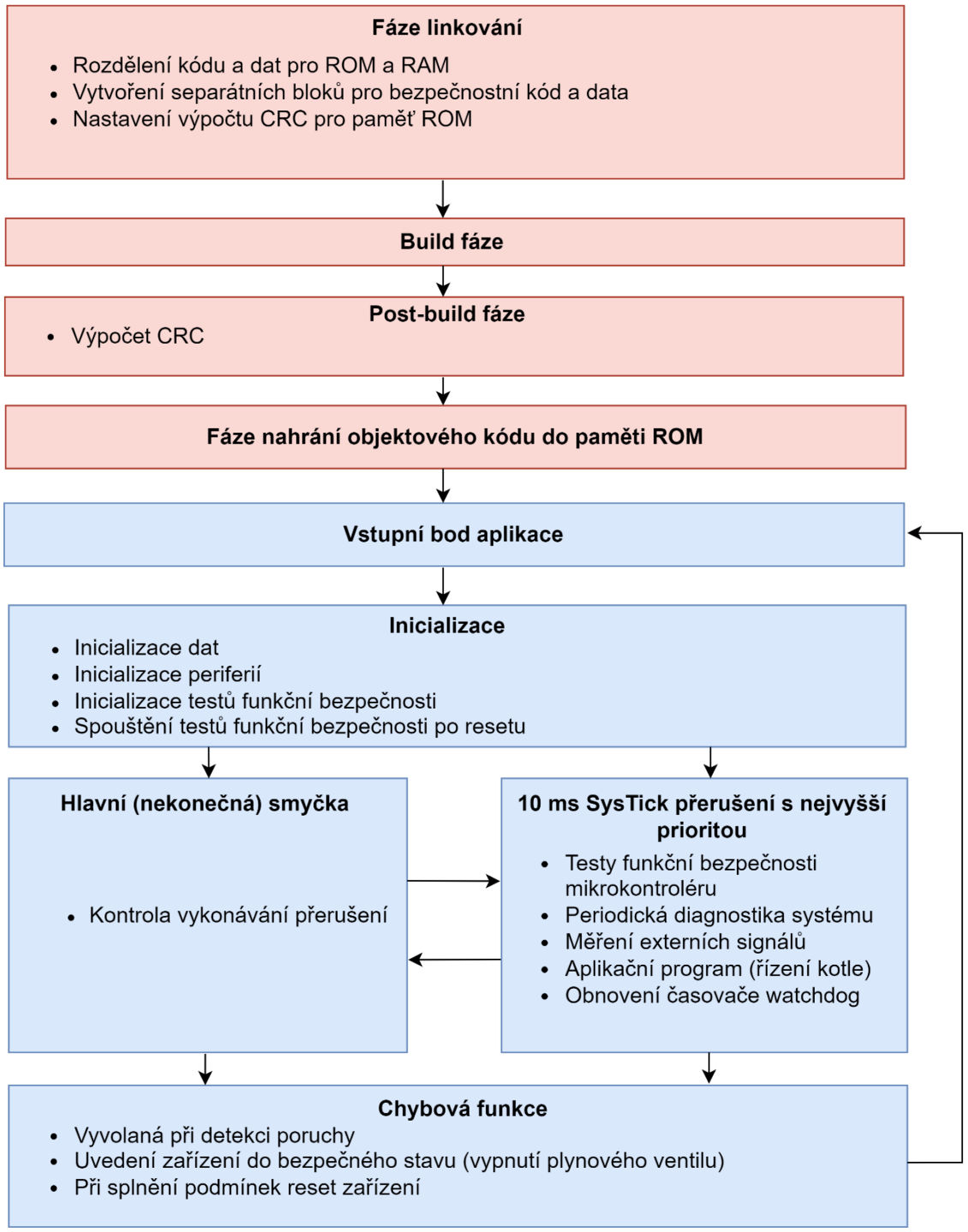
## 4.2 Návrh aplikačního softwaru

Tato podkapitola je zaměřena na návrh aplikačního softwaru pro mikrokontroléry. Jak již bylo zmíněno v hardwarové části, mikrokontroléry fungují v konfiguraci master-slave. Na straně mikrokontrolérů jsou prováděny testy funkční bezpečnosti pro třídu B, které jsou navrženy v samostatné kapitole. Mimo tyto testy jsou integrovány další mechanismy, které zvyšují celkovou bezpečnost systému. Veškerou logiku plynového kotle řídí mikrokontrolér master. Slave redundantně kontroluje (měří) důležité analogové signály. Vzájemně komunikují přes rozhraní UART.




Obrázek 36: Zjednodušené blokové schéma části mikrokontrolérů

Na obrázku 37 je vysokoúrovňový diagram aplikace funkční bezpečnosti. Červená barva odpovídá fázi návrhu, modrá barva odpovídá fázi běhu programu. Tento diagram je platný pro master.




Obrázek 37: Diagram aplikace funkční bezpečnosti pro mikrokontrolér master

Pro řízení plynového kotle byl použit stavový automat. Základem je periodické přerušení vyvolávané časovačem *SysTick* o periodě 10 ms, ve kterém je umístěn kód jak stavového automatu, tak i kód pro testy funkční bezpečnosti. I když master i slave pracují s přerušením o stejné periodě, tyto události jsou vůči sobě asynchronní, laicky řečeno vůči sobě „plavou“. Tento fakt je nutné brát v potaz z hlediska kooperace mikrokontrolérů a návrhu komunikace přes UART. Lze využít mechanismu handshaking,

kdy se vysílající dotazuje přijímajícího, zda může poslat další data (zda už je přijímající zpracoval). Tento mechanismus poměrně zesložiťuje návrh, byl tedy zvolen jiný, vycházející ze znalosti časových poměrů. Vysílající se tedy nedotazuje, zdali může poslat další data, ale po uplynutí určitého časového intervalu automaticky předpokládá, že přijímající data zpracoval a je připraven přijmout další. Master je schopen pomocí zpětnovazebních mechanismů detekovat situaci, kdy slave na příkazy nereaguje nebo je nekorektně interpretuje. Při diagnostice relátek a spínací větve probíhá pouze jednosměrná komunikace, kdy master vysílá příkazy mikrokontroléru slave k sepnutí/rozepnutí *Bot* tranzistoru a následně měří napětí v již zmíněných měřicích bodech. Zde je potřeba měření provést s určitým časovým zpožděním daným propagací dat přes UART, jejich zpracováním na straně mikrokontroléru slave a následným spínacím dějem. Jinými slovy master spoléhá, že po uplynutí určité doby bude tranzistor *Bot* sepnut. Kdyby v této fázi nedošlo k sepnutí tranzistoru *Bot* (mikrokontrolér slave), master by tento stav detekoval AD převodníkem a vyhlásil by jej automaticky jako poruchu. Ve stavech *Init* a Heat již probíhá obousměrná komunikace. Master posláním časové značky spustí převod AD převodníku na straně mikrokontroléru slavu a vyvolá též převod na své straně. Jako odpověď slave posílá paket dat, která obsahují naměřené hodnoty z AD převodníku. Na straně masteru jsou tyto naměřené hodnoty komparovány s naměřenými hodnotami mikrokontroléru master. V případě, že se rovnají v mezích tolerance, byl test úspěšný.

### 4.2.1 Komunikace přes UART

Mechanismus komunikace přes UART byl navržen tak, aby co nejvíce přispíval celkové bezpečnosti. Oba mikrokontroléry mají nastaveno počítadlo, které je vždy při příchodu zprávy přes UART resetováno. V případě, že dojde k selhání komunikace, počítadlo překročí maximální hodnotu, což je indikace poruchy a je vyvolán reset mikrokontroléru, jde tedy o tzv. timeout.

Samotná komunikace je navržena tak, aby nemusela podléhat bezpečnostním testům, z hlediska terminologie norem jde o „Black channel communication“. Systém je navržen tak, že při selhání komunikace dojde automaticky k detekci poruchy pomocí softwaru. Při indikaci poruchy je poslána chybová zpráva přes UART, vypnuta komunikace a je vyvolán softwarový reset funkcí *SoftwareReset*. Ten lze u mikrokontroléru KV46 vyvolat více způsoby. První je zápis do bitu *SYSRESETREQ* registru *AIRCR*. Druhý je simulace uvíznutí v nekonečné smyčce, což detekuje watchdog a resetuje mikrokontrolér. Byla zvolena kombinace obou, tedy zápis do zmíněného bitu následovaný nekonečnou smyčkou. Druhý mikrokontrolér detekuje chybu buď tím, že do definovaného časového intervalu neobdrží data (timeout) nebo přijetím chybové zprávy přes UART. To je indikace poruchy a zároveň požadavek na reset druhého mikrokontroléru. Tímto mechanismem jsou tedy oba mikrokontroléry schopny se vzájemně vyresetovat a uvést systém do výchozích podmínek.

Po resetu začne běžet časová perioda, která slouží k synchronizaci a navázání komunikace mezi mikrokontrololéry. Navíc může sloužit jako „ochranná“ perioda, než odezní vlivy, které způsobily reset (nárůst teploty, rušení, glitch). Tento časový interval je důležitý, jelikož mezi resety mikrokontrolérů je vždy nenulový časový interval, který je způsoben zpožděním při detekování chyby. Po resetu naváže slave komunikaci přes UART zprávou *UART_CMD_INIT_READY*. Na tuto zprávu master odpoví opět stejným příkazem. Tímto si mikrokontroléry křížově zkontrolují, že se nachází v *Init* stavu. Při nenavázání komunikace v daném časovém intervalu dojde opět k resetu obou mikrokontrolérů.

Například krátkodobé rušení nebo náraz ionizující částice mohou způsobit přechodnou chybu. Zdroj chyby však po uplynutí krátkého časového intervalu odezní a systém má předpoklad opět fungovat korektně. Uvedení systému do trvale bezpečného stavu po detekci první chyby by učinilo zařízení nespolehlivé. Byla tedy vytvořena proměnná, která ukládá celkový počet resetů zařízení. Zde je nutné umístit proměnnou do paměťového prostoru, který není startup funkcí přepsaný. Po detekci Power-on-Resetu je nutné tuto proměnou vynulovat. Při detekci poruchy se tedy mikrokontroléry resetují, a až při překročení maximálního počtu resetů je systém uveden do trvalého bezpečného stavu. V tomto stavu je zapotřebí zavolat servis, který provede následné testovaní celého systému (Proof Test) a odstranění závady.

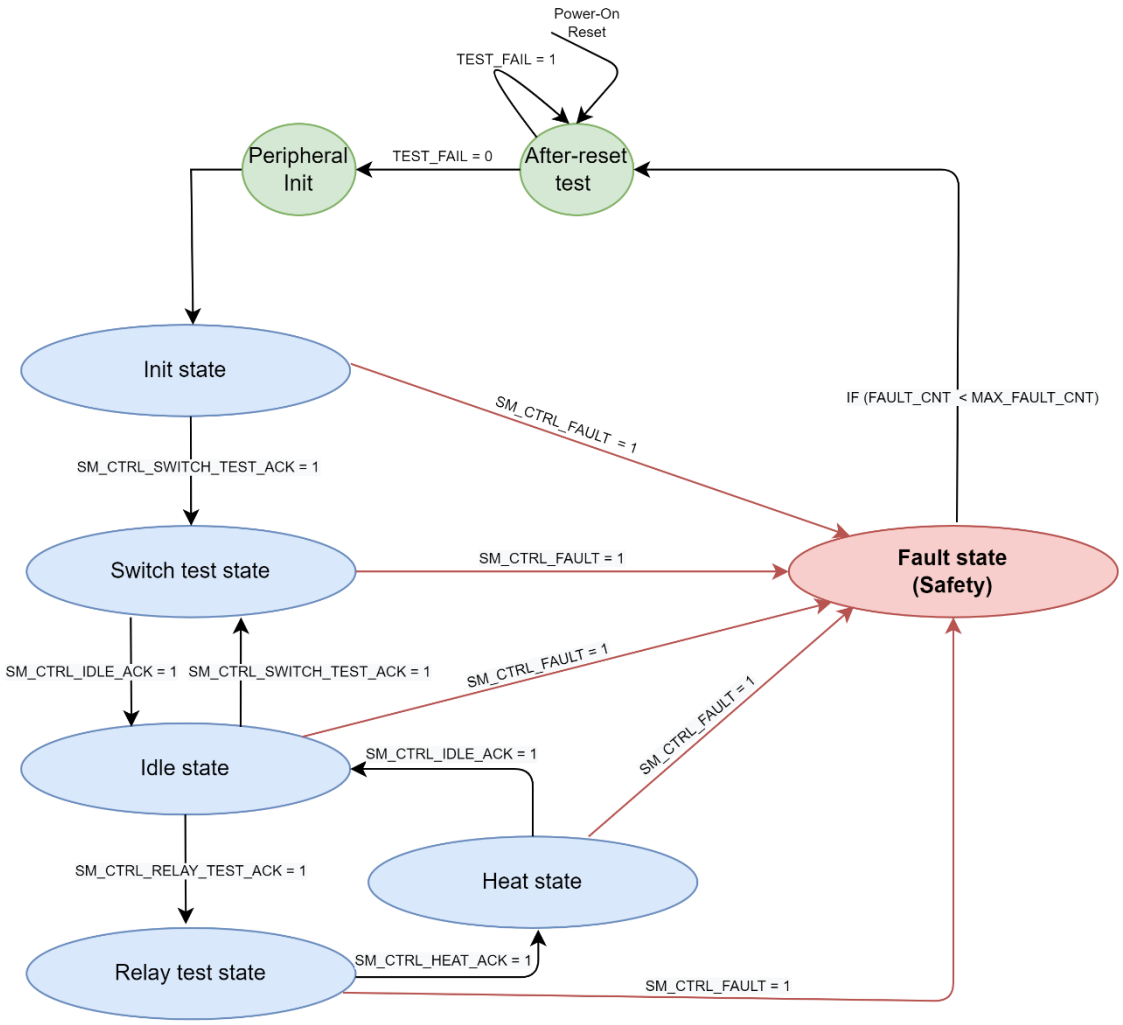
Na obrázku níže jsou průběhy komunikace mezi mikrokontroléry ve stavech *Idle* a *Heat*. Průběh *SysTick ISR* odpovídá vyvolání periodického přerušení časovače SysTick v mikrokontroléru master. V průběhu *Master TX* lze vidět poslanou časovou značku mikrokontrolérem master. V průběhu *Slave TX* lze vidět poslání paketu (naměřených hodnot) mikrokontrolérem slave jako odpověď na časovou značku poslanou mikrokontrolérem master.


Obrázek 38: Průběh komunikace přes UART

### 4.2.2 Mikrokontrolér master

Z doporučení normy byla logika aplikačního programu byla navržena pomocí stavového automatu o šesti stavech. Pro každý stav je vytvořena funkce (tedy celkem šest funkcí), ze které jsou volány dílčí funkce daného stavu (vyčítání výsledků AD převodu, komparace výsledků). Hlavní funkcí je *SM_StateMachine*. Je vytvořeno pole ukazatelů

na jednotlivé funkce, které jsou následně volány pomocí indexu pole, které odpovídají stavům automatu, volány jednotlivé funkce.




Obrázek 39: Stavový automat pro mikrokontrolér master

Vytvořené pole ukazatelů na funkce bez návratové hodnoty a bez vstupních paramterů:

```
/* State machine function table */
const PFNC_VOID_VOID SM_STATE_TABLE[6];
```

```
/* State machine control structure */
typedef struct
{
    SM_STATE_T    eState;                /* Current state */
    SM_APP_CTRL   uiCtrl;                /* Control flags */
    uint16_t      ISRWaitCnt;            /* Delay */
    uint16_t      TransientDelayCnt;     /* Delay */
    uint16_t      SMAllFaults;           /* Fault bits */
} SM_APP_CTRL_T;
```

Stavový automat je řízen strukturou s*AppCtrl*. Proměnná *eState* slouží k uchování současného stavu. Proměnná *uiCtrl* slouží k řízení stavového automatu pomocí příkazů.

Například zápis masky *SM_CTRL_FAULT* do proměnné *uiCtrl* značí poruchu systému. Na základě hodnoty *uiCtrl* se vyhodnocuje přechod do jiného stavu, v tomto příkladě do stavu *Fault*. Každému přechodu mezi stavy náleží přechodová funkce, která slouží k nastavení proměnných a výstupů pro následující stav. Proměnné *ISRWaitCnt* a *TransientDelayCnt* slouží pro řízení zpoždění přechodu do jiného stavu a jako neblokující čekací smyčka.

Pro identifikaci poruch slouží dvě proměnné. První je *SMAllFaults*, která souvisí s poruchami detekovanými stavovým automatem. Druhou je *AllFaults*, která indikuje detekované poruchy ze samočinných testů, tato proměnná bude přiblížena v kapitole zabývající se funkčními testy. Při nenulové hodnotě těchto proměnných stavový automat přechází do *Fault* stavu. Jde o chybový stav a jeho chování musí být jasně definováno. V této konkrétní aplikaci jde o stav, kdy je plynový ventil odpojen od napájecího napětí a přívod plynu je tak zastaven. Přechodové funkce, které směřují do tohoto stavu, tedy musí spolehlivě vypnout tranzistor *Top* a zároveň poslat přes UART chybový kód mikrokontroléru slave, na jehož základě je odpojen i tranzistor *Bot*.

Pro každou poruchu je vytvořena bitová maska, aby bylo možné konkrétní poruchu identifikovat. Nenulová hodnota této proměnné značí poruchu.

Tabulka 2: Význam jednotlivých bitů proměnné *SMAllFaults*

| **Bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
|---|---|---|---|---|---|---|---|---|
| **Význam** | - | DATA COMPARE | OVERHEAT | BOT TR VAL | MID TR VAL | TOP TR VAL | RELAY DOWN VAL | RELAY UP VAL |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Význam** | BOT TR TEST | MID TR TEST | TOP TR TEST | RELAY DOWN TEST | RELAY UP TEST | FLAME | INIT | UART |

**Init stav**

Výchozí stav po resetu. Inicializace parametrů a čekání na navázání komunikace se slavem.

**Switch test stav**

V tomto stavu je diagnostikována spínací větev. Mechanismus byl již popsán v hardwarové části. Při testování tranzistoru *Bot* je nutné tento tranzistor sepnout. Master musí vyslat přes UART příkaz pro sepnutí tranzistoru. Následně musí počkat určitý časový interval, než začne měřit (asynchronnost obou mikrokontrolérů, čas sepnutí tranzistoru). Perioda diagnostiky, tedy jak často bude stavový automat přecházet do tohoto stavu, je nastavitelná (násobky periody přerušení *SysTick*). Spínací větev je možné diagnostikovat bez sepnutí ventilu.

**Idle stav**

Stav nečinnosti. Jsou kontrolována napětí na všech tranzistorech a na ventilu. Jsou vysílány požadavky přes UART na měření a následně jsou obdržená data od mikrokontroléru slave komparována. Z tohoto stavu je samovolně přecházeno (po uplynutí nastavené periody) do stavu diagnostiky spínací větve. Otestovat samotnou spínací větev bez sepnutí ventilu je umožněno díky tranzistoru, který odpojí spodní relátko, při testování je sepnuto tedy jen vrchní. Perioda testování by měla být zvolena s ohledem na opotřebovávání relátka testováním.

**Relay test stav**

V tomto stavu jsou diagnostikována obě relátka. Při požadavku na spuštění kotle přejde stavový automat do tohoto stavu, aby ověřil správné fungování relátek. Při úspěšné diagnostice stavový automat přechází do stavu *Heat*. Tento test se záměrně provádí pouze při přechodu do stavu *Heat*. Při testu jsou spíná/vypínána relátka. Příliš časté testování by znamenalo zbytečné opotřebení těchto spínacích prvků.

**Heat stav**

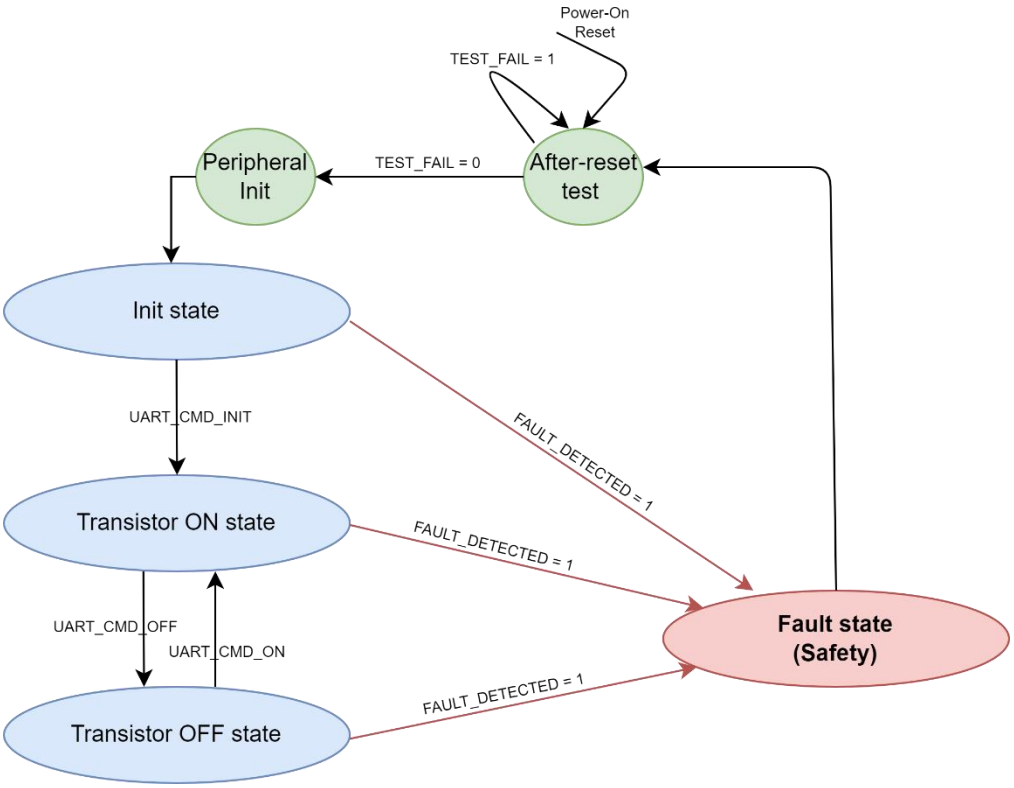
V tomto stavu je ventil sepnut a plyn tak proudí do spalovací komory (kotel topí). Jsou kontrolována napětí na všech tranzistorech a na ventilu. Při požadavku na vypnutí topení stavový automat přechází do stavu *Idle*.

**Fault stav**

Chybový stav, je poslána chybová zpráva a vypnuta komunikace. Plynový ventil je odpojen od napětí a plyn tak neproudí. V tomto stavu musí dojit k odpojení obou relátek, takže signály, které řídí relátka, musí být deaktivovány. V případě, že není překročen maximální počet resetů, tak se mikrokontroléry resetují.

### 4.2.3 Mikrokontrolér slave

Pro návrh softwaru pro slave byla snaha o softwarovu diverzitu. Nabízí se možnost s drobnými modifikacemi přenést kód z masteru na slave. V tomto případě však hrozí přenesení systematických chyb. Například chyba plynoucí ze špatně navržené logiky stavového automatu masteru se přenese na slave, čímž se sníží celková bezpečnost. Při návrhu stavového automatu pro slave bylo tedy využito jiné logiky pro volání dílčích funkcí stavového automatu a byly použity odlišné konstrukce jednotlivých funkcí. Slave se řídí pomocí příkazů, které obdrží přes UART.




Obrázek 40: Stavový automat pro mikrokontrolér slave

**Init stav**

Výchozí stav po resetu. Inicializace parametrů a čekání na navázání komunikace přes UART.

**Transistor ON stav**

Při obdržení příkazu od mikrokontroléru master pro sepnutí tranzistoru stavový automat přechází do tohoto stavu. V tomto stavu je tranzistor *Bot* sepnutý.

**Transistor OFF stav**

Při obdržení příkazu od mikrokontroléru master pro rozpojení tranzistoru stavový automat přechází do tohoto stavu. V tomto stavu je tranzistor *Bot* rozepnutý.

**Fault stav**

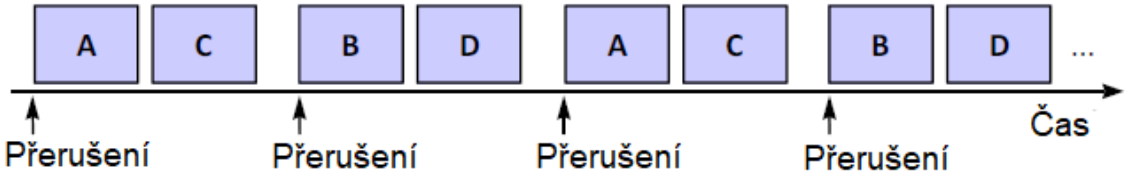
Do tohoto stavu se slave dostane buď při detekování poruchy, nebo při obdržení chybové zprávy. V tomto stavu musí dojit k odpojení obou relátek, takže signály, které řídí relátka, musí být deaktivovány. V případě, že není překročen maximální počet resetů, tak se mikrokontroléry resetují.

## 4.3 Softwarové testy

V následujících podkapitolách je uveden návrh dílčích softwarových testů. Mikrokontrolér KV46 je podporován knihovnou s funkčními testy. V opodstatněných případech bylo využito knihovních funkcí (testy jádra). V ostatních případech bylo nutné navrhnout vlastní řešení, popřípadě testy modifikovat.

Některé testy jsou nepřerušitelné z podstaty testu. Nepřerušitelné testy jsou vykonávány v periodických událostech s nejvyšší prioritou. Nabízí se řešení volat tyto funkce z jiných míst (přerušení s nižší prioritou, nekonečná smyčka) a po dobu testu globálně zakázat přerušení, čímž je zajištěna integrita testu. Tento přístup však není příliš vhodný, proto byla zvolena varianta provádění všech testů v přerušení *SYSTICK_Isr*. Z časového důvodu nelze v přerušení volat všechny funkce naráz, ale je nutné vytvořit plánovač. Ten umožňuje deterministické volání funkcí [20]. Obecný princip plánovače lze vidět na Obrázku 41, funkce jsou volány postupně. Plánovač je navržen jednoduchou switch-case konstrukcí, v níž jsou funkce volány podle hodnoty proměnné *sSafetyCommon.SafetyTestSwitch*.

Pro níže navržené testy. Norma IEC 60730 vyžaduje, aby byly všechny testy vykonány do poloviny MTTF, tedy do poloviny střední doby do poruchy.




Obrázek 41: Princip plánovače [20]

V společné struktuře pro všechny testy *sSafetyCommon* se nachází proměnná *AllFaults*. Její nenulová hodnota indikuje nastalou poruchu. Každá porucha má svou bitovou masku (kromě některých testů CPU registrů, které v případě poruchy zůstanou v nekonečné smyčce). Identifikace poruchy má význam z diagnostického hlediska.

Tabulka 3: Bity proměnné *AllFaults* pro detekci poruchy

| **Bit** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
|---|---|---|---|---|---|---|---|---|
| **Význam** | - | - | P. FLOW AND ISR | AIO TEST | CPU SPECIAL | CPU CONTROL | CPU FLOAT 2 | CPU FLOAT 1 |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Význam** | CPU NONSTA-CKED | CPU REG | CPU PRIMASK | PC TEST | CLOCK TEST | STACK TEST | RAM TEST | FLASH TEST |

### 4.3.1 Test volatilní a nevolatilní paměti

Pro správné fungování aplikace funkční bezpečnosti je nezbytné nastavit linker. Výstupem kompilátoru jsou soubory obsahující objektový kód. Linker tyto soubory sloučí do jednoho spustitelného souboru a všem symbolům přiřadí adresu v paměti.

Všechny potřebné informace pro linker jsou uloženy v tzv. „linker command file“. Je vhodné oddělit kód spojený s funkční bezpečností od kódu, který není z hlediska bezpečnosti kritický.

Pouze ta část paměti, která obsahuje data spojena s funkční bezpečností, musí být otestována. Při testování všech dat (i těch, která nejsou z hlediska bezpečnosti kritická) narůstá časová náročnost testu. V mikrokontroléru KV46 je nevolatilní paměť reprezentována pamětí typu flash, volatilní paměť je reprezentována pamětí typu SRAM. Nejprve je nutné definovat hranice (myšleno počáteční a koncové adresy) těchto pamětí.

```
/* FLASH memory boundaries. */
__ROM_start__       = 0x00000000;
__ROM_end__         = 0x0003FFFF;

/* RAM memory boundaries. */
__RAM_start__       = 0x1FFFC000;
__RAM_end__         = 0x20003FFF;
```

Pomocí skriptovacího jazyka „linker command language“ jsou nadefinovány velikosti objektů v paměti, například:

```
/* Sizes of objects in RAM. */
...
ram_test_backup_size  = 0x40;    /* Size of backup for RAM test. */
...
/* Sizes of objects in FLASH. */
...
__vector_table_size__ = 0x400;
...
```

Prostřednictvím výše zmíněného skriptovacího jazyka jsou do jednotlivých sekcí pamětí SRAM a flash rozmístěna odpovídající data.

#### 4.3.1.1 Test paměti SRAM

Test paměti SRAM je z časového hlediska náročný. Paměť není otestována naráz, ale je testována po blocích definované velikosti. Celá paměť je tak otestována až po určitém časovém intervalu. Velikost testovaného bloku musí být vhodně zvolena vzhledem k cílové aplikaci. Příliš malý testovací blok znamená dlouhý časový interval, za který je celá paměť otestována. Naopak velký testovací blok otestuje celou paměť za kratší časový interval, ale omezuje výpočetní výkon mikrokontroléru.

**Test za běhu: March C**

Pro testy za běhu programu je nutné, aby co nejméně vytěžovaly výpočetní výkon mikrokontroléru. V aplikacích běžících v reálném čase, kde se například paměť testuje v přerušení s periodou 100 µs, je čas testu paměti kritický, proto bylo nutné nejprve změřit výpočetní čas knihovních funkcí. I když se minimální a maximální čas výpočtu liší, rozhodující je ten maximální, i kdyby na 100 testů vykonaných v minimálním čase připadal 1 test vykonaný v maximálním čase. Při maximálním čase se například nemusí stihnout vypočítat data potřebná pro regulační soustavu, což může vést ke zhroucení systému. Výpočetní časy pro March C jsou oproti March X testu větší, což vychází z větší

komplexnosti testu, viz teoretický úvod. Výpočetní čas testu lze ovlivnit velikostí testovaného bloku. Uvedené časy jsou měřeny při frekvenci jádra 95.7 MHz.

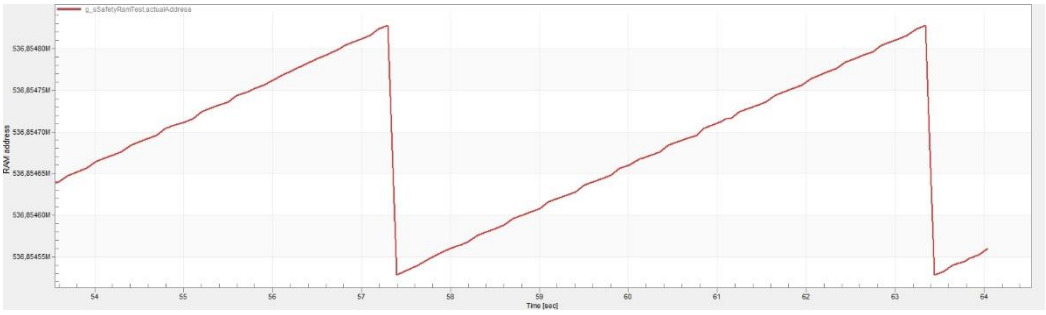
Tabulka 4: Měření výpočetního času funkcí pro March X a C test

| | March X | | | | March C | | | |
|---|---|---|---|---|---|---|---|---|
| **Velikost [byte]** | **Min [cykly]** | **Max [cykly]** | **Min. čas [µs]** | **Max. čas [µs]** | **Min [cykly]** | **Max [cykly]** | **Min. čas [µs]** | **Max. čas [µs]** |
| **0x4** | 216 | 272 | 2,25 | 2.83 | 241 | 294 | 2,51 | 3.06 |
| **0x10** | 310 | 414 | 3,23 | 4.31 | 388 | 489 | 4,04 | 5.09 |
| **0x20** | 367 | 602 | 3,82 | 6.27 | 424 | 729 | 4,42 | 7.60 |

Jelikož v této konkrétní aplikaci je voláno přerušení s nejvyšší prioritou s periodou 10 ms, není požadavek na výpočetní čas funkce tak kritický jako u jiných aplikací. Čas $T$, za který se otestuje testovaná paměť, lze spočítat dle následujícího vztahu:

$$T = \frac{velikost\ testované\ paměti\ [B]}{velikost\ testovaného\ bloku\ [B]} \cdot perioda\ testování\ [s] \qquad (6)$$

Kde *perioda testování* je časový interval, ve kterém je volána funkce pro testování bloku paměti. Na obrázku 42 je ukázán průběh proměnné v programu FreeMASTER, která odpovídá aktuálně testované adrese v paměti RAM při testu March C. Sestupné hrany průběhu odpovídají časovému okamžiku, kdy je otestována celá paměť, tedy kdy je test dokončen.




Obrázek 42: Průběh March C testu

**Test po startu: Galpat test**

Tento test je z hlediska výpočetních kroků velice náročný, ve své podstatě je pro reálné aplikace nepoužitelný. Algoritmus byl proto modifikován tak, že se testu nepodrobuje celá paměť, ale vždy jen její omezená oblast. Tím se sníží komplexnost testu, ale je možné jej vykonat v kratším čase. I přes tuto modifikaci je test stále časově náročný, je proto vhodný pro testování pouze po startu programu, nikoliv za jeho běhu. Logika testování je taková, že se daná paměťová oblast vyplní hodnotou logické 1 nebo 0. Poté se do základní buňky (Base cell) zapíše invertovaná hodnota. Následně jsou všechny ostatní buňky

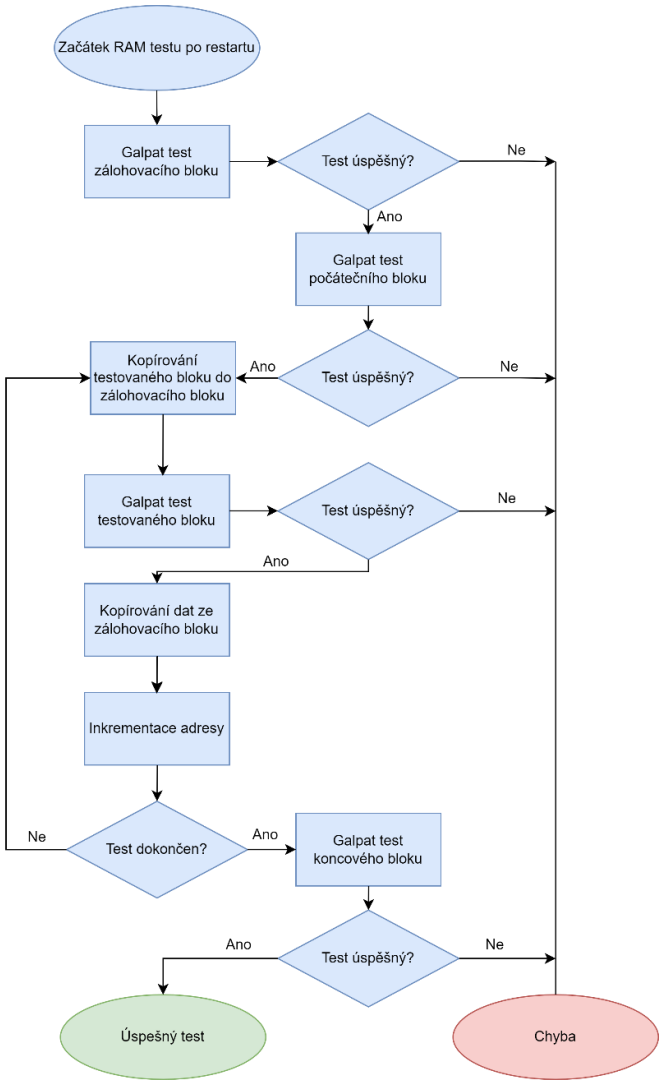
v dané oblasti (Other cell) čteny. Tento test odhalí kromě stuck-at poruch také případ, kdy zápis do paměťové buňky ovlivní i okolní paměťové buňky.

**BC:** Base cell

**OC:** Other cell

**N:** Počet buněk

1. Zápis 0 do všech buněk
2. Pro BC = 0 do N - 1
    a. Negace BC
        i. Pro OC = 0 do N – 1; OC != BC
            - Kontrola (čtení) BC
            - Kontrola (čtení) OC
    b. Negace BC
3. Zápis 1 do všech buněk
4. Zopakování kroku 2




Obrázek 43: Algoritmus Galpat testu

Tabulka 5: Měření výpočetního času pro Galpat test

| Velikost bloku [4B slovo] | Max [cykly] | Max. čas [ms] |
|---|---|---|
| 0xD | 337018 | 3.51 |
| 0x7 | 182383 | 1.90 |
| 0x5 | 130838 | 1.36 |

V tabulce 5 je uveden čas pro testování bloku o různé velikosti (počet 4bytových slov). I pro malý blok paměti (0x5) je výpočetní čas testu 1.36 ms, což je nepoužitelné pro testování za běhu programu. Po startu programu (a tedy celého zařízení) u většiny aplikací je možné „malé zpoždění“ způsobené samočinným testováním. Byla zvolena koncepce důkladnějšího Galpat testu paměti po startu programu a rychlejšího March C testu za běhu programu.

#### 4.3.1.2 Test paměti flash

Pro testování paměti flash byla zvolena metoda pomocí CRC (anglicky Cyclic Redundancy Check). Ve fázi linkování je spočítán CRC pro definovanou část paměti a je uložen do paměti flash. Zde je nutné, aby byl CRC uložen do jiné oblasti paměti, než pro jakou je počítán. Jelikož vývojové prostředí MCUXpresso neumožňuje přímo generovat CRC, je nutné použít externí nástroj, v tomto případě SRecord. Z důvodu, že mikrokontrolér KV46 disponuje periferií pro výpočet CRC, byla hardwarová metoda počítání upřednostněna před softwarovou metodou. Hardwarové počítání je časově rychlejší a méně náročné na software. Funkce *FS_CM4_CM7_FLASH_HW16* je periodicky volána v přerušení. Jelikož testovaná část paměti je poměrně velká, je CRC počítán po blocích, výsledek je tedy k dispozici až po *N* volání funkce. Po poslední iteraci je výsledek porovnán s CRC vypočteném ve fázi linkování. V případě, že se shodují, byl test úspěšný.

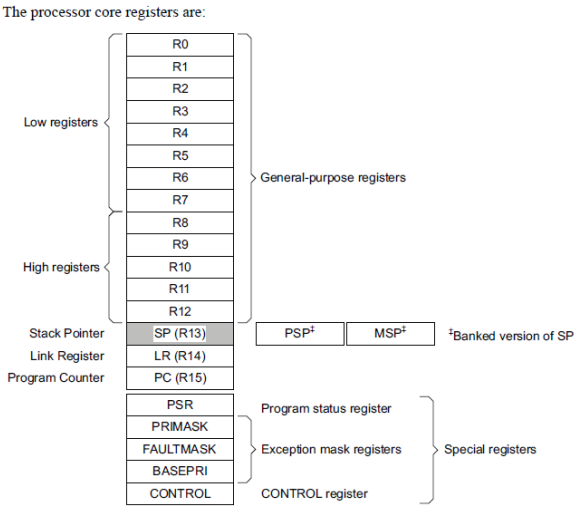
```
#
#---------------------------------
#      CRC CALCULATOR V1.3
#---------------------------------
#
# Getting CRC information:
# -------------------------------
# CRC table found at: 0x0003FFF0
# Safety FLASH start: 0x00000000
# Safety FLASH end:   0x00001D74
#
# Calculating new CRC value:
# -------------------------------
# CRC-16/AUG-CCITT value: 0x3CC0
#
# Saving new CRC value:
# -------------------------------
# CRC written at 0x0003FFFA
#
```

Obrázek 44: Výpis z konzole prostředí MCUXpresso po výpočtu CRC

### 4.3.2 Test CPU registrů

**Jádro ARM Cortex-M4 obsahuje tyto registry [25]:**

- Třináct General-Purpose registrů pro operace s daty.
- Dva Stack Pointer registry
  - Main Stack Pointer (MSP) registr
  - Process Stack Pointer (PSP) registr
- Jeden Link Registr (LR), který ukládá návratové informace pro subrutiny, volané funkce a přerušení.
- Jeden 32bitový Program Counter (PC) registr, který ukládá vykonávanou adresu programu.
- Tři Program Status Registry (PSR).
  - Application Program Status Registr (APSR)
  - Interrupt Program Status Registr (IPSR)
  - Execution Program Status Registr (EPSR)
- Jeden Priority Mask Registr (PRIMASK), který může zakázat všechna přerušení s konfigurovatelnou prioritou.
- Jeden Fault Mask registr (FAULTMASK), který může zakázat všechna přerušení kromě nemaskovatelných přerušení.
- Jeden Base Priority Mask registr (BASEPRI), který definuje minimální prioritu pro přerušení. Mikrokontrolér tedy nevykoná přerušení s větší nebo rovnou úrovní přerušení, než je hodnota v tomto registru.
- Jeden Control registr (Control), který řídí zásobník a úroveň oprávnění pro software.
- Třicet dva Single-precision extension registrů (Jádro Cortex-M4 obsahuje jednotku FPU).

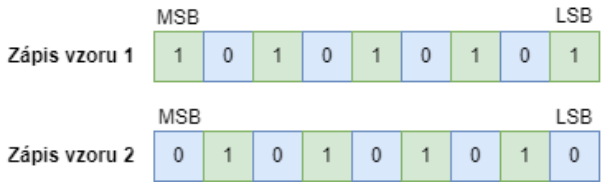



Obrázek 45: Registry jádra ARM Cortex-M4 [25]

Tento test vyžaduje zápis do konkrétního registru mikrokontroléru, proto musí být funkce napsány v jazyce symbolických adres. Jak již bylo zmíněno v teoretickém úvodu, při testování registrů je nutné brát ohled na to, jestli je potřeba při volání funkce testovaný registr zálohovat. V případě mikrokontrolérů ARM je nutné postupovat v souladu s dokumentem *Procedure Call Standard for the ARM Architecture*, který mimo jiné popisuje konvence pro volání funkce. Registry R4-R11 je dle dokumentu nutné zálohovat. Registry R0-R3 slouží jako vstupní parametry pro funkci a není potřeba je zálohovat.

Pro testování registrů je využit algoritmus šachovnice, který byl popsán v teoretické části. Funkce *SafetyCpuRegisterTest(FS_RESULT *pAllFaults)* otestuje CPU registry za běhu. V samotné funkci jsou volány subrutiny, které testují jednotlivé seskupení registrů. V úryvku kódu je ukázka volání subrutiny pro testování.

```
/* This function tests the R0-R7, R12, LR, and APSR CPU registers in
a sequence. */
TestResultRegister = FS_CM4_CM7_CPU_Register();
if (TestResultRegister > 0)
{
    *pAllFaults |= CPU_REGISTERS_FAULT;
}
```




Obrázek 46: Vzory pro testování CPU registrů

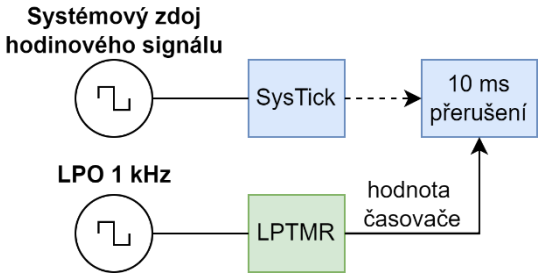
### 4.3.3 Test programového čítače

V teoretické části byla vysvětlena nevhodnost testování programového čítače pomocí skoků do definovaných paměťových prostorů. Správné fungování programového čítače je zajištěno logickým monitorováním korektního vykonávání programu v návaznosti na časovač watchdog.

### 4.3.4 Test hodinového signálu

Jelikož mikrokontrolér KV46 nedisponuje časovačem/čítačem s funkcí zachycení (anglicky capture), tak byla zvolena metoda testování pomocí nezávislého časovače LPTMR (Low-power timer). Nezávislost časovače znamená, že je zdroj jeho hodinového signálu jiný než systémový zdroj hodinového signálu. Pro LPTRM je potřeba nastavit jako zdroj hodinového signálu LPO (Low power oscilátor) o frekvenci 1 kHz. V případě použití stejných zdrojů hodinového signálu by byl poměr frekvencí stále stejný. V přerušení *SYSTICK_Isr* je vyčítána hodnota časovače a porovnávána s mezemi tolerance (ze znalosti periody přerušení lze vypočítat očekávanou hodnotu časovače).

Následně je časovač resetován a inkrementuje se opět do dalšího přerušení, kdy je jeho hodnota opět čtena.




Obrázek 47: Test hodinového signálu s prostředky mikrokontroléru KV46

### 4.3.5 Test přerušení

Tato podkapitola je záměrně hned za podkapitolou týkající se testu hodinového signálu. Příliš časté nebo málo časté vyvolání přerušení znamená, že při kontrole hodnoty LPTMR časovače je čtena nižší/vyšší hodnota než očekávaná. Norma též vyžaduje kontrolu, zdali vůbec k přerušení dochází, což samotný tento mechanismus neodhalí. V hlavní smyčce je kontrolován stavový bit LPTMR časovače indikující přetečení časovače, což značí nevykonávání přerušení. V hlavní smyčce je inkrementována proměnná. V přerušení je pak tato hodnota čtena, čímž je kontrolováno, zda se hlavní smyčka vykonává dostatečně často. Řešení je výhodné z hlediska využití periferií, jelikož není nutné použití dalšího časovače.

### 4.3.6 Test časovače watchdog

Test časovače watchdog by měl otestovat jak správné fungování, tak i časování. Správné fungování znamená, že by mělo být otestováno, zdali časovač watchdog po vypršení časového limitu skutečně resetuje mikrokontrolér. Správné časování znamená, že k resetu mikrokontroléru došlo po uplynutí definované periody.

Watchdog v mikrokontroléru KV46 má jistá specifika. Pro modifikaci nastavení periferie je nutné nejprve do registru zapsat sekvenci pro odemknutí a až poté je možné periferii konfigurovat, navíc konfigurace je možná pouze 256 cyklů po odemčení. V případě, že v tomto intervalu nedojde k modifikaci žádného konfiguračního bitu, je vyvolán reset. Podobný mechanismus je implementován i při resetování časovače, kdy je nutné zapsat do registru správnou obnovovací sekvenci, navíc mezi těmito zápisy nesmí uplynout více než 20 cyklů. Po resetu mikrokontroléru začne automaticky čítat perioda 256 cyklů. Je tedy nutné hned po startu modifikovat konfigurační bity, jinak dojde k resetu.

Samotný test je navržen tak, že po Power-On-Resetu je na začátku funkce *main* zavolána funkce *WatchdogTest*. V této funkci je inicializován časovač, je nutné, aby zdroj jeho hodinového signálu byl odlišný od zdroje hodinového signálu časovače watchdog.

Následně je simulováno uvíznutí v programu pomocí nekonečné smyčky a je periodicky vyčítána hodnota časovače:

```
while(1)
{
    LPTMR->CNR = 0;       // Counter read sync
    psWatchdogBackup->ByteCounter = LPTMR->CNR;
}
```

Po vyvolání resetu časovačem watchdog je pak hodnota porovnána s limitními hodnotami, čímž je zjištěno, zda časovač watchdog resetoval mikrokontrolér v daném časovém okně. Samotná periferie umožňuje i dva hardwarové testy. Pro tyto účely byla vytvořena funkce, která otestuje časovač watchdog pomocí tzv. Byte testu. Časovač watchdog je uveden do testovacího módu, kdy je testu podroben každý byte z 4bytového registru časovače. Tento test sice otestuje všechny byty registru, ale už netestuje čas, po kterém k watchdog resetu došlo. Navíc se pro testování používá hardware časovače watchdog, kdy v případě poruchy test nemusí být průkazný. Pro implementaci je tedy vhodnější test se simulací uváznutí v nekonečné smyčce, který je vykonán softwarově a nevyužívá přímo hardware časovače watchdog pro samotný test.

### 4.3.7 Test zásobníku

Zásobník je testován pomocí dvou funkcí. Zásobník je část paměti RAM, je pro něj použita stejná funkce jako pro testování paměti RAM, tedy algoritmus March C. Jediný rozdíl je v paměťové oblasti, pro kterou je test prováděn. V tomto případě je nutné zadat počáteční a koncovou adresu pro oblast vyhrazenou pro zásobník. Za tyto adresy je umístěn vzor. Ten je pomocí jednoduché funkce čten a komparován s očekávaným vzorem. Při nerovnosti patrně došlo k rozšíření zásobníku mimo vymezenou oblast. Adresa, která ukazuje na vrchol zásobníku, je uložena v registru Stack Pointer, který je testován spolu s ostatními registry jádra.

### 4.3.8 Test toku programu

Pro testování toku programu byla zvolena metoda CFCSS, která testuje, že testované bloky se vykonávají v požadované posloupnosti, tedy že z bloku A je přecházeno do bloku B a nikoliv do bloku C. Funkce *ProgramFlowCheckTest* je rozmístěna do důležitých uzlů, jejichž korektní sekvence vykonávání je pro aplikaci nezbytná. V tomto aplikačním případě byla funkce umístěna do stavového automatu a do bloků pro volání dílčích testů funkční bezpečnosti. V proměnné *ProgramFlowSign* je uložena značka předešlého uzlu. V aktuálně testovaném uzlu je provedena operace XOR mezi značkou předešlého uzlu a očekávaným rozdílem mezi uzly. V případě, že je přechod korektní, tak se výsledek této operace musí rovnat značce současného uzlu.

### 4.3.9 Test digitálních vstupů/výstupů

Pro test výstupů jsou v některých aplikačních nótách použity mechanismy, kde výstup mikrokontroléru je připojen se vstupem mikrokontroléru. Tím však lze jen zjistit, že se

požadovaná hodnota skutečně zpropagovala na výstup mikrokontroléru. Je však podstatné, že se daný výstupní signál zpropagoval až do koncového bodu, například spínací tranzistor *Bot*. Správná hodnota digitálního výstupu, a tedy sepnutí/rozepnutí tranzistoru, je ověřeno přes měření pomocí AD převodníku. Samotný AD převodník je také testován, viz následující podkapitola.

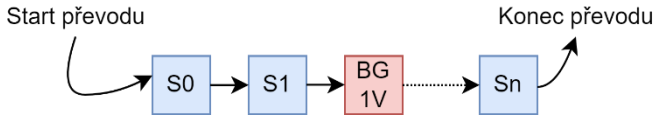
V aplikaci jsou použity dva digitální vstupy pro diagnostiku relátek. Poruchu digitálních vstupů lze odhalit diagnostickým mechanismem relátek. Při sepnutí relátka musí být na digitálním vstupu definovaná logická hodnota. Při rozpojení relátka musí být na digitálním vstupu inverzní hodnota.

### 4.3.10 Test AD převodníku

Test převodníku testuje, zda převodník převádí korektně vstupní napětí. Test multiplexoru testuje, zda jsou vstupní signály správně vybírány.

#### 4.3.10.1 Převodník

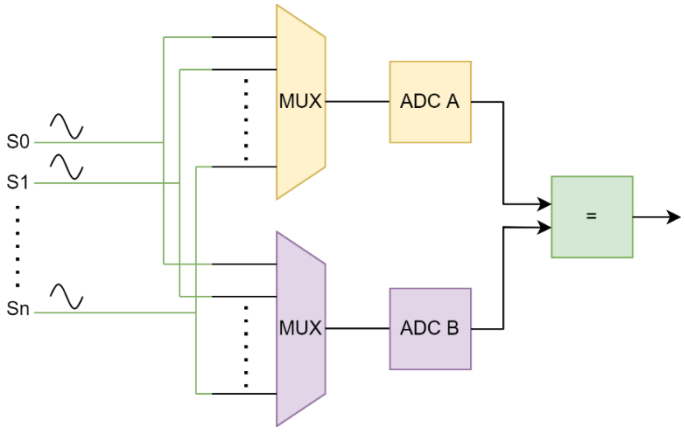
V některých aplikačních nótách funkce pro testování převodníku rekonfiguruji periferii převodníku, to však není vůbec nutné. Mikrokontrolér KV46 disponuje interní bandgap referencí o napětí 1 V. Filozofie navrženého testu je taková, že mezi převáděné vzorky je umístěn vzorek se známou hodnotou napětí, v tomto případě již zmíněný bandgap. Ten je v mikrokontroléru nutné interně propojit se vstupem AD převodníku. Hodnota referenčního vzorku je pak softwarově porovnávána s očekávanou hodnotou (v mezích tolerance). Pro správné fungování je potřeba nastavit výstupní buffer bandgap reference na vyšší proudovou schopnost.




Obrázek 48: Princip testu AD převodníku

#### 4.3.10.2 Multiplexor

Mikrokontrolér KV46 disponuje dvěma AD převodníky, *ADC_A* a *ADC_B*. Měřené signály jsou přivedeny na vstupy obou AD převodníků, signály jsou tedy převedeny dvěma nezávislými převodníky a následně jsou softwarově porovnány výsledky převodů.




Obrázek 49: Princip testu AD multiplexoru

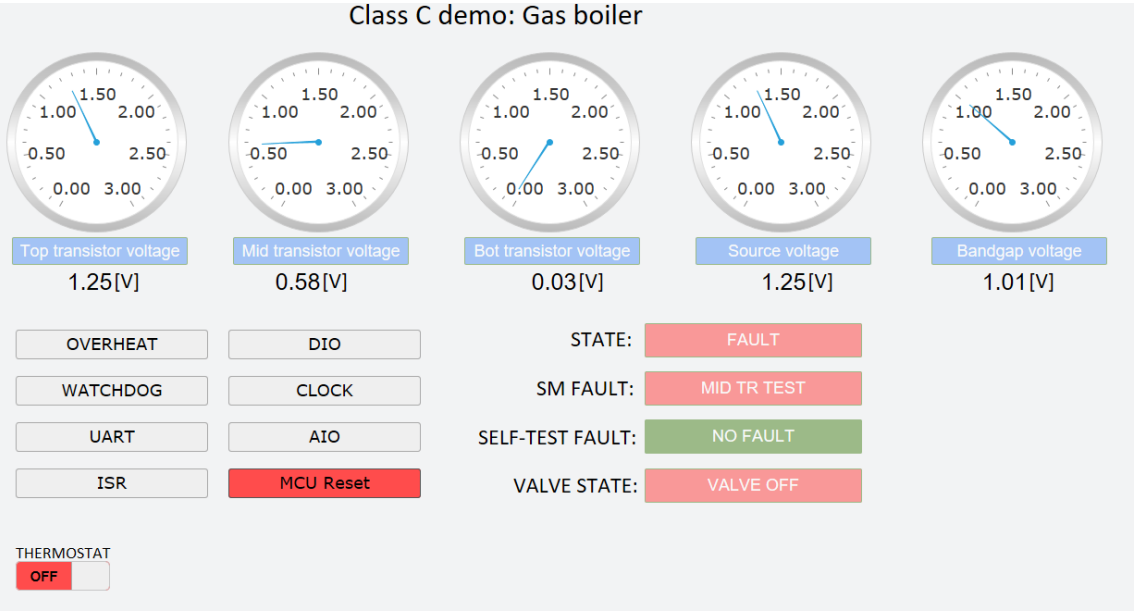
# 5. TESTOVÁNÍ

## 5.1 Grafické uživatelské rozhraní v programu FreeMASTER

Softwarový nástroj FreeMASTER je určený k ladění programu běžícího v reálném čase a zároveň podporuje tvorbu grafického uživatelského rozhraní.

Ve skriptovacím jazyce JavaScript bylo vytvořeno grafické uživatelské rozhraní pro ovládání demonstračního hardwaru. Rozhraní zobrazuje jednotlivá naměřená napětí pomocí AD převodníku mikrokontroléru master. Výjimkou je napětí *Source voltage*, což je hodnota 12V napájecího napětí měřeného mikrokontrolérem slave a posílaného přes UART. Přepínačem *THERMOSTAT* je simulován požadavek na zažehnutí kotle (otevření plynového ventilu). V rozhraní je vypisován aktuální stav stavového automatu *STATE*, chyby *SM FAULT* (proměnná *SMAllFaults*) a *SELF-TEST FAULT* (proměnná *AllFaults*) a stav plynového ventilu *VALVE STATE*.

Tlačítka v levé části spolu se zkratovacími propojkami na DPS umožňují do systému uměle vložit chybu. To slouží k demonstraci funkčnosti navrženého řešení. Pro demonstrační účely při detekované chybě dojde k odpojení ventilu a systém se dostane do stavu *FAULT*. Z něj je systém možné vyprostit tlačítkem *MCU Reset*, které resetuje oba mikrokontroléry. Na Obrázku 50 lze vidět situaci, kdy byl pomocí zkratovací propojky odpojen prostřední (*Mid*) tranzistor, což indikuje *SM FAULT: MID TR TEST*.



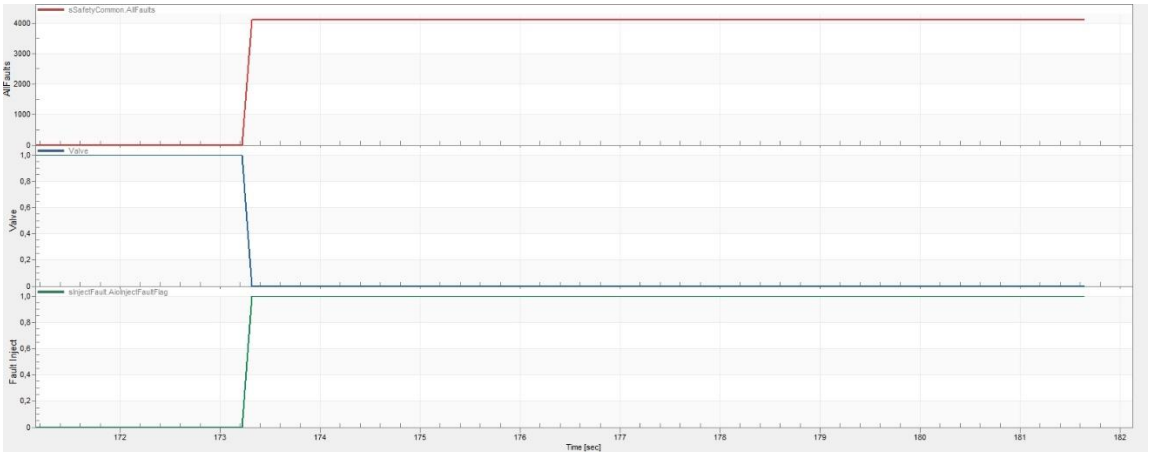

Obrázek 50: Grafické uživatelské rozhraní v programu FreeMASTER

## 5.2 Průběhy vybraných testů na straně mikrokontroléru master

Jelikož kompletní testování by znamenalo podrobit systém velké sadě testů a různých konfigurací, tak pro demonstraci funkčnosti řešení byly vybrány pouze některé testy. Po stisku příslušného tlačítka v programu FreeMASTER jsou volány funkce, které do systému uměle injektují chybu. Požadovaný výsledek testu je, aby byla vložená chyba detekována (proměnné *AllFaults* a *SMAllFaults*) a plynový ventil vypnut (proměnná *Valve*).

### 5.2.1 Test AD převodníku

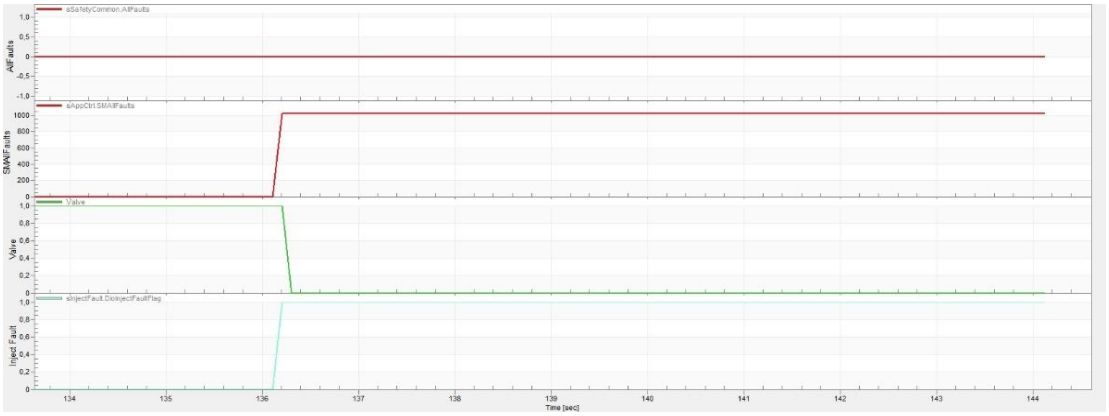
Chyba v AD převodníku byla simulována změnou referenčního vzorku periferie *ADC_A,* správný referenční vzorek je bandgap reference. Průběh proměnných je zaznamenán na Obrázku 51. Náběžná hrana zeleného průběhu značí okamžik vložení chyby. Modrý průběh odpovídá stavu plynového ventilu, červený hodnotě proměnné *AllFaults*, jejíž nenulová hodnota indikuje odhalenou poruchu.




Obrázek 51: Časový průběh proměnných pro test AD převodníku

### 5.2.2 Test digitální výstupu

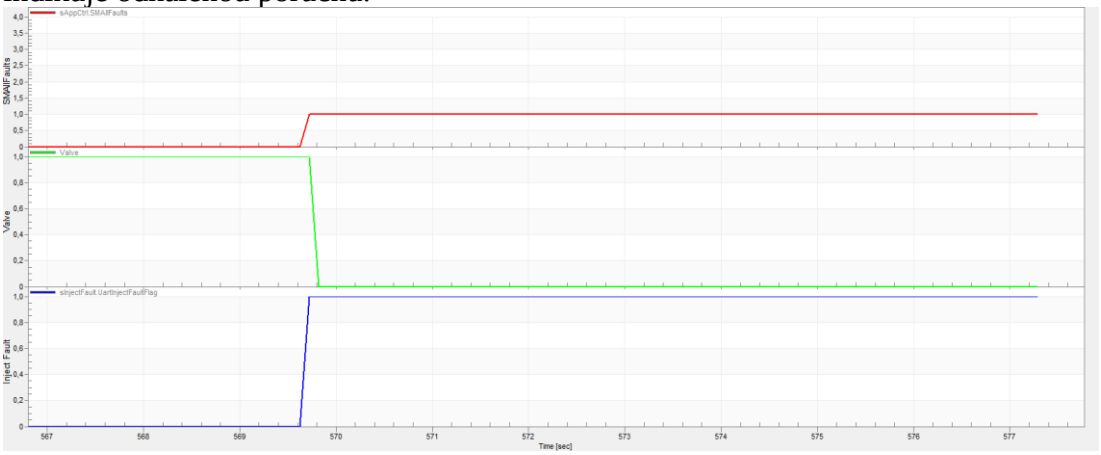
Chyba v digitálním výstupu byla simulována SA0 (stuck-at 0) chyba. Náběžná hrana světlemodrého průběhu značí okamžik vložení chyby. Zelený průběh odpovídá stavu plynového ventilu, červený hodnotě proměnné *SMAllFaults*, jejíž nenulová hodnota indikuje odhalenou poruchu. Hodnota proměnné *AllFaults* zůstává nulová.



Obrázek 52: Časový průběh proměnných pro test digitálního výstupu

### 5.2.3 Test rozhraní UART

Chyba komunikace přes UART byla simulována vypnutím přijímače periferie. Náběžná hrana modrého průběhu značí okamžik vložení chyby, zelený průběh odpovídá stavu plynového ventilu, červený průběh hodnotě proměnné *AllFaults*, jejíž nenulová hodnota indikuje odhalenou poruchu.



Obrázek 53: Časový průběh proměnných pro test UART

## 5.3 Test systému

Poslední měření byla provedena pomocí logického analyzátoru. Signály *Master switch* a *Slave switch* odpovídají hodnotám na výstupu mikrokontroléru master a slave. Náběžná hrana signálu *Master/Slave inject fault* indikuje okamžik vložení chyby do systému.

Při prvním testu byla vložena chyba do mikrokontroléru master, průběh signálu je na Obrázku 54. Po vložený chyby nejprve master rozepne tranzistor *Mid* a následně slave rozepne tranzistor *Bot*. Přechod signálu *Valve* z logické 0 do logické 1 odpovídá odpojení plynového ventilu. Časová poloha náběžné hrany signálu *Valve* neodpovídá skutečnému stavu, jelikož jde o signál měřený v obvodu pro detekci napětí na ventilu, v němž se pro měření využívá kondenzátorů. Tyto kondenzátory svým nabíjením/vybíjením vkládají zpoždění. Reálný časový údaj vypnutí plynového ventilu se pohybuje do 20 ms od

náběžné hrany signálu *Master inject fault*. Stejné měření bylo provedeno pro mikrokontrolér slave.


Obrázek 54: Průběhy při vložení chyby na straně mikrokontroléru master


Obrázek 55: Průběhy při vložení chyby na straně mikrokontroléru slave

Z obou průběhu je patrné, že při detekci poruchy mikrokontrolér vypne příslušný tranzistor a pomocí navržených mechanismů informuje druhý mikrokontrolér o detekované poruše, na což druhý mikrokontrolér zareaguje vypnutím příslušného tranzistoru. Systém je tedy schopen reagovat na chyby detekované kterýmkoliv mikrokontrolérem a zajistit požadovaný bezpečný stav, tedy vypnutí plynového ventilu.

# 6. ZÁVĚR

Cílem této diplomové práce bylo navrhnout a implementovat testy funkční bezpečnosti pro mikrokontrolér splňující požadavky vycházející z norem IEC 60730 a IEC 60335.

V teoretické části práce byla rozebrána problematika funkční bezpečnosti jako celku a popsány normy IEC 60730 a IEC 60335. Stěžejním tématem teoretické části byl rozbor normou vyžadovaných testů funkční bezpečnosti pro mikrokontroléry. Bylo potřeba důkladně nastudovat metody a principy testování. Navržené a diskutované metody byly převážně získány ze zkušeností odborníků na danou problematiku, reálných průmyslových řešení a interakcí s certifikačními autoritami VDE a UL

Normou definované testy jsou vždy navrhované s ohledem na konkrétní architekturu systému s funkční bezpečností. K těmto účelům byl navržen demonstrační hardware simulující funkci plynového kotle. Byla navržena struktura pro bezpečné řízení plynového ventilu pomocí dvou mikrokontrolérů pracujícími v konfiguraci master-slave, které mezi sebou komunikují přes rozhraní UART. Komunikace byla navržena jako „Black channel communication“. Toto řešení je výhodné, protože nepodléhá funkční bezpečnosti, a tudíž není nutné komunikaci testovat.

Master řídí logiku celého systému, slave redundantně měří důležité analogové signály a v případě poruchy je schopen nezávisle na mikrokontroléru slave odpojit plynový ventil. Taktéž slave je schopen při detekci chyby nezávisle na mikrokontroléru master odpojit plynový ventil. Analogové hodnoty naměřené mikrokontrolérem slave jsou posílány přes UART a na straně mikrokontroléru master vzájemně porovnávány. Každý analogový signál (kromě signálu pro měření úrovně napájecího napětí) je měřen čtyřmi nezávislými analogovými převodníky.

Byla navržena větev pro spínaní relátek ovládající ventil s ohledem na celkovou bezpečnost. V sérii jsou zapojeny tři tranzistory, *Top, Mid* a *Bot*. Tranzistor *Top* je řízen obvodem pro detekci poklesu napájecího napětí, tranzistor *Mid* mikrokontrolérem master a tranzistor *Bot* mikrokontrolérem slave. Fungování spínací větve lze připodobnit k logické funkci AND, jelikož všechny tři tranzistory musí být sepnuty, aby byla sepnuta i relátka. Diagnostika je z hlediska bezpečnosti velmi důležitá. Byly navrženy a implementovány účinné diagnostické mechanismy, které umožňují odhalit a přímo lokalizovat zkrat/rozpojení tranzistorů *Top, Mid* a *Bot* a zkrat/rozpojení relátek. Diagnostika tranzistorů je prováděna s nastavitelnou periodou. Diagnostika relátek je prováděna vždy při přechodu do stavu *Heat*, což odpovídá požadavku na zažehnutí kotle. Perioda diagnostik by měla být zvolena s ohledem na mechanické opotřebování relátek. Všechny tyto navržené mechanismy jsou díky svým schopnostem zamezit a předcházet chybám využitelné v reálném zapojení. Tyto obvody byly doplněny o části, které slouží pouze k imitaci plynového kotle (LED dioda pro plamen, fotodioda jako detektor plamene, zkratovací propojky pro vkládání chyb). Demonstrační hardware byl navržen v programu KiCAD.

V softwarové části byla nejprve navržena logika pro mikrokontrolér master. S ohledem na normu IEC 60730 bylo navrženo řízení pomocí stavového automatu o šesti stavech. Byla navržena softwarová struktura stavového automatu. K těmto účelům bylo vytvořeno pole ukazatelů na funkce pro jednotlivé stavy (funkce *Heat()*, funkce *Fault()* atd.). Funkce jsou volány na základě hodnoty proměnné *eState*, ve které je uložen aktuální stav stavového automatu. Stavový automat je řízen pomocí zápisu bitových masek do proměnné *uiCtrl*. Dále byl navržen stavový automat pro mikrokontrolér slave. Nabízelo by se řešení „překopírovat“ kód z mikrokontroléru master, čímž však hrozí přenesení systematických chyb. Proto byla logika fungování mikrokontroléru slave navržena pomocí odlišných konstrukcí. Dále byla navržena logika komunikace mezi mikrokontroléry přes rozhraní UART, což zahrnovalo vytvoření sady příkazů pro vzájemnou komunikaci a navržení datového paketu. Pro vyšší bezpečnost byla navržena logika tzv. timeoutu, což je mechanismus, který při selhání komunikace zajistí detekci chyby. V další části byly navrženy a implementovány testy funkční bezpečnosti v souladu s požadavky normy IEC 60730 pro třídu B. Pro každý test je volána samostatná funkce. Testy nejde volat všechny naráz, proto byl vytvořen plánovač, který periodicky volá jednotlivé funkce. Výsledky testů funkční bezpečnosti jsou ukládány do jednotlivých bitů proměnné *AllFaults*. Výsledky testů prováděných v rámci funkcí stavového automatu jsou ukládány do jednotlivých bitů proměnné *SMAllFaults*.

Funkčnost navrženého systému byla testována pomocí programu FreeMASTER, který je určený k ladění programu běžícího v reálném čase a zároveň podporuje tvorbu grafického uživatelského rozhraní. K testovacím a demonstračním účelům bylo v jazyce JavaScript navrženo grafické uživatelské rozhraní. V tomto rozhraní se vypisují naměřené hodnoty jednotlivých analogových signálů. Rozhraní též vypisuje aktuální stav systému a detekované chyby. K testovacím účelům byly vytvořeny funkce v jazyce C, které do systému umožňují uměle injektují chyby. Například funkce pro simulaci chyby hodinového signálu přenastaví děličku systémového zdroje hodinového signálu, funkce pro simulaci chyby AD převodníku změní hodnotu referenčního vzorku AD převodníku.

Závěrem byly provedeny vybrané testy a jejich výsledky zobrazeny pomocí průběhů v programu FreeMASTER a pomocí logického analyzátoru. Všechny provedené testy byly úspěšné, tedy plynový ventil byl při detekované chybě odpojen od napětí, čímž byl systém uveden do bezpečného stavu.

# LITERATURA


[1] Prášek, R.: Functional *Safety*. [Online]. [cit. 15.10.2021] Dostupné z: https://www.qmprofi.cz/33/funkcni-bezpecnostuniqueidmRRWSbk196FNf8-jVUh4EjjkeFupcvnhRmUsfGJb5uA

[2] LEPKA, Jaroslav. *Functional safety library - deep dive,* Interní dokument firmy NXP, 2021

[3] *IEC 60730-1: Automatic electrical controls - Part 1: General requirements. International Electrotechnical Commission*, 2013, 580 s.

[4] *IEC 60335-1: Household and similar electrical appliances - Part 1: General requirements. International Electrotechnical Commission,* 2013, 708 s.

[5] *Industrial Safety starts with IEC/UL 60730 Standards* [online]. [cit. 15.10.2021]. Dostupné z: https://www.nxp.com/filesstatic/training_pdf/WBNR_FTF10_ENT_F0714.pdf

[6] *Defects, Errors, and Faults* [online]. ©2017. [Cit. 27.11.2021]. Dostupné z: http://www.vlsifacts.com/defects-errors-and-faults/

[7] LEE, Kyoungwoo. *Fault, Failure & Reliability* [online]. [Cit. 11.10.2021]. Dostupné z: https://www.ics.uci.edu/~kyoungwl/forge/rep/talk/FaultFailureReliability.pdf

[8] *Failures in Integrated Circuits* [online]. [Cit. 14.10.2021]. Dostupné z http://ece-research.unm.edu/jimp/vlsi_test/slides/html/defects1.html

[9] *Using the Built-in Self-Test (BIST) on the MPC5777M* [online]. [cit. 15.12.2021]. Dostupné z: https://www.nxp.com/docs/en/application-note/AN5131.pdf

[10] PAROVINCHAK Vasyl, KUZO Taras. *PSoC 4 – IEC 60730 Class B and IEC 61508 SIL Safety Software Library* [online]. [cit. 15.12.2021]. Dostupné z: https://www.infineon.com/dgdl/Infineon-AN89056_PSoC_4_IEC_60730_Class_B_and_IEC_61508_SIL_Safety_Software_Library-ApplicationNotes-v05_00-EN.pdf?fileId=8ac78c8c7cdc391c017d073cff8f610f&utm_source=cypress&utm_medium=referral&utm_campaign=202110_globe_en_all_integration-application_note

[11] *Safety Class B with PMSM Sensorless Drive MPC5777M* [online]. [cit. 15.12.2021]. Dostupné z: https://www.nxp.com/docs/en/application-note/AN5321.pdf

[12] *IEC60730B Library User's Guide* [online]. [cit. 15.12.2021]. Dostupné z: https://www.nxp.com.cn/docs/en/user-guide/IEC60730BCM4CM7L41UG.pdf

[13] OH Namshuk, SHIRVANI Philip, MCCLUSKEY J. Edward. *Control Flow Checking by Software Signatures* [online]. [cit. 15.12.2021]. Dostupné z: https://www.nxp.com/docs/en/application-note/AN5321.pdf

[14] *KV4x Reference Manual* [online]. ©2011. [cit. 16. 10. 2021] Dostupné z: https://www.nxp.com/design/development-boards/tower-development-boards/mcu-andprocessor-modules/kinetis-modules/kinetis-kv4x-family-tower-system-module:TWRKV46F150M

[15] *Overall requirements for Safety Class B and C Software development*, Interní dokument firmy NXP, 2021

[16] *Guidelines for obtaining UL/CSA/IEC 60730-1/60335-1 Class B certification in any STM32 application* [online]. [cit. 16. 10. 2021] Dostupné z: https://www.st.com/resource/en/application_note/dm00105610-guidelines-for-obtaining-ulcsaiec-607301603351-class-b-certification-in-any-stm32-application-stmicroelectronics.pdf

[17] *NASA's 10 Coding Rules for Writing Safety Critical Program* [online]. [cit. 16. 10. 2021] Dostupné z: https://www.rankred.com/nasa-coding-rules/

[18] *KV4x Data Sheet* [online]. ©2011. [cit. 20. 12. 2021] Dostupné z: https://www.nxp.com/docs/en/data-sheet/KV4XP100M168.pdf

[19] HOBBS, Chris. *Embedded software development for safety-critical systems*. Second edition. Boca Raton: CRC Press, [2020]. ISBN 978-0-367-33885-5.

[20] PONT, Michael J. *The Engineering of Reliable Embedded Systems: Developing software for 'SIL 0' to 'SIL 3' designs using Time-Triggered architectures*. Second Edition. SafeTTy Systems Ldt., 2017. ISBN 978-0-9930355-3-1

[21] *VLSI test principles and architectures design for testability*. Editor Laung-Terng WANG, editor Cheng-Wen WU, editor Xiaoqing WEN. San Francisco: Morgan Kaufmann, 2006. ISBN 0-12-370597-5.

[22] *Are All Random Failures Systematic?* [online]. [cit. 11.5.2022]. Dostupné z: https://ez.analog.com/ez-blogs/b/engineerzone-spotlight/posts/are-all-random-failures-systematic

[23] *Integration of Soft Errors in Functional Safety: a conceptual study* [online]. [cit. 16. 4. 2022] Dostupné z: https://www.researchgate.net/publication/281967937_Integration_of_Soft_Errors_in_Functional_Safety_a_conceptual_study

[24] *MISRA Development Guidelines for Vehicle Based Software, ISBN 0 9524156 0 7, Motor Industry Research Association*

[25] *Cortex-M4 Devices Generic User Guide* [online]. ©2011. [cit. 16. 4. 2022] Dostupné z: https://developer.arm.com/documentation/dui0553/latest/

[26] Bushnell, M. L.: *Essentials Of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, 2000, ISBN 9780792379911.

[27] *EN 13611: Safety and control devices for burners and appliances burning gaseous and/or liquid fuels - General requirements*, 2019, 153 s.

# SEZNAM SYMBOLŮ A ZKRATEK

## Symboly

| Symbol | Význam | Jednotka |
|---|---|---|
| $I_{load}$ | Proud tekoucí zátěží | [A] |
| $I_B$ | Proud tekoucí do báze tranzistoru | [A] |
| $I_c$ | Proud tekoucí kolektorem tranzistoru | [A] |
| $U_{load}$ | Napětí na zátěži | [V] |
| $U_{CE(sat)}$ | Saturační napětí tranzistoru | [V] |
| $U_{CC}$ | Napájecí napětí | [V] |
| $U_{R4}$ | Napětí na rezistoru R4 | [V] |
| $R_{REL1}$ | Hodnota odporu relátka 1 | [Ω] |
| $R_{REL2}$ | Hodnota odporu relátka 2 | [Ω] |
| $R_{load}$ | Hodnota odporu zátěže | [Ω] |
| $R_B$ | Hodnota odporu rezistoru v bázi tranzistoru | [Ω] |
| **R1, 2, 3, 4** | Hodnoty odporu rezistorů | [Ω] |

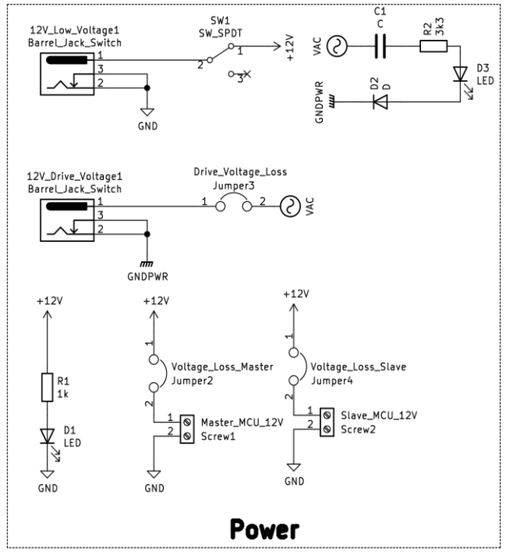
## Zkratky

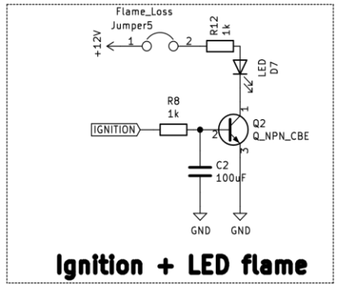
| Zkratka | Význam |
|---|---|
| **AC** | Alternating Current |
| **AD** | Analog to Digital |
| **ADC** | Analog to Digital converter |
| **ARM** | Advanced RISC Machines |
| **BIST** | Built In Self Test |
| **CFCSS** | Control Flow Checking by Software Signatures |
| **CPU** | Central Processing Unit |
| **CRC** | Cyclic Redundancy Check |
| **DA** | Digital to Analog |
| **DAC** | Digital to Analog Converter |
| **DPS** | Deska plošných spojů |
| **EMC** | Electromagnetic Compatibility |
| **ESD** | Electrostatic Discharge |
| **EUC** | Equipment Under Control |
| **FMEA** | Failure Mode and Effects Analysis |
| **FPU** | Floating Point Unit |
| **GPIO** | General Purpose Input Output |
| **I2C** | Inter-Integrated Circuit |
| **IEC** | International Electrotechnical Commission |
| **LBIST** | Logic Built-in Self-test |
| **LIFO** | Last In First Out |
| **LPO** | Low Power Oscillator |
| **LPTMR** | Low Power Timer |
| **MBIST** | Memory Built-in Self Test |

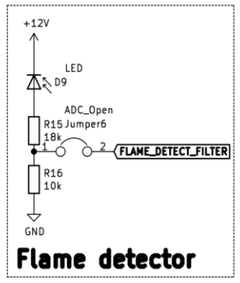
| | |
|---|---|
| **MISRA** | Motor Industry Software Reliability Association |
| **MTTF** | Mean Time To Failure |
| **POR** | Power-on Reset |
| **PWM** | Pulse-width modulation |
| **RAM** | Random Access Memory |
| **ROM** | Read Only Memory |
| **S/H** | Sample and Hold |
| **SA** | Stuck-at |
| **SBST** | Software Based Self Testing |
| **SIL** | Safety Integrity Level |
| **SPI** | Serial Peripheral Interface |
| **UART** | Universal Asynchronous Receiver Transmitter |

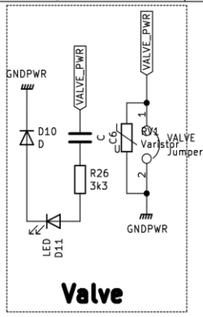
# Příloha A - Návrh zařízení

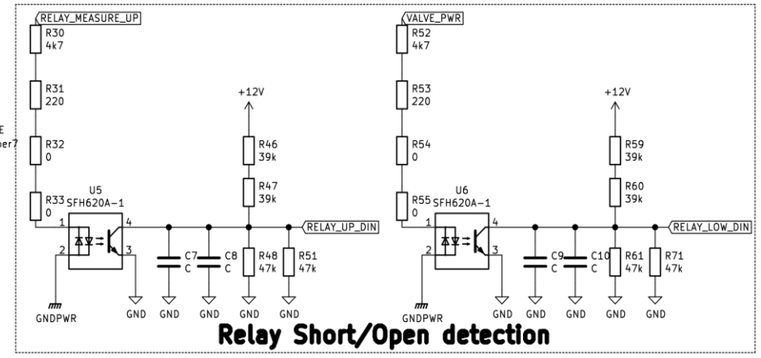
## A.1 Obvodové zapojení demonstračního přípravku

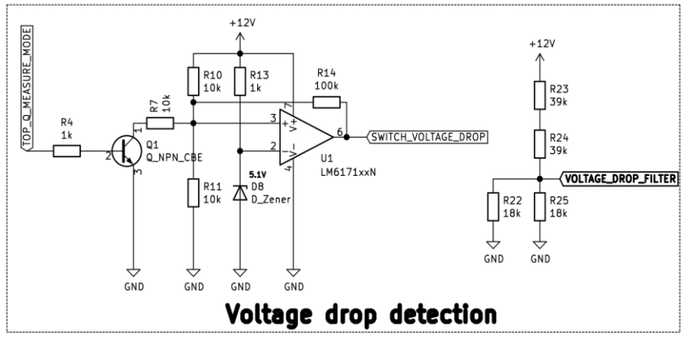


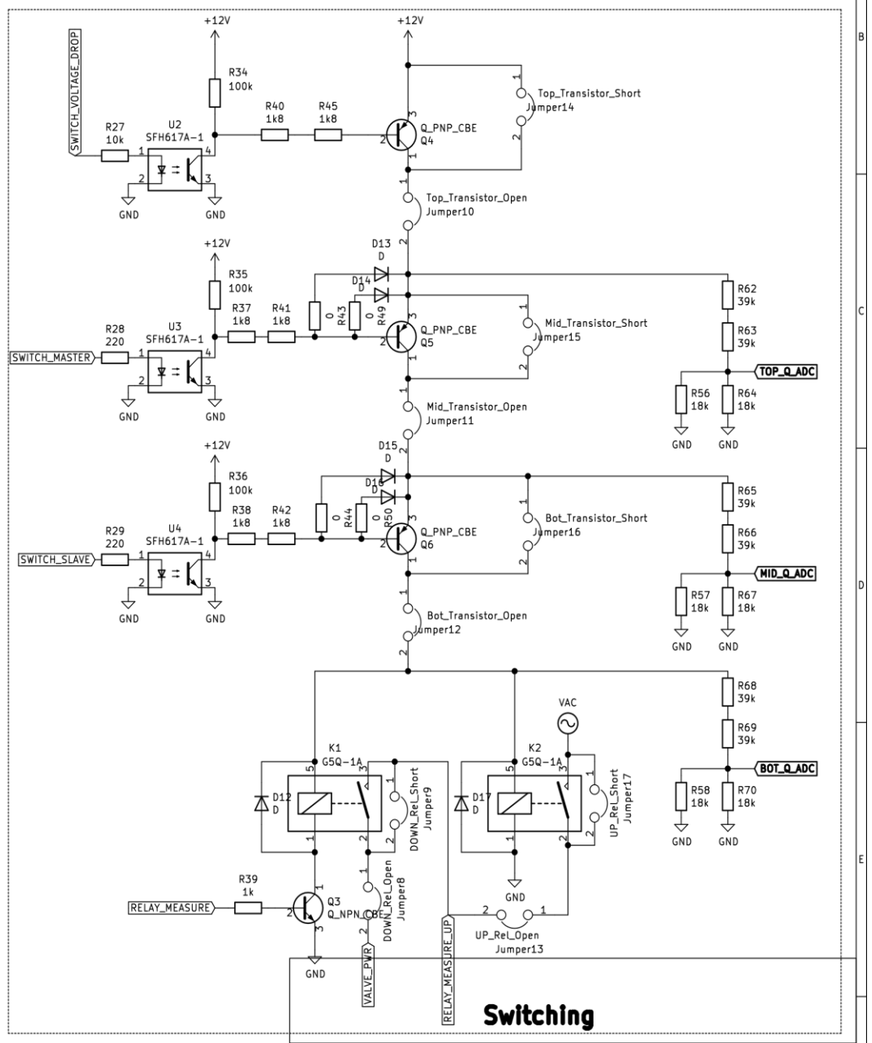



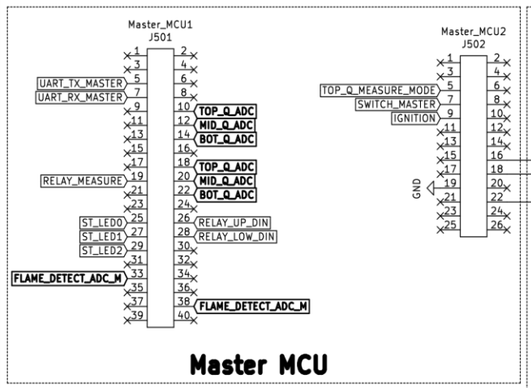



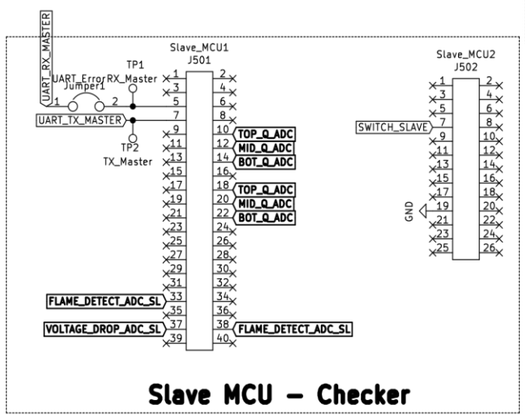



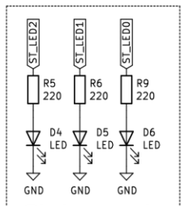


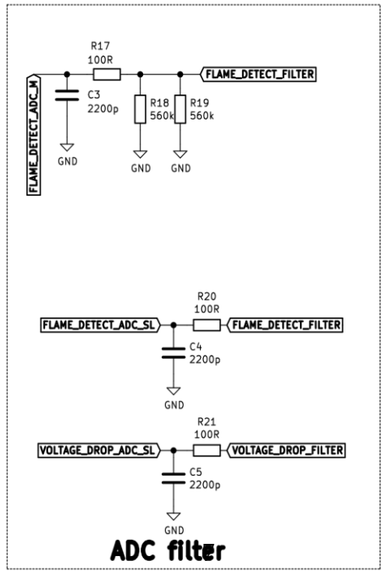


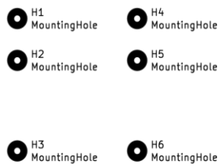









Sheet: /
File: block_sch.sch

**Title:**

| Size: A3 | Date: | **Rev: 1.0** |
|---|---|---|
| KiCad E.D.A. kicad (5.1.5)-3 | | Id: 1/1 |