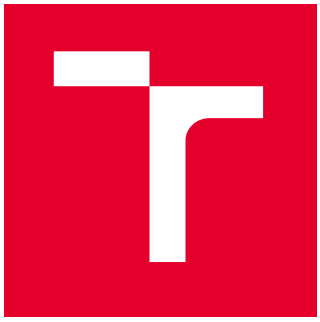


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

APLIKACE PRO AUTOMATIZOVANÉ MĚŘENÍ PROUDOVÉ SPOTŘEBY

APPLICATION FOR AUTOMATED POWER TRACE MEASUREMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Lukáš Karabina**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Gerlich**
SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Lukáš Karabina ***ID:*** 203458

Ročník: 2 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Aplikace pro automatizované měření proudové spotřeby

POKYNY PRO VYPRACOVÁNÍ:

Hlavním cílem diplomové práce je návrh a implementace komplexní aplikace pro realizaci automatizovaného měření proudové spotřeby kryptografických zařízení s cílem zjištění odolnosti proti útokům postranními kanály. Cílená aplikace bude ve formě desktopové aplikace, která bude komunikovat současně s měřicím zařízením (osciloskop) a testovaným na vývojové desce SAKURA (USB rozhraní). Aplikace musí být vytvořena modulárně pro možnost dalšího rozšíření. Aplikace bude umožňovat nastavení osciloskopu (např. posun časové základny, změna rozlišení, nastavení odesílaných dat) a generovat vstupní data pro kryptografické zařízení. Aplikace bude umožňovat ukládání dat z osciloskopu. Výslednou aplikaci otestujte na experimentálním pracovišti.

DOPORUČENÁ LITERATURA:

[1] HEJLSBERG, Anders, et al. The C# programming language. Pearson Education, 2008.

[2] Digital Oscilloscopes Programmer Manual. Tektronix [online]. [cit. 2021-9-14].

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

Vedoucí práce: Ing. Tomáš Gerlich

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Diplomová práce je zaměřená na poskytnutí informací k návrhu a implementaci aplikace pro automatizované měření proudové spotřeby. První a druhá část této práce shrnuje poznatky o vlastním měření proudové spotřeby a využití výsledků při analýze odolnosti kryptografických systémů proti útokům postranními kanály. V této části je uvedeno několik metod, které lze při takové analýze využít. Je zde uvedena kapitola o přístrojích, se kterými bude aplikace komunikovat. Praktická část této práce se zabývá vlastním návrhem a vývojem aplikace. Jsou zde uvedeny využité návrhové vzory a postupy, podle kterých je aplikace vytvořena. Jednotlivé podkapitoly dále popisují vlastní vývoj, implementaci, testování a problémy s tím spojené.

KLÍČOVÁ SLOVA

aplikace, automatizované měření, postranní kanály, osciloskop, proudová analýza

ABSTRACT

Master's thesis is focused on providing information for the design and implementation of an application for automated power trace measurement. The first and second sections of this thesis summarise the findings on the actual measurement of the power consumption and the use of the results in the analysis of the robustness of cryptographic systems against side channel attacks. In this sections, several methods that can be used in such an analysis are presented. A section on the devices with which the application will communicate is also included. The practical part of this thesis deals with the actual design and development of the application. The design patterns used and the procedures by which the application is created are presented. The individual subsections further describe the actual development, implementation, testing and associated problems.

KEYWORDS

application, automated measurement, side channels, oscilloscope, power analysis

KARABINA, Lukáš. *Aplikace pro automatizované měření proudové spotřeby*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2022, 69 s. Diplomová práce. Vedoucí práce: Ing. Tomáš Gerlich

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Lukáš Karabina
VUT ID autora:	203458
Typ práce:	Diplomová práce
Akademický rok:	2021/22
Téma závěrečné práce:	Aplikace pro automatizované měření proudové spotřeby

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

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

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Tomášovi Gerlichovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

V dnešní době, kdy jsou různá kryptografická zařízení používána na denní bázi a zajišťují relativně bezpečné fungování komplexní infrastruktury služeb a různých systémů, panuje přesto určitá nejistota, zda jsou konkrétní fyzická zařízení opravdu tak bezpečná, jak často jejich výrobci uvádějí. Tato zařízení implementují mnoho různých kryptografických protokolů, které poskytují dostatečnou teoretickou bezpečnost celého systému. Kryptografické protokoly samotné však často neposkytují fyzickou bezpečnost celého systému. I systém používající ten nejlepší kryptografický protokol může podlehnout fyzickým útokům. V takovém případě se jedná především o útoky postranními kanály. Útoky postranními kanály jsou právě mířeny na vlastní implementace protokolů a na zařízení, která je implementují, ne na protokol samotný a jeho matematickou bezpečnost. Útoků postranními kanály je celá řada a liší se od způsobu provedení, komplexity, časové náročnost apod. Pokud takové možnosti existují, jak napadnout kryptografický systém a kompromitovat jej, bez nutnosti lámání složitých matematických funkcí, které zajišťují bezpečnost vlastních protokolů, je velmi žádané ověřit vlastní fyzickou bezpečnost zařízení. Příkladem jedné kategorie útoků postranními kanály jsou útoky využívající proudovou spotřebu zařízení. Prakticky je téměř nemožné odstranit všechny korelace mezi chráněným tajemstvím a proudovou spotřebou zařízení, jelikož při zpracování těchto dat v elektronických zařízení musí z principu fungování docházet ke změnám proudové spotřeby.

Cílem ochrany proti takovému způsobu útoku je různými technikami a metodami, co nejvíce zamaskovat tuto korelaci a ztížit potenciálnímu útočníkovi analýzu vlastní proudové spotřeby. Provedení analýzy proudové spotřeby je klíčové k zjištění, zda je zařízení dostatečně chráněno proti útoku proudovým postranním kanálem. Tato analýza může být velmi obtížná a časově náročná, proto je důležité mít možnost celý proces automatizovat.

Cílem této práce je vytvoření desktopové aplikace, která bude umožňovat automatizované provedení měření proudové spotřeby na kryptografickém zařízení. Aplikace bude generovat a odesílat vstupní data pro kryptografické zařízení a zároveň umožňovat automatizovanou obsluhu osciloskopu, který měří vlastní proudovou spotřebu. Tento soubor dat bude poté umožněno uložit a použít pro další matematické analýzy. Další požadavek na aplikaci je vysoká modulárnost, aby mohla být později rozšířena o nové funkce.

1 Postranní kanály

Jako postranní kanál lze chápat jakoukoliv závislost, přímou i nepřímou, kterou lze využít k provedení tzv. *útok postranním kanálem – Side-Channel Attack* (SCA) na daný kryptografický systém či protokol. Jedná se tak o útoky na konkrétní implementace systémů, nikoliv na systémy či algoritmy samotné. Z toho vyplývá, že postranní kanály jako takové jsou závislé na zařízení, které implementuje část kryptografického systému. SCA se snaží o získání jakékoliv informace, kterou lze zneužít k prolomení protokolu. Existuje široké a pestré množství druhů SCA a příslušných postranních kanálů, kterými jsou prováděny. K využívaným postranním kanálům patří:

- Proudový postranní kanál,
- Časový postranní kanál,
- Elektromagnetický postranní kanál,
- Akustický postranní kanál,
- Chybový postranní kanál.

Některé postranní kanály vyžadují rozsáhlé znalosti technických specifikací daných zařízení nebo dlouhodobý přístup ke konkrétnímu zařízení. S ohledem na postranní kanál a konkrétní SCA se liší i komplexita měření, kryptoanalýzy a výsledného útoku. Komplexita se může rozpínat od pouhého pozorování veličin ovlivňující zařízení, až po odbrušování vrstev polovodičové struktury a zkoumání či interakce s vnitřními stavy jednotlivých součástek [1, 2].

Teoreticky lze uvažovat i lidský postranní kanál, kdy jsou informace získány prostřednictvím např. sociálního inženýrství. V praxi se toto označení a zařazení mezi postranní kanály nepoužívá.

1.1 Proudový postranní kanál

Závislost kryptografických systémů na elektronických zařízeních a tedy na proudové spotřebě, lze zneužít a provést SCA na daný kryptografický systém a protokol. I přesto, že použité kryptografické protokoly jsou považovány za bezpečné, jejich praktická implementace na elektronických zařízeních může vést k prolomení ochrany, kterou mají zajistit. V elektronickém zařízení jsou bity reprezentovány jako stavy elektronických součástek uvnitř obvodů. V rámci provádění jednotlivých operací dochází ke změnám vnitřních stavů obvodu. Tyto změny ovlivňují okamžitou proudovou spotřebu, která tak koreluje s bitovými řetězci a operacemi, které zařízení v danou dobu provádí.

Tuto korelaci lze využít k získání utajených informací, které za normálních okolností zaručují bezpečnost systému. Zpravidla se jedná o kryptografické klíče, které by

jinak nebylo možné ze systému získat. Tento způsob útoku vyžaduje alespoň částečný fyzický přístup k zařízení, na které je útok prováděn. Útok tak úplně obchází teoretickou bezpečnost použitých protokolů a spoléhá na míru korelace mezi utajenými informacemi a proudovou spotřebou. Zařízení je možné proti tomuto typu útoku chránit různými protiopatřeními, mezi které patří maskování a skrývání. Tato protiopatření narušují korelaci zpracovávaných dat na proudové spotřebě a dokáží ji snížit či zcela eliminovat [3].

1.1.1 Jednoduchá proudová analýza

Jednoduchá proudová analýza – Simple Power Analysis (SPA) je nejzákladnější metoda měření, analýzy a interpretace proudové spotřeby. Jde o přímé pozorování a zaznamenání dat, které v reálném čase měří osciloskop skrze sondu na zkoumaném zařízení, které provádí kryptografické operace. Pokud zařízení není proti útoku postranním kanálem nijak chráněno, může být tato metoda plně postačující k získání tajných parametrů kryptografického procesu a prolomení kryptografického systému, bez ohledu na sílu a bezpečnost použitého protokolu. Použití této metody i na chráněné zařízení může přinést cenné informace. I přesto, že zařízení má implementovány protiopatření proti útoku postranním kanálemmůže tato analýza odhalit použité protokoly, použité časování integrovaných obvodů a jiné důležité informace, které mohou být dále využity k provedení jiného typu útoku. Další výhoda této metody je, že lze použít na malé množství změřených proudových průběhů. V ideálním případě stačí pouze jeden změřený proudový průběh celé kryptografické operace k získání tajného kryptografického klíče.

Touto metodou lze odhalit použitý kryptografický algoritmus například na základě počtu rund, které algoritmus vykonává během kryptografické operace. Jakmile je odhalen použitý kryptografický algoritmus, lze podle jeho definice v naměřených datech identifikovat dokonce jednotlivé kroky, které algoritmus provádí. Z těchto korelací a znalostí samotného algoritmu lze v jistých případech přímo odvodit použitý kryptografický klíč. Obecně tuto metodu lze použít na běžná kryptografická zařízení, která implementují kryptografické protokoly jako např. DES, AES, RSA aj. [4, 5].

Předpokladem použití této metody je však přímé a precizní měření, jelikož téměř jakýkoliv šum, který při měření ovlivní dané zařízení a měřící soustavu, může zanést do naměřených hodnot takovou míru náhody, že nebude možné z naměřených dat získat potřebné informace k prolomení kryptografického systému. V případě, že metoda SPA nedostačuje, je nutné použít metodu DPA, která se opírá o statistickou diferenciální analýzu.

1.1.2 Diferenciální proudová analýza

Diferenciální proudová analýza – Differential Power Analysis (DPA) je pokročilejší metoda měření, analýzy a interpretace proudové spotřeby. Tato metoda je založena na využití statistické analýzy k nalezení korelace mezi změřenými hodnotami a tajnými informacemi kryptografického systému. Na rozdíl od metody SPA vyžaduje DPA velké množství proudových průběhů, změřené na kryptografickém zařízení, které mohou být ovlivněny vysokým šumem. Množinu proudových průběhů získáme zaznamenáním velkého množství kryptografických operací s generovanými daty, které zařízení provádí. Tato metoda je proto podstatně časově náročnější, ale přináší lepší výsledky než SPA. Metoda analyzuje celkovou korelaci měřených hodnot proudové spotřeby a zpracovávaných datech v celé množině změřených průběhů. Požadavek této metody je znalost kryptografického algoritmu, které dané zařízení používá. Vzhledem k velkému počtu proudových průběhů, které tato metoda vyžaduje, je ale schopna identifikovat části proudových průběhů, které korelují s kryptografickými parametry. Tyto části zahrneme do tzv. leakage modelu, který reprezentuje části, ovlivněné hodnotami kryptografického klíče. Statistickou diferenciální analýzou pak můžeme z daných částí odvodit kryptografický klíč, který byl zařízením použit. [6, 7, 8].

Jelikož metoda vyžaduje velké množství proudových průběhů, musí mít případný útočník delší dobu přístup ke kryptografickému zařízení, aby získal dostatečně velkou množinu naměřených proudových průběhů. Tuto množinu poté lze porovnat se zkoumaným proudovým průběhem, ze kterého chceme odvodit kryptografický klíč.

2 Měření proudové spotřeby

K provedení analýzy kryptografického zařízení, se zaměřením na odolnost proti útokům postranními kanály, je nutné nejprve získat dostatečné množství dat. V případě proudového postranního kanálu, jsou součástí těchto dat změřené proudové charakteristiky zařízení. Moderní kryptografie téměř ve všech případech spoléhá na elektronická zařízení, která implementují dané kryptografické protokoly. Implementace těchto kryptografických protokolů je v digitálním případě vždy založena na zpracování logických jedniček a nul neboli bitů. Každá informace, ať už se jedná o zprávu, kryptografický klíč, nebo jiný parametr, je reprezentována řetězcem bitů.

Kryptografické systémy skrze definované protokoly a operace zpracovávají tyto informace a zajišťují kryptografické aspekty. Fyzické zpracování těchto informací probíhá na elektronických zařízeních v rámci integrovaných obvodů, kde dochází ke změnám elektrického napětí a průtoku proudu. Kryptografické protokoly implementované na elektronických zařízeních i vlastní kryptografické operace a parametry jsou tedy svým způsobem závislé na proudové spotřebě zařízení, případně jednotlivých integrovaných obvodů [3, 9].

2.1 Automatizované měření

Měření proudové spotřeby s cílem zjištění odolnosti zařízení proti útokům postranními kanály může být časově náročné a těžko proveditelné bez použití zařízení a nástrojů, které umožní podstatnou část procesu automatizovat a zaznamenat. Některá měření mohou trvat hodiny i dny, záleží pouze na typu zařízení a počtu měření, která je třeba provést. Pro analýzu některých kryptografických zařízení mohou být vyžadovány velké počty měření jednotlivých proudových průběhů a jejich zaznamenání. Komplexnější analýzy vyžadují porovnání tisíců naměřených proudových průběhů.

Měření samotné je prováděno měřící sondou, která musí být schopna zachytit signál z měřeného zařízení. S ohledem na typ sondy může být připojena přímo k některému z elektrických kontaktů zařízení či integrovanému obvodu, nebo může snímat elektromagnetické vlny, které vyzařuje při průchodu proudem každý vodič i polovodič. Měřící sonda je dále připojena do osciloskopu, který signál dokáže zpracovat a zobrazit. Osciloskop může být připojený k jinému zařízení, zpravidla stolnímu nebo přenosnému počítači, pomocí dostupného rozhraní, do kterého odesílá naměřená data kde poté probíhá vlastní analýza.

2.2 Metody měření proudové spotřeby

Vlastní proudovou spotřebu je možné měřit několika způsoby. Obecně přítomnost napětí a průchod proudu způsobuje v závislosti na médiu několik fyzikálních jevů, které lze různými způsoby měřit a z měření vyvodit vlastní proudovou spotřebu kryptografického zařízení nebo integrovaného obvodu. Konkrétní metodu lze zvolit na základě dostupného vybavení, možnostech přístupu, vlastnostech a charakteru měřeného zařízení. Tyto metody poskytují různou přesnost měření a výsledků, jsou různě obtížné či nákladné a vyžadují jinou míru manipulace s měřeným zařízením. Některé metody mohou proudovou spotřebu měřit bezkontaktně a bez zásahu do kryptografického zařízení a jiné vyžadují přímou manipulaci se zařízením nebo integrovaným obvodem. Manipulace se zařízením může zahrnovat překonávání překážek a krytů, ale i přemosťování obvodových tras nebo vytržení některých pinů integrovaných obvodů [4].

Následující kapitoly budou věnovány několika používaným metodám pro měření proudové spotřeby. Metody budou obecně a stručně popsány. Dále bude vysvětlen základní fyzikální princip metody a přiloženo modelové schéma, jak měření danou metodou probíhá.

2.2.1 Měřící bočník

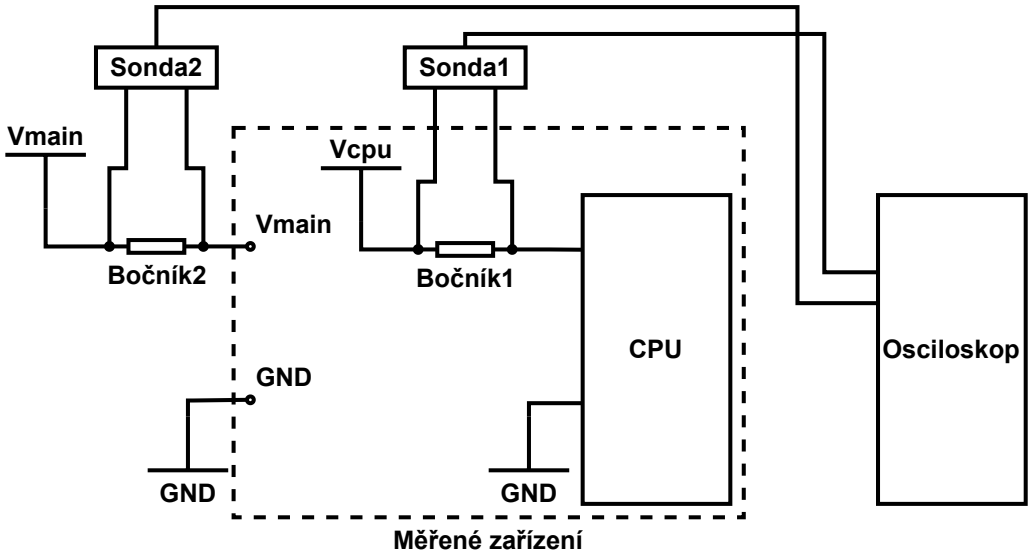
Metoda měření proudové spotřeby s použitím měřícího bočníku je založena na upraveném vztahu z Ohmova zákona [10]:

$$I = \frac{U}{R} \tag{2.1}$$

Jako měřící bočník volíme vysoce stabilní rezistor nízkého odporu, který je zapojen do série s měřeným zařízením, případně přímo s integrovaným obvodem. Připojení bočníku přímo k integrovanému obvodu je preferovanější a přesnější, protože tato varianta odstraní nepřesnosti, které vznikají fungováním ostatních součástek celého zařízení. Odpor měřícího bočníku bývá v řádu jednotek mΩ do maximálně jednotek Ω, v závislosti na měřeném zařízení. Jelikož bočníkem přímo protéká proud, musí být bočník dostatečně teplotně a výkonově odolný. Při znalosti přesné rezistivity bočníků, ji můžeme dosadit do vztahu 2.1. Na bočníku pomocí osciloskopu skrze sondu změříme úbytek napětí. Z těchto dvou veličin je jednoduše vypočítatelný proud, který prochází bočníkem a kryptografickým zařízením, případně integrovaným obvodem. Nízká rezistivita bočníku ovlivňuje celkový proud obvodem jen minimálně, ale je možné ji výpočtem pomocí Kirchhoffova zákona odstranit. Výsledná hodnota je rovna proudu, které kryptografické zařízení, případně samotný integrovaný obvod při svém fungování odebírá [11].

Z principu je tato metoda nejjednodušší a nejlevnější variantou samotného měření. Metoda vyžaduje přímý přístup k měřenému zařízení či integrovanému obvodu a manipulaci s napájecími kontakty. Mezi napájecí kontakty zařízení nebo integrovaného obvodu je nutné vložit do série měřící bočník, kterým bude procházet proud, a na jehož svorky poté připojíme měřící sondu. Změřený úbytek napětí na bočníku a jeho rezistivitu dosazujeme do vztahu 2.1.

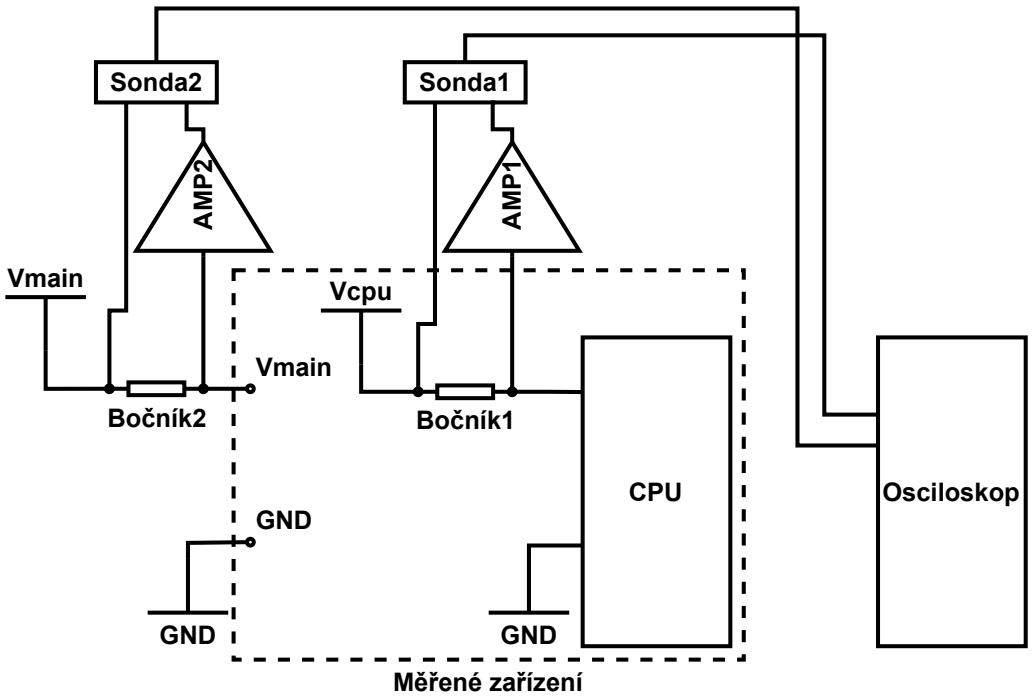
Schéma na obrázku 2.1 zobrazuje model měření s měřícím bočníkem na napájecí části samotného integrovaného obvodu (Sonda1 a Bočník1) i na napájecí části celého měřeného zařízení (Sonda2 a Bočník2). Z obrázku 2.1 je patrné, že měření probíhá na samotné napájecí větvi integrovaného obvodu případně zařízení. Principiálně by bylo možné měřit i na zemnící větvi (ve schématu svorky GND). Varianta s měřením na zemnící větvi by byla výhodná, pokud by integrovaný obvod měl více napájecích svorek pro různé hodnoty napětí a společnou zemnící svorku. V takovém případě by úbytek napětí na bočníku byl závislý na celkovém proudu, který zařízením prochází ze všech napájecích svorek. Nevýhodou by bylo, pokud by integrovaný obvod měl více zemnících kontaktů. V takovém případě by bylo nutné měřit úbytek napětí na každém z nich.



Obr. 2.1: Schéma měření proudu měřícím bočníkem.

2.2.2 Měření s napěťovým zesilovačem

Tato metoda je založena na měření bočníkem podobně jako na obrázku 2.1. Rozdíl této metody je přidání napěťového zesilovače na svorky měřícího bočníku. Nové schéma je patrné na obrázku 2.2. V závislosti na velikosti rezistivity bočníku nemusí být úbytek napětí na bočníku přesně měřitelný. Z tohoto důvodu je na jeho svorky připojen operační zesilovač s poměrem zesílení např. 1/10 nebo 1/100 v závislosti na úbytku napětí, které je na bočníku sondou měřitelné. Požadavek na napěťový operační zesilovač je vysoká stabilita a přesný poměr zesílení [12].



Obr. 2.2: Schéma měření proudu se zesilovačem.

2.2.3 Hallova sonda

Metoda měření Hallovou sondou je založena na Hallově jevu, který objevil americký fyzik Edwin Hall. Hallův jev vzniká jako rozdíl napětí na stranách elektrického vodiče i polovodiče, kterým prochází elektrický proud za působení vnějšího magnetického pole. Hall pozoroval, že pokud vodičem prochází elektrický proud, elektrony se pohybují v rovině. Pokud však na vodič, kterým prochází elektrický proud, působí vnější magnetické pole, elektrony ve vodiči jsou vychýleny magnetickým polem ze své dráhy pohybu.

Na elektrony v takovém případě působí tzv. Lorentzova síla [13]:

$$F_L = e(E_e + v_n \times B), \text{kde } e = -q \text{ pro elektrony} \tag{2.2}$$

e je náboj částic, $\mathbf{E_e}$ je elektrické pole, $\mathbf{v_n}$ rychlost částic a **B** intenzita působícího magnetického pole.

Toto vychýlení elektronů vyvolá na stranách vodiče rozdílný elektrický potenciál, a tím vzniká Hallovo napětí. Hallovo napětí kompenzuje vliv Lorentzovy síly, která působí na elektrony v magnetickém poli. Toto vzniklé napětí je měřitelné a je úměrné síle působícího magnetického pole. Pro Hallovo napětí tak obecně platí [14]:

$$U_H = \frac{R_H}{d}\ I_C\ B \tag{2.3}$$

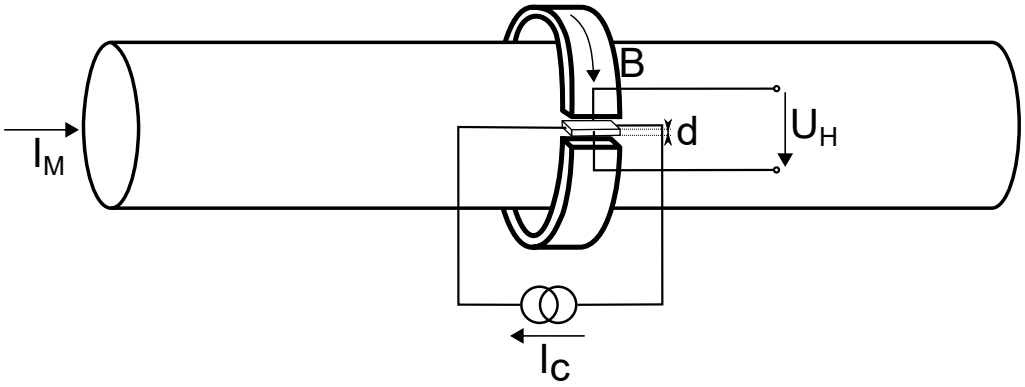
$\mathbf{R_H}$ je Hallův koefiecient, **d** tloušťka vodiče, $\mathbf{I_C}$ proud vodičem a **B** intenzita působícího magnetického pole.

Hallova sonda je zařízení, které je uzpůsobené k měření Hallova jevu. Zpravidla se jedná o úzkou polovodičovou destičku, kterou prochází v jedné ose proud, a v druhé ose jsou připevněny elektrody pro měření Hallova napětí.

Toto zařízení lze použít k mnoha aplikacím. Jednou z nich je právě měření proudu. Zařízení pro takové měření se skládá z Hallovy sondy, která je převážně obklopena neuzavřeným prstencovým jádrem, které poskytne magnetické pole. Princip této metody vychází z fyzikálního jevu, kdy při průchodu elektrického proudu vodičem vzniká přiměřené magnetické pole. Vodič je umístěn do středu prstencového jádra. Pokud vodičem prochází proud, vytváří kolem sebe magnetické pole, které je posíleno jádrem. Vzniklé magnetické pole působí na destičku, kterou prochází řídící proud. V tomto okamžiku dojde v destičce k Hallově jevu a na měřících kontaktech se projeví Hallovo napětí [15].

Výhodou této metody je absence přidaného odporu v hlavním měřeném obvodu, který by narušil přesnost měření. Nevýhodou může být nepřesnost měření, kterou může způsobit přítomnost parazitního magnetického pole, které ovlivní měření.

Na obrázku 2.3 je zobrazen model jedno–osé Hallovy sondy. Polovodičovou destičkou o tloušťce d protéká řídící proud I_C. Pokud měřeným vodičem protéká proud I_M, vytváří spolu s jádrem magnetické pole o intenzitě B. Působením Hallova jevu na měřících svorkách destičky vzniká Hallovo napětí U_H.

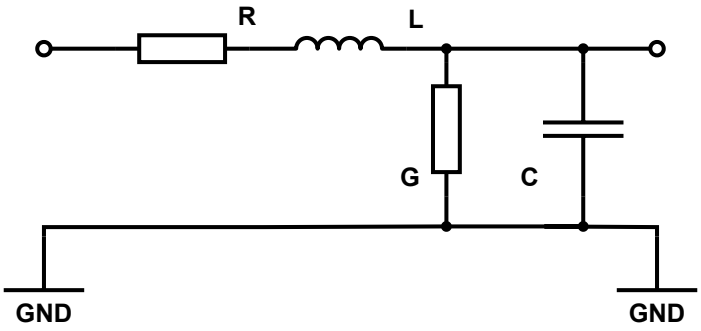


Obr. 2.3: Model jedno–osé Hallovy sondy.

2.2.4 Měření s elektromagnetickou cívkou

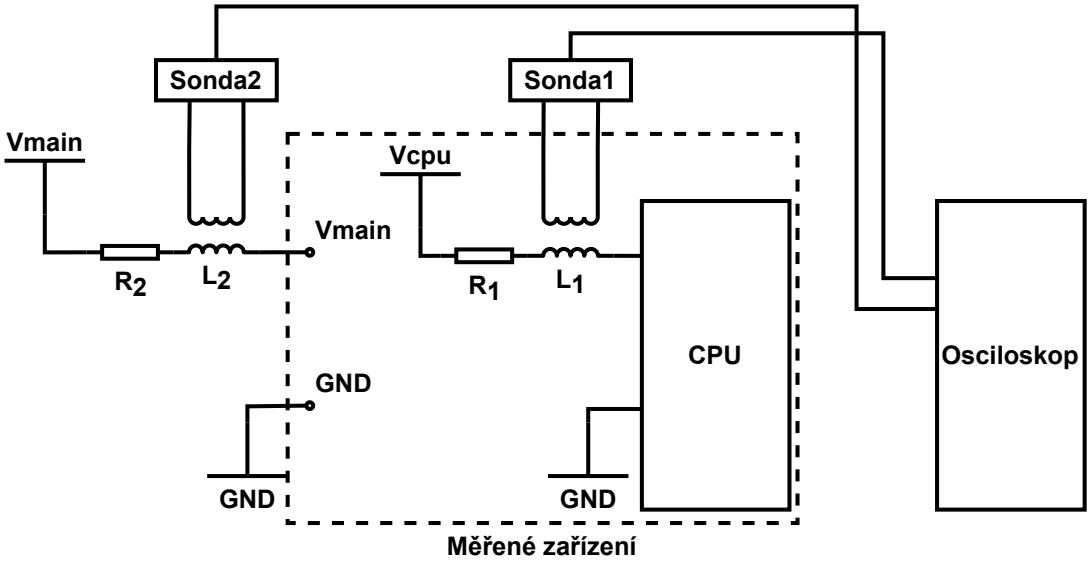
Metoda měření proudu elektromagnetickou cívkou je založena na jevu elektromagnetické indukce. Tento jev se projevuje tak, že při průchodu proudu vodičem je indukováno elektromagnetické pole kolem vodiče, a analogicky pokud je vodič vystaven působení elektromagnetického pole, je na tento vodič indukováno napětí. Každý reálný vodič lze charakterizovat pomocí parametrů rezistivity, parazitní kapacity a parazitní indukčnosti. Právě indukčnost udává závislost chování vodiče při průchodu elektrickým proudem a vliv elektromagnetického pole na vodič. Jevu elektromagnetické indukce využívá široké spektrum elektronických zařízení a prvků, jako např. transformátory, antény, různé senzory, elektrické generátory, elektromotory atd. Nejčastěji tento jev využívají obecně cívky. Cívka je v principu zařízení uzpůsobené k maximálnímu využití jevu elektromagnetické indukce. V podstatě se jedná o vodič, který je navinut na magnetické jádro, které zvyšuje účinnost vyzařování nebo přijímání elektromagnetické indukce. Tato obecná konstrukce má za cíl maximalizaci indukčnosti zařízení a minimalizaci rezistivity a parazitní kapacity [16].

Reálný elektrický vodič můžeme nahradit modelovým schématem vedení, kde jsou znázorněny parametry elementárního úseku vedení. Parametry tohoto modelu jsou rezistivita (**R**), indukčnost(**L**), vodivost(**G**) a kapacita(**C**), které působí vždy na jednotku délky [17]. Schéma na obrázku 2.4 zobrazuje modelový případ reálného vedení, kde jsou parametry vedení reprezentovány elektronickými součástkami ve schématu. Každé reálné vedení lze pro účel zjištění elektrických parametrů nahradit tímto schématem.



Obr. 2.4: Náhradní schéma vedení.

Každý vodič lze popsat modelových schématem vedení. Tohoto modelu lze využít při měření elektromagnetickou cívkou. Pokud vodičem prochází proud, tak skrze vlastní indukčnost vyzařuje elektromagnetické pole, které na měřící elektromagnetické cívce indukuje napětí. Měřící schémata 2.1 a 2.2 upravíme do podoby pro měření elektromagnetickou cívkou tak, že vedení napájecí větve zařízení nebo integrovaného obvodu nahradíme částí modelového schéma vedení 2.4. Schéma na obrázku 2.5 zobrazuje upravené napájecí vedení zařízení nebo integrovaného obvodu. Vlastní indukčnost napájecího vedení je reprezentována cívkami L_1 resp. L_2. Tato vlastní indukčnost indukuje napětí do měřících cívek, kde je skrze sondy měřeno.



Obr. 2.5: Schéma měření s elektromagnetickou cívkou.

Výhodou této metody je možnost bezkontaktního měření, kdy není třeba jakkoliv zasahovat do integrity zařízení nebo integrovaného obvodu a jeho kontaktů. Cívku lze umístit do blízkosti kontaktu integrovaného obvodu a měřit procházející proud bez nutnosti manipulace s vlastním obvodem. Do obvodu není přidán žádný dodatečný odpor, žádné měřící svorky nebo kontakty, stačí pouze překonat případná stínění, kterým může být zařízení chráněno. Velkou nevýhodou této metody je vysoká náchylnost na rušení. Samotné vedení má velmi nízké hodnoty indukčnosti a tak indukované napětí do měřící cívky je sotva rozeznatelné od šumu.

3 Nástroje a zařízení pro měření proudové spotřeby

3.1 Osciloskop

Osciloskop je elektrotechnické zařízení, které umožňuje přesné měření elektronických proudových a napěťových průběhů a frekvenční analýzu měřeného signálu. Obecně je osciloskop pravděpodobně nejpoužívanější zařízení a vybavení každé laboratoře pro měření elektronických parametrů. První osciloskop byl založen na analogové technologii obrazovky katodové trubice, kde byl na stínítko promítán obraz měřeného signálu, který nešlo téměř nijak upravovat. Dnešní osciloskopy jsou plně digitalizované a umožňují podstatně preciznější měření a analýzu signálu. Vzorkování a kvantování signálu, které zanášejí do měřené části určité nepřesnosti, je vyváženo řadou funkcí, které moderní osciloskopy nabízejí. Například se může jednat o různé analýzy, možnosti nastavení spouští pro měření, zmrazení signálu, měřící kurzory atd. Osciloskop je běžně obsluhován přes řídící panel, na kterém lze pomocí funkčních tlačítek a otočných spínačů upravovat obraz měřeného signálu a interagovat s různými funkcemi osciloskopu. Digitální osciloskopy rovněž umožňují určitou automatizaci celého procesu měření. Osciloskop může podporovat automatickou konfiguraci, ukládání a odesílání měřených dat do počítače, který je k osciloskopu připojen dostupným rozhraním. Tato automatizace může nahradit fyzickou obsluhu osciloskopu a provádět dlouhotrvající měření, nebo měření na dálku, bez přítomnosti lidské obsluhy [18].

Komunikace s osciloskopem probíhá v závislosti na výrobci skrze rozhraní s definovaným standardem IEEE 488, nebo proprietárním protokolem, který si výrobce pro svá zařízení definuje. Standard IEEE 488.2 definuje kromě vlastní komunikace i *standardní příkazy pro programovatelné přístroje – Standard Commands for Programmable Instruments* (SCPI). SCPI jsou textově orientované příkazy, které umožňují konfiguraci osciloskopu. [19]

3.2 Vývojová deska SAKURA

Projekt *Side-channel AttacK User Reference Architecture* (SAKURA) byl založen v roce 2012 ve spolupráci s japonskou *The University of Electro-Communications* (UEC)[1], která zastřešuje centrum SATOH Lab.[2] a společností MORITA TECH

[1] `https://www.uec.ac.jp/eng/`

[2] `https://satoh.cs.uec.ac.jp/en/index.html`

CO., LTD.[3], která se od roku 1993 věnuje vývoji a návrhu elektronických zařízení. Projekt SAKURA je nástupce projektu *Side-channel Attack Standard Evaluation BOard* (SASEBO), který v roce 2007 odstartovala UEC a AIST[4]. Původní projekt SASEBO jako reakce na každodenní používání kryptografických zařízení, a přesto rostoucí trend bezpečnostních hrozeb jako např. datové úniky a podvržení informací. Pro uživatele těchto systémů bylo obtížné zjistit, zda jsou kryptografické algoritmy implementovány s dostatečnou mírou bezpečnosti. Není výjimkou, aby byly nalezeny bezpečnostní hrozby v zařízeních, které výrobce prezentuje jako perfektně bezpečné. Projekt SASEBO obsahoval několik typů vývojových desek, které obsahovaly různé druhy FPGA obvodů. Jmenovitě se jednalo o FPGA obvody firem Xilinx®[5] a ALTERA®, kterou odkoupila společnost Intel Corp.®. V rámci projektu byl vytvořen i vlastní integrovaný LSI obvod, který implementuje kryptografické algoritmy.

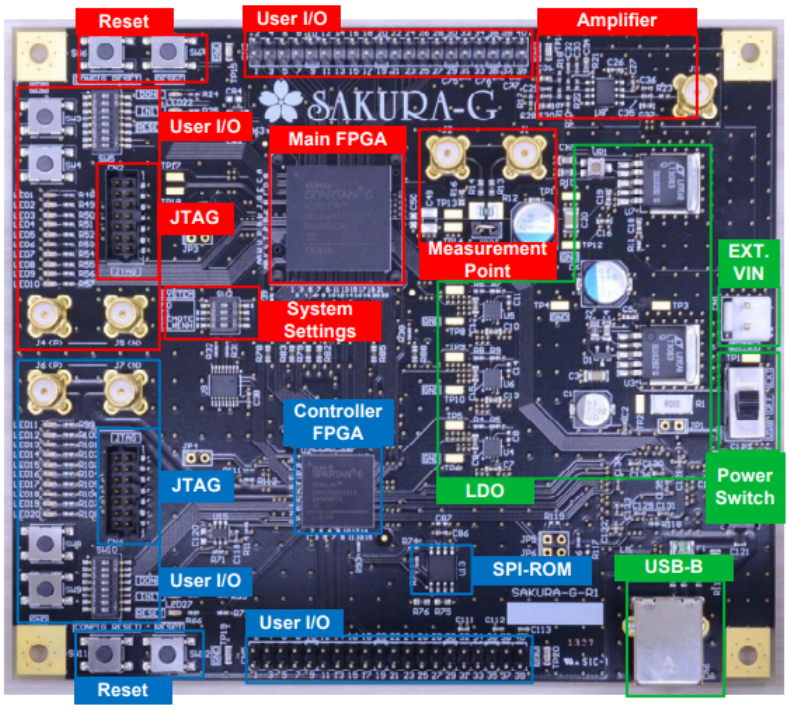
Projekt SAKURA je pokračování projektu SASEBO, kdy většina hardwaru a softwaru jednoduše přešla pod nový projekt. Z již vyvinutých byly odvozeny desky nové, které od roku 2016 prodává společnost TROCHE Co.,Ltd[6]. Jedná se především o desky SAKURA–G, SAKURA–W, sadu SAKURA–G/W a SAKURA–X. V projektu jsou i další desky, jejichž podpora a prodej byly ukončeny. Rovněž byl ukončen prodej vlastních LSI obvodů a jejich použití na deskách.

Vývojové desky jsou navrženy tak, aby zjednodušily a umožnily analýzu hardwarové zranitelnosti. Desky zpravidla obsahují jeden nebo dva integrované obvody, které provádí potřebné kryptografické operace a příslušné části, které zajišťují fungování integrovaných obvodů. Dále jsou na deskách přítomny již vyvedené potřebné kontakty pro různá měření, která mohou být pro analýzu nezbytná. Jmenovitě je na desce přítomný měřící bočník a zabudovaný zesilovač pro měření proudové spotřeby.

SAKURA–G

Deska SAKURA–G obsahuje 2 integrované FPGA obvody typu Xilinx Spartan-6. Vychází z původní desky SASEBO–GII, na které byly nahrazeny starší obvody Xilinx Virtex-5. Jeden integrovaný FPGA obvod slouží jako hlavní řadič desky a druhý implementuje kryptografické algoritmy. Nízkošumový návrh a uspořádání desky usnadňují případné analýzy. Deska je navržena k analýze pro SCA, *útok zavedením chyb – Fault Injection Attack* (FIA) a *fyzicky neklonovatelné funkce – Physical Unclonable Function* (PUF). Rovněž může být použita jako mateřská deska pro usazení desky SAKURA–W k práci s čipovými kartami [20].

[3] `https://morita-tech.co.jp/en/`
[4] `https://www.aist.go.jp/index_en.html`
[5] `https://www.xilinx.com/`
[6] `http://www.troche.com/sakura/`



Obr. 3.1: Blokové schéma desky SAKURA–G. Převzato z manuálu k desce [20].

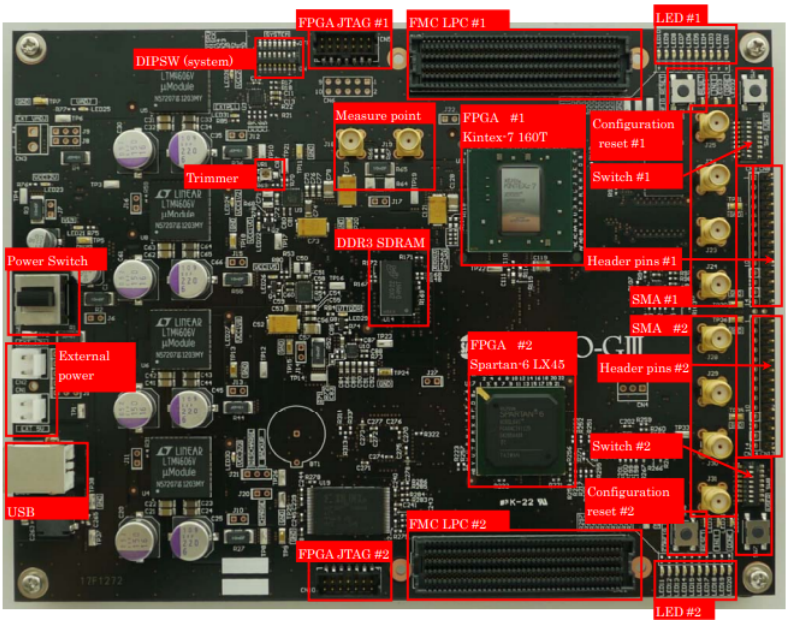
SAKURA–W

Deska SAKURA–W je navržena k analýze chování čipových karet (IC card). Deska je navržena jako modul k usazení na desku SAKURA–G. SAKURA–W není navržena k samostatnému použití bez desky SAKURA–G. K desce je navíc poskytována předprogramovatelná čipová karta s čipem Atmel ATMega8515, která již obsahuje binární implementaci algoritmů DES a AES. Tento modul rozšiřuje možnosti měření proudové spotřeby čipových karet při provádění kryptografických operací [21].

Obr. 3.2: Usazení desky SAKURA–W na desce SAKURA–G. Převzato z manuálu k desce [21].

SAKURA–X

Deska SAKURA–X je komerční označení desky SASEBO–GIII, která je nástupcem desky SASEBO–GII, ze které vychází SAKURA–G. SAKURA–X je zatím nejpokročilejší deskou v projektu SAKURA. Deska obsahuje 2 FPGA obvody. Jako hlavní řadič slouží obvod Xilinx Spartan-6 a pro implementaci kryptografických algoritmů slouží novější obvod Xilinx Kintex-7. Na desce je k dispozici integrovaná DDR3 SDRAM paměť s kapacitou 1 GB. Rovněž je zajištěna podpora připojení až dvou rozšiřujících FMC LPC karet [22].



Obr. 3.3: Blokové schéma desky SAKURA–X. Převzato z manuálu k desce [22].

3.3 Program

Programem lze obecně označit jakýkoliv soubor instrukcí a příkazů, které jsou určeny k interpretaci a vykonání počítačem. Tento soubor instrukcí a příkazů je zpravidla napsán programátorem ve zvoleném programovacím jazyce. Tento textový soubor je poté zpravidla přeložen a zkompilován do kódu, který dokáže počítač implementovat a vykonat. Přesný postup přeměny textového souboru do počítačem vykonatelné podoby se liší v závislosti na programovacím jazyku a cílovém počítači, pro který je program vyvíjen. Jednotlivé programovací jazyky se dělí podle různých vlastností do mnoha skupin. Velmi důležité dělení programovacích jazyků je na kategorii imperativních a deklarativních. Deklarativní jazyky se zaměřují na konkrétní cíl, kterého má program dosáhnout, nikoliv na podrobnou proceduru, kterou musí vykonat. Imperativní jazyky jsou zaměřeny na proceduru a algoritmus, který je nutné

vykonat, aby bylo dosaženo očekávaného výsledku. Většina programovacích jazyků jsou zařazeny do kategorie imperativních. V těchto programovacích jazycích programátor konkrétně specifikuje přesný postup, který musí počítač vykonat. Jedním z dalších možných dělení programovacích jazyků může být podle míry poskytnuté abstrakce řešení problému na jazyky funkční, skriptovací, von Neumannovy a objektově orientované. Jednotlivé kategorie dělení programovacích jazyků nejsou vždy přesně exaktní a často jsou předmětem debat, zda daný jazyk spadá do jedné, či druhé kategorie [23, 24].

3.3.1 Programovací jazyk C#

C# je objektově orientovaný jazyk navržený dánským softwarovým inženýrem Andersem Hejlsbergem ve společnosti Microsoft v roce 2000. Jazyk je klíčovou součástí softwarové platformy .NET. Jazyk C# je dnes stále vyvíjen a dostupný na platformách, které podporují použití .NET. V posledních letech dochází k značnému rozšiřování podpory platformy .NET a dokonce vytvoření několika open-source implementací pro multiplatformní podporu. Jedná se o velmi moderní, víceúčelový objektově orientovaný jazyk s širokým spektrem využití. C# poskytuje plnou podporu externích předkompilovaných knihoven, které implementují řešení běžných podproblémů. Pro jazyk C# existuje velké množství těchto knihoven, podstatnou část z nich poskytuje samotná společnost Microsoft [25].

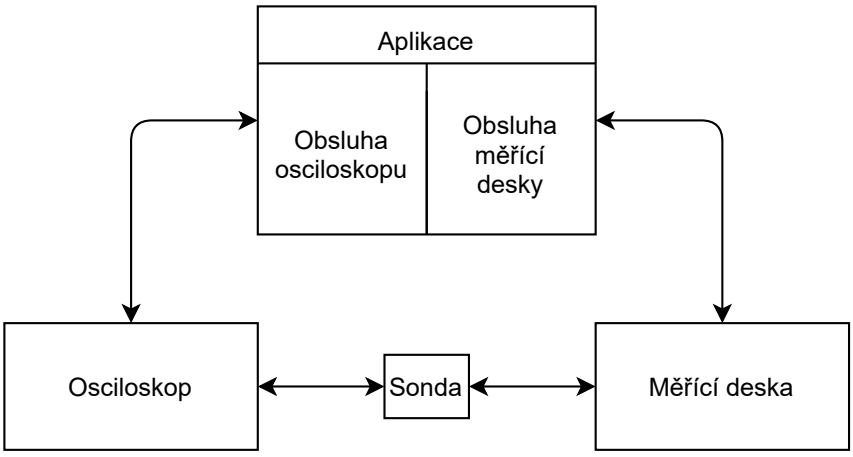
4 Aplikace P–SCAAT

Aplikace pro realizaci automatizovaného měření proudové spotřeby je napsána v jazyce C# ve verzi 7.3 jako desktopová aplikace s GUI cílenou na platformu .NET Framework 4.8. Přestože existují novější verze programovacího jazyka C#, je aplikace napsaná ve starší verzi 7.3. Novější verze jsou podporované v platformách .NET Core, které umožňují psaní multiplatformních aplikací. V aplikaci použité knihovny jsou však podporovány pouze na platformách .NET Framework. Z tohoto důvodu je aplikace cílena na tuto platformu a tím pádem pouze na operační systémy Microsoft Windows [26] [27].

Aplikace nese název *Power Side-Channel Attack Analysis Tool* (P–SCAAT), čteno „pískat“. Jelikož aplikace obsahuje GUI je vhodné, aby běh a odezva aplikace byla co nejméně ovlivněna vstupně-výstupními operacemi, které je nutné provádět. Při běhu je důležité, aby GUI zůstalo co nejvíce nezávislé na operacích, které probíhají na pozadí. Výpočetně nebo časově náročné operace jsou proto v maximální míře prováděny v oddělených vláknech, které neovlivňují odezvu GUI. Schéma aplikace je na obrázku 4.1

Dále je kladen důraz na modulárnost výsledné aplikace tak, aby ji bylo možné jednoduše dále rozšířit o nové funkce. Proto byl zvolen programovací jazyk C#, který je objektově orientovaný. Právě správné použití principů objektově orientovaného programování by mělo zajistit modulárnost aplikace.

Vývoj probíhal v prostředí Microsoft Visual Studio Enterprise 2019 s instalovanou verzí .NET Framework 4.8. Pro podporu vývoje byla použita repozitářová služba GitHub[1]



Obr. 4.1: Základní návrh aplikace.

[1] `https://github.com/`

Při návrhu a implementaci aplikace je rovněž kladen důraz na dodržení principu návrhu SOLID a především MVVM. Inspirací pro využití těchto zdrojů, návrh a implementaci byly znalosti z předmětu Programování v .NET a C#[2] a výukové materiály uživatelů SingletonSean[3] a IAmTimCorey[4].

4.0.1 Komunikace s vývojovou deskou

Pro komunikaci s vývojovou deskou je použita systémová knihovna *System.IO.Ports*, která poskytuje rozhraní pro práci a synchronní komunikaci přes sériový port, kterým je zařízení připojeno. Tato knihovna umožňuje otevření sériového portu s použitím parametrů jména rozhraní, modulační rychlosti, parity, velikosti datových jednotek a stop bitů. Pro správně fungující komunikace s použitým kryptografickým zařízením jsou tyto parametry nezbytné.

Ke generování náhodných dat, které jsou desce odesílány jako vstupní data je použit kryptografický *generátor náhodných čísel – Random Number Generator* (RNG) z nativního jmenného prostoru *System.Security.Cryptography* a příslušné knihovny *System.Security.Cryptography.Algorithms.dll*. Z této knihovny je použita třída *RandomNumberGenerator*, která generuje náhodné hodnoty a ukládá je do pole bytů zvolené délky. Takto je zajištěna generace vstupních dat potřebné délky pro kryptografické zařízení.

4.0.2 Komunikace s osciloskopem

Komunikace s osciloskopem je zajištěna pomocí knihoven NI.VISA ve verzi 21.0.0.-49304 a IVI.VISA ve verzi 5.11.0.0 od společnosti *National Instruments* (NI) a *Interchangeable Virtual Instruments* (IVI) *Foundation*. Tyto knihovny umožňují navázat textově orientovanou relaci s osciloskopem a odesílat SCPI příkazy, na které osciloskop reaguje. Konkrétně se z knihoven jedná o třídu *MessageBasedSession*. Takto je zajištěno nastavení osciloskopu před začátkem samotného měření a i zpětná komunikace osciloskopu pro odesílání změřených výsledků do počítače. [28]

[2] `https://www.vut.cz/studenti/predmety/detail/242605`
[3] `https://github.com/SingletonSean/`
[4] `https://iamtimcorey.com/`

4.1 Návrhový vzor SOLID

SOLID je zkratkou pro 5 návrhových principů software, kterou zavedl v roce 2004 Michael Feathers v reakci na práci Roberta C. Martina [29]. Maximální možné dodržení těchto principů poskytuje výhody v modulárnosti a přehlednosti výsledného kódu i celého programu [30, 31, 32].

Single-responsibility

Každá třída má pouze jednu jedinou zodpovědnost. Jednotlivé části fungování aplikace jsou spravovány pouze jedinou třídou. Příkladem může být třída, která pouze otevírá komunikační rozhraní s osciloskopem. Pokud lze činnost třídy popsat jedinou větou, je navržena v duchu tohoto principu.

Open–close

Programové části jsou otevřené rozšíření, ale uzavřené modifikaci. Třídy lze libovolně rozšiřovat o další metody, vlastnosti a funkce bez nutnosti zásahu do již existujících částí. Tento princip má za cíl maximální možné zobecnění rozhraní kódu.

Liskov substitution

Poděděné třídy musí být zaměnitelné za třídy rodičovské. Třídy, které jsou děděné z rodičovských tříd nesmí obsahovat méně funkcí, než samotné rodičovské třídy. Rodičovské třídy pak mohou být nahrazeny děděnými třídami bez nutnosti zásahu do jejich kódu.

Interface segregation principle

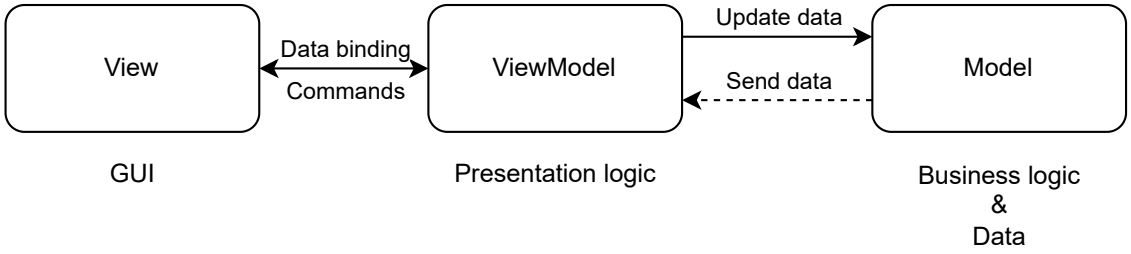
Třída, která implementuje rozhraní by neměla být závislá na metodách, které nepotřebuje. Rozhraní by měla být cíleně zaměřená na specifické úkoly a neobsahovat zbytečně množství metod, které implementující třídy nevyužívají.

Dependency inversion principle

Obrácený princip závislostí požaduje, aby konkrétní záviselo na obecném a nikdy ne naopak. Vyšší míra abstrakce nesmí být závislá na nižší. Zjednodušeně pokud navrhujeme závislou třídu, měla by být závislá na třídě abstraktnější, nikoliv konkrétnější.

4.2 Návrhový vzor MVVM

MVVM neboli Model–View–ViewModel je velmi populární návrhový vzor, zejména pokud se jedná o aplikace s GUI. MVVM podstatně zjednodušuje a zpřehledňuje psaní kódu tak, že odděluje části zodpovědné za prezentaci dat uživateli skrze GUI od částí zodpovědné za zpracování aplikačních dat. Jinými slovy MVVM vytváří části konkrétně zodpovědné za zpracování dat, převedení dat do prezentovatelné podoby a jejich vlastní prezentaci uživateli. Vice versa poskytuje uživateli přehledné prostředí pro interakci s aplikací prostřednictvím prezentační části, která předává data částím, které je dále upravují a zpracovávají pro konkrétní potřeby uživatele. Jednoduché schéma tohoto návrhového vzoru je patrné na obrázku 4.2, kde jsou jednotlivé části zobrazeny včetně principů předávání dat mezi nimi [33].



Obr. 4.2: Schéma návrhového vzoru MVVM.

K použití tohoto návrhového vzoru přímo vybízí aplikace psané v *Windows Presentation Foundation* (WPF), což je framework pro vývoj desktopových aplikací s GUI. Tento framework poskytuje funkce, které dovolují vytvořit moderní GUI, skrze které uživatel interaguje s aplikací. Framework zahrnuje funkce použití *Extensible Application Markup Language* (XAML), řízení vrstev, ovládací prvky, provázání aplikačních dat, použití šablon, stylů a mnoho dalšího [34]. Zároveň použití WPF téměř eliminuje nutnost psaní tzv. code-behind. Code-behind je v aplikacích vysoce nežádoucí, protože prohlubuje přímou závislost mezi GUI a vlastní aplikací, což komplikuje případné rozšíření, či znovupoužití některých částí aplikace.

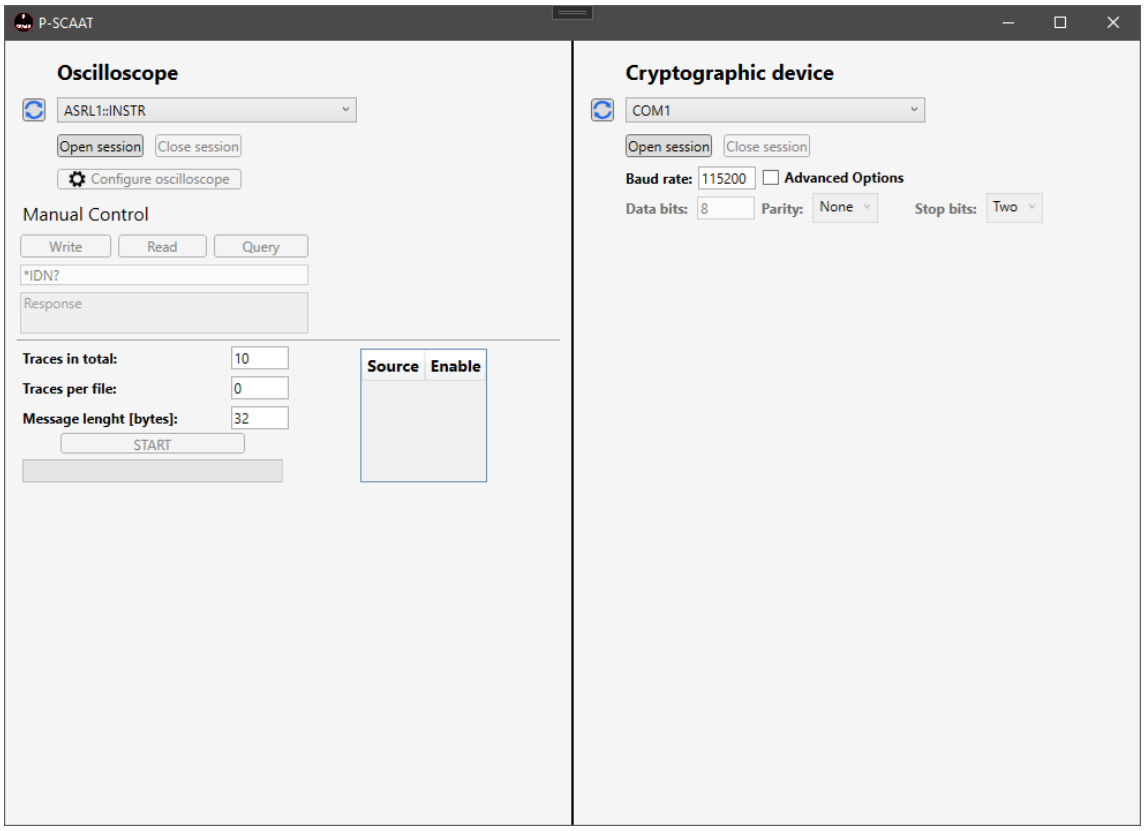
MVVM je poměrně robustní a komplexní. Z tohoto důvodu se nehodí pro jednoduché a jednoúčelové aplikace, kde použití tohoto návrhového vzoru přináší jen nechtěnou složitost. Nicméně do komplexnějších aplikací přináší organizaci a strukturu kódu pro maximalizaci přehlednosti, rozšiřitelnosti a udržovatelnosti aplikace [35]. Při vývoji byly brány v potaz i jiné principy. Například pravidlo tří nebo princip 90–90.

4.3 Vlastní implementace v MVVM

Při návrhu a vývoji aplikace jsem se snažil maximalizovat využití MVVM vzoru, aby byla aplikace přehledná a případně rozšiřitelná. Aplikaci jsem dle tohoto vzoru rozdělil do 3 hlavních částí.

4.3.1 Část View

View je zodpovědné za čistou prezentaci dat a interakci s uživatelem prostřednictví GUI. Tato část obsahuje 3 části a je zde použit značkovací jazyk XAML z WPF. Aplikaci jsem navrhoval tak, aby fungovala pouze v jediném okně a uživatel měl nad celou aplikací kontrolu právě v tomto jediném okně. Chtěl jsem se vyhnout přehnanému vytváření a zavírání několika samostatných oken, které by přispívaly k fungování jediné aplikace. Náhled hlavního okna je zřejmý z obrázku 4.3 Jedinou výjimkou jsou okna pro uložení či načtení souboru a vyskakovací okna s upozorněním pro uživatele. Jinak vše běží v jediném okně. Samotné okno aplikace je rozděleno vertikálně na dvě podčásti. Levá podčást uživateli umožňuje práci s osciloskopem a měřením jako takovým. V této části uživatel vybírá, se kterým osciloskopem si přeje pracovat. Po navázání příslušné relace je uživateli umožněno tuto levou část přepnout do konfigurační nabídky, kde lze pomocí aplikačního GUI nakonfigurovat některé parametry osciloskopu. Při navázání relace je však možno bez jakéhokoliv zásahu do konfigurace osciloskopu začít měřit. Základní část obsahuje kromě ovládacích prvků pro relaci s osciloskopem také nastavení pro samotné měření. Pravá část aplikačního okna je zodpovědná za prezentaci dat spojenými s měřeným kryptografickým zařízením. Jelikož se v tomto případě jedná o jednoduchý výběr sériového portu, navázání a ukončení relace a minimální konfigurace, je tato část neměnná a nemá žádnou přepínatelnou nabídku.

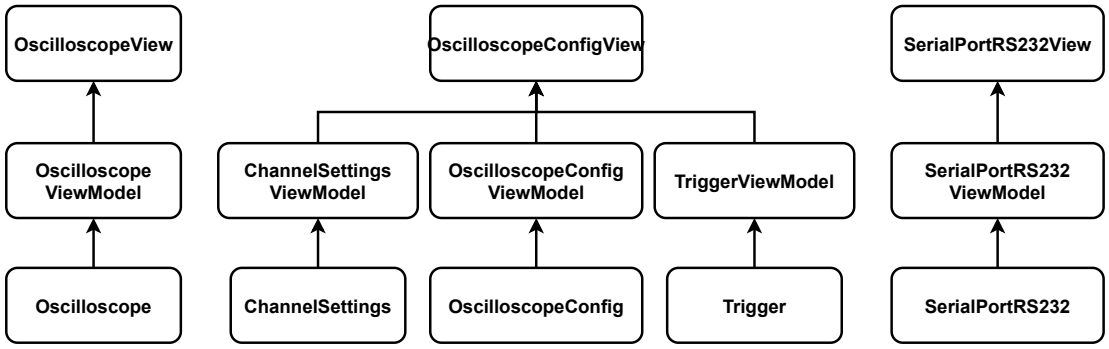


Obr. 4.3: Hlavní okno aplikace P–SCAAT.

4.3.2 Část ViewModel

ViewModel část připravuje data pro zobrazení uživateli. Tato část aplikace je nejobsáhlejší, co se týče čistého kódu. Zde se nacházejí třídy, které reprezentují téměř všechny funkce aplikace. Tato část rovněž zpravidla obsahuje části kódu, které se starají o správné fungování a zobrazování dat v případě, že na nich dojde ke změně. O této změně, ať už byla způsobena aplikací nebo uživatelem, je nutné informovat aplikační GUI, aby uživateli byly zobrazeny aktuální data. S těmito změnami se může pojit další režie, kterou je třeba nad daty provádět. Konkrétním příkladem takové změny může být právě změna konfigurace osciloskopu skrze GUI. Data, která uživatel ve View změnil, jsou provázána na vlastnosti ve ViewModelech. Změna konfigurace osciloskopu se vždy pojí vytvořením patřičného SCPI příkazu a jeho zařazení do seznamu příkazů, které budou osciloskopu při potvrzení konfigurace předány. Změny jsou rovněž kontrolovány tak, aby nedocházelo k chybným vstupům ze strany uživatele. Tato část je tak jakousi abstrakcí komunikace mezi vstupy uživatele přes View a zpracování reálných aplikačních dat v části Model. Zde rovněž implementuji několik příkazů, které jsou provázány na ovládací prvky v GUI. Těmito příkazy uživatel vy-

nutí změnu aplikačních dat Model. Obecně by tato abstrakce měla umožnit použití jakékoliv formy GUI s minimálními úpravami pro stejnou aplikaci. Vlastní logika aplikace je tak nezávislá na zobrazované formě a univerzálně použitelné s libovolným GUI. Schéma provázání jednotlivých Model ViewModel a View je na obrázku 4.4



Obr. 4.4: Základní MVVM schéma aplikace P–SCAAT.

V rámci běhu aplikace jsou jednotlivé ViewModely vytvářeny a zahazovány dle aktuálních potřeb aplikace a právě zobrazených dat. Tento postup šetří zdroje, které aplikace při svém běhu potřebuje. Jinými slovy, pokud není zobrazeno konkrétní View, není důvod vytvářet konverzi a kontrolu dat z Modelu do tohoto View. Zde je však důležité při zahození ViewModelu opravdu uvolnit všechny reference, které ViewModel vytvořil. Pokud k tomuto uvolnění nedojde, objekt není systémem úplně odstraněn a zůstává uchován v paměti. Při delším běhu aplikace toto chování může vytvářet závažné problémy se správou paměti. Vyčištění referencí ViewModelu, který má být zahozen probíhá z větší části automaticky, jediné co je nutné v rámci kódu dělat manuálně je přihlašování a odhlašování odběrů oznámení o změnách vlastností. Jelikož aplikační GUI musíme aktualizovat při každé změně, je třeba program o těchto změnách informovat. Toho MVVM dosahuje pomocí přihlašování k odběru události pro změnu vlastnosti. A vyvolání této události při každé změně dané vlastnosti.

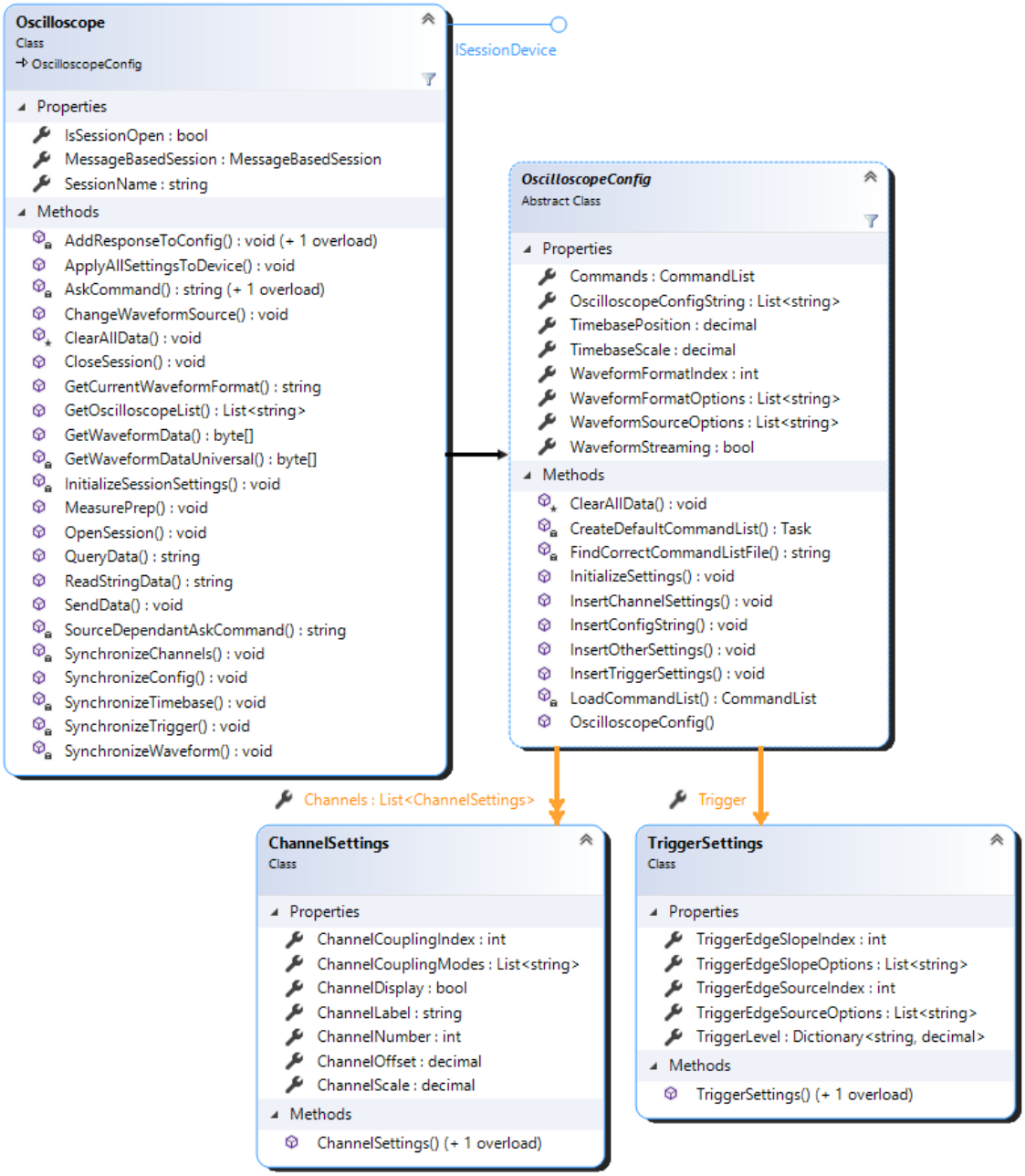
Jednotlivá View a ViewModely jsou prezentovány jediným oknem aplikace, které je rozděleno na části. Třída v příloze B.3 poskytuje způsob, jak v rámci jediného okna pracovat s více třídami najednou.

4.3.3 Část Model

Model obsahuje celé pozadí aplikace. Jedná se o reprezentaci a abstrakci reálných vlastností a objektů. V rámci této aplikace to je například třída, která reprezentuje osciloskop, jeho konfiguraci, konfiguraci jednotlivých kanálů, příkazovou sadu osciloskopu apod. Modely implementují konkrétní metody, jak s reálnými objekty aplikace

interaguje. Jedná se o metody pro otevírání a uzavírání komunikačních relací, posílání a čtení dat z osciloskopu nebo měřeného zařízení. V rámci modelu osciloskopu je rovněž implementována metoda, která je zodpovědná za vlastní měření.

Modely jsem rovněž použil jako uchování stavů těchto objektů při celém běhu aplikace. Toto však není ideální řešení a modely by měly sloužit pouze jako abstrakce reálných objektů, nikoliv přímo k uložení stavů v rámci aplikace. Nicméně tímto porušení čistého vzoru MVVM se snažím dosáhnout o něco snazší implementace. V podstatě tak limituji aplikaci na komunikaci s jedním osciloskopem a jedním měřeným kryptografickým zařízením. Pro upřesnění aplikace není vázána na jeden konkrétní typ osciloskopu, nebo kryptografického zařízení, ale pouze k práci s jedním osciloskopem a jedním měřeným zařízením najednou. Pokud by aplikace měla pracovat s více osciloskopy a více měřenými zařízeními najednou, bylo by třeba pro ukládání stavů objektů implementovat vlastní třídy, které by uchovávaly informace o konkrétních objektech. Taková funkce by v aktuální implementaci způsobila paměťovou neefektivitu. V rámci celé aplikace je nejdůležitější Model pro osciloskop, který reprezentuje reálné zařízení. Diagram obsažených tříd pro osciloskop je zobrazen na obrázku 4.5.



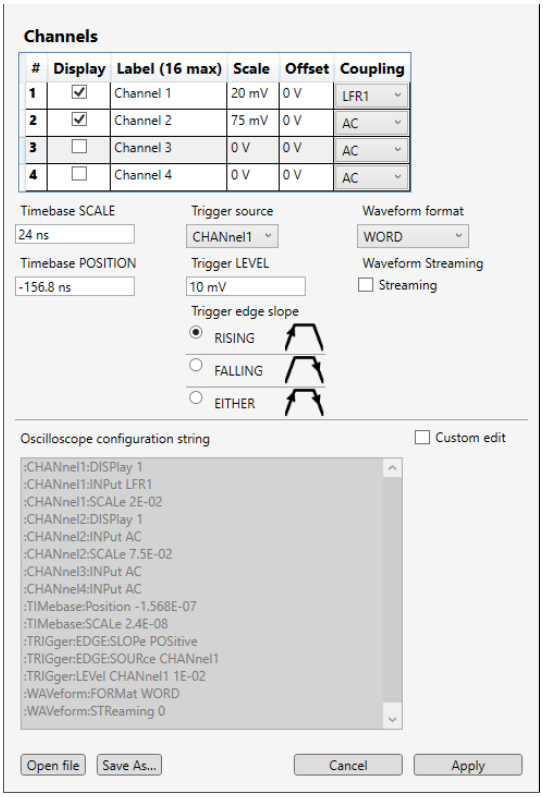
Obr. 4.5: Část diagramu tříd osciloskopu.

4.4 Konfigurace osciloskopu skrze aplikaci

Jelikož modernější osciloskopy obsahují velké množství nastavení, je skrze tuto konfigurační nabídku možné provádět pouze základní konfiguraci. Jmenovitě se jedná o jednoduché nastavení parametrů analogových kanálů osciloskopu, práce s časovou základnou, měřící spouští a nastavení formátu dat, ve kterém bude osciloskop

změřené hodnoty posílat do připojeného počítače. Při zvolení konfigurace osciloskopu se aplikace automaticky synchronizuje s osciloskopem. Aplikace osciloskopu během synchronizace pošle sérii příkazů, kterými se dotáže na hodnoty parametrů, které může uživatel v rámci GUI upravovat. Uživateli jsou tak prezentovány aktuální hodnoty nastavení osciloskopu. Synchronizace se sestává z několika metod v třídě *Oscilloscope*. Jelikož téměř každý parametr kromě vlastního příkazu vyžaduje i jiný postup synchronizace, je vytvořena synchronizační metoda pro každý parametr zvlášť. I přesto však tyto metody obsahují společné prvky. Jako například sestavení a odeslání dotazovacího příkazu a zápis výsledného příkazu do seznamu příkazů pro případné uložení nebo úpravu konfigurace.

Náhled konfiguračního okna je na obrázku 4.6. Konfigurační část obsahuje interaktivní prvky a textové pole, do kterého se automaticky vypisují patřičné SCPI příkazy, které budou odeslány do osciloskopu. Obsah textového pole je možné uložit nebo načíst ze souboru pro případné opakované použití stejné konfigurace. Do textového pole může rovněž uživatel manuálně zadat vlastní SCPI příkazy, které chce, aby byly zapsány do konfigurace osciloskopu. Uživatel tak není limitován pouze nastavením, které má vlastní GUI prvky. Nicméně manuálně zadané příkazy nejsou aplikací nijak kontrolovány a je na uživateli, aby zaručil, že formát příkazů bude odpovídat konfigurační sadě pro vybraný osciloskop.



Obr. 4.6: Okno konfigurace osciloskopu.

Parametry, které obsahují hodnotu a jednotku, jsou automaticky převáděny tak, aby osciloskopu byla odesílána vždy správná hodnota a zároveň aby hodnoty zůstaly čitelné pro uživatele v rámci GUI. Proto jsou v této části zejména číselné parametry převáděny tak, že automaticky doplňují hodnoty o metrický prefix a jednotku. Aplikace tak podporuje obousměrné použití jednotek a vybraných metrických prefixů.

4.4.1 Příkazová sada

Jednotlivé osciloskopy mají vlastní příkazové sady. Pro správné fungování aplikace a její komunikace s osciloskopem je nutné aplikaci předat soubor, který obsahuje příkazovou sadu pro daný osciloskop. Příklad souboru s příkazovou sadou pro osciloskop Agilent Technologies MSO9104A je uveden v příloze C.1. Z tohoto souboru je po připojení k osciloskopu načtena příkazová sada jako třída *CommandList*, jejíž diagram je na obrázku 4.7. Aplikace poté skrze tuto třídu sestavuje SCPI příkazy s proměnnými parametry tak, aby je bylo možné odeslat do osciloskopu. V souboru je vždy uvedeno označení daného příkazu, tělo příkazu a umístění proměnných parametrů. Formátováním a skládáním textových řetězců aplikace sestaví požadovaný příkaz pro daný osciloskop. S ohledem na tuto skutečnost je vyžadováno přesné dodržení formátu příkazového souboru.

Z předešlého odstavce tedy vyplývá, že pokud je dodržen formát příkazového souboru, je aplikace schopná obsluhovat kterýkoliv osciloskop. Samozřejmě za předpokladu, že daný osciloskop komunikuje skrze SCPI příkazy. Po připojení k osciloskopu se aplikace automaticky pokusí nalézt patřičnou příkazovou sadu pro daný osciloskop. Pokud je soubor s příkazovou sadou nalezen, rovnou se načte a s osciloskopem je možné pracovat. V případě, že příkazová sada nalezena není, je načten soubor s výchozí příkazovou sadou. S vysokou pravděpodobností, ale bude komunikace s osciloskopem odkázána na manuální odesílání příkazů. Výchozí soubor je načítán primárně proto, aby v případě použití různých osciloskopů, které podporují stejnou příkazovou sadu, nebylo nutné vytvářet tuto příkazovou sadu pro každý zvlášť.

Z tohoto souboru se kromě vlastních SCPI načítají i jiné parametry osciloskopu. Například počet kanálů, které lze v osciloskopu konfigurovat. Případně další nastavení, jako možnosti pro výběr zdroje spouště osciloskopu, formáty a zdroje pro odesílání změřených dat. A další parametry, které mohou být aplikací vyžadovány pro potřeby rozpoznání odpovědí osciloskopu.

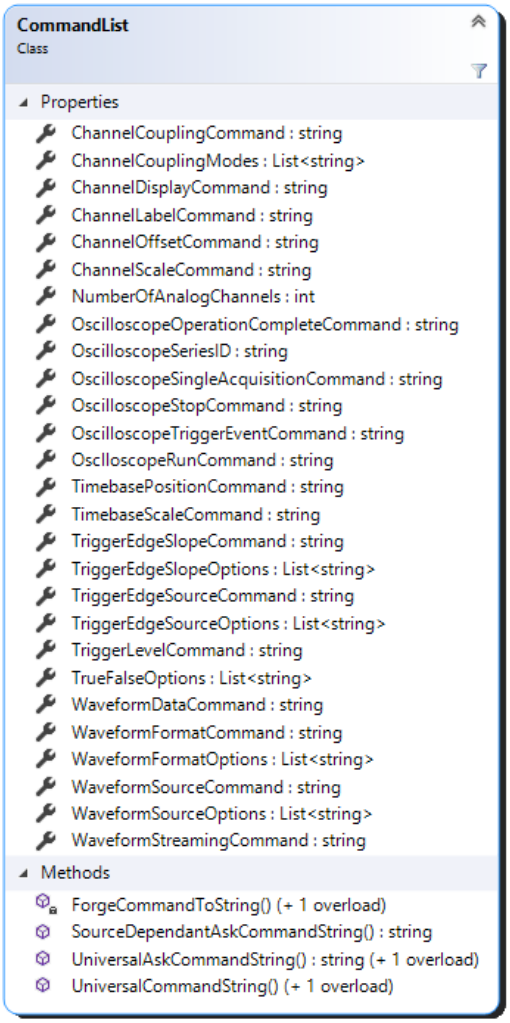
Přestože jsou SCPI příkazy zahrnuty ve standardu IEEE 488.2, jejich formát není přesně definovaný. Proto může nastat stav, kdy nebude možné předložit správně formátovaný příkaz pro některou z funkcí aplikace. V takovém případě by bylo nutné implementovat novou metodu, která by s příkazem dokázala pracovat. Příkladem ne-

standardně formátovaného příkazu je příkaz *:TRIGger:LEVel <zdroj><hodnota>* z příkazové sady osciloskopu MSO9104A. Při synchronizaci se aplikace dotazuje na hodnoty parametrů tak, že osciloskopu odešle příkaz zakončený znakem **?**. V případě tohoto příkazu je však validní formát *TRIGger:LEVel? <zdroj>* místo očekávaného *TRIGger:LEVel <zdroj>?*.

Příkazové sady pro osciloskopy jsou zpravidla uváděny v referenčním programovacím manuálu, který poskytuje výrobce daného osciloskopu.[5]

Použití příkazové sady může být nahrazeno proprietárním ovladačem, který může výrobce k osciloskopu poskytnout. Tyto ovladače pokud jsou dostupné, tak zpravidla mnohem efektivněji implementují komunikaci s konkrétním osciloskopem a nabízí širší možnosti konfigurace. Nicméně v případě použití ovladače by aplikace byla schopná komunikovat pouze s konkrétním typem osciloskopu. Případné použití jiného osciloskopu by vyžadovalo separátní implementaci použití jiného ovladače.

[5]Při vývoji byl primárně použit manuál osciloskopu MSO9104A



Obr. 4.7: Diagram třídy *CommandList*.

V rámci aplikace uvedená příkazová sada obsahuje všechny potřebné příkazy pro správné fungování aplikace. Přesto je však jednoduše rozšiřitelná. Pro rozšíření příkazové sady stačí přidat nové vlastnosti do třídy *CommandList* a doplnit správně formátované příkazy do patřičného souboru. Samozřejmě je dále nutné implementovat metody, které budou nové příkazy využívat.

4.5 Měřící smyčka

Automatizaci celého procesu měření proudové spotřeby v rámci aplikace zajišťuje soubor metod a tříd. Celý proces měření zastřešuje třída *MeasureCommand*. Spuštění měřící smyčky provádí uživatel stisknutím tlačítka pro měření v GUI. Prvním stiskem tlačítka dojde k zavolání metody pro spuštění měřící smyčky. Měřící smyčku může uživatel opětovným stiskem tlačítka kdykoliv přerušit. Opětovným stiskem

tlačítka dochází k přerušení, nikoliv pozastavení měřící smyčky. Další stisk tlačítka spustí nové měření.

Před začátkem vlastního měření je třeba mít navázanou komunikaci s osciloskopem a měřeným zařízením. Měřící smyčka je konečná, proto musí uživatel uvést, kolik průběhů požaduje změřit. Dále je možné nastavit, kolik průběhů bude uloženo do souboru předtím, než aplikace vytvoří další soubor, kam bude ukládat změřená data. Nastavitelný je i počet bytů, jaký budou mít náhodná vstupní data pro kryptografické zařízení. Před začátkem měřící smyčky jsou proto provedeny patřičné inicializace, aby měřící smyčka nebyla příliš časově náročná. Z tohoto důvodu nelze během měřící smyčky data měnit.

Vlastní měřící smyčka se sestává ze 4 základních operací:

- nastavení osciloskopu do čekacího režimu,
- odeslání dat kryptografickému zařízení,
- stáhnutí změřených dat z osciloskopu,
- uložení dat do textového souboru.

Vývojový diagram měřící smyčky je patrný na obrázku 4.8. Počet průchodů touto smyčkou určuje uživatel jím zvolenými parametry.

Nastavení osciloskopu do čekacího režimu je pouhé odeslání zprávy osciloskopu pro přepnutí do režimu, kdy čeká na sepnutí spouště. Jakmile je příkaz odeslán, aplikace se periodicky dotazuje, zda je osciloskop připraven k měření. Po obdržení odpovědi, že je osciloskop připraven, je tato část ukončena.

Při každém průběhu měřící smyčkou jsou vygenerována nová vstupní data. Po vygenerování je aplikaci ohlášena událost o vytvoření nové zprávy. Třída, která zastřešuje komunikaci s kryptografickým zařízením, je přihlášena k odběru této události. Jakmile je událost ohlášena, třída novou zprávu rovnou odešle do kryptografického zařízení.

Po odeslání vstupních dat aplikace přechází do části, kdy se snaží získat naměřená data z osciloskopu. Osciloskop může data předávat v několika různých formátech s ohledem na přesnost výsledků. Formát změřených dat může během konfigurace změnit uživatel. Před vlastním stáhnutím dat z osciloskopu se aplikace dotáže, jakým formátem osciloskop dat odesílá. Jelikož v rámci knihoven NI.VISA a IVI.VISA jsou speciálně přizpůsobené metody pro stáhnutí jednotlivých formátů. Pokud se však aplikaci nepodaří data skrze tyto metody stáhnout, pokusí se data získat univerzálním čtením výstupu osciloskopu. Pokud i tento postup selže, aplikace nahradí data z osciloskopu zprávou o chybě a pokračuje dalším průchodem smyčkou. Tento stav může nastat i v případě, kdy je s osciloskopem během měření manuálně manipulováno. Tímto způsobem pak lze z výsledného souboru ověřit, zda a kdy osciloskop neposlal aplikaci očekávaná data. Speciální metody pro čtení konkrétních formátů dat jsou podstatně rychlejší než univerzální přístup, proto má

Obr. 4.8: Vývojový diagram měřící smyčky.

aplikace tendenci prvně použít speciální metody. I přesto je tato část měřící smyčky časově nejnáročnější částí.

Časová náročnost je způsobena rychlostí, jakou osciloskop data odesílá, a také množstvím dat. Množství dat může být mezi jednotlivými měřeními proměnné. Osciloskop si množství dat může nastavit automaticky pro dodržení určité přesnosti, případně si množství dat může uživatel nastavit manuálně. Pro tato nastavení však v rámci aplikace nejsou vytvořeny ovládací prvky GUI ani nejsou zahrnuty příkazy v příkazové sadě. Tuto změnu tak musí uživatel provést buď přímo na osciloskopu, nebo skrze aplikaci odeslat manuálně zadaný příkaz.

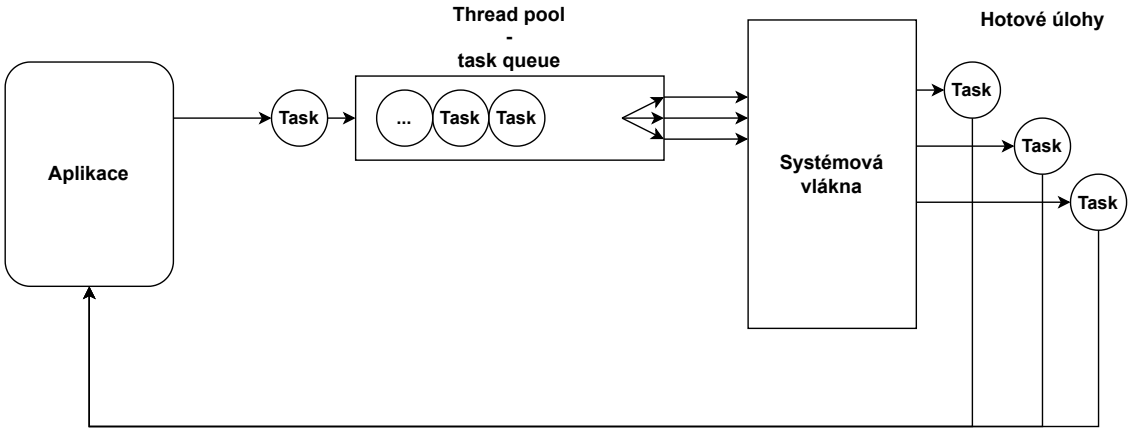
Před začátkem měření si uživatel může vybrat z jakých zdrojů chce měření provádět. Zpravidla se jedná o jednotlivé kanály osciloskopu. Samozřejmě může zvolit i více zdrojů najednou. V takovém případě aplikace cyklí zvolené zdroje a žádá jednotlivé změřené průběhu z osciloskopu pro daný kanál. Pokud uživatel žádný zdroj nezvolil, je použit zdroj předchozího nastavení osciloskopu.

Uložení dat do souboru zahrnuje případné vytvoření daného souboru s datem a časem měření. Dále je třeba uvést číslo souboru v případě, že jsou data v rámci jednoho měření ukládána do více souborů. V rámci souboru je jeden průběh smyčkou reprezentován jedním řádkem. Začátek řádku obsahuje časové razítko vytvoření zprávy pro kryptografické zařízení, dále je uveden zdroj dat (zpravidla se jedná o označení kanálu osciloskopu), pokud byl zvolen zdroj uživatelem, odeslaná zpráva pro kryptografické zařízení a změřená data z osciloskopu. Jednotlivé hodnoty jsou ukládány jako textové řetězce a jsou odděleny čárkou. V některých formátech osciloskop data odesílá v netextové podobě, zpravidla bytové hodnoty. Aby taková data mohla být uložena do textového souboru, provádí aplikace bezeztrátovou konverzi pomocí kódování Base64 k převedení dat do textového řetězce. S ohledem na množství průběhů a velikost dat, která osciloskop zašle při změření jednoho průběhu, mohou textové soubory nabývat značné velikosti. Vše ale záleží na množství průběhů na soubor a velikost dat odeslaných z osciloskopu. Zpravidla se ale velikost souborů může pohybovat v řádech mezi několika desítek MB do několika GB.

Zdrojový kód hlavní části měřící smyčky je uveden v příloze B.1

4.6 Vícevláknová implementace

Pro zachování a udržení maximální responzivity aplikace je využito principů vícevláknového programování. Operace a metody, které by mohly způsobit zamrznutí nebo jinak ovlivnit odezvu aplikace, jsou prováděny v oddělených vláknech. Jádro aplikace a tím pádem i část, která se stará o vykreslování a interakci s aplikačním GUI běží pouze v jednom vlákně. Z toho důvodu by mohlo docházet k zásekům aplikace, pokud zrovna provádí časově náročnou operaci. Vícevláknová implementace tomuto dokáže zabránit, nebo alespoň zmírnit dopady operací na responzivitu. Konkrétně je zde využit tzv. *asynchronní vzor založený na úloze – Task-based asynchronous pattern* (TAP)[36]. Důležitou součástí TAP jsou úlohy (tasks), které je možné spouštět a vykonávat v oddělených vláknech tak, aby nebylo zatíženo hlavní aplikační vlákno. V rámci programovacího jazyka C# je možné využít jmenného prostoru *System.Threading.Tasks* z knihovny *System.Runtime.dll*. Knihovna mimo jiné obsahuje třídu *Task*, která poskytuje jednoduchý způsob, jak v aplikaci provádět operace asynchronně. Pro takto prováděné operace je aplikaci v rámci platformy .NET Framework systémem vyčleněn tzv. thread pool. Z thread pool si aplikace může vyžádat spuštění nového vlákna, ve kterém provede patřičnou operaci, a po skončení vlákno uvolní pro další použití [37]. Toto fungování je patrné z obrázku 4.9.



Obr. 4.9: Fungování thread pool v .NET Framework.

V aplikaci je využito asynchronního programování pro několik operací. Zejména se jedná o čtení a zápis souborů, navázání spojení s osciloskopem nebo kryptografickým zařízením a synchronizace nebo změna nastavení osciloskopu. Zároveň měřící smyčka provádí některé operace asynchronně.

Využití TAP má značné výhody proti klasickému manuálnímu spouštění a opouštění vláken. Aplikace tak využívá systémové zdroje více efektivně a dynamicky, než v případě manuálního řízení vláken přímo v rámci zdrojového kódu aplikace.

Další výhodou asynchronního programování je možnost dočasného přerušení provádění dané metody. Pokud v rámci metody probíhá časově náročná operace, lze provádění metody dočasně pozastavit, dokud není operace dokončena. V takovém případě se nadřazená volající metoda může vrátit k jiným operacím a po dokončení časově operace je provádění automaticky obnoveno.

4.7 Vybrané metody

Tato kapitola se věnuje několika vybraným metodám, které jsou v rámci aplikace použity. V rámci programu jsou jednotlivé metody zdokumentovány pomocí XML dokumentačních komentářů. Nicméně pro rozšíření a podrobnější vysvětlení jsou zde uvedené metody dále vysvětleny.

Metoda v části 4.1 se stará o sestavení správně formátovaného SCPI příkazu, který lze validně odeslat do osciloskopu. Tato konkrétní metoda akceptuje jako parametr tělo příkazu a jeden parametr, který je do příkazu vložen. Těla příkazů jsou uvedeny v příloze C.1. Štítky ve složených závorkách označují místa, kam jsou pomocí interpolace řetězců vkládány parametry. Návratová hodnota této metody se sestává ze dvou řetězců. První řetězec obsahuje pouze tělo příkazu bez štítků

pro umisťování parametrů. Druhý řetězec obsahuje celý výsledný příkaz. Tělo příkazu je v dalších metodách použito k odstranění duplikátů v rámci výsledného souboru příkazů, které budou osciloskopu předány. Pokud by k odstraňování duplikátů nedocházelo, mohlo by dojít k znepřehlednění. Každá změna parametru by sestavila a vložila nový příkaz, bez odstranění příkazu s předešlým parametrem.

Výpis 4.1: Metoda pro sestavení SCPI příkazu.

```
private static (string, string) ForgeCommandToString(string selectedCommand,
    strincommandParameter1)
{
    StringBuilder stringBuilder = new StringBuilder();
    string partCommand = stringBuilder.AppendFormat(CultureInfo.InvariantCulture,
        selectedCommand, string.Empty).ToString();
    _ = stringBuilder.Clear();
    string resultCommand = stringBuilder.AppendFormat(CultureInfo.InvariantCulture,
        selectedCommand, commandParameter1).ToString();
    return (partCommand, resultCommand);
}
```

Při synchronizaci se aplikace dotazuje osciloskopu na hodnoty jednotlivých parametrů, aby uživateli skrze GUI byly prezentovány aktuální hodnoty nastavení osciloskopu. Kromě číselných hodnot nastavení rozsahů a pozic kanálů nebo časové základny, odpovědi mohou zahrnovat i řadu textových parametrů. Příkladem může být formát odesílaných dat nebo zdroj pro spoušť osciloskopu. Příklady těchto textových parametrů jsou rovněž patrné v příloze C.1. Pro jednoduchost a šetření komunikace, osciloskop v odpovědích vynechává malá písmena z hodnot těchto parametrů. Nicméně v příkazové sadě jsou pro přehlednost, čitelnost a úplnost uvedeny celé parametry. Následující metoda ve výpisu 4.2 je zodpovědná za porovnání shody mezi odpovědí osciloskopu a parametrem uvedeným v příkazové sadě. Toho metoda dosahuje s použitím LINQ, kde z výčtového typu vybere jednotlivé položky a dočasně pro porovnání odstraní malá písmena. V tomto dočasném seznamu nalezne shodu s odpovědí osciloskopu a propojí odpověď s celým parametrem pomocí indexu.

Výpis 4.2: Čtení odpovědi osciloskopu při synchronizaci zdroje spouště osciloskopu.

```
private void SynchronizeTriggerEdgeSource(TriggerSettings trigger)
{
    string commandPart = Commands.TriggerEdgeSourceCommand;
    string oscilloscopeResponse = AskCommand(commandPart);
    trigger.TriggerEdgeSourceIndex = Commands.TriggerEdgeSourceOptions
        .Select(x => x = Regex.Replace(x, @"[a-z]+", string.Empty))
        .Select((item, index) => (item, index))
        .Where(x => x.item.Contains(oscilloscopeResponse))
        .Select(x => x.index)
        .FirstOrDefault();
    AddResponseToConfig(commandPartCommands.TriggerEdgeSourceOptions.ElementAt(
        trigger.TriggerEdgeSourceIndex));
}
```

Následující metoda ve výpisu 4.3 má za úkol zpřehlednit a zlepšit čitelnost číselných údajů v okně konfigurace osciloskopu. Jelikož aplikace standardně zobrazuje číselné hodnoty výpisem celé desetinné části, mohou být hodnoty v řádech tisícin a níže špatně čitelné a nepřehledné. V běžném životě pro zápis takových hodnot využíváme metrických prefixů uváděným před jednotkou. Pro nízké hodnoty to jsou zpravidla prefixy m – mili pro tisíciny, μ – mikro pro miliontiny, n – nano pro miliardtiny atd. Nicméně program si s takovým zápisem běžně neporadí a je třeba implementace podobné metody, která takový zápis přeloží mezi vstupem uživatele pro zápis do promměné v programu a vice versa. V rámci aplikace je na podobných místech, kde se využívá práce s textovými řetězci a jejich převody, využito tzv. regulerních výrazů, které poskytují robustní způsob zpracování takových dat. V tomto případě maximalizuje přesnost a variabilitu s jakou lze vstupní data zadávat.

Tato metoda rovněž podporuje několik způsobů vyjádření prefixu μ. Pro prefix je standardně používáno řecké písmeno, které lze zapsat dvěma různými UNICODE znaky. Pro běžné použití je nahrazováno písmenem u z běžné znakové sady.

Výpis 4.3: Převodník pro metrické prefixy číselných hodnot.

```
protected decimal StringToDecimal(string stringToConvert)
{
    stringToConvert = Regex.Replace(stringToConvert, @"\s", string.Empty);
    stringToConvert = Regex.Replace(stringToConvert, @",", ".");
    Match regexMatch = new Regex(@"([-]*\d*[.]*\d+)([mu\u03BC\u00B5np]*)([sV]*)").
        Match(stringToConvert);
    _ = decimal.TryParse(regexMatch.Groups[1].Value, NumberStyles.AllowDecimalPoint
         NumberStyles.AllowExponent | NumberStyles.AllowLeadingSign, CultureInfo.
        InvariantCulture, out decimal numberResult);
    string unitPrefix = regexMatch.Groups[2].Value;
    if (unitPrefix.Equals("m", StringComparison.Ordinal))
    {
        numberResult *= 1E-03M;
    }
    else if (unitPrefix.Equals("u", StringComparison.Ordinal) || unitPrefix.Equals
        ("\u03BC"StringComparison.Ordinal) || unitPrefix.Equals("\u00B5",
        StringComparison.Ordinal))
    {
        numberResult *= 1E-06M;
    }
    else if (unitPrefix.Equals("n", StringComparison.Ordinal))
    {
        numberResult *= 1E-09M;
    }
    else if (unitPrefix.Equals("p", StringComparison.Ordinal))
    {
        numberResult *= 1E-12M;
    }
    return numberResult;
}
```

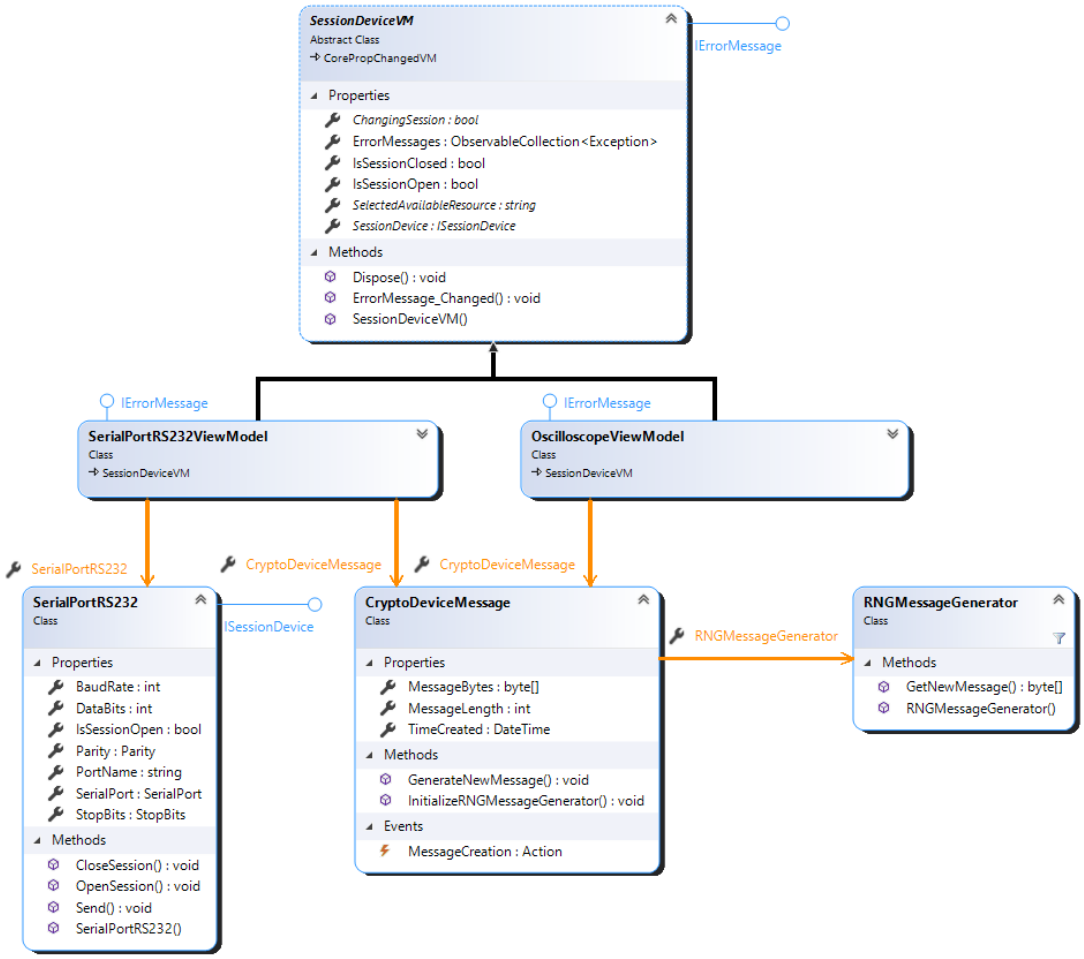
4.8 Diagram tříd

Pro přehlednost je v této kapitole a v příloze D uvedeno několik diagramů, které reprezentují výslednou kompozici aplikace. Na jednotlivých diagramech je patrné, které třídy dědí z jiných tříd, případně jaká aplikační rozhraní interpretují. Tyto diagramy byly vygenerovány přímo ze zdrojového kódu pomocí nástroje *Class designer*, který je volitelnou součástí Visual Studio 2019. Diagramy rovněž prezentují některé vlastnosti a metody daných tříd.

S ohledem na místo, které celé diagramy zabírají, jsou zde uvedeny pouze jejich nejdůležitější části. Náhled diagramu celé aplikace je uveden v příloze D.1

Následující obrázek 4.10 zobrazuje implementaci tříd ViewModel pro osciloskop a měřené zařízení. Jelikož obě tyto části vyžadují komunikaci skrze vstupně-výstupní rozhraní počítače, sdílí určité prvky, které jsou s touto komunikací spojeny. Tyto části rovněž vyžadují podobné zacházení ze strany aplikačního GUI, které obsahuje prvky pro navázání komunikace a hlídá, aby uživatel nemohl začít navazovat komunikační relaci v případě, že je již ustanovena. Dále tyto třídy implementují mechanizmy hlášení chyb komunikace se zařízením. Pokud není možné komunikaci navázat, nebo je náhle přerušena aplikace skrze tyto třídy informuje uživatele o vzniklém problému. Stejný mechanizmus se aktivuje, pokud vyprší časový limit pro čtení nebo odeslání zprávy zařízením.

Obrázek dále naznačuje, že třídu, která reprezentuje vstupní data pro kryptografické zařízení, využívá jak část pro komunikaci s kryptografickým zařízením, tak i část pro komunikaci s osciloskopem. V rámci ViewModel pro komunikaci s osciloskopem je implementováno spouštění měřící smyčky, ve které se periodicky generuje nová zpráva a obsah zprávy je ukládán spolu se změřenými hodnotami do souboru.



Obr. 4.10: Část diagramu ViewModel pro komunikující zařízení.

Zdrojový kód třídy *SessionDeviceVM* je uveden v příloze B.2

Další diagram je uveden v příloze D.2. Na tomto diagramu je patrná struktura implementovaných příkazů. Příkazy jsou iniciovány uživatelem skrze aplikační GUI a provádí konkrétní operace. Příkazy, které obsahují časově náročné operace jsou implementovány asynchronně, aby nezpůsobovaly zamrznutí GUI. Jelikož jsou ale tyto příkazy prováděny v oddělených vláknech, je třeba aby zde byl implementován mechanizmus, který zabrání vícenásobnému spuštění stejného příkazu.

Synchronní příkazy jsou spouštěny v rámci hlavního vlákna, proto je nemožné před dokončením příkazu vynutit nové spuštění. Operace probíhají v sekvenci, kterou v rámci hlavního vlákna nelze narušit. Příkaz *MeasureCommand*, který implementuje měřící smyčku je jediný asynchronní příkaz, který lze iniciovat vícenásobně během jeho vykonávání.

Nicméně je zde jiný mechanizmus, který umožňuje opětovnou inicializací přerušit průběh měření. Abstraktní tělo asynchronních příkazů s ochranou proti vícenásobnému spuštění je uvedeno v příloze B.4.

4.9 Průběh vývoje

V průběhu vývoje byla aplikace několikrát refaktorována. Původně byla aplikace vytvořena bez použití MVVM návrhového vzoru a místo WPF bylo použito Windows Form Application, zkráceně WinForms, jako rozhraní pro vytvoření GUI. S přidáváním metod a tříd se však vývoj rychle komplikoval a bylo obtížné udržet zdrojový kód přehledný. Bez použití vzoru MVVM rovněž vzniká téměř 100% závislost jednotlivých tříd a jejich vlastností na ovládacích prvcích v rámci GUI a vice versa. Tyto nedostatky byly eliminovány použitím MVVM společně s WPF.

Po první refaktorizaci zde bylo několik pokusů zprovoznit aplikaci v novějším a modernějším .NET Core 5 proti staršímu .NET Framework 4.8. Mezi výhody .NET Core patří dostupnost novější verze jazyka C#, konkrétně pro .NET Core 5 se jedná o verzi C# 9.0. Tato novější verze jazyka C# nabízí funkce pro psaní přehlednějšího a efektivnějšího zdrojového kódu aplikace, které nejsou v předešlých verzích dostupné. Dále .NET Core podporuje vývoj multiplatformních aplikací, oproti .NET Framework, který podporuje pouze platformy s operačním systémem Windows. Pokud by aplikace byla napsaná v .NET Core mohla by být spuštěna na kterékoliv platformě s podporou tohoto rozhraní. Jedinou limitací by zůstávalo WPF, které je podporováno pouze v operačních systémech Windows. Nicméně s použitím MVVM by bylo třeba pouze vytvořit jiné GUI a provázat jej na již existující kód aplikace.

V aplikaci použité knihovny NI.VISA a IVI.VISA bohužel nepodporují použití s .NET Core a proto bylo třeba použití staršího a méně univerzálního .NET Framework.

4.9.1 GitHub

Pro jednotlivé refaktory a verze aplikace byla využita platforma GitHub a její repozitářová služba. Použití podobné služby rovněž podnítila okolnost, že bylo nutné často aplikaci přemisťovat mezi stolním počítačem, na kterém probíhal vývoj, na přenosný počítač pro testování komunikace přímo na pracovišti.

Během vývoje bylo založeno několik repozitářů, protože pokusy o refaktorizaci aplikace z .NET Core na .NET Framework případně z WinForm a WPF vždy vyžadují založení nového projektu v rámci Visual Studio. Z tohoto důvodu je pro finální verzi aplikace vytvořen i vlastní repozitář. V tomto finálním repozitáři již není obsažena kompletní historie vývoje aplikace v podobě jednotlivých verzí (*commit*) a vývojové větve dle funkcí (*branch*). Struktura finálního repozitáře je tak více chaotičtější oproti předešlým instancím. Tento stav však není ideální a při vývoji software by neměl nastávat. Vlastnímu vývoji by mělo předcházet rozsáhlejší plánování, než bylo přítomno při vývoji této aplikace. Použitý postup by působil velké problémy

při vývoji aplikace v týmu nebo v případě, že by bylo nutné se v rámci verzí vracet k předešlým implementacím. Při samostatném vývoji tento stav nepůsobil značné problémy, nicméně stále není žádaný pro reálné prostředí.

Finální repozitář a výsledná aplikace jsou dostupné ke stažení přes odkaz [6].

4.9.2 Plíživý vývoj funkcí

Vývoj se potýkal s jevem, kterému se říká feature creep, přeloženo jako plíživý vývoj funkcí. Jedná se o jev, který postihuje zejména vývoj software. Plíživý vývoj funkcí se projevuje potřebou neustále přidávat a implementovat nové funkce. Při vývoji aplikace je třeba samozřejmě funkce přidávat a implementovat, je však nutné při vývoji a implementaci stále myslet na užitečnost a využití dané funkce. Přesto některé funkce přímo aplikace nevyžaduje a jejich přidání působí spíše problémy než užitek. Neustálé přidávání funkcí zvyšuje nežádanou komplexitu aplikace a její použití. Zároveň implementace těchto funkcí odebírá čas z celého vývoje aplikace, který by mohl být využit efektivněji, například pro uhlazení aplikace, testování a vyřešení jiných problémů a chyb.

Pro vývoj je tak nejdůležitější plánovat, které funkce je třeba přidat a implementovat, aby aplikace efektivně plnila účel, pro který byla vytvořena. Zároveň je třeba dávat pozor, aby nová implementovaná funkce příliš nekomplikovala jednoduchý návrh aplikace.

Přesto by aplikace měla obsahovat pouze funkce, které skutečně pro svůj chod potřebuje. Jaké konkrétní funkce bude uživatel aktivně využívat je v průběhu vývoje těžké odhadovat. V ideálním případě by se aplikace měla dočasně používat v testovacím režimu, kdy uživatel komunikuje zpětnou vazbu a poznatky k aplikaci vývojáři, a ten má možnost aplikaci dále ladit a upravovat. Nicméně i v tomto případě může nastat jev plíživého vývoje funkcí ze strany uživatele, který klade požadavky na implementace funkcí, které ve výsledku nebude využívat.

Pokud v budoucnu do aplikace mají být přidány další funkce, měl by vývojář vždy pevně zvážit, zda jsou dané funkce potřebné pro účel aplikace, či zda stejného výsledku nelze dosáhnout použitím funkcí stávajících.

4.10 Užití aplikace

Výsledkem vývoje je desktopová aplikace P–SCAAT. Tato aplikace běží na operačních systémech Windows s podporou .NET Framework. Aplikace jako taková usnadňuje uživateli automatizaci měření na osciloskopu. Umožňuje tak měření pro analýzu odolnosti proti útokům postranními kanály. Aplikace uživateli poskytuje mož-

[6] `https://github.com/Synexyx/P_SCAAT_DP`

nost skrze aplikační GUI konfigurovat některé parametry připojeného osciloskopu. V rámci aplikace lze spustit automatizované měření, které odesílá vstupní data kryptografickému zařízení, jehož analýzu uživatel provádí. Zároveň během tohoto měření aplikace periodicky stahuje změřená data z osciloskopu a ukládá je spolu s vstupními daty pro zařízení do souboru. Cílem aplikace je tak sběr těchto dat, které mohou být později použity pro vlastní analýzu odolnosti proti SCA. Aplikaci s drobnými úpravami lze použít pro měření s kterýmkoliv osciloskopem, který podporuje použití SCPI příkazů. Co se týče měřeného kryptografického zařízení, vyžaduje aplikace pouze možnost komunikace přes rozhraní sériového portu. Aplikace by měla být bez problémů jednoduše rozšiřitelná o další funkce.

Aplikace je zkompilována do spustitelného souboru, který je přenositelný na kterýkoliv počítač s operačním systémem Windows a podporou .NET Framework verze 4.8. Pro fungování aplikace dále k počítači musí být rozhraním USB připojen osciloskop a sériovým portem měřené zařízení.

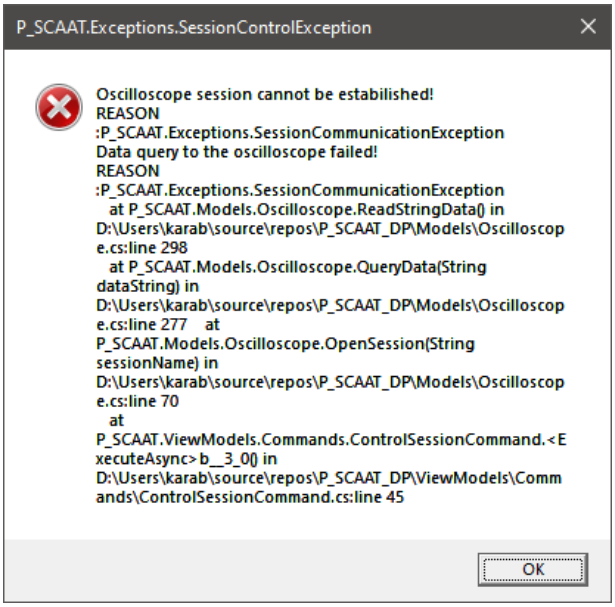
4.11 Testování

Vývoj z většiny času probíhal odděleně mimo pracoviště. Nedostupnost osciloskopu a měřeného zařízení v průběhu vývoje poměrně limitovaly rozsah a kvalitu výsledného testování. Nicméně i přesto byla aplikace v průběhu vývoje několikrát otestována přímo na pracovišti. Funkčnost hotové aplikace byla v provozním režimu otestována na výsledném pracovišti a aplikace nejevila známky chyb a problémů.

V aplikaci jsou implementovány postupy, které se dokáží vypořádat s jistou řadou chyb. Pokud se jedná o kritickou chybu, která brání aplikaci pokračovat v práci, je o takové chybě informován uživatel skrze vyskakující okno s výstrahou. Aplikace tak při výskytu chyby nespadne, ale přeruší práci a informuje o chybě uživatele. Této funkce je dosaženo pomocí odchytu výjimek, které vyvolají jednotlivé části aplikace, pokud při jejich provádění dojde k problému. V rámci aplikace je použita metoda tzv. probublávání výjimek. Skrze tuto metodu jsou výjimky odchyceny na kritických místech, kde by mohly ohrozit plynulý běh programu. Zároveň probublávání umožňuje včasné přerušení provádění dané operace a předchází tak stavu, kdy je operace tiše dokončena s chybami, které v průběhu nastaly. Příklad chybového okna, kterým aplikace ohlásila chybu při navázání komunikace s osciloskopem je na obrázku 4.11.

V rámci tohoto okna jsou údaje, kde chyba nastala. Okno obsahuje výpis procesu, který k chybě vedl. Tento postup však v ostrém provozu není ideální, protože není třeba uživatele o procesu informovat. Nicméně takovýmto způsobem je usnadněno případně nalezení a odstranění chyby vývojáři, kterému tato chyba vznikne, jelikož je v okně uvedeno, při které konkrétní operaci nastal problém. Správný postup

pro vypořádání se s chybami, je uživateli předat jednoduchou a jasnou informaci o vzniku problému a komplexnější a přesnější detaily o chybě uložit do logovacího souboru. Tento postup je do jisté části implementován v třídách, které rozšiřují rozhraní *IErrorMessage*. Postup však není používaný a je zakomentovaný, protože chybí implementace jednoduchých zpráv pro uživatele a filtrace, které chyby je třeba logovat, a které nikoliv a periodické promazávání logového souboru, aby svou velikostí nezabíral příliš mnoho paměti. Dalším vylepšením by byly detailnější informace o vstupech, které mohly ovlivnit vznik dané výjimky.



Obr. 4.11: Vyskakující okno s chybou navázání komunikace s osciloskopem.

Pro testování software se v ideálním případě dále používají tzv. jednotkové testy (unit test), které usnadňují a zkvalitňují testování a běh celé aplikace. Jednotkové testy mají za úkol maximálně otestovat funkčnost použitých metod v rámci programu. Jedná se tak vlastně o části programu, které testují jiné části programu.

V rámci této implementace však jednotkové testy použité nejsou a testování probíhalo pouze manuálně. V ideálním případě by jednotkové testy měly pokrývat všechny důležité funkce aplikace. Nicméně psaní jednotkových testů může být časově náročné a z toho důvodu jsou často opomenuty.

Závěr

V rámci této diplomové práce jsem se seznámil s teoretickou částí práce. Z uvedených zdrojů jsem popsal problematiku vlastního měření proudové spotřeby v kapitolách 2.1 a 2.2. Uvádím zde některé konkrétní metody pro měření proudové spotřeby. Dále v práci vysvětluji základní princip útoků postranními kanály a způsobů analýzy změřených výsledků. Následující kapitola 3 popisuje základní nástroje, které umožňují automatizaci celého procesu měření proudové spotřeby. Kapitola 3.2 je věnována projektu SAKURA, který poskytuje vývojové desky pro analýzu útoků postranními kanály. Dle slov mého vedoucího má fakulta k dispozici právě desky projektu SAKURA, proto jsem projekt popsal a zmínil konkrétní desky, které jsou dostupné pro měření.

Vlastnímu vývoji aplikace P–SCAAT v programovacím jazyce C# se věnuje kapitola 4, kde jsem nastínil základní myšlenky a principy, které vedly k návrhu a následné implementaci vlastní aplikace. Konkrétně se jedná o části 4.1 a 4.2. Vlastní implementaci se věnují části následující. V těchto částech jsou popsány a rozvedeny postupy, kterých jsem se při vývoji držel. Rovněž jsou zde rozepsány klíčové části aplikace, které zaručují její funkčnost. V této kapitole jsou v jednotlivých částech uvedeny i případné nedostatky a problémy, které byly s jejím vývojem spojeny. Přestože aplikace stále obsahuje jisté nedostatky, věřím, že většinu problémů jsem v průběhu vývoje vyřešil.

Aplikaci jsem manuálně otestoval v provozním režimu přímo na pracovišti. Při testování aplikace nejevila žádné známky chyb nebo nefunkčnosti. Výslednou aplikaci a její funkčnost jsem na pracovišti předvedl vedoucímu práce.

Výsledkem mé práce je tedy funkční desktopová aplikace pro operační systém Windows. Aplikace poskytuje možnost automatizovaného měření proudové spotřeby prostřednictvím osciloskopu. Aplikace zároveň generuje vstupní data pro měřené zařízení, jehož proudovou spotřebu osciloskop měří. V rámci tohoto automatizovaného měření aplikace stahuje změřená data z osciloskopu a ukládá je do souboru. V rámci aplikaci si uživatel může přizpůsobit nastavení osciloskopu svým požadavkům. Aplikaci jsem psal s využitím asynchronního a vícevláknového programování tak, abych zaručil maximální responzivitu aplikačního GUI v okamžicích, kdy aplikace provádí časově náročné operace. I během těchto operací aplikace zůstává responzivní a uživatel s ní může dle potřeby pracovat.

K aplikaci jsem nenapsal separátní dokumentaci, veškerá dokumentace mé aplikace je buď obsažena v této práci, nebo je formou dokumentačních XML komentářů uvedena ve zdrojovém kódu přímo u jednotlivých metod. Dále je pro přehlednost k dispozici diagram tříd celé aplikace.

Literatura

[1] HONG, Seokhie. Side Channel Attacks. MDPI - Multidisciplinary Digital Publishing Institute, 2019. ISBN 3039210017.

[2] MARTINÁSEK, Zdeněk. Kryptoanalýza postranními kanály [online]. Brno, 2013 [cit. 2021-12-12]. Disertační práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Václav Zeman. Dostupné z: http://hdl.handle.net/11012/27988

[3] RANDOLPH, Mark a William DIEHL. Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman. Cryptography [online]. 2020, 4(2) [cit. 2021-10-05]. ISSN 2410-387X. Dostupné z: doi:10.3390/cryptography4020015

[4] MANGARD, Stefan, Elisabeth OSWALD a Thomas POPP. Power Analysis Attacks. New York, NY: Springer-Verlag, 2007. ISBN 0387308571. Dostupné z: doi:10.1007/978-0-387-38162-6

[5] Side-channel attacks explained: everything you need to know. Rambus [online]. Rambus Press, 2021 [cit. 2021-10-10]. Dostupné z: https://www.rambus.com/blogs/side-channel-attacks/

[6] KOCHER, Paul, Joshua JAFFE, Benjamin JUN a Pankaj ROHATGI. Introduction to differential power analysis. Journal of Cryptographic Engineering [online]. 2011, 1(1), 5-27 [cit. 2021-10-07]. ISSN 2190-8508. Dostupné z: doi:10.1007/s13389-011-0006-y

[7] BIHAM, Eli a Adi SHAMIR. Differential fault analysis of secret key cryptosystems. KALISKI, Burton S., ed. Advances in Cryptology — CRYPTO '97 [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, 1997-5-17, s. 513-525 [cit. 2021-10-12]. Lecture Notes in Computer Science. ISBN 978-3-540-63384-6. Dostupné z: doi:10.1007/BFb0052259

[8] GERLICH, Tomáš. Ochrana proti kryptoanalýze postranními kanály [online]. Brno, 2015 [cit. 2021-10-12]. Bakalářská práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Zdeněk Martinásek. Dostupné z: http://hdl.handle.net/11012/41334

[9] GAMAARACHCHI, Hasindu a Harsha GANEGODA. Power Analysis Based Side Channel Attack [online]. 2018 [cit. 2021-10-07]. Dostupné z: https://arxiv.org/abs/1801.00932

[10] MILLIKAN„ Robert A. a E.S. BISHOP. Elements of Electricity: A Practical Discussion of the Fundamental Laws and Phenomena of Electricity and Their Practical Applications in the Business and Industrial World [online]. Chichago: American Technological Society, 1917 [cit. 2022-05-15]. LCCN 17001802. Dostupné z: https://babel.hathitrust.org/cgi/pt?id=mdp.39015039610756&view=1up&seq=7&skin=2021

[11] SMITH, Gary. ANALOG MEASURMENTS IN A DIGITAL WORLD. Motor [online]. New York: Hearst Business Publishing, 2020, 234(1), 16-23 [cit. 2021-12-07]. ISSN 0027-1748.

[12] ZHEN, Yang. Current Sensing Circuit Concepts and Fundamentals. U.S.A.: Microchip Technology, 2011. ISBN 978-1-61341-590-0.

[13] HUGHES, Scott. Magnetic force; Magnetic fields; Ampere's law [online]. Massachusetts Institute of Technology Department of Physics, 10 March 2005, 9 [cit. 2022-05-15]. Dostupné z: https://web.mit.edu/sahughes/www/8.022/lec10.pdf

[14] SANFILIPPO, S. Hall probes: physics and application to magnetometry [online]. Paul Scherrer Institut, Villigen, Switzerland, 2011, 40 [cit. 2022-05-15]. Dostupné z: doi:10.48550/ARXIV.1103.1271

[15] Current Sensing Using Linear Hall Sensors [online]. 2009 [cit. 2021-10-07]. Dostupné z: https://www.infineon.com/dgdl/Current_Sensing_Rev.1.1.pdf?fileId=db3a304332d040720132d939503e5f17

[16] GANDOLFI, Karine, Christophe MOURTEL a Francis OLIVIER. Electromagnetic Analysis: Concrete Results. In: Cryptographic Hardware and Embedded Systems — CHES 2001 [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, s. 251-261 [cit. 2021-10-09]. ISBN 9783540425212. ISSN 0302-9743. Dostupné z: doi:10.1007/3-540-44709-1_21

[17] NOŽKA, Marek. Metalické vedení. VOŠ a SPŠE Olomouc [online]. Olomouc [cit. 2021-11-07]. Dostupné z: https://mamut.spseol.cz/nozka/psk/056-vedeni_parametry/

[18] KULARATNA, Nihal. Digital and Analogue Instrumentation: Testing and measurement. No. 11. Stevenage: The Institution of Engineering and Technology, 2003. ISBN 0852969996.

[19] Standard Commands for Programmable Instruments (SCPI). SCPI Consortium [online]. May 1999, 1999(VERSION 1999.0), 819 [cit. 2022-05-07]. Dostupné z: https://www.ivifoundation.org/docs/scpi-99.pdf

[20] SAKURA-G Side-channel AttacK User Reference Architecture. SAKURA Hardware Security Project [online]. Kawasaki, Kanagawa, 2013 [cit. 2021-11-11]. Dostupné z: https://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G_Spec_Ver1.0_English.pdf

[21] SAKURA-W Side-channel AttacK User Reference Architecture. SAKURA Hardware Security Project [online]. Chofu, Tokyo, 2014 [cit. 2021-11-11]. Dostupné z: https://satoh.cs.uec.ac.jp/SAKURA/doc/SAKURA-W_Quik_Start_Guide_Ver1.0_English.pdf

[22] SAKURA-X Side-channel AttacK User Reference Architecture. SAKURA Hardware Security Project [online]. Tsukuba-shi, Ibaraki, 2013 [cit. 2021-11-11]. Dostupné z: http://www.risec.aist.go.jp/project/sasebo/download/SASEBO-GIII_Spec_v1_1_English.pdf

[23] SCOTT, Michael L. Programming Language Pragmatics. San Francisco: Elsevier Science & Technology, 2009. ISBN 0123745144.

[24] CHIVERS, Ian. Essential C# Fast. London: Springer London, Limited, 2003. ISBN 1852335629. ISSN 1439-975X. Dostupné z: doi:10.1007/978-1-4471-0075-1

[25] MONTALBANO, Elizabeth. 25 innovators: Anders Hejlsberg. CRN [online]. Westborough: The Channel Company, 2003, (1062), 28 [cit. 2021-12-06]. ISSN 1539-7343. Dostupné z: https://www.proquest.com/trade-journals/25-innovators-anders-hejlsberg/docview/227594822/se-2?accountid=17115

[26] C# language versioning. Microsoft .NET documentation [online]. Microsoft, 2022 [cit. 2022-04-29]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/configure-language-version

[27] .NET Glossary. Microsoft .NET documentation [online]. Microsoft, 2022 [cit. 2022-04-29]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/glossary

[28] VPP-4.3.6: VISA Implementation Specification for .NET. Interchangeable Virtual Instruments [online]. 7 June 2016, (Revision 5.8), 257 [cit. 2022-05-07]. Dostupné z: https://www.ivifoundation.org/docs/vpp436_2016-06-07.pdf

[29] KOSS, Robert. MARTIN ROBERT C. Agile software development. New Jersey: Prentice-Hall, 2003, 529 s. ISBN 0-13-597444-5.

[30] LEE, Roger Y. Software Engineering. 2013. Paris: Springer, 2013. ISBN 9462390517. Dostupné z: doi:10.2991/978-94-6239-006-5

[31] MARTIN, Robert C. Design Principles and Design Patterns. Object Mentor [online]. 2000 [cit. 2021-12-07]. Dostupné z: http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf [2018-10-28] přes WayBack–Machine

[32] JONÁŠ, Martin. Návrhové principy: SOLID. Zdrojak.cz [online]. 2012 [cit. 2021-12-07]. Dostupné z: https://zdrojak.cz/clanky/navrhove-principy-solid/

[33] MVVM - Quick Guide. TutorialsPoint [online]. 2022 [cit. 2022-04-29]. Dostupné z: https://www.tutorialspoint.com/mvvm/mvvm_quick_guide.htm

[34] Desktop Guide (WPF .NET). Microsoft technical documentation [online]. Microsoft, 2022 [cit. 2022-04-29]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-6.0

[35] SMITH, Josh. Patterns - WPF Apps With The Model-View-ViewModel Design Pattern. MSDN Magazine [online]. February 2009, 2009(Volume 24) [cit. 2022-04-29]. Dostupné z: https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern

[36] Task-based asynchronous pattern (TAP) in .NET: Introduction and overview. Microsoft technical documentation [online]. Microsoft, 2022 [cit. 2022-05-08]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/task-based-asynchronous-pattern-tap

[37] CARMONA, David. Programming the Thread Pool in the .NET Framework. Microsoft technical documentation [online]. Microsoft, 2002 [cit. 2022-05-08]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/dotnet/articles/ms973903(v=msdn.10)?redirectedfrom=MSDN

Seznam symbolů a zkratek

AES	Advanced Encryption Standard
AIST	The National Institute of Advanced Industrial Science and Technology
DES	Data Encryption Standard
DPA	diferenciální proudová analýza – Differential Power Analysis
FIA	útok zavedením chyb – Fault Injection Attack
FMC LPC	FPGA Mezzanine Card – Low Pin Count
FPGA	programovatelné hradlové pole – Field-programmable gate array
GB	Giga byte
GUI	grafické uživatelské rozhraní – Graphical User Interface
IC card	čipová karta – Integrated Circuit Card
IVI	Interchangeable Virtual Instruments
LINQ	Language Integrated Query
LSI	Large-scale integration
MB	Mega byte
MVVM	Model–View–ViewModel
NI	National Instruments
P–SCAAT	Power Side-Channel Attack Analysis Tool
PUF	fyzicky neklonovatelné funkce – Physical Unclonable Function
RNG	generátor náhodných čísel – Random Number Generator
RSA	Rivest–Shamir–Adleman
SAKURA	Side-channel AttacK User Reference Architecture
SASEBO	Side-channel Attack Standard Evaluation BOard
SCA	útok postranním kanálem – Side-Channel Attack

SCPI standardní příkazy pro programovatelné přístroje – Standard Commands for Programmable Instruments

SPA jednoduchá proudová analýza – Simple Power Analysis

TAP asynchronní vzor založený na úloze – Task-based asynchronous pattern

UEC The University of Electro-Communications

VISA Virtual Instrument Software Architecture

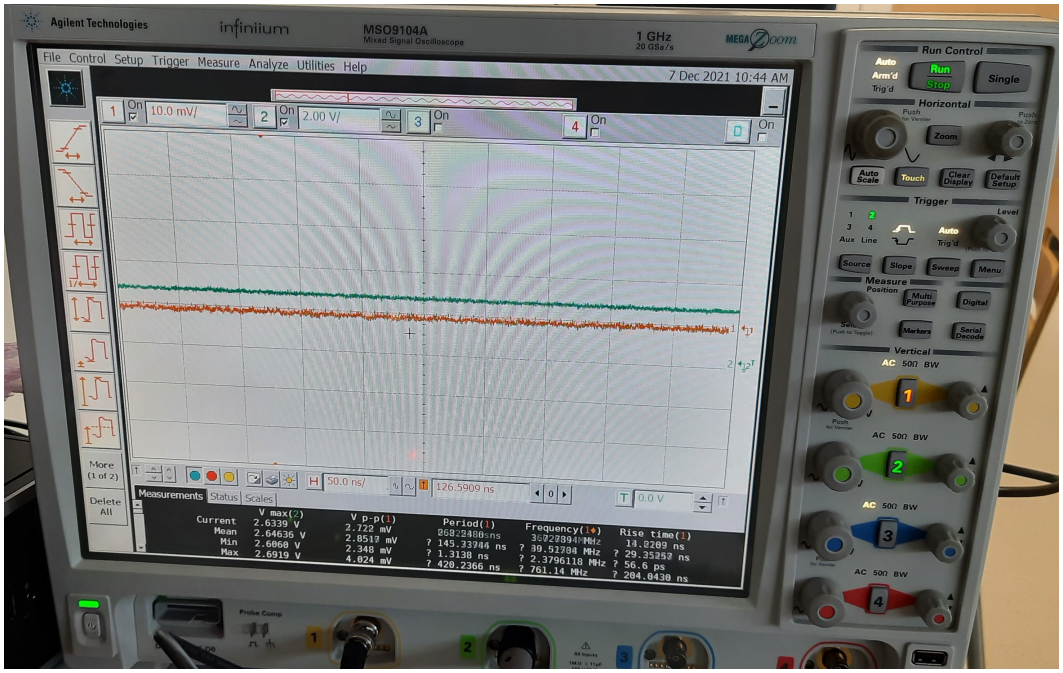
WPF Windows Presentation Foundation

XAML Extensible Application Markup Language

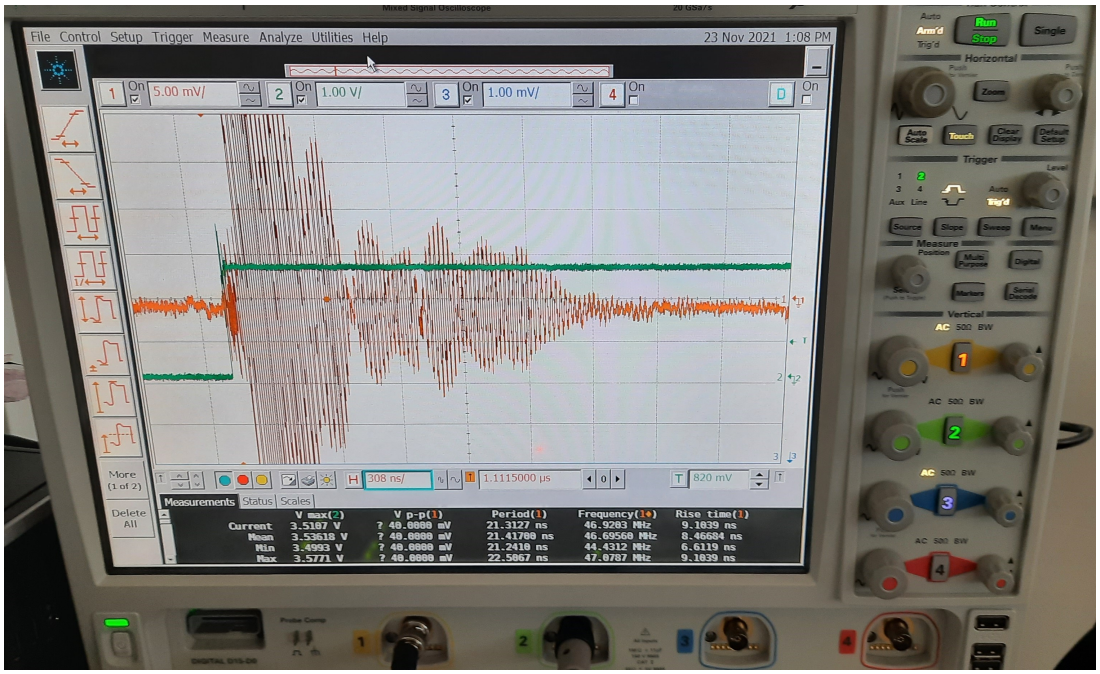
XML Extensible Markup Language

Seznam příloh

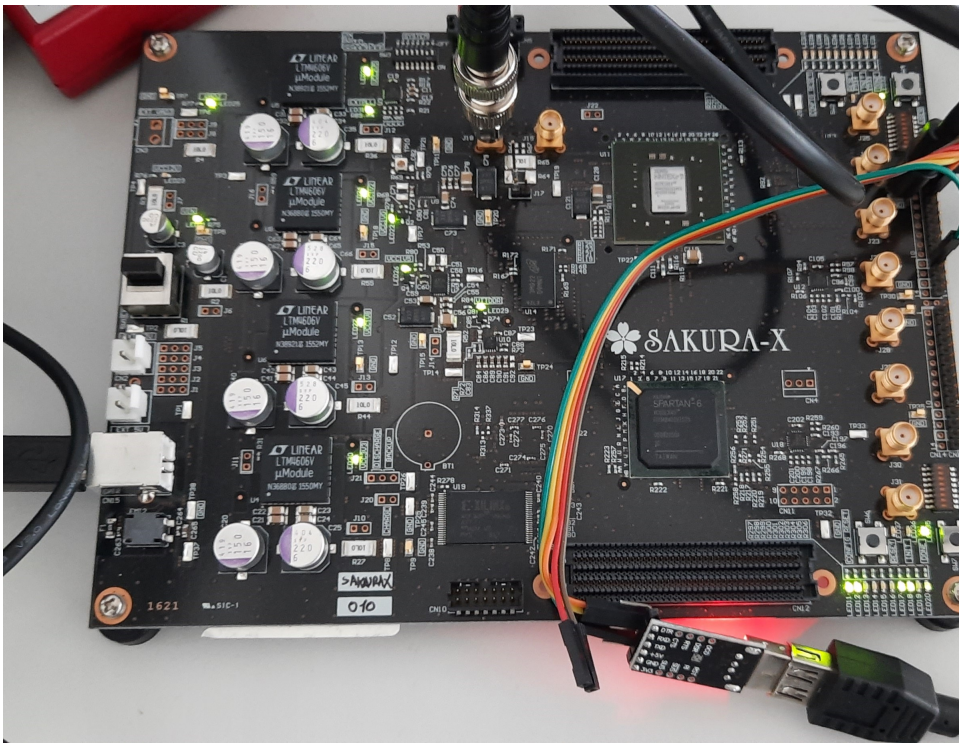
A Obrázky pracoviště



Obr. A.1: Osciloskop Agilent MSO9104A



Obr. A.2: **Osciloskop Agilent MSO9104A**, na kterém probíhá měření proudové spotřeby



Obr. A.3: Deska SAKURA-X s připojenými sondami pro měření

B Výpis kódu jazyka C#

Výpis B.1: Tělo měřící smyčky

```
public override async Task ExecuteAsync(object parameter)
{
    if (!_oscilloscopeViewModel.MeasurementInProgress)
    {
        _oscilloscopeViewModel.MeasurementInProgress = true;
        _oscilloscopeViewModel.ProgressBarValue = 0
        tokenSource = new CancellationTokenSource()

        string fileNameSessionID = DateTime.Now.ToString("HHmmss");
        _cryptoDeviceMessage.InitializeRNGMessageGenerator(_oscilloscopeViewModel.
            MessageLength)
        await Task.Run(async () =>
        {
            try
            {
                List<WaveformSourceViewModel> sourcesToMeasure
                    _oscilloscopeViewModel.WaveformSource.Where(x => x.IsSelected).
                    ToList();
                string selectedFormat = _oscilloscope.GetCurrentWaveformFormat()
                for (int totalTraces = 0; totalTraces < _oscilloscopeViewModel.
                    TracesTotal; totalTraces++)
                {
                    tokenSource.Token.ThrowIfCancellationRequested()
                    _oscilloscope.MeasurePrep()
                    await Task.Run(() => { _cryptoDeviceMessage.GenerateNewMessage
                        (); })
                    tokenSource.Token.ThrowIfCancellationRequested()
                    ///<summary>
                    ///Measure even if no source is selected => measure with
                        unknown source previsously set in device. (Don't change
                        source)
                    ///</summary>
                    if (!sourcesToMeasure.Any())
                    {
                        string response = _oscilloscope.GetWaveformData(
                            selectedFormat);
                        await SaveToFile(fileNameSessionID, "N/A", response);
                    }
                    ///<summary>
                    ///Cycle through all sources. Change them and acquire data.
                    ///</summary>
                    else
                    {
                        foreach (WaveformSourceViewModel source in sourcesToMeasure
                            )
                        {
                            string selectedSource = source.SourceName;
                            _oscilloscope.ChangeWaveformSource(selectedSource);
                            string response = _oscilloscope.GetWaveformData(
                                selectedFormat);
                            await SaveToFile(fileNameSessionID, selectedSource,
                                response);
                        }
```

```
                    perFile++;
                    _oscilloscopeViewModel.ProgressBarValue = totalTraces + 1;
                    tokenSource.Token.ThrowIfCancellationRequested();
                }
            }
            catch (OperationCanceledException ex) when (ex.CancellationToken ==
                tokenSource.Token)
            {
                Debug.WriteLine("TASK CANCELED");
                _oscilloscopeViewModel.ProgressBarValue = 0;
            }
            catch (Exception ex)
            {
                _oscilloscopeViewModel.ErrorMessages.Add(ex);
                tokenSource.Cancel();
            }
        }, tokenSource.Token);
    }
    else
    {
        tokenSource.Cancel();
    }
    _oscilloscopeViewModel.MeasurementInProgress = false;
}
```

Výpis B.2: Abstraktní třída ViewModel pro komunikující zařízení

```
internal abstract class SessionDeviceVM : CorePropChangedVM, IErrorMessage
{
    public abstract ISessionDevice SessionDevice { get; }
    public bool IsSessionOpen => SessionDevice.IsSessionOpen;
    public bool IsSessionClosed => !SessionDevice.IsSessionOpen;
    public abstract bool ChangingSession { get; set; }
    public abstract string SelectedAvailableResource { get; }
    public ObservableCollection<Exception> ErrorMessages { get; } =
        newObservableCollection<Exception>();
    public SessionDeviceVM()
    {
        ErrorMessages.CollectionChanged += ErrorMessage_Changed;
    }
    public virtual void ErrorMessage_Changed(object sender,
        NotifyCollectionChangedEventArgs e)
    {
        if (e.NewItems != null)
        {
            foreach (Exception item in e.NewItems)
            {
                _ = MessageBox.Show($"{item.Message} {item.StackTrace}", $"{item.
                    GetType()}",MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }
    }
    public override void Dispose()
    {
        ErrorMessages.CollectionChanged -= ErrorMessage_Changed;
        base.Dispose();
    }
}
```

Výpis B.3: Třída *MainViewModel* pro hlavní okno aplikace

```
internal class MainViewModel : CorePropChangedVM
{
    #region Properties
    private readonly OscilloscopeViewControlState _oscilloscopeViewControlState;
    private readonly SerialPortRS232ViewModel _serialPortRS232ViewModel;
    public CorePropChangedVM OscilloscopeSelectedVM =>_oscilloscopeViewControlState
        .OscilloscopeSelectedVM;
    public CorePropChangedVM SerialPort232 => _serialPortRS232ViewModel;
    #endregion
    public MainViewModel(OscilloscopeViewControlState oscilloscopeViewControlState,
        SerialPortRS232ViewModel serialPortRS232ViewModel)
    {
        _oscilloscopeViewControlState = oscilloscopeViewControlState;
        _serialPortRS232ViewModel = serialPortRS232ViewModel;
        _oscilloscopeViewControlState.OscilloscopeConfigViewSwitched +=
            OnOscilloscopeConfigViewSwitch;
    }
    private void OnOscilloscopeConfigViewSwitch()
    {
        OnPropertyChanged(nameof(OscilloscopeSelectedVM));
    }
    public override void Dispose()
    {
        _oscilloscopeViewControlState.OscilloscopeConfigViewSwitched -=
            OnOscilloscopeConfigViewSwitch;
        base.Dispose();
    }
}
```

Výpis B.4: Abstraktní tělo asynchronního příkazu s ochranou proti vícenásobnému spuštění

```
internal abstract class AsyncIsExcecutingCoreCommand : AsyncCoreCommand
{
    private bool _isExecuting;
    protected bool IsExecuting
    {
        get => _isExecuting;
        set { _isExecuting = value; OnCanExecuteChanged(); }
    }
    public override bool CanExecute(object parameter)
    {
        return !IsExecuting && base.CanExecute(parameter);
    }
    public override async void Execute(object parameter)
    {
        IsExecuting = true;
        try
        {
            await ExecuteAsync(parameter);
        }
        finally
        {
            IsExecuting = false;
        }
    }
}
```

C Příklad příkazové sady

Výpis C.1: Příklad příkazové sady osciloskopu Agilent Technologies MSO9104A

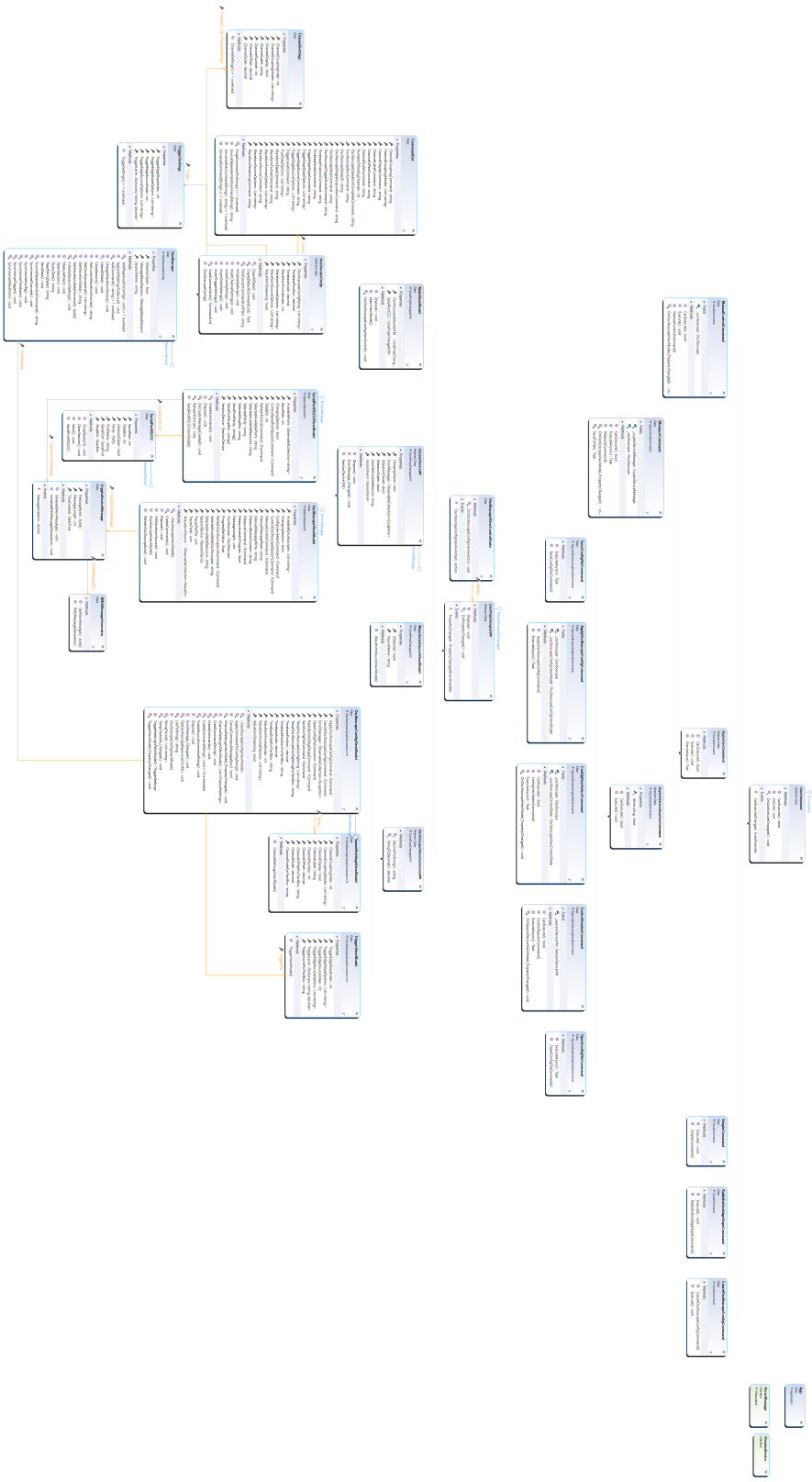
```
{
  "OscilloscopeSeriesID": "Agilent Technologies,MSO9104A,No Serial,04.20.0005",
  "NumberOfAnalogChannels": 4,
  "TrueFalseOptions": [
    "1",
    "0",
    "ON",
    "OFF"
  ],
  "OscilloscopeSingleAcquisitionCommand": ":SINGle",
  "OscilloscopeStopCommand": ":STOP",
  "OscilloscopeRunCommand": ":RUN",
  "OscilloscopeOperationCompleteCommand": "*OPC?",
  "OscilloscopeTriggerEventCommand": ":TER?",
  "ChannelDisplayCommand": ":CHANnel{0}:DISPlay {1}",
  "ChannelLabelCommand": ":CHANnel{0}:LABel {1}",
  "ChannelScaleCommand": ":CHANnel{0}:SCALe {1}",
  "ChannelOffsetCommand": ":CHANnel{0}:OFFSet {1}",
  "ChannelCouplingCommand": ":CHANnel{0}:INPut {1}",
  "ChannelCouplingModes": [
    "AC",
    "DC",
    "LFR1",
    "LFR2",
    "DC50"
  ],
  "TimebasePositionCommand": ":TIMebase:Position {0}",
  "TimebaseScaleCommand": ":TIMebase:SCALe {0}",
  "TriggerEdgeSourceCommand": ":TRIGger:EDGE:SOURce {0}",
  "TriggerEdgeSourceOptions": [
    "CHANnel1",
    "CHANnel2",
    "CHANnel3",
    "CHANnel4"
  ],
  "TriggerEdgeSlopeCommand": ":TRIGger:EDGE:SLOPe {0}",
  "TriggerEdgeSlopeOptions": [
    "POSitive",
    "NEGative",
    "EITHer"
  ],
  "TriggerLevelCommand": ":TRIGger:LEVel {0} {1}",
  "WaveformDataCommand": ":WAVeform:DATA?",
  "WaveformSourceCommand": ":WAVeform:SOURce {0}",
  "WaveformSourceOptions": [
    "CHANnel1",
    "CHANnel2",
    "CHANnel3",
    "CHANnel4"
  ],
  "WaveformFormatCommand": ":WAVeform:FORMat {0}",
  "WaveformFormatOptions": [
    "ASCii",
    "BINary",
```

```
    "BYTE",
    "WORD"
  ],
  "WaveformStreamingCommand": ":WAVeform:STReaming {0}"
}
```

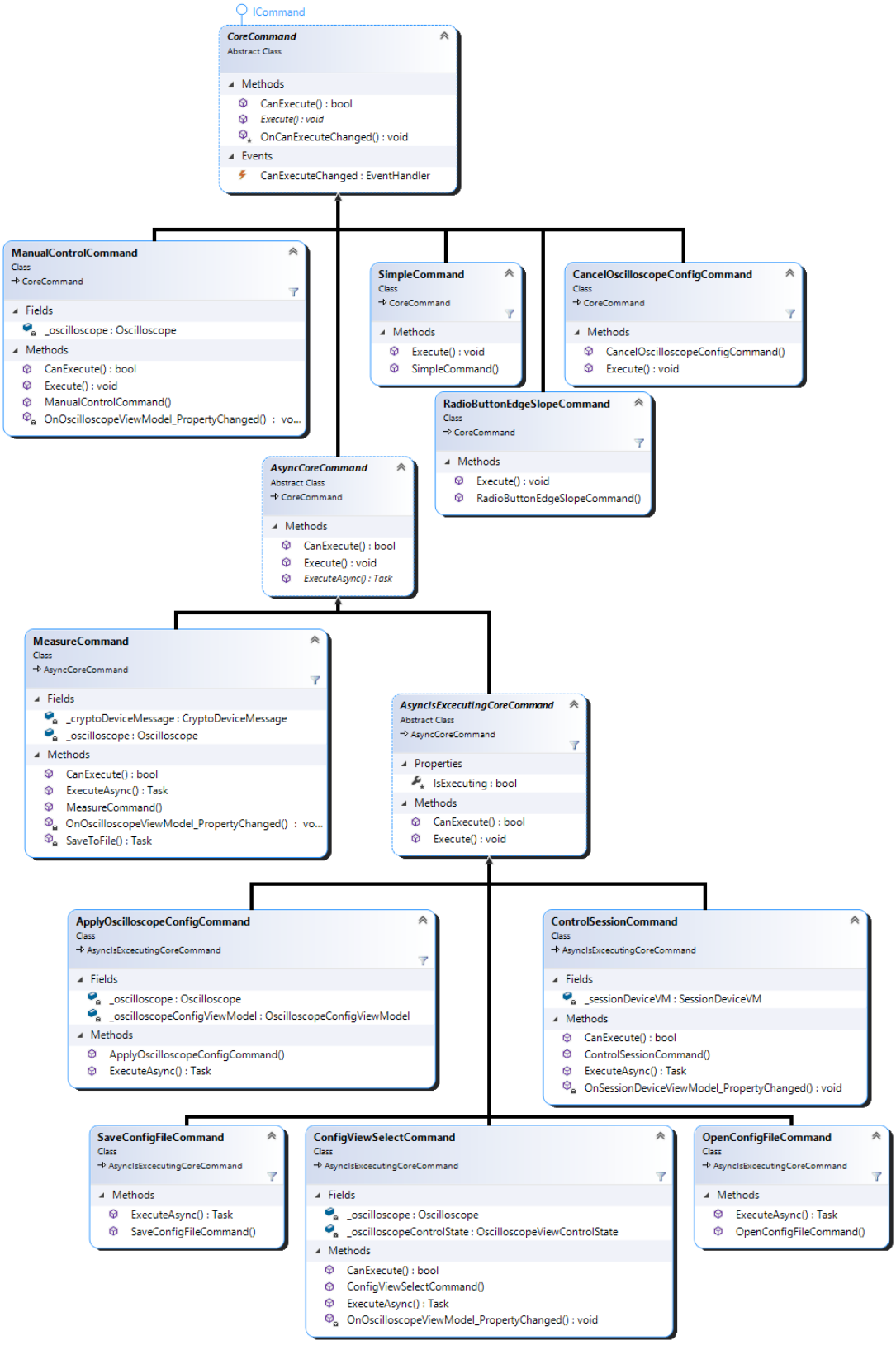
Výpis C.2: Výchozí příkazová sada

```
{
  "OscilloscopeSeriesID": null,
  "NumberOfAnalogChannels": 4,
  "TrueFalseOptions": [
    "1",
    "0"
  ],
  "OscilloscopeSingleAcquisitionCommand": "SINGLe",
  "OscilloscopeStopCommand": "STOP",
  "OscilloscopeRunCommand": "RUN",
  "OscilloscopeOperationCompleteCommand": null,
  "OscilloscopeTriggerEventCommand": null,
  "ChannelDisplayCommand": null,
  "ChannelLabelCommand": null,
  "ChannelScaleCommand": null,
  "ChannelOffsetCommand": null,
  "ChannelCouplingCommand": null,
  "ChannelCouplingModes": [
    "AC",
    "DC"
  ],
  "TimebasePositionCommand": null,
  "TimebaseScaleCommand": null,
  "TriggerEdgeSourceCommand": null,
  "TriggerEdgeSourceOptions": [],
  "TriggerEdgeSlopeCommand": null,
  "TriggerEdgeSlopeOptions": [],
  "TriggerLevelCommand": null,
  "WaveformDataCommand": null,
  "WaveformSourceCommand": null,
  "WaveformSourceOptions": [],
  "WaveformFormatCommand": null,
  "WaveformFormatOptions": [],
  "WaveformStreamingCommand": null
}
```

D Digramy tříd



Obr. D.1: Náhled diagramu tříd celé aplikace



Obr. D.2: Diagram tříd příkazů