VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

MODULÁRNÍ SOFTWAROVÝ NÁSTROJ PRO I.MX

MODULAR SOFTWARE TOOL FOR I.MX

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **RADIM KRATOCHVÍL**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JOSEF STRNADEL, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/20002/2016/xkrato35

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Kratochvíl Radim, Bc.**
Obor: Počítačové a vestavěné systémy
Téma: **Modulární softwarový nástroj pro i.MX**
Modular Software Tool for i.MX
Kategorie: Vestavěné systémy

Pokyny:

1. Zdokumentujte i) základní rysy mikrokontrolérů rodiny i.MX od firmy NXP a ii) prostředky a základy tvorby vestavných aplikací na nich založených.
2. Zdokumentujte současný stav v oblasti nástrojů pro správu, analýzu a testování vývojových kitů na bázi i.MX, shrňte výhody a nevýhody těchto nástrojů.
3. Zdokumentujte možnosti tvorby modulárního programového vybavení prostředky QT5.
4. Navrhněte architekturu modulárního softwarového nástroje pro správu, analýzu a testování vývojových kitů na bázi i.MX založeného na QT5.
5. Nástroj navržený v bodě 4 implementujte, vhodně ověřte a prokažte jeho vlastnosti a použitelnost pro programování reálné aplikace, v C++ za použití QT5, určené k běhu na reálném vývojovém kitu.
6. Vlastnosti vytvořeného nástroje zhodnoťte a diskutujte možné směry jeho dalšího vývoje.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 4 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Strnadel Josef, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Tato diplomová práce pojednává o vytvoření multiplatformní modulární aplikace pro práci s mikrokontroléry řady i.MX od firmy NXP. Tato aplikace je napsaná v programovacím jazyce C++ za použití frameworku Qt5. Následně je hotová aplikace porovnávána s již existujícími nástroji pro práci s mikrokontroléry i.MX

Abstract

The aim of this work is to create multiplatform plug-in based application for work with microcontrollers i.MX from company NXP. This application is writen in programming language C++ with framework Qt5. Completed application is compared with already existing applications for work with microcontrollers i.MX

Klíčová slova

i.MX, mikrokontrolér, Qt5, modulární aplikace

Keywords

i.MX, microcontroller, Qt5, plug-in based application

Citace

KRATOCHVÍL, Radim. *Modulární softwarový nástroj pro i.MX*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Strnadel Josef.

Modulární softwarový nástroj pro i.MX

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Josefa Strnadela, Ph.D. Další informace mi poskytli zaměstnanci firmy NXP. V práci jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Radim Kratochvíl
22. května 2017

Poděkování

Rád bych poděkoval mému vedoucímu práce Ing. Josefu Strnadelovi, Ph.D., a zaměstnancům firmy NXP, za ochotnou pomoc, cenné rady a věcné připomínky.

Obsah

Kapitola 1

Úvod

Mikrokontroléry rodiny i.MX od firmy NXP jsou určeny pro průmysl, výrobu automobilů, internetu věcí, chytrých hodinek, a podobně. Zejména využití v automobilovém průmyslu se v dnešní době neustále rozrůstá a vyvíjí. Firma NXP v současné době nabízí několik nástrojů pro tyto mikrokontroléry, ale ty jsou buď zaměřeny do výroby nebo na návrh desek.

Cílem mé diplomové práce je vytvořit nástroj, který by usnadnil vývoj a ladění na vývojových kitech založených na mikrokontrolérech i.MX. Nástroj by měl být modulární tak, aby bylo možno nastavit v aplikaci jen ty funkce, které daný vývojář potřebuje. Kromě základních funkcí, jako je terminál a VNC, by aplikace měla obsahovat i komunikaci s ROM kódem, programování, úpravu registrů a postupem času i další funkce.

Kapitola 2

Zaměření práce a realizační prostředky

2.1 Vymezení aplikační oblasti

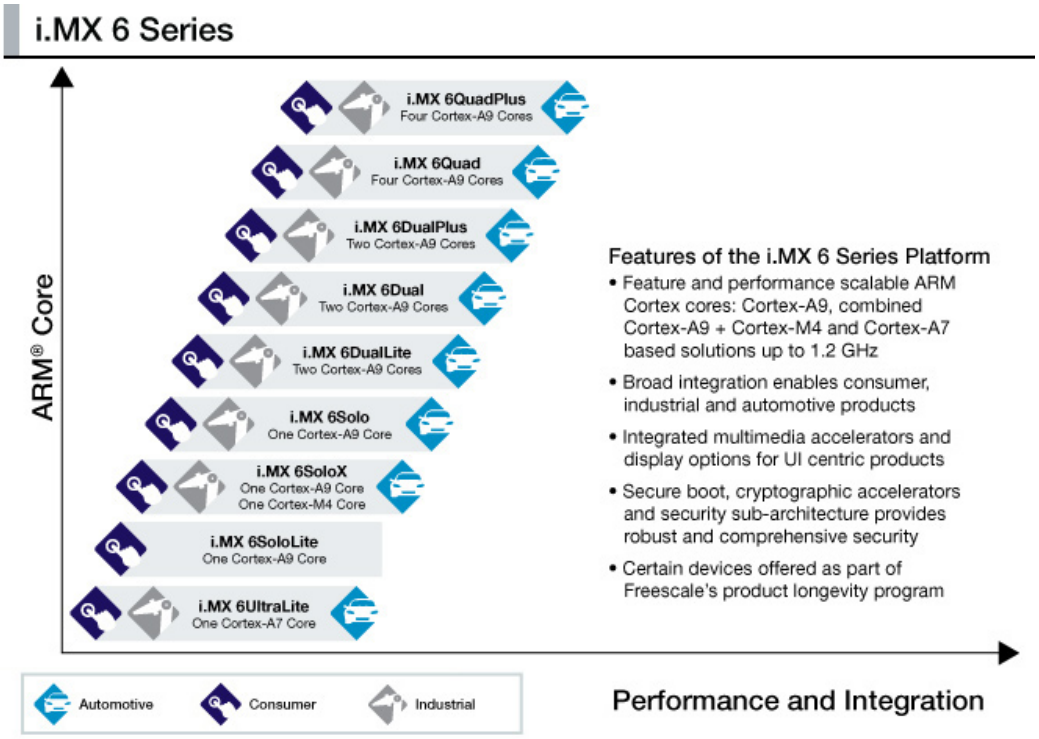
Rodina mikrokontrolérů pochází od firmy NXP. Jsou to aplikační procesory, které jsou založeny na jádrech ARM Cortex-A. Jedná se tedy o výkonné procesory, převážně určené do aut, konzumní elektroniky (tablety, chytré hodinky, apod.) a do strojů v průmyslu. V jednom čipu může být více procesorových jader i grafický čip. První i.MX se objevily již v roce 2001.

V dnešní době jsou k dispozici série i.MX6, i.MX7 a i.MX8. Série i.MX6 je založena na jádrech ARM Cortex-A9, nebo ARM Cortex-A7 a pokrývá široké spektrum využití. Obsahuje od jedno-jádrových úsporných mikrokontrolerů (i.MX6ULL), až po výkonné čtyř-jádrové mikrokontrolery obsahující i grafiku (i.MX6QP). Nalezneme zde také i.MX6SX, což je nepřímý nástupce rodiny mikrokontrolerů Vybrid, který kromě jádra ARM Cortex-A9 obsahuje i nízkopříkonový procesor ARM Cortex M4. Na obrázku 2.1 nalezneme všechny mikrokontrolery i.MX6 seřazeny podle výkonu.

Série i.MX7 prozatím obsahuje jen dva procesory a to i.MX7D a i.MX7S. Tyto procesory jsou spolu pinově kompatibilní a jediný rozdíl mezi nimi je, že i.MX7D obsahuje dvě jádra. Jádra jsou založena na ARM Cortex-A7 a oba procesory obsahují i jedno jádro ARM Cortex-M4. Na rozdíl i.MX6SX ale neobsahují integrovanou grafiku. I přesto mají dost výkonu pro zobrazování 2D grafického uživatelského rozhraní. Jsou velmi úsporné a díky možnosti uspání jádra A7 z jádra M4 se energetická náročnost ještě může snížit (například jádro A7 bude v provozu jen tehdy, když bude potřeba většího výkonu a nebo bude potřeba něco vykreslit na displej). Pro testování bude využit mikrokontrolér i.MX7D, který je detailněji popsán
v kapitole 2.1.1

Nejnovější přírůstek do této rodiny je série i.MX8. Tato řada je zaměřena na velký výkon, a to nejen procesorový, ale i grafický. Je určena pro zpracování obrazu a podporuje až čtyři displeje. Série i.MX8 není následníkem i.MX7, ale navzájem se doplňují. Nejvýkonnější z nich je i.MX 8QuadMax, který obsahuje čtyři jádra ARM Cortex-A53, dva 2x ARM Cortex-A72, tři ARM Cortex-M4F (jedno z nich je dedikované pro řízení zdrojů ostatních jader) a dvě grafické jádra.

Na těchto procesorech je možnost provozovat operační systémy jako je Linux, Android, QNX a FreeRTOS. V budoucnu možná přibude Windows Embedded.



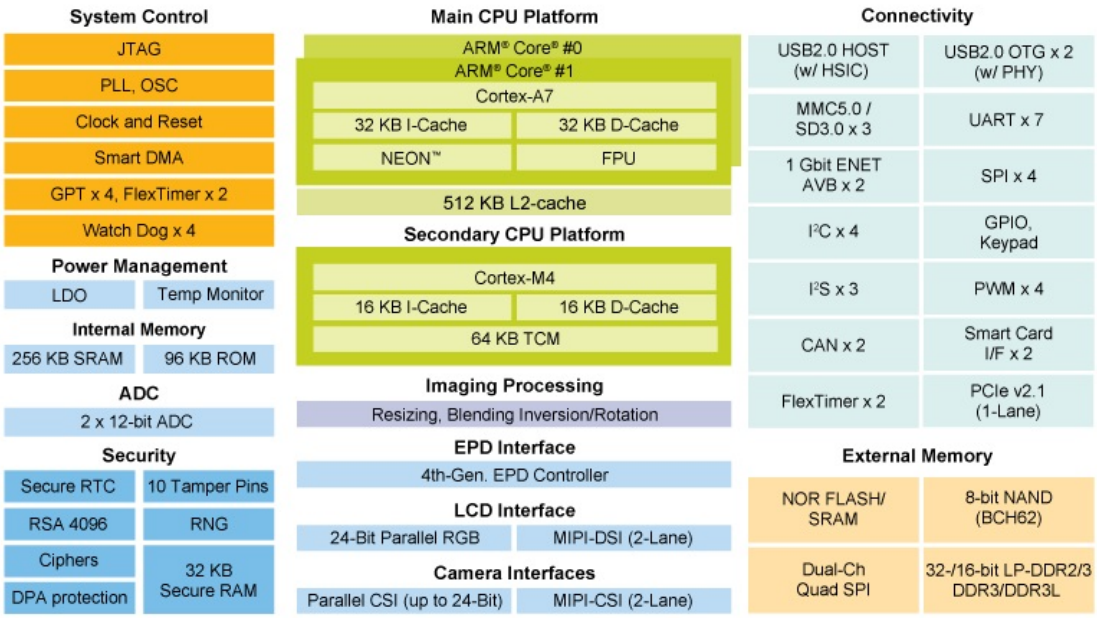
Obrázek 2.1: Vyobrazení všech mikrokontrolerů rodiny i.MX6 [1]

2.1.1 Mikrokontrolér i.MX7D

Jedná se o výkonný procesor určený pro nízkopříkonové případy užití. Má specifické vlastnosti pro mobilní bateriemi poháněné aplikace. Přímo v tomto mikrokontroléru je integrovaný řadič pro e-papírové displeje. Nabízí rozhraní pro síťové periferie jako jsou Wifi, Bluetooth a další. Na obrázku 2.2 je vyobrazen blokový diagram s nejdůležitějšími prvky mikrokontroléru i.MX7D, velikosti vyrovnávacích pamětí a interních pamětí. Z obrázku je také patrné, že neobsahuje grafický čip.

Mikrokontrolér obsahuje dvě jádra, která jsou založena na architektuře ARM Cortex-A7 (taktované až na 1 GHz) a jedno jádro na ARM Cortex-M4 (taktované až na 200 MHz). Jádra ARM Cortex-A7 obsahují:

- Jednotku pro práci s médii, založené na technologii NEON (SIMD instrukce),
- rozšíření pro zabezpečení,
- rozšíření pro virtualizaci,
- rožšíření pro Large Physical Addres (LPA),
- 512KB L2 vyrovnávací paměti,
- snoop control unit (SCU),
- globální řadič přerušení s podporou 128 sdílenými přerušeními od periferií,
- in-order pipeline s přímou a nepřímou predikcí skoku,



Obrázek 2.2: Blokový diagram mikrokontroleru i.MX7D [3]

- jednotku pro práci s čísly v plovoucí desetinné čárce.

Jádro ARM Cortex-M4 obsahuje:

- Integrovaný Nested Vectored Interrupt Controller (NVIC),
- jednotku pro práci s čísly v plovoucí desetinné čárce,
- CoreMPU (memory protection unit).

2.1.2 Podpora pro více jader

Pro podporu více jader a jejich spravování obsahuje:

- Resource domain controller (RDC) který přiřazuje jednotlivým jádrům práva oblasti pamětí a periferie. Také hlídá, aby jádro nepoužívalo prostředky, ke kterým nemá práva,
- jednotku pro zasílání zpráv,
- hardwarové semafory,
- shared bus topology,
- ARMv7 exkluzivní přístup do externí paměti.

2.1.3 Startování systému

Při zapnutí napájení se jako první startuje jádro Cortex-A7 a jádro Cortex-M4 může být spuštěno během startovaní (například skriptem v bootloaderu, nebo až po nastartování operačního systému).

Po restartování systému začne jádro ARM startovat z boot ROM kódu, který je umístěn na čipu. Tento ROM kód se rozhodne podle vnitřního dvoubitového registru *BOOT_MODE*, nastavení pojistek a pinů, jak bude probíhat startování systému. V registru *BOOT_MODE* můžou být čtyři hodnoty, z toho je jedna rezervovaná. Zbývající tři možnosti jsou startování podle pojistek, nahrávání přes USB nebo vnitřní startování.

Mezi vlastnosti ROM kódu patří:

- Podpora startování z různých zařízení a pamětí,
- nahrávání obrazů přes USB OTG,
- konfigurační data zařízení (DCD),
- digitální podpisy a šifrování založené na High Assurance Boot (HAB),
- probuzení z nízkopříkonových režimů.

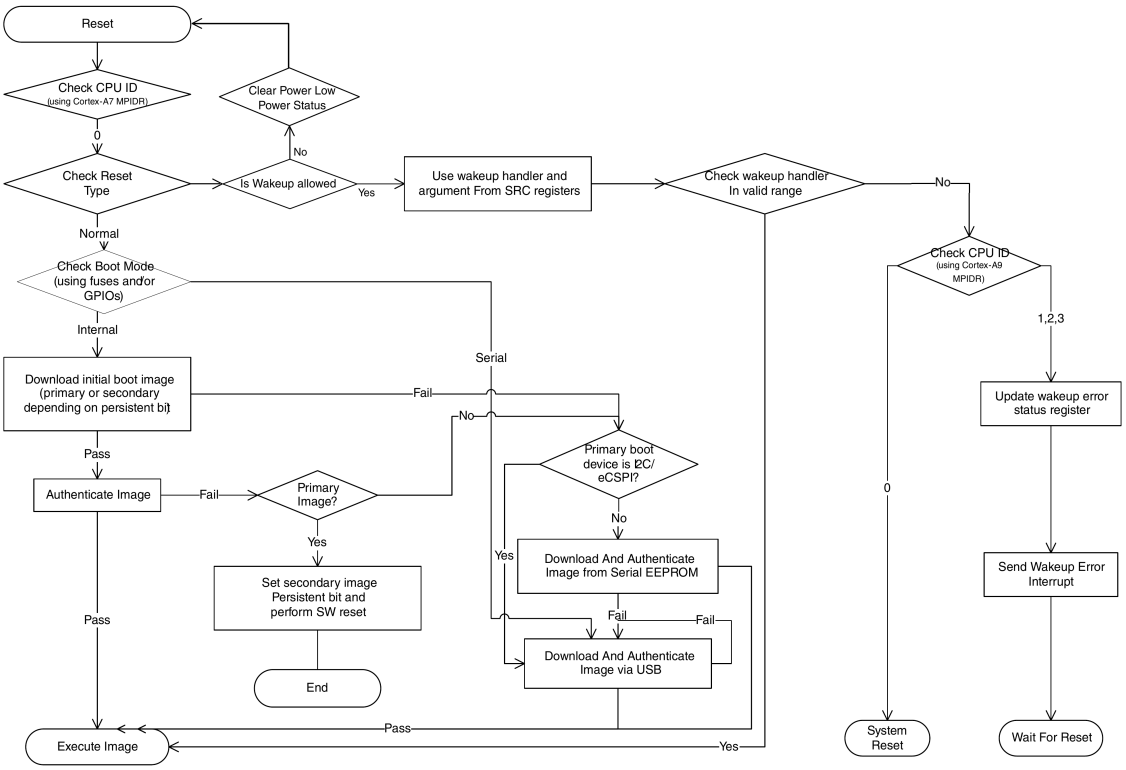
ROM kód hledá obraz pro procesor na těchto zařízeních:

- NOR flash,
- NAND flash,
- SDIO/MMC/SDXC,
- eSD 3.0/eMMC 5.0,
- SPI,
- USB (recovery mode),
- USB paměť,
- Ethernet,
- QSPI,
- PCIe.

ROM kód může být kromě startování systému využit pro nahrávání obrazů operačního systému (nebo i baremetal programů). Kromě nahrání do různých pamětí typu flash, může být i kód nahrán přímo do paměti RAM a z ní spuštěn. Komunikace s ROM kódem probíhá přes USB OTG v módu HID viz odstavec 2.1.4.

Probuzení z níkopříkonových režimů probíhá také přes ROM kód, přičemž kontroluje v registrech, jestli se jedná o probuzení a pokud ano, tak nenahrává aplikaci, ale rovnou skočí na adresu uloženou v *PERSISTENT_ENTRY0* a pokračuje v provádění kódu.

Nastavování konfiguračních dat zařízení (DCD) umožňuje ROM kódu získat data o SOC z obrazu systému uloženého na startovacím zařízením. Například takto získá optimální



Obrázek 2.3: Startování procesoru za pomoci ROM kódu [4]

informace o nastavení DDR pamětí. DCD je omezeno jen na určité regiony v paměti a adresy periferií, jenž jsou pro tyto data určeny.

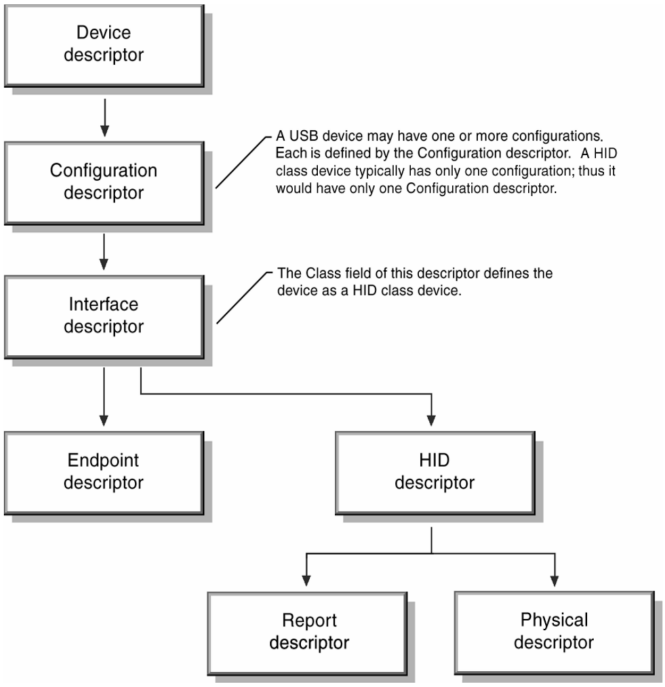
Hlavní vlastností ROM kódu je zajištění secure bootu za pomoci High Assurance Boot (HAB). HAB je knihovna, která je komponentou ROM kódu. Za pomocí této knihovny, kombinací hardwaru a softwaru a využití Public Key Infrastructure (PKI) protokolu je možnost chránit procesor před spuštěním neautorizovaného kódu. Pokud je tato vlastnost aktivovaná, musí být každý program, respektive obraz operačního systému podepsaný veřejným klíčem. Privátní klíč k tomuto veřejnému klíči se uloží do registrů (může být až pět privátních klíčů). Kromě zajištění secure bootu může být využito i šifrování paměti [4].

Na obrázku 2.3 lze vidět zjednodušený postup startování systému. V levé části obrázku je vyobrazeno startování systému po resetu a v pravé části je probouzení z úsporných režimů.

2.1.4 USB OTG a USB HID

USB On-The-Go (OTG) umožňuje produktům, jako telefony nebo tablety, chovat se ne jako USB zařízení, ale jako USB host (Například lze k telefonu připojit USB disk a získat z něj data). Přičemž při připojení do USB sběrnice, kde host už existuje, může se tento produkt chovat jako klasická USB periferie.

Human interface device (HID) je druh zařízení, jenž jsou určena pro komunikaci mezi počítačem a člověkem. Tento druh komunikace využívají například myši, klávesnice, sluchátka, grafické tablety, webkamery a další. Standart definuje konfiguraci, komunikační protokol a základní funkčnost těchto zařízení, ale ponechává prostor pro rozšíření.



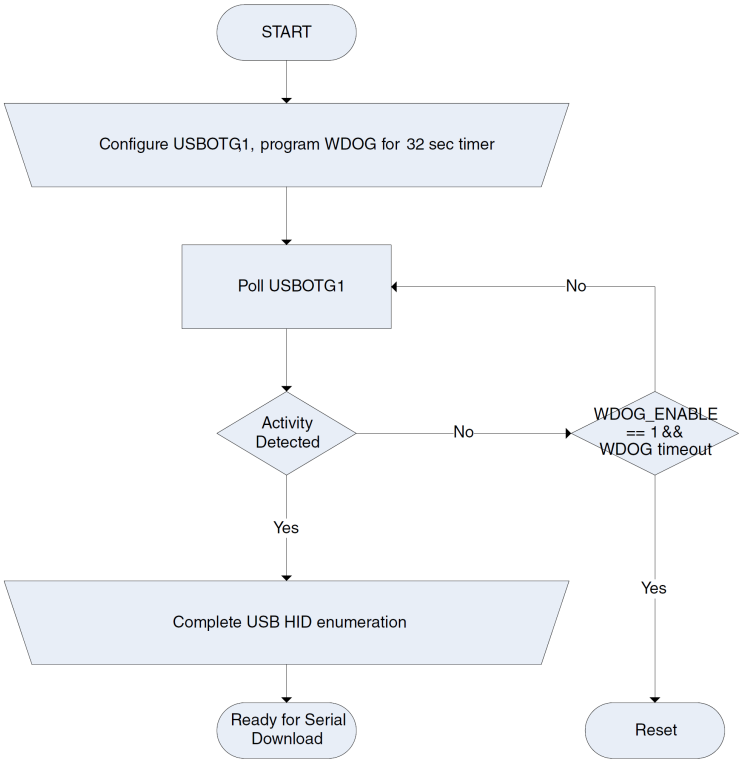
Obrázek 2.4: USB HID deskriptory [7]

V ROM paměti zařízení je uloženo nastavení v podobě deskriptorů, přičemž třída HID používá class-specific descriptors, které se liší od standardních USB deskriptorů, viz obrázek 2.4.

- **HID Descriptor** určuje počet, typ a velikost zprávy HID deskriptorů a fyzické deskriptory, které jsou se zařízením spojeny,
- **HID Descriptor Report** určuje formát dat, která mohou zařízení posílat nebo přijímat a přitom nejsou definována ve specifikaci třídy HID. Pro každé pole v HID Descriptor report je jeden deskriptor, který definuje, kolik bitů konkrétní datová položka zabírá, jaké má využití, jaké je rozmezí hodnot, atd.

Pro získání informací lze zaslat požadavek na stav, nastavení nebo přímo nastavit stav. Využívají se následující požadavky:

- GET_REPORT (získej zprávu),
- SET_REPORT (nastav zprávu),
- GET_IDLE (získej nečinnost),
- SET_IDLE (nastav nečinnost),
- GET_PROTOCOL (získej protokol),
- SET_PROTOCOL (nastav protokol).



Obrázek 2.5: Startování procesoru v režimu Serial downloader [4]

2.1.5 Komunikace s ROM kódem

Při startu systému v režimu Serial Downloader ROM kód nastaví watchdog na čas nastavený v pojistkách a zkouší spojení přes USB OTG. Pokud nenaváže spojení, tak se procesor restartuje. Toto je ilustrováno na obrázku 2.5.

Samotná komunikace mezi počítačem a ROM kódem probíhá za pomoci USB HID (viz odstavec 2.1.4). Pro komunikaci se používají typy reportů, jenž jsou definované v tabulce 2.1. Je zde i maximální délka dat, jenž je možné pod tímto report ID poslat a v případě přijmutí je potřeba přebytečná data ignorovat.

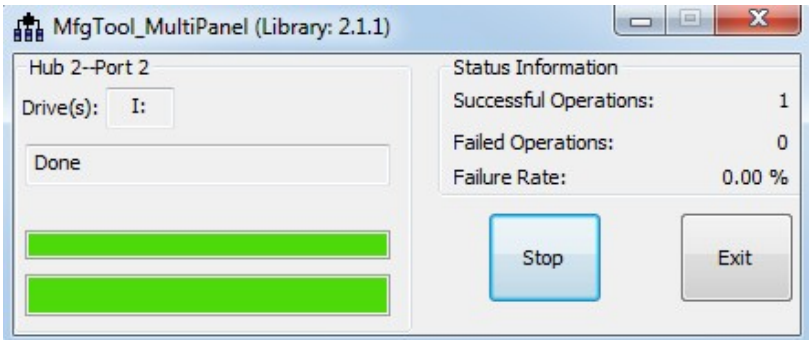
Jméno reportu	Report ID	Délka dat (bajty)
CMD (příkaz)	0x01	1024
DAT (data)	0x02	1024
SEC (zabezpečení)	0x03	4
RET (návratová hodnota)	0x04	64

Tabulka 2.1: Typy reportů, jejich report ID a délka dat.

Report CMD slouží pro zasílání příkazů, jenž jsou definované v tabulce 2.2 a pro přenos samotných dat se používá DAT. Pro zjištění, jestli je procesor zamčený, lze využít report SEC a v něm po vyčtení můžou být dvě možné hodnoty a to *0x12343412* v případě že je deska zamčená a nebo *0x56787856* pokud je deska odemčená. Poslední report RET slouží pro získání návratové hodnoty z určitého příkazu.

Příkaz	hodnota (hex)	Potvrzovací kód
READ	0x0101	
WRITE	0x0202	0x128A8A12
PC	0x0303	
WFILE	0x0404	0x88888888
ERROR	0x0505	
WCSF	0x0606	0x128A8A12
HWCFG	0x0707	
EXEC	0x0808	
REENUM	0x0909	
WDCD	0x0A0A	0x128A8A12
JUMP	0x0B0B	0
SKIPDCD	0x0C0C	0x900DD009

Tabulka 2.2: Příkazy pro CMD.



Obrázek 2.6: Uživatelské rozhraní nástroje MFG tool

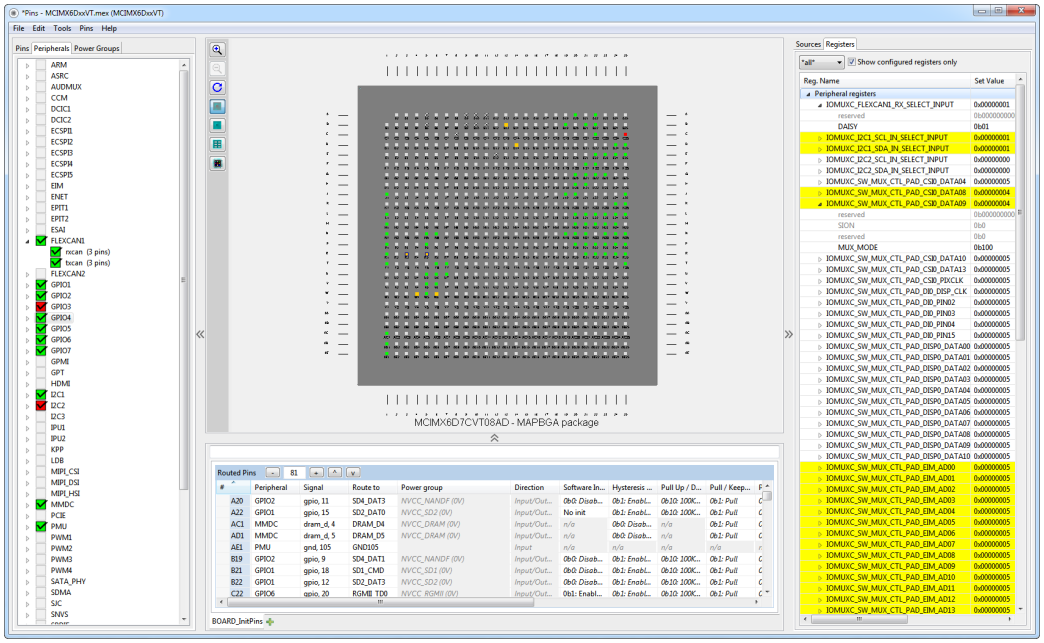
Příkazy jako READ a WRITE se dají použít například pro přístup do registrů a nebo do paměti a ERROR pro vyčtení poslední chyby. Pro nahrání obrazu operačního systému jsou nejdůležitější příkazy:

- **WDCD** slouží pro zápis konfiguračních dat zařízení,
- **WFILE** umožňuje zapsat blok dat,
- **SKIPDCD** je důležitý, aby se dalo vyhnout znovunačtení konfiguračních dat,
- **JUMP** slouží pro spuštění kódu z určité adresy.

2.2 Nástroje pro práci s vývojovými kity na bázi i.MX

2.2.1 MFG tool

Nástroj Manufacturing Tool (zkráceně MFG tool) slouží pro nahrání obrazů operačního systému a nebo aplikace. Na obrázku 2.6 je vidět uživatelské rozhraní tohoto nástroje. Tento nástroj je především určen do výroby, z čehož plyne jeho jednoduché rozhraní. Zaměstnanec připojí desku a zmáčkne tlačítko start. Pro pokročilé ovládání této aplikace je potřeba upravovat skripty a konfigurační soubory, přičemž tyto skripty mají stovky řádků. Což při vývoji aplikace není moc vhodné.



Obrázek 2.7: Uživatelské rozhraní i.MX Pin Mux Tool

Kromě této špatné uživatelské přívětivosti, obsahuje aplikace řadu technických nedostatků. Největší z nich je, že provádí kontrolu obrazu za pomocí kontrolního součtu až po dokončení celého nahrávání. Pokud je kontrolní součet špatný, začne nahrávat celý obraz od začátku. V případě velkého obrazu a horšího kabelu, může nahrávání trvat velmi dlouho a v extrémních případech může dojít i k poškození flash paměti, z důvodu příliš velkého počtu zápisů. Další problém je se zpětnou kompatibilitou a roztříštěností (velké množství verzí, přičemž každá verze funguje s určitým mikrokontrolérem) tohoto nástroje. Pokud by byl využit pro některý starší procesor z rodiny Vybrid, tak hledáním funkční verze pro daný procesor může vývojář strávit i několik dní.

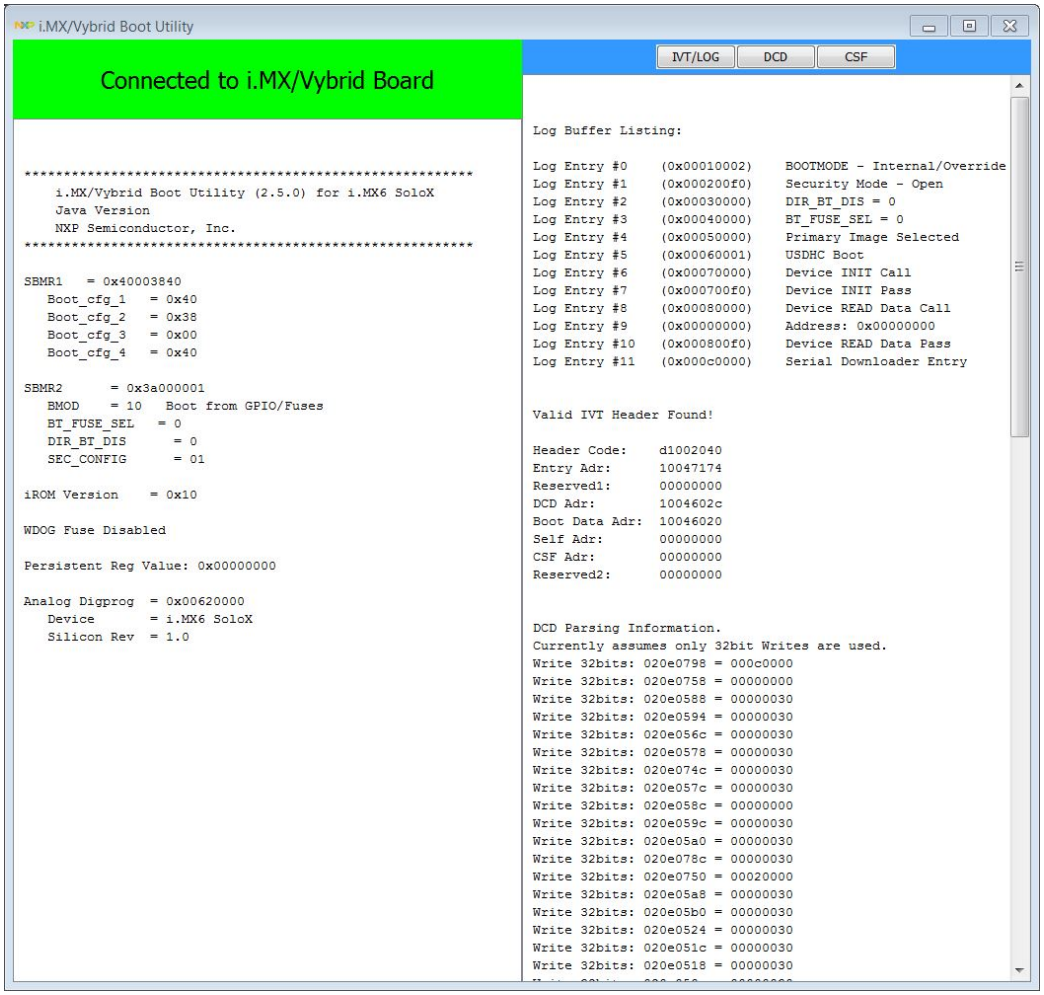
2.2.2 i.MX Pin Mux Tool

Jedná se o vyspělý nástroj napsaný v jazyce Java, který slouží pro nastavování pinů mikrokontrolerů. Využívá rozsáhlou databázi, ale na rozdíl od nástrojů pro mikrokontrolery Kinetis, nepoužívá vše, co se v ní nachází (například adresy všech registrů, jenž procesor obsahuje). Proto bych chtěl ve své aplikaci implementovat nastavování registrů, jejichž seznam je ve jmenované databázi.

Na obrázku 2.7 je vyobrazeno uživatelské rozhraní nástroje i.MX Pin Mux Tool. Toto rozhraní je velmi přehledné a nabízí i spoustu pokročilých funkci.

2.2.3 SB loader

Nástroj SB loader slouží pro nahrávaní obrazů operačního systému. Jedná se o konzolovou aplikaci, jenž nabízí základní příkazy jako nahrát obraz ze souboru, spuštění systému z určité adresy a podobně. Není tedy možné využít pokročilé funkce, jenž nabízí ROM kód (čtení a zápis na určité adresy apod.). Další nevýhodou je pouze podpora operačního systému Windows.



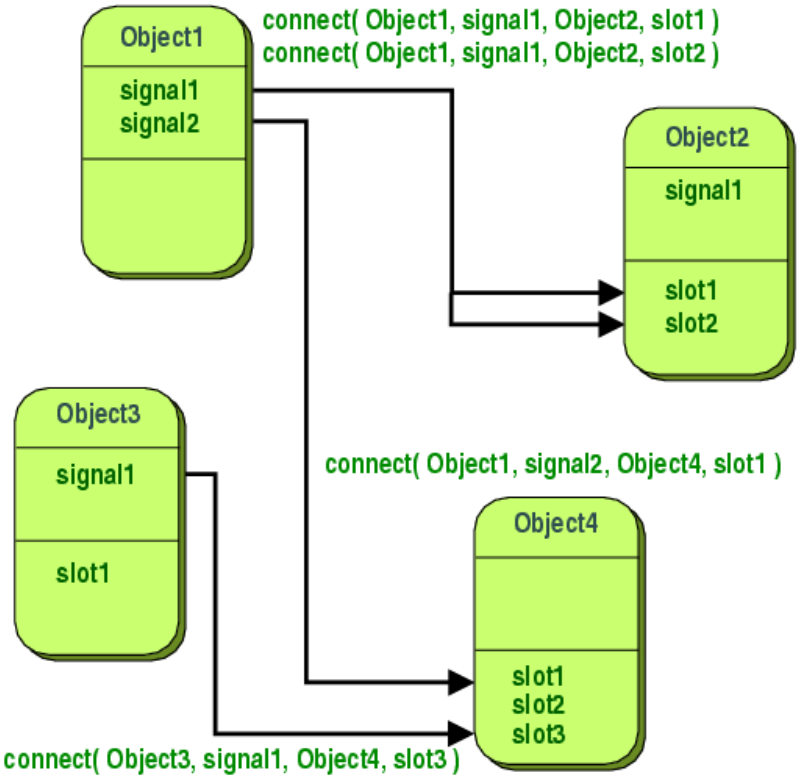
Obrázek 2.8: Nástroj i.MX and Vybrid Boot Utility

2.2.4 i.MX and Vybrid Boot Utility

Po zahájení prací na mém nástroji se objevil nástroj i.MX and Vybrid Boot Utility. Je zatím v beta verzi a má velmi omezené schopnosti. Jak je vidět na obrázku 2.8, neobsahuje žádné grafické uživatelské rozhraní a jedná se spíše o skript napsaný v jazyce Java. Využívá komunikaci s ROM kódem přes USB HID a prozatím dokáže jen vypsat základní informace a chyby při startu procesoru.

2.3 Qt5

Qt je multiplatformní framework nad jazykem C++. Umožňuje vytvářet aplikace s grafickým uživatelským rozhraním a přidává funkci signálů a slotů 2.3.1. Pro vytváření grafického uživatelského rozhraní nabízí dvě možnosti a to widgety nebo Qt Quick. Widgety jsou třídy v jazyce C++ a jejich skládáním se vytváří GUI. Qt Quick využívá QML, což je jazyk podobný JavaScriptu, ze kterého se můžou volat C++ funkce. GUI se vytváří za pomocí značkovacího jazyka.



Obrázek 2.9: Propojení signálů se sloty [5]

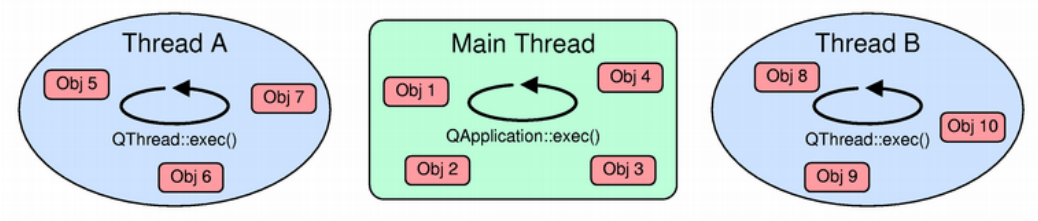
2.3.1 Signály a sloty

Signály a sloty jsou v Qt použity jako komunikace mezi objekty. Jedná se o jednu ze stěžejních vlastností frameworku Qt a tímto systémem nahrazují Callback funkce.

Signál představuje nějakou událost, kterou chce daný objekt informovat ostatní (například stisk tlačítka). Slot je funkce, která má reagovat na nějakou událost. Kromě předdefinovaných signálů a slotů si může programátor vytvořit vlastní signály a sloty.

Na obrázku 2.9 je zobrazeno propojení signálů a slotů. Toto propojení se vytváří za pomocí funkce *connect()*, které se předá zdrojový objekt a požadovaný signál a cílový objekt s požadovaným slotem. Toto propojení může být vytvořeno mezi libovolnými objekty a signál může být propojen s více sloty a naopak. Také je možné propojit signál se signálem a po vyvolání prvního signálu se automaticky zavolá i druhý signál. Signály a sloty můžou mít i parametry, přes které lze předávat dodatečné informace a při jejich propojování je musí mít kompatibilní. Sloty můžou být i volány jako normální členské funkce daného objektu a také mohou být definovány jako virtuální metody.

Po vyslání signálu za pomocí klíčového slova *emit* se okamžitě začnou provádět přidružené sloty. Toto je výhodné, pokud nechceme čekat na spuštění *event loop* daného objektu a je výchozí. Chování se dá přepnout, když se ve funkci *connect* nastaví příznak *Qt::QueuedConnection*. Poté se vyvolané signály ukládají do fronty objektu, jenž obsahuje slot (běh programu pokračuje kódem, jenž je po volání *emit*). Vlastník příslušného slotu obslouží všechny příchozí signály, jakmile dokončí svou práci a dostane se do *event loop*. Toto chování je žádoucí v případě použití vláken, kdy při klasické obsluze ihned by daný



Obrázek 2.10: Diagram tří vláken[6]

slot provádělo vlákno, jenž zavolalo signál, kdežto u řazení do fronty ho provede vlákno, jenž vlastní daný objekt [5].

2.3.2 Moduly

Framework Qt nabízí možnost vytvoření modulů, které se můžou do aplikace linkovat jak dynamicky tak i staticky. Tyto moduly nejsou nijak omezené a můžou libovolně rozšířit funkci aplikace.

Aby byla aplikace rozšiřitelná o moduly, musí se v kódu provést tyto kroky:

1. Definovat rozhraní modulů (třída, která má jen pure virtual funkce),
2. použít makro *Q_DECLARE_INTERFACE()*, které řekne meta-object systému o daném rozhraní,
3. použít objekt *QPluginLoader* v aplikaci pro nahrání modulů,
4. použít *qobject_cast()* pro otestování, zda modul implementuje dané rozhraní.

Při vytváření modulu je potřeba provést tyto kroky:

1. Deklarovat třídu modulu tak, že dědí z třídy *QObject* a z požadovaného rozhraní,
2. použít makro *Q_INTERFACES()*, které řekne meta-object systému o daném rozhraní,
3. exportovat modul za pomoci makra *Q_PLUGIN_METADATA()*,
4. sestavit modul dobře nastaveným projektovým souborem. Je potřeba v něm nastavit:
   - TEMPLATE = lib,
   - CONFIG += plugin,
   - uložení výstupu kompilace (nejlépe do podsložky plugins u výstupu aplikace).

2.3.3 Vlákna

Pro využití vláken je možnost použít standardních vláken z C++ a nebo Framework Qt má vlastní implementaci vláken v objektu *QThread*.

Po startu aplikace se nastartuje hlavní vlákno, ve kterém se mohou vytvořit další vlákna. Po jejich vytvoření můžeme do nich za pomoci funkce *moveToThread()* přesunout určitý

objekt, který mají vykonávat (těchto objektů může být i více). Vlákna můžou mezi sebou komunikovat a dokonce i využívat signály a sloty. Vykreslovat může jen hlavní vlákno a tudíž objekty, jenž jsou ve vlastním vlákně, nemohou vytvářet grafické uživatelské rozhraní [6].

Na obrázku 2.10 je vyobrazen diagram tří vláken, kdy v každém z nich je několik objektů a ukazuje, že každé vlákno má svoji vlastní *event loop* a díky tomu je možné mezi vlákny zasílat signály a události.

Kapitola 3

Návrh aplikace

Aplikace, která bude v rámci diplomové práce navržena, se bude skládat ze správce a modulů. Mezi moduly nebude komunikace nijak omezena a záleží jen na programátorovi modulu, jak bude komunikovat. Každý modul, který nebude vykreslovat, poběží ve vlastním vlákně, a tudíž určitá funkčnost může být rozdělena do více modulů tak, aby nedocházelo ke zpomalování vykreslování grafického uživatelského rozhraní.

3.1 Aplikace

Samotná aplikace se bude starat o nahrávání modulů, správu modulů a zajišťování komunikace mezi nimi. Grafické uživatelské rozhraní je vytvořeno za pomocí záložek (jako v prohlížeči), kde každý modul, který vykresluje, dostane vlastní záložku.

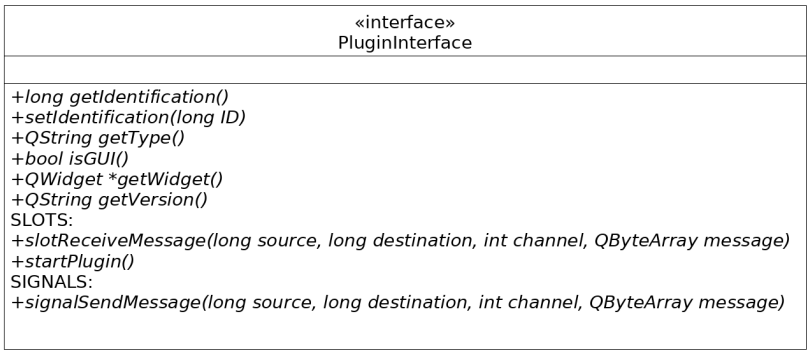
Aplikace načte moduly ze složky *plugins*. Načítá je po jednom s tím, že pokud se jedná o modul, který vykresluje grafické uživatelské rozhraní, tak aplikace vytvoří novou záložku a do ní přiřadí *QWidget*, který získá od modulu. Pokud nevykresluje, tak ho přesune do vlastního vlákna. Po vytvoření modulu se vloží jeho záznam v podobě obalovacího objektu do pole a přiřadí se mu ID, které slouží zároveň také jako index do tohoto pole. Tento objekt kromě odkazu na modul bude obsahovat také další informace o tomto modulu proto, aby nebylo nutné přistupovat do ostatních vláken, a hlavně za použití pomocného signálu umožní propojení, kdy veškerá komunikace proudí přes hlavní aplikaci (viz 3.3).

3.2 Moduly

Moduly by neměly být nijak závislé na hlavní aplikaci, ale můžou být závislé na jiných modulech. Proto bude potřeba, aby moduly byly označeny svou aktuální verzí. Při hledání svých závislostí zjistí od hlavní aplikace jejich verze, díky kterým si potom může ověřit kompatibilitu.

Každý modul musí implementovat rozhraní, které je vyobrazeno na obrázku 3.1. Význam jednotlivých položek je následující:

- **setIdentification:** přes tuto funkci může hlavní aplikace nastavit ID modulu,
- **getIdentification:** tato funkce vrátí nastavený identifikátor,
- **getType:** tato funkce vrátí název modulu, přes který se dá modul vyhledat,



Obrázek 3.1: Rozhraní modulů

- **isGUI:** pokud modul bude vykreslovat, tak vrátí hodnotu *true*, podle toho se hlavní aplikace rozhodne, jestli modul přesune do vlastního vlákna. viz odstavec 2.3.3,
- **getWidget:** jestliže aplikace vykresluje, tak přes tuhle funkci hlavní aplikaci vrátí QWidget, do kterého bude vykreslovat,
- **getVersion:** modul vrátí svoji verzi,
- **slotReceiveMessage:** tento slot slouží pro komunikaci viz odstavec 3.3,
- **startPlugin:** tento slot se spustí po přesunutí modulu do vlastního vlákna,
- **signalSendMessage:** tenhle signál slouží pro komunikaci viz odstavec 3.3.

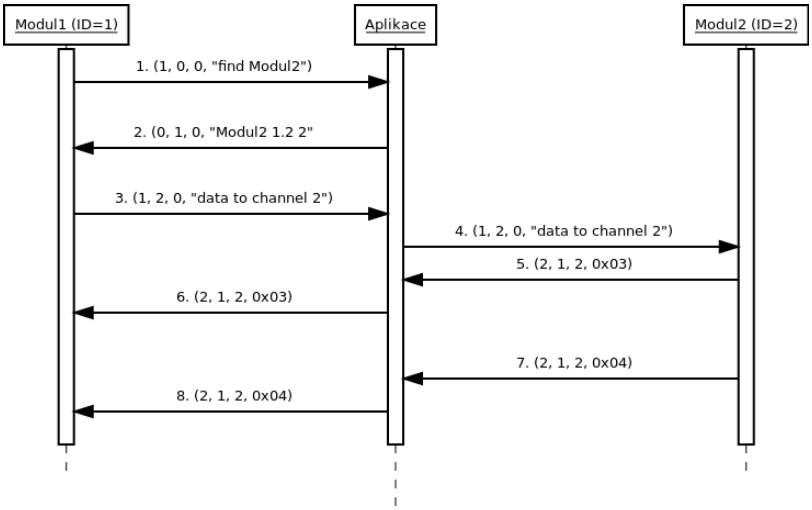
3.3 Komunikace mezi moduly

Pro komunikaci jsem se rozhodl využít signálů a slotů, které nabízí framework Qt, viz odstavec 2.3.1. Protože komunikace bude probíhat mezi různými vlákny, bude se využívat příznak *Qt::QueuedConnection*, aby nedocházelo ke konfliktům.

Na obrázku 3.1 je vidět, že v rozhraní modulů je definován signál (*signalSendMessage*) a slot(*slotReceiveMessage*) pro komunikaci. Mají shodné parametry a jsou to tyto:

- **source** - ID odesílatele,
- **destination** - ID příjemce,
- **channel** - využití tohoto parametru je čistě na programátorovi. Může být využito pro odlišení různých typů zpráv,
- **message** - samotná data, která jsou typu QByteArray, jenž může být čten jako pole bajtů a nebo jako řetězec.

Hlavní aplikace bude mít speciální příkaz *find*, který je určen pro vyhledávání ID modulů (hlavní aplikace má vždy ID=0). Jako parametr se předá jméno hledaného modulu. Hlavní aplikace na tuto zprávu odpoví tak, že mu vrátí jméno hledaného, jeho verzi a jeho ID.



Obrázek 3.2: Komunikace mezi dvěma moduly

Na obrázku 3.2 je vyobrazena komunikace mezi dvěma moduly. Stojí za povšimnutí, že veškerá komunikace proudí přes hlavní aplikaci. Pro tuto variantu jsem se rozhodl z důvodu, že kdyby byl propojen každý modul s každým, tak by se každá zpráva zaslala všem modulům a jen příslušný modul by si ji přečetl a zbytek modulů by ji zahodil. V systému by proudilo velké množství zbytečných zpráv. Tenhle případ komunikace nastává, když chce Modul1 získávat pravidelně data od Modul2 (například změna připojeného zařízení). Probíhá to v těchto krocích:

1. Modul1 vyhledá přes hlavní aplikaci ID Modulu2,
2. hlavní aplikace vrátí Modulu1 ID a verzi Modulu2,
3. Modul1 zašle řídící příkaz pro Modul2 (zvoleno programátorem Modulu2) a číslo kanálu, na který chce data posílat,
4. hlavní aplikace přepošle zprávu od Modulu1 do Modulu2,
5. Modul2 posílá už surová data na daný kanál,
6. hlavní aplikace přepošle zprávu od Modulu2 do Modulu1.

Kapitola 4

Implementace

Tato kapitola se zaobírá samotnou implementací aplikace a všech vytvořených modulů. Kromě funkce jednotlivých částí budu popisovat i komunikaci mezi moduly, jenž tvoří klíčovou složku celé aplikace. Dále zde budu ukazovat diagramy tříd, slovní popis fungování a ve složitějších situacích i pseudokód.

Celý zdrojový kód je možno nalézt na přiloženém CD, kde adresářová struktura je následující:

- **bin** do této složky se ukládají zkompilované binární soubory,
- **src** obsahuje zdrojové kódy,
- **src/app** jsou v ní zdrojové kódy správce modulů,
- **src/plugins** každý modul v této složce má svojí podsložku se zdrojovými kódy,
- **common** hlavičkové soubory, jenž využívá více modulů,
- **scripts** modul pro skripty bere tuto složku jako výchozí.

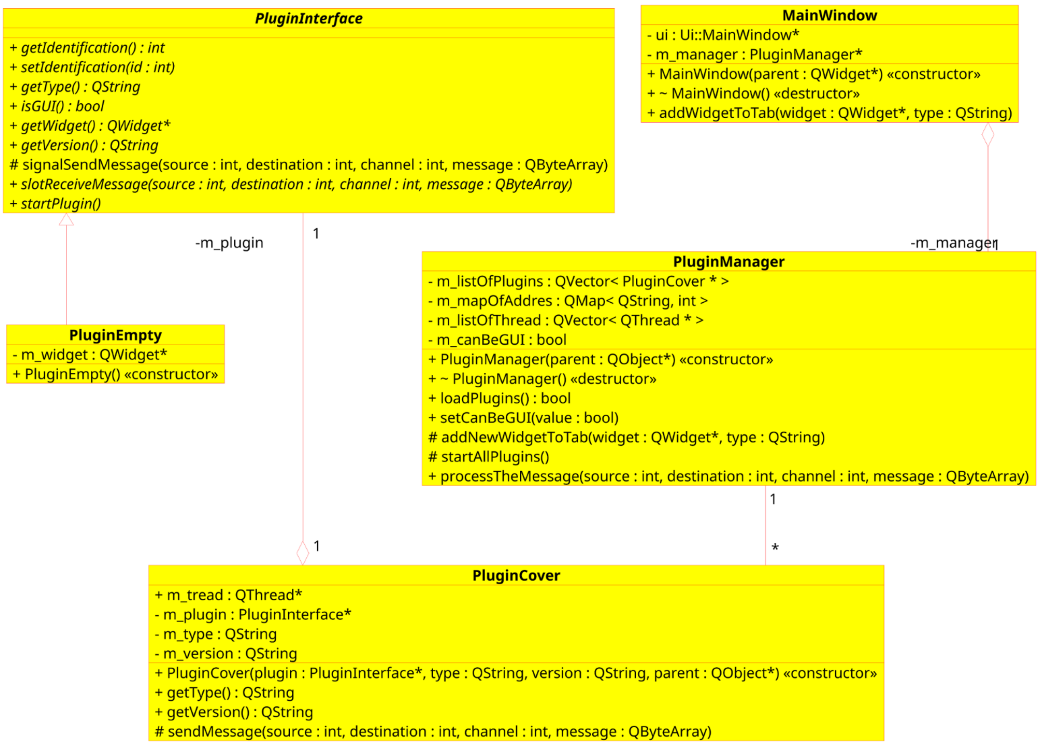
4.1 Správce modulů

Implementace správce modulů vychází z odstavce 3.1. Hlavní třída v této části je *MainWindow*, která dědí z *QMainWindow*, což je třída pro vytvoření hlavního okna aplikace a obsahuje tedy základ pro grafické rozhraní. Na obrázku 4.1 je vyobrazen diagram tříd. Zdrojové kódy pro tuto část jsou ve složce *src/app* na přiloženém CD. Jednotlivé třídy mají v této složce vždy své odpovídající soubory s koncovkou *.h* a *cpp* (název třídy odpovídá názvu souboru). Společné hlavičkové soubory jsou ve složce *common*.

Moduly jsem se rozhodl zabalit do pomocné třídy *PluginCover*. Do tohoto objektu se kromě odkazu na modul ukládá i základní informace o tomto modulu, jako je typ a verze. A to z toho důvodu, aby se pro tyto informace nemuselo přistupovat do jiného vlákna. Dále obsahuje i odkaz na vlákno, aby se dal modul ovládat z hlavního vlákna.

Samotná funkce správy modulů je implementována ve třídě *PluginManager*. Po startu aplikace je potřeba nejdříve načíst moduly, jenž jsou umístěny ve složce *plugins*. Postup načítání je nastíněn v pseudokódu 4.1.

Propojení komunikace mezi moduly je realizováno ve slotu, na který jsou napojeny všechny moduly. Nejdříve se zkontrolují ID zdroje a cíle. Pokud je ID cíle rovno nule, tak



Obrázek 4.1: Diagram tříd správce modulů

je zpráva určena pro něho a vyřídí se požadavek (prozatím jen příkaz *find*), pokud ne, tak správce modulů zprávu přepošle cílovému modulu za pomocí mapy ID, která byla vytvořena při načítaní modulů. Tato třída je před-připravená pro konzolovou verzi aplikace.

Kód 4.1: Pseudokód načítání modulů

```
1  foreach(files in folder plugins){
2          if(plugin is valid){
3                  if(is there already same plugin){
4                          error;
5                  }
6
7                  add pluginCover with plugin to list;
8                  add plugin ID to map;
9                  connect plugin to message system;
10
11                 if(plugin have gui){
12                         get plugin widget;
13                         assign widget to new tab;
14                 } else{
15                         create new thread;
16                         assign thread to pluginCover;
17                         move plugin to thread;
18                 }
19         }
20 }
21 start all plugins;
```

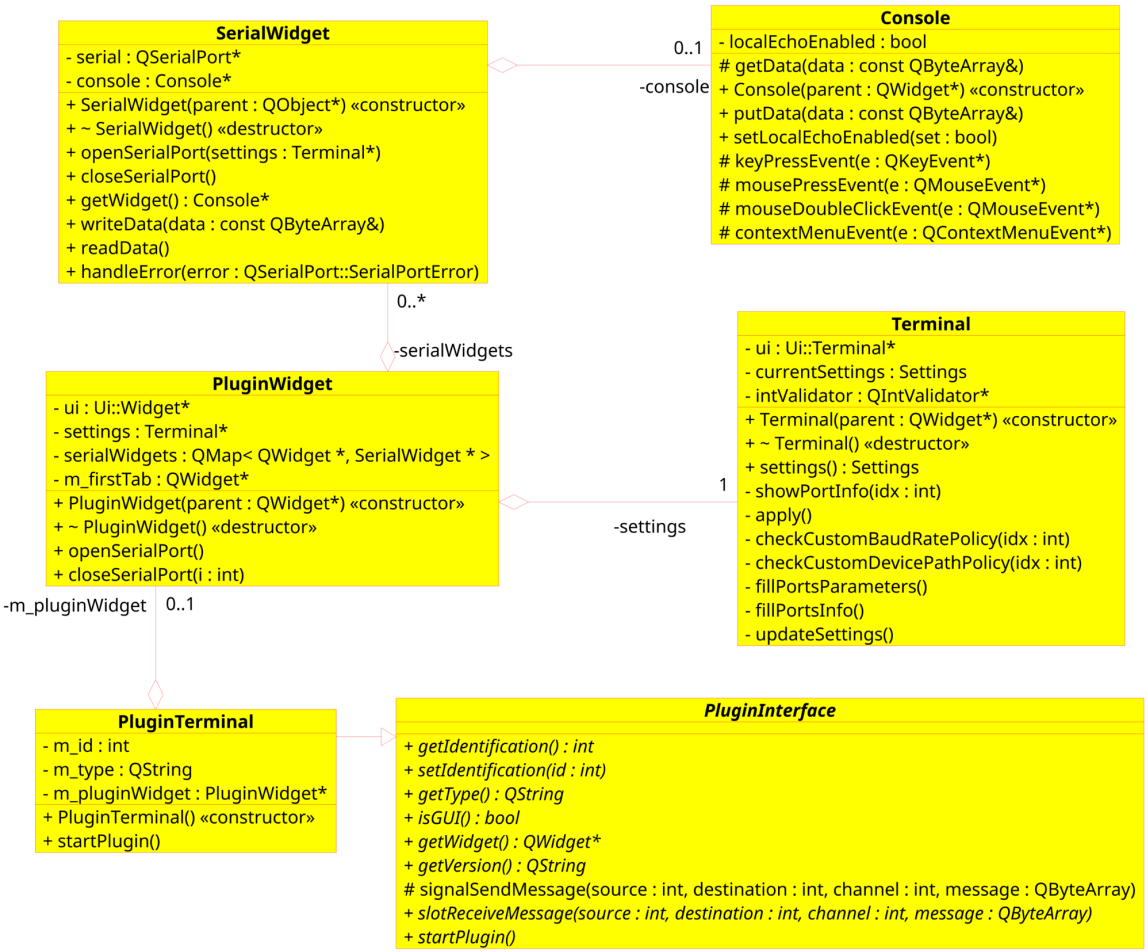
4.2 Ukázkové moduly

Jako první jsem implementoval dva ukázkové moduly, které fungují jako producent a konzument. Jsou napsány co nejjednodušeji tak, aby se z nich dalo co nejlépe vycházet při implementaci dalších modulů. Obsahují tudíž jen nejnutnější nastavení a minimum věcí pro svůj běh (jenž je definováno v sekci 3.2 Moduly). Tudíž je nutno definovat funkce, jenž jsou ve třídě *PluginInterface* a příklad toho, jak to může být co nejjednodušeji provedeno, je v kódu 4.2.

Zdrojové kódy pro tuto část jsou ve složkách *src/plugins/plugin1Test* a *src/plugins/plugin1TestGUI* na přiloženém CD. Jednotlivé třídy mají v této složce vždy své odpovídající soubory s koncovkou *.h* a *cpp* (název třídy odpovídá názvu souboru). Společné hlavičkové soubory jsou ve složce *common*.

Kód 4.2: Potřebné minimum pro funkci modulu

```
1  PluginTerminal::PluginTerminal(){
2          m_type = "terminal";
3          m_pluginWidget = new PluginWidget();
4  }
5
6  void PluginTerminal::startPlugin(){
7
8  }
9
10 int PluginTerminal::getIdentification(){
11         return m_id;
12 }
13
14 void PluginTerminal::setIdentification(int id){
15         m_id = id;
16 }
17
18 QString PluginTerminal::getType() {
19         return m_type;
20 }
21
22 bool PluginTerminal::isGUI() {
23         return true;
24 }
25
26 void PluginTerminal::slotReceiveMessage(int source,
27         int destination, int channel, QByteArray ){
28
29 }
30
31 QWidget *PluginTerminal::getWidget(){
32         return m_pluginWidget;
33 }
34
35 QString PluginTerminal::getVersion(){
36         return QString().number(APP_VERSION);
37 }
```



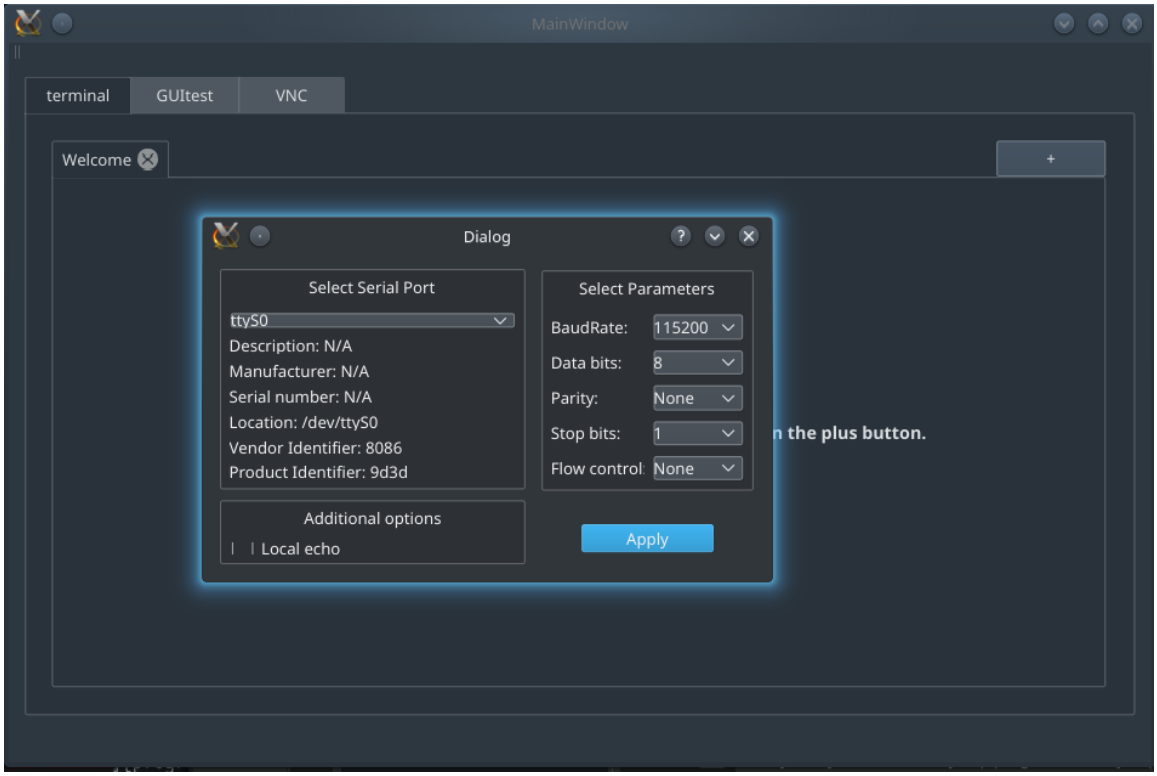
Obrázek 4.2: Diagram tříd modulu terminál

Jeden modul, jenž slouží jako producent, neobsahuje uživatelské grafické rozhraní a druhý, jenž je konzument, ho obsahuje (při implementaci vlastních modulů může programátor vyjít právě z toho modulu, podle toho, jestli chce uživatelské grafické rozhraní). Moduly fungují tak, že konzument se zaregistruje u producenta a ten každých 100 ms zvedne číslo o jedničku a pošle zaregistrovaným konzumentům novou hodnotu. Konzument po příjmu nové hodnoty vykreslí do svého okna tuto hodnotu.

4.3 Terminál

Tento modul implementuje funkcionalitu pro komunikaci přes sériovou linku. Modul funguje samostatně a ke své funkci nepotřebuje žádný jiný modul. Na obrázku 4.2 je vyobrazen diagram tříd modulu terminál. Třída *PluginTerminal*, jenž dědí ze třídy *PluginInterface* obsahuje pouze potřebné minimum (jenž je definováno v sekci 3.2 Moduly a ukázka kódu je v kódu 4.2) pro správnou komunikaci se správcem modulů. Vytváří také objekt z třídy *PluginWidgets*, jenž obsahuje grafické uživatelské rozhraní.

Zdrojové kódy pro tuto část jsou ve složce *src/plugins/pluginTerminal* na přiloženém CD. Jednotlivé třídy mají v této složce vždy své odpovídající soubory s koncovkou *.h* a *cpp* (název třídy odpovídá názvu souboru). Společné hlavičkové soubory jsou ve složce *common*.



Obrázek 4.3: Uživatelské grafické rozhraní modulu terminál

Grafické uživatelské rozhraní, jenž je vyobrazeno na obrázku 4.3, využívá panely. Každý panel představuje jeden otevřený terminál. Po kliknutí na tlačítko pro přidání nového terminálu, se zobrazí dialogové okno, jenž je implementováno ve třídě *Terminal*. V tomto okně je možné vybrat, která sériová linka se má využít a nastavit vlastnosti komunikace. Po stisknutí tlačítka *Apply* se vytvoří nový panel s danou konfigurací. U každého panelu je tlačítko pro zavření.

Jednotlivé terminály jsou implementovány ve třídě *SerialWidget*. Otevřené terminály jsou uloženy v mapě, jenž je součástí třídy *PluginWidget* a pro jednoduché vyhledávání jsou indexovány za pomocí odkazu na jejich *QWidget* (ten se dá získat z panelu). Rozhodl jsem se pro toto indexování z důvodu, že panely se dají přesouvat a lze jim tak měnit pořadí. Třída *SerialWidget* obsahuje třídu *console*, jenž slouží pro zobrazování terminálu a třídu *QSerialPort*, která realizuje samotnou komunikaci přes sériovou linku. Třída *QSerialPort* je součástí frameworku Qt, pro čtení se využívá signál *readyRead* a pro zápis funkce *write*. Obsahuje i funkce pro nastavení parametrů sériové linky. Do těchto funkcí pro nastavení parametrů se předává konfigurace z dialogového okna *Terminal*.

Samotné grafické uživatelské rozhraní, v němž je text ze sériového portu, je implementováno ve třídě *Console*, jenž dědí z Qt třídy *QPlainTextEdit* (jedná se o prvek grafického uživatelského rozhraní, který slouží pro editaci textu). Ve třídě *Console* je implementována funkce *PutData*, která slouží pro vkládání textu a bere v potaz i mazací sekvence (např. 080820200808 pro smazání dvou znaků), které chodí přes sériovou linku. Také je zde přepsána funkce *keyPressEvent*, jenž slouží pro zachytávání znaků z klávesnice, a to z toho důvodu, aby bylo možno zajistit správné chování směrových kláves a klávesy *backspace*.

Obrázek 4.4: Zjednodušený diagram tříd modulu VNC

4.4 VNC

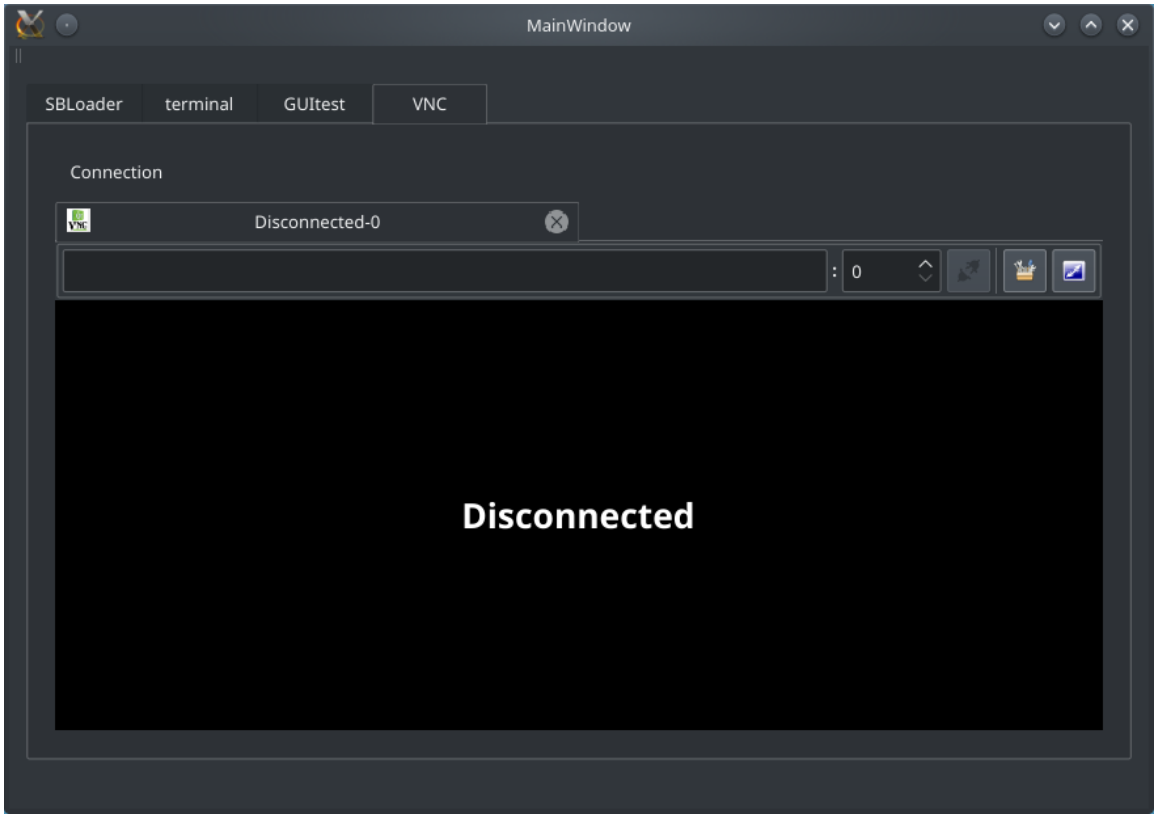
Tento modul složí jako VNC klient a je zcela nezávislý na ostatních modulech. Implementace tohoto modulu vycházela z již hotových řešení a za využití několika knihoven. Byly využity tyto knihovny:

- **zlib** knihovna pro kompresi,
- **libvncclient** knihovna pro klienta VNC,
- **libjpeg** knihovna pro práci s formátem JPEG.

Zdrojové kódy pro tuto část jsou ve složce *src/plugins/pluginVNC* na přiloženém CD. Jednotlivé třídy mají v této složce vždy své odpovídající soubory s koncovkou *.h* a *cpp* (název třídy odpovídá názvu souboru). Společné hlavičkové soubory jsou ve složce *common*. Kromě zmiňovaných tříd, jsou zde v podsložkách zdrojové kódy použitých knihoven.

Na obrázku 4.4 je vyobrazen zjednodušený diagram tříd modulu VNC. Je zde vidět třída *PluginVNC*, která komunikuje se správcem modulů a umožňuje tak VNC využívat jako modul. Tato třída obsahuje jen potřebné minimum pro vytvoření modulu (jenž je definováno v sekci 3.2 Moduly a ukázka kódu je v kódu 4.2).

Třída *MainWindow* vytváří grafické uživatelské rozhraní, jenž je vyobrazeno na obrázku 4.5. Tato třída je už poměrně velká a využívá i další třídy pro svoji funkci. Například třída *PreferencesDialog* umí vytvořit dialog s nastavením pro aplikaci. Třída *MainWindow* tento dialog vytváří po stisknutí tlačítka v kontextové nabídce.



Obrázek 4.5: Uživatelské grafické rozhraní modulu VNC

Realizace okna samotných VNC spojení je implementována ve třídě *ConnectionWindow*, která už využívá knihovnu *libvncclient* pro přenos obrazových dat. Tato data jsou poté vykreslena do okna.

Grafické uživatelské rozhraní je založeno na panelech, díky čemuž může být otevřeno více různých instancí VNC. Nová instance se vytvoří přes nabídku *Connection* a položku *New*. V každém panelu jsou dvě pole pro zadávání textu, jedno slouží pro zadání IP adresy a druhé pro číslo obrazovky. Po zadání těchto hodnot je možné se připojit přes tlačítko pro připojení, jenž je napravo od pole pro číslo obrazovky. Dále je možné před připojením nastavit vlastnosti přenosu a po připojení je možné obraz přepnout na celou obrazovku.

4.5 Komunikace s ROM kódem

Pro komunikaci jsem funkcionalitu rozdělil do tří modulů. Jeden slouží pro spouštění skriptů, jenž běží ve vlastním vlákně, pak pro řízení a překlad komunikace (vykresluje do okna) a pro komunikaci s USB, které také běží ve vlastním vlákně.

Pro tyto tři moduly jsem se rozhodl implementovat složitější komunikační protokol, ve kterém se kromě dat přenáší i hlavička, která obsahuje toto:

- **ID** označuje unikátní číslo zprávy,
- **timeout** čas v milisekundách do vypršení platnosti,
- **counter** určuje o kolikátý pokus se jedná,

- **acknowledge** obsahuje potvrzení nebo požadavek na potvrzení,
- **size** je velikost dat.

Tato hlavička slouží pro zabezpečení dat v tom smyslu, že pokud se nepodaří data zpracovat do času, jenž je v položce *timeout*, tak se vrátí chyba přes položku *acknowledge*. Počet těchto pokusů se ukládá do *counter*. Položka *acknowledge* může nabývat těchto hodnot:

- **NOACK** při odesílání zprávy, potvrzení není potřeba,
- **ACKREQUEST** při odesílání zprávy, požadavek na potvrzení,
- **ACKOK** při příjímání zprávy, operace proběhla úspěšně,
- **ACKFAIL** při příjímání zprávy, vypršel čas,
- **ERROR** při příjímání zprávy, nastala jiná chyba (např. USB zařízení není dostupné).

Pro zpřehlednění komunikace jsem vytvořil ještě dvě další hlavičky. Obě dvě hlavičky využívá modul pro komunikaci s ROM kódem. Jedna slouží pro komunikaci s USB modulem a druhá modulem pro skripty. Hlavička pro skripty vypadá následovně:

- **command** příkaz, jenž se má vykonat,
- **address** cílová adresa,
- **value** hodnota, která se má uložit,
- **format** v jakém formátu jsou data,
- **count** je počet dat k zapsání.

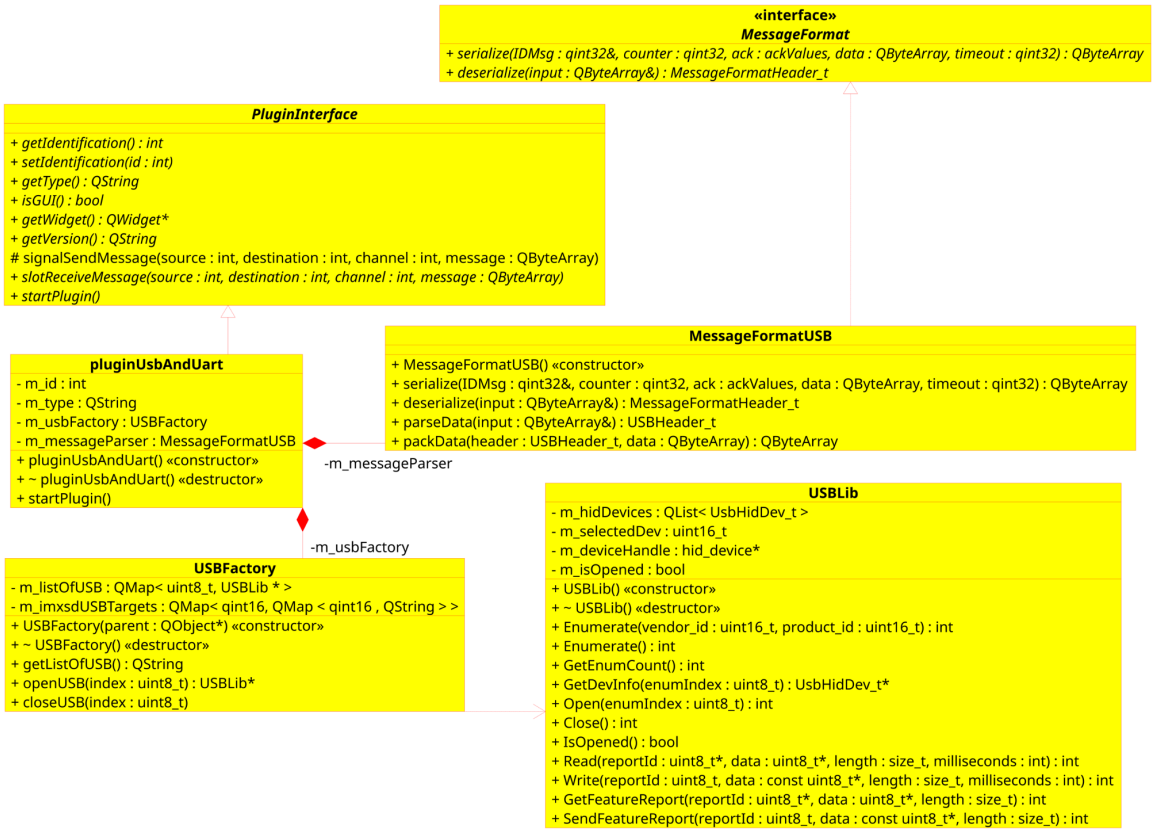
Ne všechny příkazy využívají každou položku z hlavičky, například příkaz pro zapsání DCD využívá jen *command* a *address*. Příkazy jsou definovány v tabulce 2.2.

Hlavička pro USB je jednoduší a obsahuje jen:

- **command** příkaz, jenž má USB vykonat,
- **reportID** označuje příslušný report ID,

Povolené hodnoty pro *reportID* jsou uvedeny v kapitole 2.1, v tabulce 2.1. Příkazy *command* mohou v této hlavičce nabývat hodnot *WRITE*, *READ*, *SENDFEATUREREPORT*, *GETFEATURREPORT*, jenž odpovídají příkazům USB HID.

Komunikace s ROM kódem je možné realizovat i přes UART, ale tato vlastnost není implementována. V současné době jsou na několika místech kódu jsou před-připravená rozhraní tak, aby bylo tuto funkcionalitu možné v budoucnu jednodušeji implementovat.



Obrázek 4.6: Diagram tříd modulu USB

4.5.1 USB modul

Modul pro USB slouží pro komunikaci s USB HID zařízeními. Tento modul by v budoucnu měl také podporovat komunikaci přes UART a je na to patřičně před-připraven. Diagram tříd je vyobrazen na obrázku 4.6. V třídě *pluginUSBAndUart* je implementována komunikace s ostatními moduly. Pokud modul USB přijme prázdnou zprávu na kanálu -1, tak vrátí odesílateli seznam připojených USB HID zařízení. Tento seznam je v podobě řetězce a obsahuje tyto položky:

- Přiřazené číslo,
- název výrobce,
- název produktu,
- název procesoru.

Zdrojové kódy pro tuto část jsou ve složce *src/plugins/pluginUsbAndUart* na přiloženém CD. Jednotlivé třídy mají v této složce vždy své odpovídající soubory s koncovkou *.h* a *cpp* (název třídy odpovídá názvu souboru). Společné hlavičkové soubory jsou ve složce *common*.

Název procesoru je získán za pomocí ID výrobce a ID produktu. Hodnoty a názvy pro procesory jsou v tabulce 4.1. Jak je vidět z této tabulky, tak i procesory Vybrid jsou podporovány. Data z této tabulky jsou uložena ve 2D mapě *m_imxsdUSBTargets* (třída

ID výrobce	ID produktu	Název procesoru
0x15A2	0x0054	MX6DQP
0x15A2	0x0061	MX6SDL
0x15A2	0x0063	MX6SL
0x15A2	0x0071	MX6SX
0x15A2	0x007D	MX6UL
0x15A2	0x0080	MX6ULL
0x15A2	0x0128	MX6SLL
0x15A2	0x0076	MX7SD
0x15A2	0x006A	VYBRID

Tabulka 4.1: Seznam procesorů a jejich ID produktu a výrobce.

USBFactory), jenž je indexována ID výrobcem a ID produktu. Názvy procesorů jsou v této mapě uloženy jako řetězce.

Pokud chce některý modul komunikovat s USB, tak musí zaslat zprávu, jenž má hlavičku pro zabezpečení dat a USB. Definice hlaviček a funkce pro jejich zpracování jsou definovány ve *MessageFormatUSB*, jenž dědí ze třídy *MassageFormat*.

Podle kanálu, na který je zpráva poslána, se určí, pro které USB zařízení jsou data určena (pro USB jsou určeny kanály 0 až 255). Pokud se příkaz povede, vrátí USB modul *AKCOK* a v případě příkazu *READ* a *GETFEATUREREPORT* i data, jenž byla vyčtena z USB zařízení. Pokud vyprší čas, vrátí *ACKFAIL* a v případě chyby *ERROR* (v datech zašle chybovou hlášku).

Třída *USBFactory* se stará o inicializaci a otvírání USB zařízení. Nejdůležitější funkce je *openUSB*, jenž má jako parametr číslo USB zařízení (získáno podle kanálu) a vrací odkaz na objekt třídy *USBLib*. Díky této třídě není potřeba se starat o zrovna otevřená zařízení ani mít uložen jejich seznam.

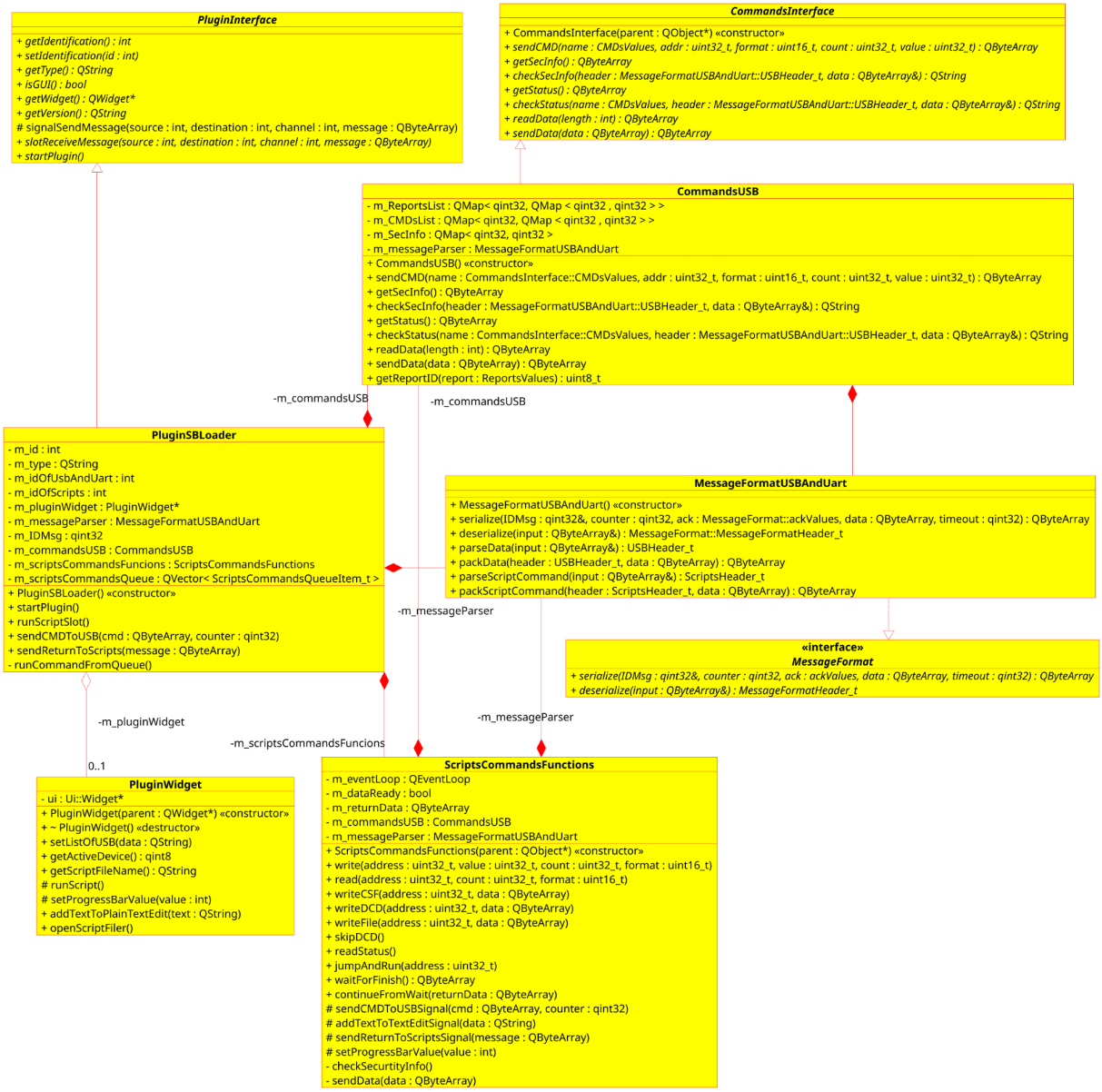
Třída *USBLib* je určena pro práci s USB HID zařízeními. Mezi její funkce patří:

- Získání počtu připojených zařízení,
- seznam připojených zařízení,
- získání informací o zařízení,
- otevření a zavření zařízení,
- čtení a zápis,
- práce s feature reports,
- a další.

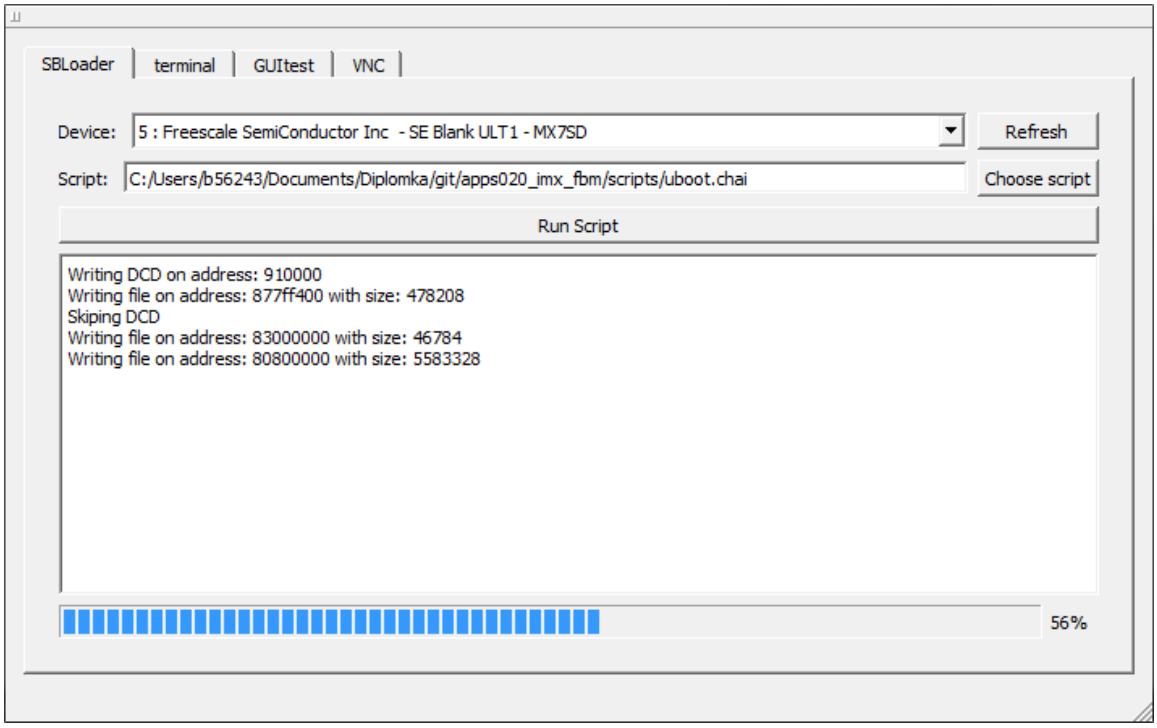
Třída obsahuje dvě rozdílné implementace, a to jednu pro operační systém Windows a druhou pro Linux, a to z toho důvodu, že oba operační systémy pracují s USB HID zařízeními rozdílně.

4.5.2 Modul pro komunikaci s ROM kódem

Tento modul slouží pro řízení komunikace s ROM kódem. Kromě této hlavní funkce slouží i jako mezivrstva mezi skripty a USB modulem.



Obrázek 4.7: Diagram tříd modulu pro komunikaci s ROM kódem



Obrázek 4.8: Ukázka uživatelského rozhraní modulu pro komunikaci s ROM kódem

Zdrojové kódy pro tuto část jsou ve složce *src/plugins/pluginSBLoader* na přiloženém CD. Jednotlivé třídy mají v této složce vždy své odpovídající soubory s koncovkou *.h* a *cpp* (název třídy odpovídá názvu souboru). Společné hlavičkové soubory jsou ve složce *common*.

Zjednodušený diagram tříd tohoto modulu je na obrázku 4.7. Hlavní třídu zde představuje *PluginSBLoader*, jenž dědí z *PluginInterface*. Je zde realizováno zabalování a vybalovaní hlaviček a dat.

Komunikace s USB je hlavně řešena ve třídě *CommandsUSB*. V této třídě jsou definovány reportID, adresy, kódy a další potřebné věci pro komunikaci s USB. Obsahuje funkce, jenž podle parametrů vygenerují pole bajtů, které se následně mohou poslat USB modulu.

Příkazy ze skriptů se zpracovávají ve třídě *ScriptsCommandsFunctions*, kde jsou ekvivalenty pro funkce ze skriptů. Tyto příkazy se dávají do fronty a postupně se zpracovávají. Jeden příkaz ze skriptů odpovídá několika příkazům pro USB. Navíc oba dva další moduly běží ve vlastních vláknech a USB ještě může vrátit chybu nebo vyprší čas (a je nutnost data znova odeslat) a je tedy potřeba všechno dělat asynchronně. Toto je ošetřeno za pomocí přejití do *QEventLoop*, ve kterém se čeká na návrat do dané funkce. Návratová hodnota se vrací přes pole bajtů, jenž je třídní proměnná.

Grafické uživatelské rozhraní je implementováno ve třídě *PluginWidget*. Uživateli se zde zobrazí USB HID zařízení, jenž jsou připojeny k počítači, vybírání ze skriptů za pomocí procházení souborů, tlačítko pro spuštění skriptu, pole, do kterého se vypisují průběžné informace a ukazatel průběhu při delších zápisech. Toto rozhraní je vyobrazeno na obrázku 4.8. Seznam USB zařízení se získá z modulu USB, za pomocí prázdné zprávy na kanál -1. USB modul na toto odpoví řetězcem, jenž obsahuje seznam zařízení. V položce seznamu je uloženo přiřazené číslo, název výrobce, název produktu a název procesoru.

4.5.3 Modul pro skripty

Modul pro skripty slouží k možnosti snadného rozšíření funkcionality pro práci s ROM kódem. V těchto skriptech se také píšou postupy pro nahrávání operačních systémů do procesoru.

Zdrojové kódy pro tuto část jsou ve složce *src/plugins/pluginScripts* na přiloženém CD. Jednotlivé třídy mají v této složce vždy své odpovídající soubory s koncovkou *.h* a *cpp* (název třídy odpovídá názvu souboru). Společné hlavičkové soubory jsou ve složce *common*. V podsložce *src/plugins/pluginScripts/ChaiScript* jsou zdrojové kódy knihovny ChaiScript.

Aby byly skripty přehledné, rozhodl jsem se využít open source knihovnu ChaiScript [2]. Díky této knihovně je možné skripty psát v beztypovém jazyku, jenž má syntaxi podobnou jazyku C, který nabízí i spoustu pokročilých funkcí jako jsou:

- Smyčky,
- podmínky,
- vlastní funkce,
- práce s řetězci,
- vlastní objekty,
- práce s formátem JSON,
- struktury,
- výjimky,
- přetěžování,
- pole.

V knihovně ChaiScript je také možno definovat objekty a funkce v C++, které lze poté používat ve skriptech. Je tedy možnost takto propojit skripty s hostitelskou aplikací. Například se může vytvořit funkce pro načítání ze souboru a data předat do skriptu přes pole. Skript poté zavolá funkci, která má jako parametr název souboru.

Ukázkový kód, jak může skript vypadat, je v kódu 4.3. V tomto skriptu se definují dvě vlastní funkce (na řádcích 1 a 5), jenž mají stejný název a jsou přetížené za pomocí podmínky parametru. Poté je zde ukázka smyčky *for* (řádek 9), ve které se volají tyto funkce. Jako poslední lze vidět ukázku použití výjimek (řádek 14).

Kód 4.3: Ukázka skriptu ChaiScirpt

```
def myFunc(x) {
        print(x);
}

def myFunc(x) : x > 2 && x < 5 {
        print(to_string(x) + " is between 2 and 5")
}

for (var i = 1; i < 10; ++i)
{
        myFunc(i);
}

try {
        var vec = [1,2]
        var val = vec[3]
} catch (e) {
        print("Oops, index out of range: " + e.what());
}
```

Zjednodušený diagram tříd (není zde vyobrazena knihovna ChaiScript) je zobrazen na obrázku 4.9. Ve třídě *PluginScripts* je použita knihovna ChaiScript. Z této knihovny je potřeba zahrnout hlavičkové soubory *chaiscript.hpp* a *chaiscript_stdlib.hpp*. Poté se může vytvořit objekt *ChaiScript*, za pomocí kterého se skripty načítají a ovládají. Skript se dá jednoduše spustit zavoláním třídní funkce *eval_file*, která má jako parametry název souboru se skriptem a specifikaci výjimek. Přidání funkce z C++ se dá provést za pomocí funkce *add* a je ukázáno v kódu 4.4. První parametr je odkaz na tuto funkci a objekt, jenž ji obsahuje, a druhý je název funkce, přes nějž se dá volat ze skriptu.

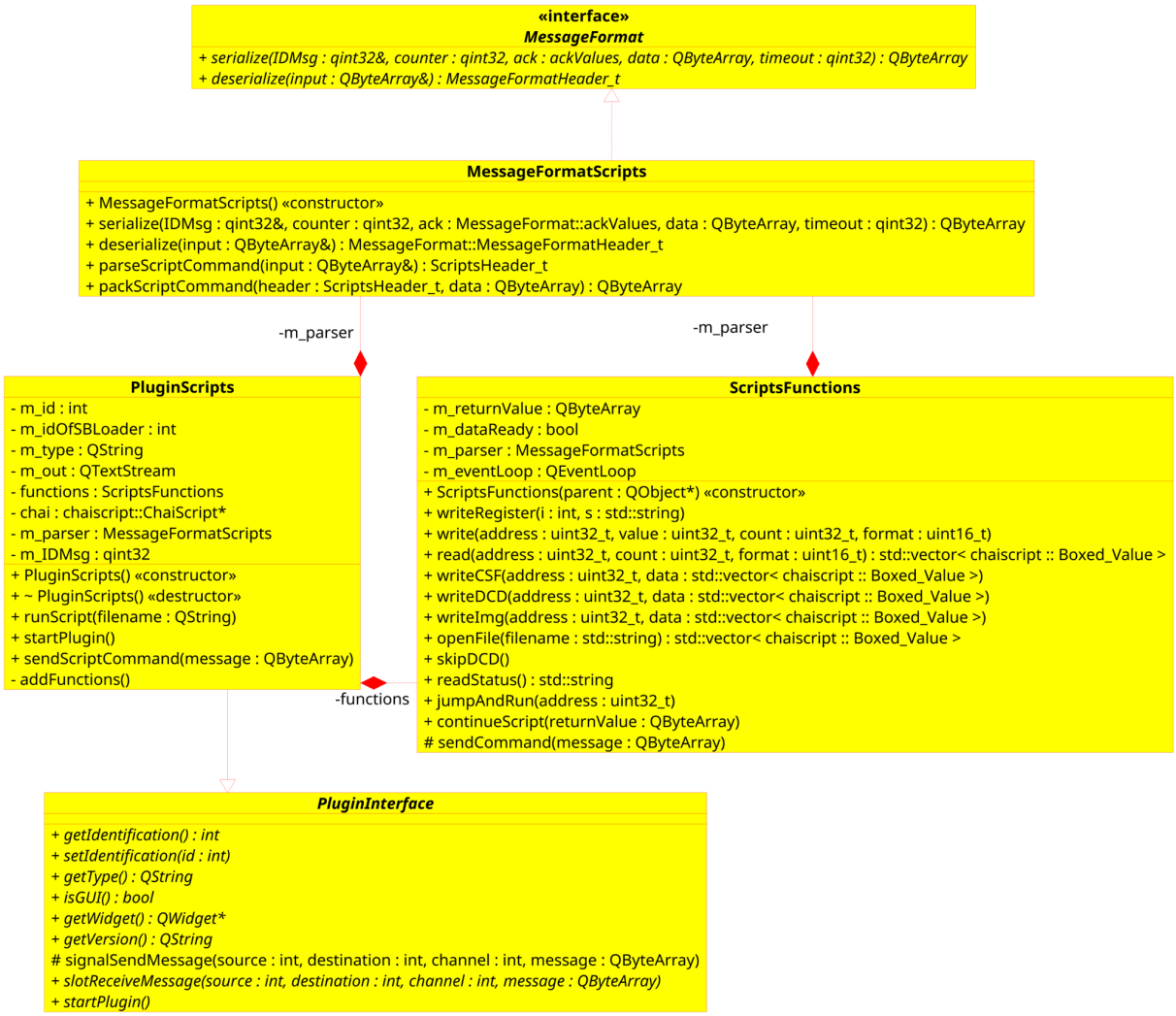
Kód 4.4: Ukázka přidání funkce do Chaiscript

```
add(chaiscript::fun(&ScriptsFunctions::read, &functions),
    "read");
```

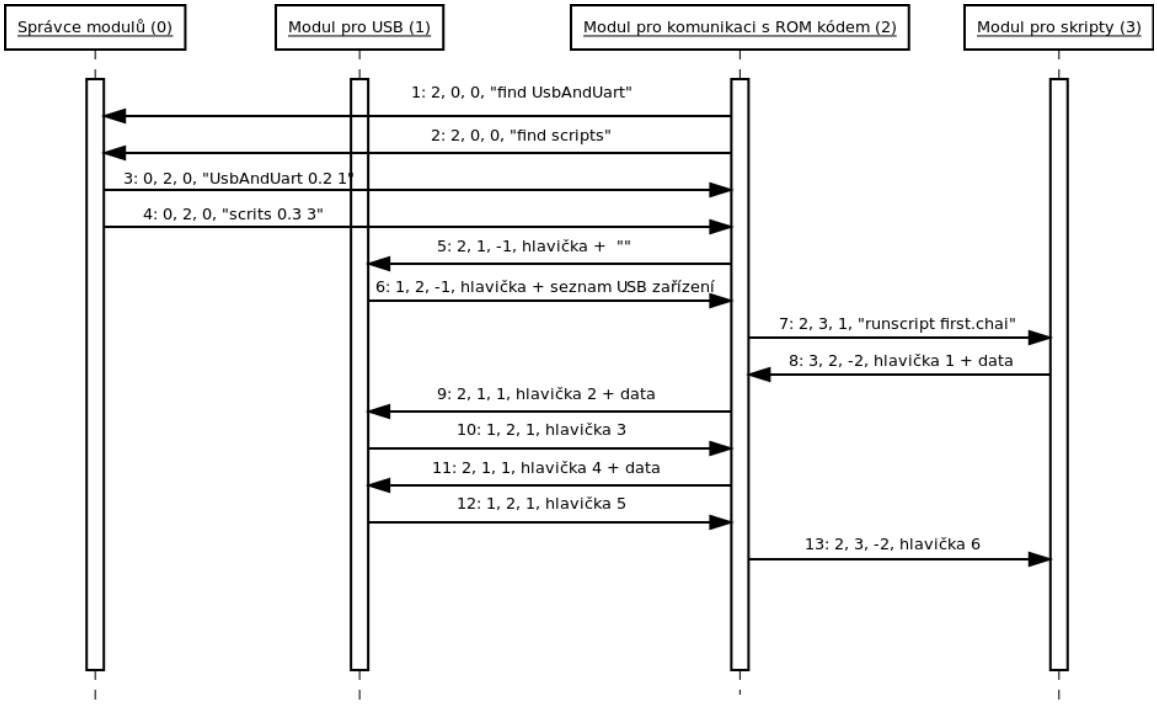
Funkce, které můžou skripty využívat, jsou definovány ve třídě *ScriptsFunctions*. Jejich implementace je taková, že se vytvoří zpráva, která se zasílá modulu pro komunikaci s ROM kódem (všechny funkce je možné vidět v diagramu tříd na obrázku 4.9). Kromě těchto funkcí je zde i funkce pro načítání souboru. Skript je nutno pozastavit, dokud modul pro komunikaci s ROM kódem neprovede požadovanou funkci. Proto se v těchto funkcích přechází do *QEventLoop*, kde se čeká na návratovou hodnotu.

Třída *PluginScripts* se stará o komunikaci s ostatními moduly. Pokud dostane na kanál 1 zprávu **runscript jmenoSouboru.chai**, tak spustí skript ze souboru *jmenoSouboru.chai*. S modulem pro práci s ROM kódem komunikuje přes kanál -2, kde se kromě dat posílají i hlavičky. Používá se zde hlavička pro zabezpečení dat a speciální hlavička, jenž je definovaná v odstavci 4.5.

Jedinou nevýhodou knihovny ChaiScript je její velikost. Při kompilování se vytváří velký object file. Pod Linuxem ani ve Visual C++ Build Tools (je možno aktivovat podporu pro velké object file za pomoci přepínače */bigobj*) s tím není problém. Bohužel MinGW32, jenž je výchozí pro framework Qt pod operačním systémem Windows, tuto možnost nepodporuje a není tedy možné s ním zkompilovat tento modul (MinGW64 toto umožňuje, ale není oficiálně podporovaný).



Obrázek 4.9: Diagram tříd modulu pro skripty



Obrázek 4.10: Diagram sekvence pro příklad komunikace

4.5.4 Příklad komunikace

V této části diplomové práce budu vysvětlovat komunikaci mezi moduly pro USB, skripty a komunikaci s ROM kódem. Předpokládejme, že po startu aplikace přidělí správce modulů ID takto:

- Modul USB = 1,
- modul pro komunikaci s ROM kódem = 2,
- modul pro skripty = 3,
- správce modulů = 0.

Diagram sekvence této komunikace je na obrázku 4.10. Formát zpráv je pořadí: id zdroje, id cíle, kanál, data. Očíslované hlavičky budou vysvětleny v tabulkách. Každá hlavička navíc ještě obsahuje velikost dat a čas do vypršení platnosti, jenž je vždy jedna sekunda.

Komunikaci zahajuje modul pro komunikaci s ROM kódem, kdy od správce modulů zjišťuje ID modulu pro skripty a USB (zprávy 1 a 2). Správce modulů mu odpoví a kromě ID zašle i verzi daného modulu (zprávy 3 a 4). Následně od modulu pro USB získá seznam připojených USB zařízení přes kanál -1 (zprávy 5 a 6).

Poté, co uživatel v grafickém uživatelském rozhraní vybere skript, jenž se má spustit a stiskne tlačítko pro spuštění skriptu (skript slouží pouze pro zápis DCD a obsahuje tudíž jen funkci pro načtení ze souboru a volání funkce *writeDCD*), modul pro komunikaci s ROM kódem vyšle zprávu modulu pro skripty na kanál, která má v datech příkaz *runscript* a název souboru, jenž obsahuje daný skript (zpráva 7). Modul pro skripty spustí daný skript a jakmile dojde na funkci *writeDCD*, tak pošle modulu pro komunikaci zprávu na kanál -2 s hlavičkou 1, jenž je definována v tabulce 4.2, a data načtená ze souboru (zpráva 8).

ID zprávy	čítač	potvrzení	příkaz	adresa
0	0	ACKREQUEST	SCRIPTSWRITEDCD	0x00910000

Tabulka 4.2: Hlavička 1

Po přijetí zprávy od modulu pro skripty začne modul pro komunikaci s ROM kódem daný příkaz zpracovávat. Rozdělí *SCRIPTSWRITEDCD* na čtyři podčásti, které bude postupně zasílat modulu pro USB. V tomto příkladě jsou znázorněny jen dvě podčásti, protože zanedbávám kontrolu, jestli je zařízení zamčené a status příkazu. Modul pro komunikaci s ROM kódem tedy zašle modulu USB nejdříve zprávu 9, jenž obsahuje hlavičku 2 (viz tabulka 4.3) a data, jenž jsou naznačena v tabulce

ID zprávy	čítač	potvrzení	příkaz pro USB	report ID
0	0	ACKREQUEST	WRITE	0x01

Tabulka 4.3: Hlavička 2

příkaz	adresa	formát	počet	hodnota
WDCD	0x00910000	0	velikost souboru	0

Tabulka 4.4: Data 2

Pokud při zápisu příkazu nenastane chyba a nebo dojde k vypršení času, odpoví USB modul zprávou 10, ve které je hlavička 3, jejíž obsah je v tabulce 4.5.

ID zprávy	čítač	potvrzení	příkaz pro USB	report ID
0	0	ACKOK	WRITE	0x01

Tabulka 4.5: Hlavička 3

Nyní může modul pro komunikaci s ROM kódem zaslat samotná data, jenž získal od modulu pro skripty. Protože je možno zapsat maximálně 1024 bajtů dat najednou, musí data rozdělit a poslat je v samostatných zprávách, proto uvažujme, že se vejdou do jedné zprávy. Jedna zpráva pro USB modul (zpráva 11) obsahuje hlavičku 4 (viz tabulka 4.6) a maximálně 1024 bajtů dat.

ID zprávy	čítač	potvrzení	příkaz pro USB	report ID
1	0	ACKREQUEST	WRITE	0x02

Tabulka 4.6: Hlavička 4

Na tuto zprávu odpoví USB modul při úspěchu zprávou, která obsahuje jen hlavičku, jenž je naznačena v tabulce 4.7

ID zprávy	čítač	potvrzení	příkaz pro USB	report ID
1	0	ACKOK	WRITE	0x02

Tabulka 4.7: Hlavička 5

Po úspěšném zapsání všech dat, modul pro komunikaci s ROM kódem může vrátit modulu pro skripty zprávu s potvrzením, že byl příkaz proveden. Protože se jedná o funkci

bez návratové hodnoty, tak ve zprávě bude pouze hlavička, jenž bude obsahovat pouze ID zprávy (0) a potvrzení *ACKOK* (zpráva 12).

Kapitola 5

Testování a zhodnocení

V této kapitole budu provádět srovnání vlastností vytvořeného nástroje a nástrojů které už existují a jsou popsány v kapitole 2.1 (kromě nástroje i.MX Pin Mux tool, jenž má jiné zaměření). Dále zde bude ukázáno porovnání skriptů pro vytvořený nástroj a MFG tool a také grafického uživatelského rozhraní těchto dvou aplikací.

Testování aplikace proběhlo na referenčním kitu i.MX 7Dual SABRE Board od firmy NXP a na kitu The phyCORE-i.MX6 Single Board Computer od firmy PHYTEC, jenž obsahuje procesor i.MX6Q.

Pro testování funkčnosti komunikace s ROM kódem jsem vytvořil skript, jenž nahraje DCD, U-Boot, DTB a linuxové jádro do paměti RAM a poté spustí U-Boot. Tento skript je v kódu 5.1 (adresy jsou pro procesor i.MX7D).

Kód 5.1: Testovací skript

```
1  var DataDCD = openFile("../scripts/imx7d_dcd.bin");
2  writeDCD(0x00910000, DataDCD);
3
4  var DataUboot = openFile("../scripts/u-boot.imx");
5  writeImg(0x877FF400 , DataUboot);
6  skipDCD();
7
8  var DataDTB = openFile("../scripts/imx7d_sdb.dtb");
9  writeImg(0x83000000 , DataDTB);
10
11 var DataZImage = openFile("../scripts/imx7d_zImage");
12 writeImg(0x80800000 , DataZImage);
13
14 jumpAndRun(0x877FF400);
```

5.1 Porovnání vlastností

Přehled vlastností srovnávaných nástrojů je vyobrazen v tabulce 5.1 a nyní budu některé vlastnosti postupně rozepisovat více podrobně.

Nástroje MFG tool, SB loader a mnou vytvořený umožňují programování přes komunikaci s ROM kódem. SB loader nabízí jen pár základních příkazů, které se spouští jako parametry programu přes příkazovou řádku. Jedná se například o inicializaci RAM, zápis obrazu a spuštění. U zbylých dvou se programování řeší za pomoci skriptů. Všechny tři nástroje umožňují i načítání obrazu do RAM paměti, pro rychlé testování. Jediný SB lo-

Vlastnost	vytvořený nástroj	MFG tool	SB loader	i.MX and Vybrid Boot Utility
GUI	Ano	Ano	Ne	Ano
Programování	Ano	Ano	Ano	Ne
USB HID	Ano	Ano	Ano	Ano
UART	Ne	Ne	Ano	Ne
Práce s registry	Ano	Ne	Ne	Ne
Skripty	Ano	Ano	Ne	Ne
Boot log	Ano (bez zpracování)	Ne	Ne	Ano
Modulární	Ano	Ne	Ne	Ne
Více vláknový	Ano	Ano	Ne	Ne
Multiplatformní	Ano	Ne	Ne	Ano
Terminál	Ano	Ne	Ne	Ne
VNC	Ano	Ne	Ne	Ne

Tabulka 5.1: Hlavička 5

ader nabízí také programování přes UART. Vytvořený nástroj umí navíc zapisovat a číst hodnoty jednotlivých registrů, ale je nutné znát adresu daného registru.

Hlavní funkcí i.MX and Vybrid Boot Utility je vypisování boot logu (záznam o startu procesoru). Stejná data jsem schopen s vytvořeným nástrojem také vyčíst, ale tato data jsou v binárním tvaru a je ještě nutno k nim dodělat parser.

Velkou výhodou vytvořeného nástroje je, že se dá snadno rozšířit o další funkcionalitu za pomocí modulů. Také je možno jednoduše přidávat a odstraňovat moduly, takže se dá upravit na míru pro danou situaci a schopnosti uživatele (například do výroby by byly jen moduly pro skripty, komunikaci s ROM kódem a USB, jenž by umožnily operátorovi jen stiskem tlačítka desku naprogramovat). Moduly se po startu aplikace automaticky načítají ze složky *scripts* a je tedy možné z této složky moduly, jenž jsou navíc odstranit a nebo je tam přidat.

5.2 Porovnání grafického uživatelského rozhraní a skriptů

Pokud se podíváme na grafické uživatelské rozhraní aplikace i.MX and Vybrid Boot Utility na obrázku 2.8, tak lze vidět, že nabízí jen funkcionalitu vyčítání záznamu o startu. Je zde jen pár tlačítek, která slouží pro připojení a vyčtení dat. Zpracovaná data se vypíšou do textového pole.

Grafické uživatelské rozhraní aplikace MFG tool, jenž je vyobrazeno na obrázku 2.6, nabízí pro uživatele jen minimální možnost konfigurace zápisu (vše se řeší ve skriptech) a téměř žádné informace o průběhu nahrávání (pouze progress bar). Uživatel dokonce nemůže vybrat ani zařízení (automaticky se vybere první nalezené), na rozdíl od mého nástroje, který kromě toho nabízí také možnost vybrání skriptu a vypisuje i aktuálně prováděnou funkci, jak je vidět na obrázku 4.8.

Skripty, jenž se v nástroji MFG tool starají o nahrávání obrazu, jsou psané ve značkovacím jazyce XML. Skripty to jsou celkem dlouhé a proto jsem se rozhodl zde ukázat pouze nahrání DTB (Device Tree Blob) do procesoru. Skript je naznačen v kódu 5.2.

Kód 5.2: VBS skript

```
1  <CMD state="BootStrap" file="firmware/zImage-imx7d-%7ddtb%.dtb"
2  type="load" address="0x83000000" loadSection="OTH" setSection="OTH"
3  HasFlashHeader="FALSE" ifdev="MX7D">Loading device tree.</CMD>
4
5  <!-- burn dtb -->
6  <CMD state="Updater" type="push" body="send"
7   file="files/zImage-imx7d-%7ddtb%.dtb"
8   ifdev="MX7D">Sending Device Tree file</CMD>
9
10 <CMD state="Updater" type="push"
11 body="$ cp $FILE /mnt/mmcblk%mmc%p1/imx7d-%7ddtb%.dtb"
12 ifdev="MX7D">write device tree to sd card</CMD>
```

Naproti tomu skript v mém nástroji, jenž provádí stejný úkon a to nahrání DTB, by vypadal jako v kódu 5.3. S tím, že adresa 0x83000000, jenž je v kódu na řádku dva, by mohla být pro zpřehlednění uložena v jiném souboru ve formátu JSON. Přesto, že mé skripty vypadají jednodušeji, dokáží udělat více věcí než skripty v MFG tool. Na Příklad dokážou číst a zapisovat do registrů nebo určité části paměti.

Kód 5.3: VBS skript

```
1  var DataDTB = openFile("../scripts/imx7d_sdb.dtb");
2  writeImg(0x83000000 , DataDTB);
```

Kapitola 6

Závěr

Ve své práci jsem se zaobíral tvorbou nástroje pro práci s procesory i.MX v programovacím jazyce C++ za pomocí frameworku Qt. Pro potřeby návrhu aplikace jsem prostudoval architekturu těchto procesorů, startování systému, USB HID a komunikaci s ROM kódem. Zjištěné poznatky jsou v kapitole 2.1. Dále jsem prostudoval framework Qt5 a jeho možnosti v tvorbě modulů, tyto poznatky jsou v kapitole 2.3. Navrhl jsem nástroj, jenž je modulární, využívá více vláken, a také moduly pro tento nástroj, více v kapitole 3. Samotná implementace nástroje je popsána v kapitole 4. Pro účely porovnání jsem zvolil již několik existujících nástrojů, které jsou prezentovány v kapitole 2.2.

Výsledný nástroj, kromě samotného správce modulů, obsahuje 7 modulů, z čehož dva jsou ukázkové, tři realizují nahrávání obrazu přes komunikaci s ROM kódem, jeden je pro terminál a poslední pro VNC. Pro nahrávání obrazů jsou využity skripty, které nabízejí spoustu pokročilých funkcí (více v sekci 4.5.3 a umožňují tak pohodlnou komunikaci s ROM kódem.

Při přímém srovnání s nástrojem MFG tool, mnou vytvořená aplikace vychází ve všech bodech lépe, ať už se jedná o funkcionalitu a nebo složitost skriptů. Pokud ji srovnáme s nástrojem i.MX and Vybrid Boot Utility, tak tento nástroj je schopen provádět jen jednu věc, a to vypisování záznamu startu. Výsledná aplikace je tato data schopna vyčíst, ale v rámci diplomové práce tato funkce není zpracovaná. Nástroj SB loader nabízí jen konzolové rozhraní a pracuje se s ním za pomocí přepínačů a nedokáže využít všechny možnosti, jenž ROM kód nabízí, na rozdíl od mé aplikace.

Hlavní výhodou vytvořeného nástroje je možnost dalšího rozšiřování funkcionality za pomocí modulů, možnost uzpůsobení dané situaci a multiplatformnost. Při vývoji byl největší problém s knihovnami, jenž měly být multiplatformní, ale nebylo je možné zkompilovat pod Visual C++ Build Tools. Proto jsem je musel upravit tak, aby to bylo možné, vzhledem ke knihovně ChaiScript viz sekce 4.5.3. Další překážkou bylo samotné testování, kdy po nahrání DCD a U-Boot se buď procesor nastartoval a nebo ne. Při implementaci bylo nejnáročnější, kromě nastudování fungování ROM kódu, vytvořit programování přes ROM kód a to z toho důvodu, že bylo potřeba propojit tři moduly tak, aby si správně předávaly informace a vzájemně na sebe čekaly.

Literatura

[1] Brown, E.: Freescale pumps out three new Linux-friendly i.MX6 SoCs [online]. http://linuxgizmos.com/freescale-pumps-out-three-new-i-mx6-socs, 2015 [cit. 2017-12-28].

[2] ChaiScript: ChaiScript [online]. http://chaiscript.com/index.html, 2017 [cit. 2017-12-28].

[3] NXP: i.MX7D: i.MX 7Dual Processors - Heterogeneous Processing with dual ARM® Cortex®-A7 cores and Cortex-M4 core [online]. http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/i.mx-appl 2017 [cit. 2017-12-28].

[4] NXP: i.MX 7Dual Applications Processor Reference Manual [online]. www.nxp.com/assets/documents/data/en/reference-manuals/IMX7DRM.pdf, 2017 [cit. 2017-4-20].

[5] The Qt Company: Qt Documentation: Signals & Slots [online]. http://doc.qt.io/qt-5/signalsandslots.html, 2017 [cit. 2017-12-28].

[6] The Qt Company: Qt Documentation: Threads and QObjects [online]. http://doc.qt.io/qt-5/threads-qobject.html, 2017 [cit. 2017-12-28].

[7] USB Implementers Forum: Universal Serial Bus (USB): Device Class Definition for Human Interface Devices (HID) [online]. http://www.usb.org/developers/hidpage/HID1_11.pdf, 2001 [cit. 2017-12-28].

Přílohy

Příloha A

Obsah přiloženého paměťového média

Na přiloženém CD lze nalézt zdrojové kódy programu, které lze zkompilovat v Qt creator, při otevření projektu *workspace.pro* ve složce src, nebo za použití příkazu qmake nad tímto souborem se vytvoří Makefile.

Složky na tomto CD mají tento obsah:

- **bin** zkompilované zdrojové kódy pro Windows,
- **src** zdrojové kódy,
- **common** společné hlavičkové soubory pro moduly,
- **doc** text diplomové práce,
- **scripts** skripty a obrazy pro nahrávání.