F3

Fakulta elektrotechnická
Katedra měření

Bakalářská práce

Implementace ONVIF serveru pro zabezpečené streamování videa přes Ethernet rozhraní

Michal Peterka
Kybernetika a robotika

Květen 2023
Vedoucí práce: Ing. Radek Sedláček, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Peterka** Jméno: **Michal** Osobní číslo: **492327**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Implementace ONVIF serveru pro zabezpečené streamování videa přes Ethernet rozhraní

Název bakalářské práce anglicky:

Implementation of ONVIF server for secure video streaming via Ethernet interface

Pokyny pro vypracování:

1.Seznamte se s profilem S bezpečnostního standardu ONVIF pro IP video zařízení a prostudujte možnosti jeho využití zejména v oblastní termovizních systémů.
2.Navrhněte a implementujte „test pattern generator“ v programovacím jazyce C++, simulující dynamický obraz s rozlišením až 4K a snímkovou frekvencí 30 Hz.
3.Navrhněte a implementujte aplikaci ONVIF serveru v programovacím jazyce C++ přenositelnou na architekturu procesoru ARM CORTEX A-53.
4.Otestujte implementovaný ONVIF server na platformě Xilinx Zynq Ultrascale+ využívající čtyřjádrový procesor ARM CORTEX A-53 a generovaný obraz zobrazte pomocí univerzálního VMS systému.

Seznam doporučené literatury:

[1] https://www.onvif.org/profiles/profile-s/
[2] Využití standardu ONVIF v kamerových systémech. Ostrava, 2012. Bakalářská práce. VŠB-TUO, FEI, Katedra informatiky. Vedoucí práce Doc. Dr. Ing. Eduard Sojka.
[3] SOAP Tutorial [online]. [cit. 2022-06-25]. Dostupné z: https://www.w3schools.com/xml/xml_soap.asp
[4] Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit [online]. [cit. 2022-06-28]. Dostupné z: https://www.xilinx.com/products/boards-and-kits/zcu104.html

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Radek Sedláček, Ph.D. katedra měření FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **01.07.2022** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **09.02.2024**

Ing. Radek Sedláček, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

Poděkování / Prohlášení

Rád bych poděkoval Ing. Radkovi Sedláčkovi, Ph. D. za vedení této práce.

Také bych rád vyjádřil velké díky společnosti Workswell s.r.o. za poskytnutí hardwaru potřebného pro realizaci mé práce.

Nakonec bych chtěl poděkovat své rodině za jejich podporu a motivaci v průběhu celého studia. Bez jejich podpory bych nedokázal splnit své akademické cíle.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 26. května 2023

.....

Abstrakt / Abstract

Tato bakalářská práce se zaměřuje na analýzu a implementaci standardu ONVIF, který je dnes již běžným standardem bezpečnostních systémů. Cílem práce bylo implementovat tento standard v jazyce C++ pro platformu Zynq Ultrascale+ MPSoC od společnosti Xilinx. K implementaci byla použita knihovna gSOAP, která poskytuje implementaci protokolu SOAP. Díky tomu bylo možné zaměřit se na implementaci konkrétních webových služeb standardu ONVIF. K přenosu implementace na cílovou platformu byly použity nástroje PetaLinux od společnosti Xilinx. Výsledkem práce je knihovna umožňující jednoduchou integraci standardu ONVIF na platformě Zynq Ultrascale+ MPSoC, ale i na jiných platformách. Navíc vznikla také aplikace, která byla použita pro testování knihovny.

Klíčová slova: ONVIF, Zynq, bezpečnostní systémy, webové služby, gSOAP, PetaLinux, zabezpečení, streamování videa

This bachelor thesis focuses on the analysis and implementation of the ONVIF standard, which is now a common standard for security systems. The aim of the thesis was to implement this standard in C++ for the Zynq Ultrascale+ MPSoC platform from Xilinx. The gSOAP library, which provides a SOAP implementation, was used for the implementation. This made it possible to focus on the implementation of specific web services of the ONVIF standard. Xilinx's PetaLinux tools were used to port the implementation to the target platform. The result of this work is a library that allows easy integration of the ONVIF standard on the Zynq Ultrascale+ MPSoC platform, but also on other platforms. In addition, an application was also created and used to test the library.

Keywords: ONVIF, Zynq, security systems, web services, gSOAP, PetaLinux, security, video streaming

Title translation: Implementation of ONVIF server for secure video streaming via Ethernet interface

/ Obsah

Tabulky / Obrázky

Kapitola 1
Úvod

Během posledních let se stále častěji setkáváme s různými bezpečnostními systémy. Kamery často sledují naši bezpečnost na ulicích a pracovištích, nebo dohlíží na náš majetek. Bezpečnostní systémy ale vyrábí velké množství výrobců, čímž vznikl problém s kompatibilitou produktů. Kamery a další prvky bezpečnostního systému od jednoho výrobce zpravidla nebylo možné použít v kombinaci s produkty jiného výrobce. Z tohoto důvodu byl v roce 2008 založen ONVIF (Open Network Video Interface Forum).

Cílem této práce je seznámit se se standardem ONVIF a cílovou platformou Zynq Ultrascale+ MPSoC (Multiprocessor System on a Chip) od firmy Xilinx a následně navrhnout metodu pro implementaci rozhraní podle standardu ONVIF. Konkrétně se budu v práci zaměřovat na ONVIF profil S, který specifikuje rozhraní pro streamování videa. Již předem nastíním, že rozhraní definované standardem ONVIF je velmi rozsáhlé, a proto se zaměřím pouze na jeho vybranou část v rámci profilu S. Dílčím cílem je také návrh a implementace testovacího generátoru obrazu, který má být následně streamován.

Zadání práce vzniklo v rámci spolupráce se společností Workswell s.r.o [1], která se zaměřuje především na vývoj termovizních kamer. Z tohoto důvodu byla jako cílová platforma zvolena právě Zynq Ultrascale+ MPSoC, která je svými vlastnostmi vhodná k použití v oblasti termovize. Pro účely implementace rozhraní podle standardu ONVIF byl použit hardware poskytnutý společností Workswell s.r.o. Konkrétně se jednalo o testovací desku ZCU104 od společnosti Xilinx.

[1] `https://workswell.cz/`

Kapitola 2
ONVIF

Zkratka ONVIF zastupuje Open Network Interface Forum. ONVIF je tedy fórum, které má jako cíl poskytovat otevřený standard rozhraní pro prvky bezpečnostních systémů na bázi protokolu IP (Internet Protocol). Základní myšlenky fóra jsou standardizace komunikace mezi prvky bezpečnostních systémů, kompatibilita bez ohledu na výrobce a otevřenost všem společnostem a organizacím.

Fórum bylo založeno jako nezisková organizace 25. listopadu 2008 společnostmi Axis Communications, Bosch Security Systems a Sony Corporation, a již následující měsíc byla vydána první verze specifikací základu standardu. V roce 2009 už mělo fórum 100 členů a přes 200 produktů v souladu se standardem. Během roku 2010 vznikla druhá verze specifikací, která je od té doby rozšiřována pomocí takzvaných profilů a modulů. Během následujících let se fórum stále rozrůstalo, a tak v roce 2018 byl překročen milník 10 000 produktů odpovídajících standardu. V současnosti má ONVIF téměř 500 členů a přes 20 000 odpovídajících produktů.

Členství ve fóru je rozděleno do pěti úrovní. Pro lepší pochopení zůstanu u anglických výrazů Full member, Contributing member, Registered affilate, User a Observer. Pro každou úroveň jsou přesně vymezená práva. První tři ze zmíněných úrovní jsou určeny pro členy, kteří se chtějí aktivně podílet na vývoji standardu. Členové na úrovní User mohou po splnění testu přidávat své produkty na seznam zařízení podporujících standard ONVIF. Úroveň Observer má práv nejméně, slouží především pro integraci ONVIF zařízení do bezpečnostního softwaru.

2.1 Webové služby

Standard ONVIF je založen na principu webových služeb (anglicky Web Service). Tyto služby poskytuje zařízení, kterým v praxi často bývá například bezpečnostní kamera, a využívá je klient, což může být například bezpečnostní software. Součástí definice služby je popis v jazyce WSDL (Web Service Description Language) [2] založeném na jazyce XML (Extensible Markup Language). Ten pro každou službu obsahuje definici požadavku k jejímu vykonání, který odesílá klient, a definici odpovědi, kterou zařízení sděluje klientovy výsledku provedení služby.

Zjednodušený příklad definice služby v jazyce WSDL je v ukázce 2.1. Požadavek na službu *GetDeviceInformation* zde nemá definované žádné parametry, které mají být předány. Naopak pro odpověď *GetDeviceInformationResponse* jsou definovány parametry, prostřednictvím kterých se předávají informace o výrobci a modelu zařízení, verzi firmwaru, sériovém čísle a hardwarovém ID (Identifier). Popis v jazyce WSDL lze pomocí kompilátorů převést na kód, který může být použit k integraci standardu.

Zařízení a klient spolu komunikují pomocí protokolu SOAP (Simple Object Access Protocol) [3]. Tento protokol je založený na předávání zpráv ve formátu XML. Ve zprávách jsou zabaleny informace o požadavku na vykonání webové služby nebo odpovědi na něj. Zprávy mohou být předávány prostřednictvím různých internetových protokolů,

přičemž nejpoužívanější z nich je HTTP (Hypertext Transfer Protocol). Zjednodušená zpráva protokolu SOAP je v ukázce 2.2. Jedná se o odpověď zařízení z ukázky 2.1.

```
<xs:element name="GetDeviceInformation">
    <xs:complexType>
       <xs:sequence/>
    </xs:complexType>
</xs:element>
<xs:element name="GetDeviceInformationResponse">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Manufacturer" type="xs:string"/>
            <xs:element name="Model" type="xs:string"/>
            <xs:element name="FirmwareVersion" type="xs:string"/>
            <xs:element name="SerialNumber" type="xs:string"/>
            <xs:element name="HardwareId" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Ukázka kódu 2.1. Příklad popisu služby v jazyce WSDL

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=...>
    <SOAP-ENV:Body>
        <tds:GetDeviceInformationResponse>
            <tds:Manufacturer>manufacturer</tds:Manufacturer>
            <tds:Model>model</tds:Model>
            <tds:FirmwareVersion>1.0.0</tds:FirmwareVersion>
            <tds:SerialNumber>123456789</tds:SerialNumber>
            <tds:HardwareId>123456789</tds:HardwareId>
        </tds:GetDeviceInformationResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Ukázka kódu 2.2. Příklad zprávy protokolu SOAP

2.2 Profily

Standard ONVIF se snaží pokrýt co nejširší spektrum prvků bezpečnostního systému, a proto je specifikace rozdělena do profilů. Zařízení či klient mohou splňovat specifikaci jednoho nebo i více profilů. Každý profil se zaměřuje na jinou problematiku v rámci bezpečnostních systémů a definuje jinou sadu webových služeb. Služby definované v profilu mohou být povinné, volitelné a nebo podmíněné. Například pokud zařízení odpovídající profilu S nemá možnost vysílání zvukového záznamu, nemusí poskytovat služby týkající se nastavení jeho přenosu. Jelikož nemusí všechna zařízení poskytovat všechny služby profilu, existují zde i služby, které pomáhají klientovi zjistit, jaké služby jsou dostupné.

2.2.1 Profil A

Tento profil je určen pro zařízení zajišťující síťovou kontrolu přístupu. Zařízení splňující specifikace profilu A umožňují získávat informace o stavu a událostech zařízení a konfiguraci přístupových práv, pověření a rozvrhy. Profil A byl zveřejněn v roce 2017, ale nikdy nebyl moc používaný. V současné době se na oficiálním seznamu produktů splňujících specifikace profilu a nachází pouze dvě zařízení od společnosti Axis Communications AB.

2.2.2 Profil C

Profil C je určen pro zařízení používaná v elektronických systémech přístupové kontroly. Zařízení a klienti, kteří splňují normu profilu C, umožňují zjistit informace o místech, řízení přístupu ke dveřím a správu událostí a alarmů. Profil C existuje od roku 2013 a na rozdíl od profilu A, zde se již na seznamu vyhovujících produktů nachází 82 položek od různých výrobců.

2.2.3 Profil D

Profil D definuje rozhraní pro periferní vstupní zařízení, jako jsou čtečky pro karty, klíče, mobilní telefony nebo čárové kódy, biometrické čtečky pro rozpoznání otisků prstů, kamery pro rozpoznávání duhovek, obličejů nebo SPZ, klávesnice a senzory pro sledování stavu zámku, dveří, teploty nebo pohybu. Dále specifikuje rozhraní pro výstupní zařízení, jako jsou zámky, displeje a LED diody. Vstupní zařízení předává přihlašovací údaje klientovi profilu D, kterým je typicky jednotka pro řízení přístupu. Klient pak podle přístupových pravidel odešle zpět k periferním zařízení příkaz k udělení nebo neudělení přístupu.

Jelikož byl profil D vydán v roce 2021, jedná se zatím o nejmladší z profilů a v současné době nejsou žádné produky, které by profil podporovaly. Profil D doplňuje profily A a C v možnostech standardizované komunikace v elektronickém systému kontroly přístupu.

2.2.4 Profil G

Profil G je navržen pro systémy na záznam videa. Zařízení Profilu G (například IP kamera) jsou taková, která mohou zaznamenávat video data přes síť nebo na samotném zařízení. Klient Profilu G je takový, který může konfigurovat a ovládat nahrávání videa přes síť ze zařízení Profilu G. Specifikace profilu G vznikla v roce 2014. Seznam produktů splňujících specifikace tohoto profilu obsahuje více než 16 tisíc různých položek.

2.2.5 Profil M

Profil M umožňuje konfiguraci a získávání informací o metadatech, filtraci a streamování metadat. Poskytuje rozhraní pro obecnou klasifikaci objektů a specifikovaná metadata pro geolokaci, vozidla, registrační značky, lidskou tvář a tělo. Pokud mají konformní produkty nativní podporu pro funkce, jako je správa profilů médií, streamování videa, přidávání obrázků do metadatových streamů, zpracování událostí nebo konfigurace pravidel, musí být podporována také rozhraní pro tyto funkce v rámci profilu M. A pokud konformní produkty podporují analytiku pro počítání objektů (např. lidí, vozidel), rozpoznávání registračních značek nebo rozpoznávání obličeje nebo protokol MQTT (Message Queuing Telemetry Transpor) používaný pro systémy IoT (Internet of Things), pak by měla existovat také rozhraní pro zpracování událostí v rámci profilu M pro tyto funkce.

Stejně jako profil D i profil M vznikl v roce 2021 a řadí se tak mezi nejmladší profily. Na rozdíl od profilu D zde však již v seznamu vyhovujících produktů nalezneme téměř 700 položek.

2.2.6 Profil S

Profil S je navržen pro video systémy. Kompatibilní zařízení mohou odesílat video data přes síť do klienta. Klient profilu S může konfigurovat, ovládat a využívat video streamování. Profil S zahrnuje také ONVIF specifikace pro PTZ ovládání, audio vstup, multicasting a reléové výstupy pro zařízení a klienty, které podporují tyto funkce. Vzhledem k tomu, že se tato práce zabývá právě streamováním videa, bude implementace vycházet ze specifikace právě tohoto profilu.

Jedná se o nejstarší z profilů. Od původního vydání specifikace v roce 2011 však již došlo k několika úpravám. V současné době je existuje více než 26 tisíc produktů splňujících specifikace tohoto profilu.

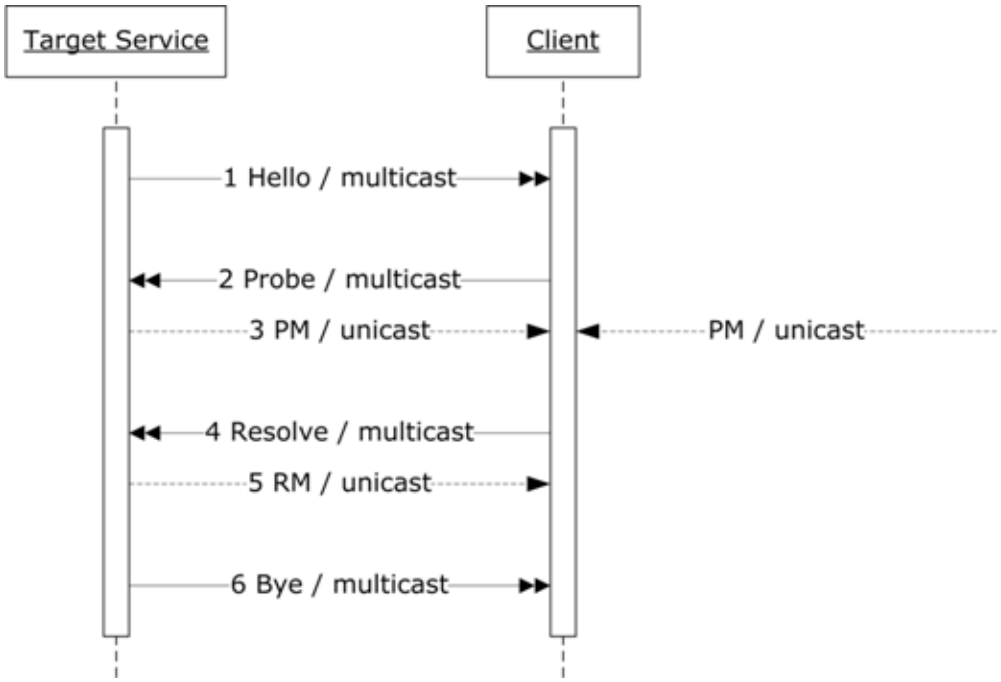
2.2.7 Profil T

Profil T je určen pro video systémy a podporuje funkce streamování videa jako například použití kódovacích formátů H.264 a H.265, nastavení obrazu a událostí nebo například detekce pohybu. Mezi povinné funkce pro zařízení patří zobrazení na obrazovce a streamování metadat a pro klienty jsou povinné funkce ovládání PTZ (Pan-Tilt-Zoom). Profil T také zahrnuje ONVIF specifikace pro HTTPS (Hypertext Transfer Protocol Secure) streamování, konfiguraci PTZ, konfiguraci oblastí pohybu, digitální vstupy a reléové výstupy a duplexní audio pro zařízení a klienty, které podporují tyto funkce. Je nutno dodat, že i když se profil T zaměřuje na streamování videa jako profil S, nejedná se o jeho náhradu s jinou cílovou aplikací, nicméně profily S a T mohou být kombinovány.

2.3 Vyhledání zařízení

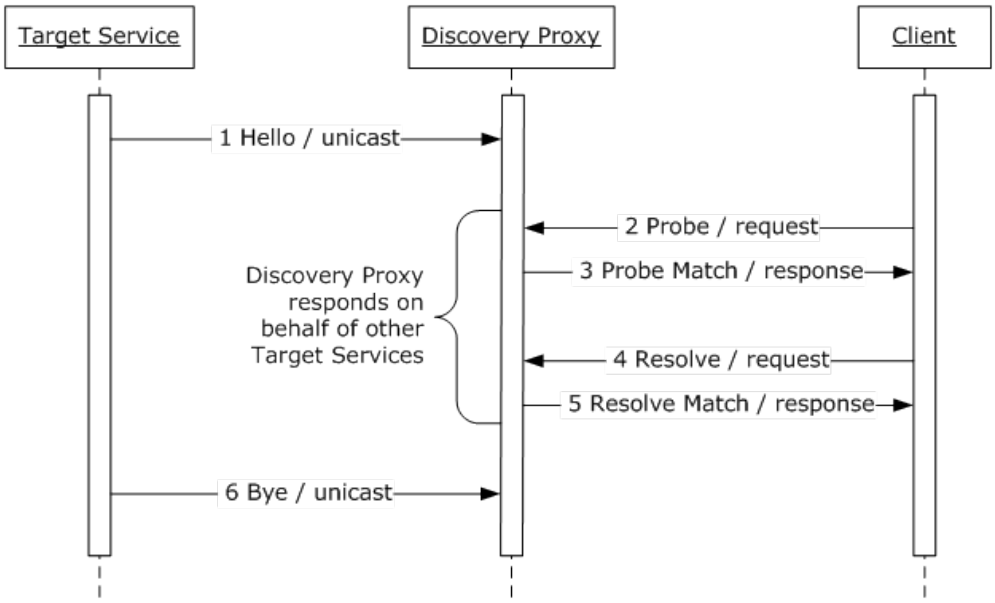
Standard ONVIF definuje, že zařízení musí podporovat vyhledávání pomocí WSDD (Web Service Dynamic Discovery) [4] protokolu. Tento protokol pro komunikaci využívá protokol SOAP [3], stejně jako standard ONVIF. Na rozdíl od standardu ONVIF, který je velice rozsáhlý a definuje mnoho různých typů zpráv, protokol WSDD definuje pouze 6 zpráv. Konkrétně se jedná o zprávy *Hello*, *Bye*, *Probe*, *ProbeMatch*, *Reslove* a *ResolveMatch*. Pro zařízení jsou definovány dva režimy komunikace - ad hoc a řízený.

V režimu ad hoc zařízení po připojení do sítě vysílá zprávu *Hello* pomocí multicastu. Tato zpráva obsahuje základní informace o zařízení, včetně jeho adresy, takže každý, kdo zprávu přijme, může se zařízením začít komunikovat. Pokud se do sítě připojí klient a chce vyhledat dostupná zařízení na síti, vysílá zprávu *Probe* pomocí multicastu. Zpráva *Probe* obsahuje typ hledaných zařízení. Všechna zařízení odpovídajícího typu pak pomocí unicastu odpoví klientovi zprávou *ProbeMatch*, jejíž obsah je podobný zprávě *Hello*. Tento způsob vyhledání se používá především k objevení nových zařízení v síti. Pokud klient chce vyhledat v síti zařízení, u kterého zná jeho unikátní identifikátor, může k tomu použít zprávu *Resolve* vyslanou pomocí multicastu. Zařízení se shodným identifikátorem na tuto zprávu odpovídá unicastovým vysláním zprávy *ResolveMatch*. Toto se může hodit, například pokud došlo ke změně adresy zařízení. Před odpojením od sítě zařízení vysílá pomocí multicastu zprávu *Bye*, kterou informuje ostatní členy sítě o svém odchodu. Znázornění komunikace v ad hoc režimu se nachází na obrázku 2.1



Obrázek 2.1. Schéma komunikace v ad hoc režimu. Převzato z [4]

Další možností je provozovat WSDD v řízeném režimu. V takovém případě se v síti vyskytuje tzv. *Discovery Proxy*, která slouží jako prostředník pro vyhledávání zařízení. Jak pak vypadá komunikace najdeme v obrázku 2.2. V tomto případě zařízení vysílá pouze zprávu *Hello* při připojení do sítě a zprávu *Bye* před odpojením od sítě. Tyto zprávy jsou pomocí unicastu doručeny pouze *Discovery Proxy*. Při přijetí zprávy *Hello* si vytvoří *Discovery Proxy* záznam o zařízení a při přijetí *Bye* záznam smaže. Klient



Obrázek 2.2. Schéma komunikace v řízeném režimu. Převzato z [4]

v tomto případě při vyhledávání zařízení posílá zprávy *Probe* a *Resolve* přímo *Discovery Proxy*, která odpovídá pomocí údajů ve svých záznamech.

2.4 Zabezpečení služeb

Aby nemohl k zařízení prostřednictvím ONVIF rozhraní přistupovat kdokoliv, je v rámci standardu ONVIF aplikován koncept uživatelů. Každé zařízení má svůj seznam uživatelů, který lze dále prostřednictvím webových služeb rozšiřovat a upravovat. Standard ONVIF definuje čtyři úrovně uživatelů lišící se přístupovými právy. Tyto úrovně jsou anonymní, uživatel, operátor a administrátor. Úroveň anonymní slouží pro případy, kdy v požadavku na vykonání služby nejsou validní iniciály uživatele. Přístupová práva pro jednotlivé úrovně jsou v tabulce v obrázku 2.1.

	Administrator	Operator	User	Anonymous
PRE_AUTH	x	x	x	x
READ_SYSTEM	x	x	x	
READ_SYSTEM_SENSITIVE	x	x		
READ_SYSTEM_SECRET	x			
WRITE_SYSTEM	x			
UNRECOVERABLE	x			
READ_MEDIA	x	x	x	
ACTUATE	x	x		

Tabulka 2.1. Tabulka přístupových práv. Převzato z [9]

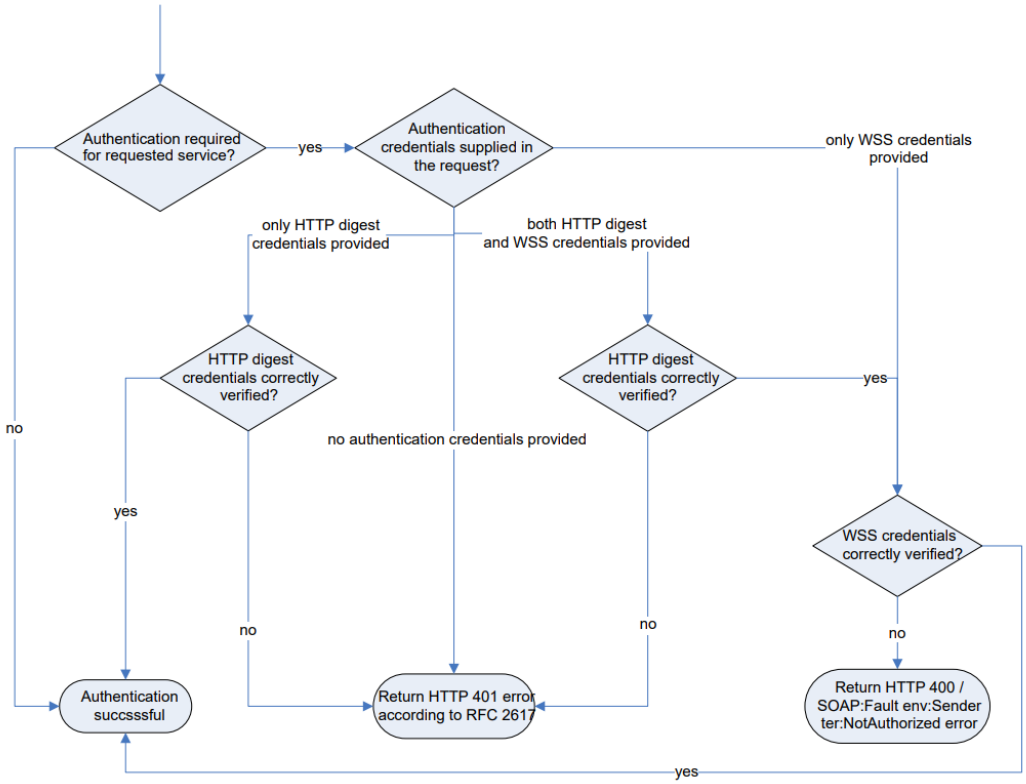
V rámci definice webové služby ONVIF navíc uvádí, jaká přístupová práva jsou zapotřebí k jejímu volání. Při volání webové služby je tedy nejdříve nutné ověřit, jestli je volána platným uživatelem s příslušnými přístupovými právy, a až poté přejít k vlastní exekuci webové služby.

Pro ověřování uživatelů umožňuje ONVIF použít dva různé protokoly. Již méně podporovanou možností je protokol WSS-UsernameToken (Web Service Security Username

Token) [5]. Jedná se o zabezpečovací protokol na úrovni protokolu SOAP. Iniciály uživatele, jako jsou jméno a heslo, jsou zde přidány do hlavičky XML zprávy protokolu SOAP, kterou posílá klient do zařízení. Pro větší úroveň zabezpečení je možné heslo předávat zašifrované pomocí algoritmu SHA1.

Druhou možností pro ověřování uživatelů je protokol HTTP Authentication [6]. Jak již z názvu vyplývá, tento protokol je založený na protokolu HTTP, který protokol SOAP používá k přenosu zpráv. Iniciály uživatele jsou obsaženy v hlavičce HTTP zprávy jedním ze dvou způsobů. V případě Basic Authentication hlavička zprávy obsahuje přímo jméno a heslo uživatele. Mnohem více zabezpečenou variantou je Digest Authentication, kde se využívá šifrovacího algoritmu MD5 a klient odesílá pouze kontrolní součet jména, hesla a řetězce obdrženého od zařízení.

Jak má probíhat ověření uživatele při požadavku na vykonání webové služby je znázorněno na obrázku 2.3. Z obrázku je zřejmé, že klient může používat libovolný ověřovací protokol nebo i jejich kombinaci.

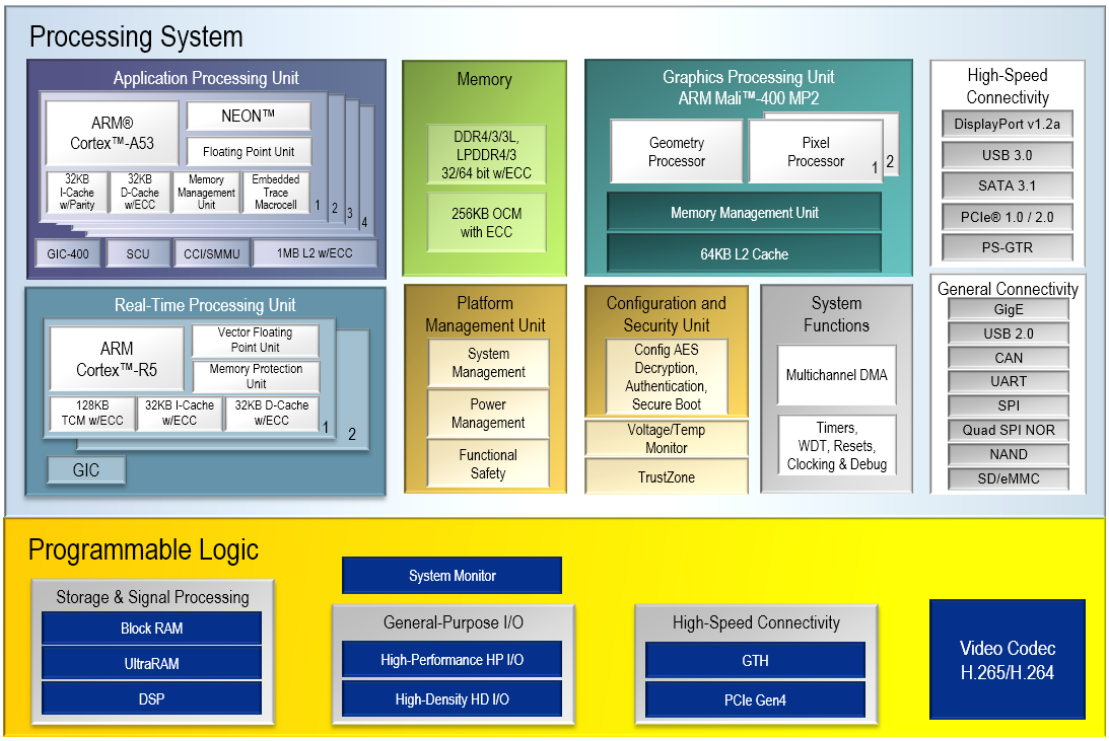


Obrázek 2.3. Ověření příchozího požadavku. Převzato z [9]

Alternativnou k již zmíněným zabezpečovacím protokolům je použití protokolu HTTPS, který je dnes již prakticky standardem webových serverů. Vzhledem k tomu, že protokol HTTPS vyžaduje vytváření certifikátu pro každé zařízení a navíc se standard ONVIF využívá převážně v izolovaných sítích, není tato alternativa příliš často používána.

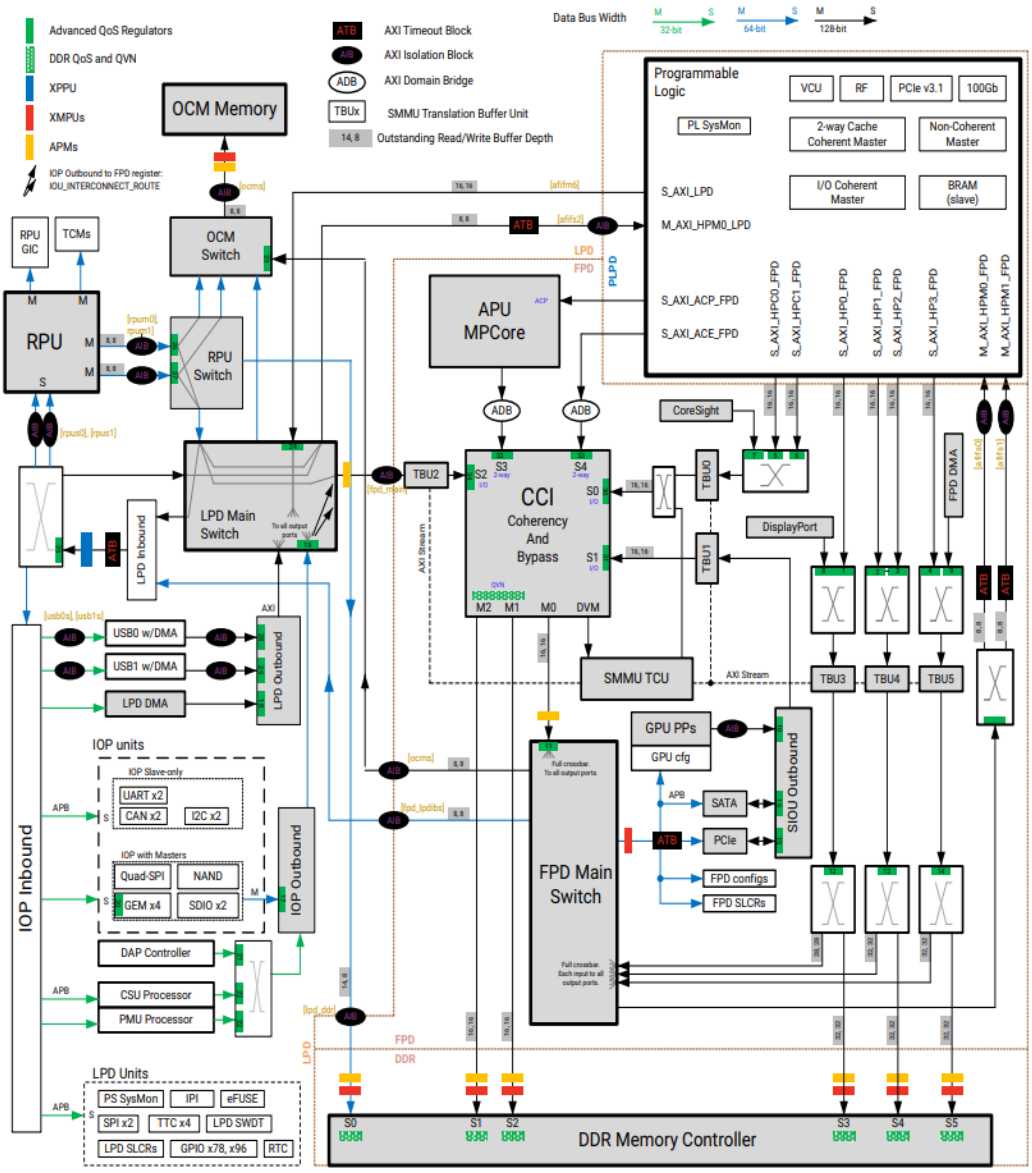
Kapitola 3
Platforma Zynq UltraScale+ MPSoC

Hlavním znakem platformy Zynq UltraScale+ [10] je spojení programovatelné logiky a procesorového systému, složeného z několika výpočetních jednotek, na jednom čipu. Platforma nabízí 3 varianty čipů, a to konkrétně CG, EG a EV. Varianta CG je určena pro heterogenní zpracování dat. Varianta EG je určena pro proužití v co nejširším spektru aplikací. Varianta EV je určena na zpracování multimediálních dat a strojové vidění. Čip, na kterém bude práce vyvíjena, je variantou EV, proto se popis v této kapitole bude zaměřovat na tuto variantu.



Obrázek 3.1. Blokové schéma Zynq UltraScale+ MPSoC EV. Převzato z [10]

Blokové schéma čipu varianty EV můžeme vidět na obrázku 3.1. Procesorový systém tohoto čipu obsahuje výpočetní jednotky APU (Application Processing Unit), RTPU (Real-Time Processing Unit) a GPU (Graphics Processing Unit), sdílenou pamět, řídící jednotky a řadu periferních sběrnic. Interní propojení čipu je na obrázku 3.2



Obrázek 3.2. Interní propojení Zynq UltraScale+ MPSoC EV. Převzato z [10]

3.1 Application Processing Unit

APU je tvořeno čtyřmi jádry, přičemž každé jádro obsahuje:

- ARM Cortex-A53
- jednotku NEON pro SIMD (Single Instruction Multiple Data) instrukce
- výpočetní jednotku pro floating point aritmetiku FPU (Floating Point Unit)
- pamětí L1 cache 32kB pro instrukce a 32kB pro data
- jednotku pro správu paměti MMU (Memory Management Uni)
- jednotku pro sledování jádra v reálném čase ETM (Embedded Trace Macrocell)

Dále jednotka obsahuje také 1MB L2 cache sdílenou mezi jádry a další jednotky zajišťující řízení jáder a přístupu k paměti. Těmi jsou GIC (Generic Interrupt Controller), SCU (Snoop Control Unit) a CCI (Cache Coherent Interconnec) spojený s SMMU (System Memory Management Uni). Varianta EG má tuto jednotku identickou, avšak varianta CG má pouze dvě jádra.

3.2 Real-Time Processing Unit

Oproti APU, je RTPU tvořeno pouze dvěma jádry, která se skládají z:

- ARM Cortex-R5
- vypočetní jednotky pro vektorovou floating point aritmetiku VFPU (Vector Floating Point Unit)
- vypočetní jednotky pro floating point aritmetiku FPU
- paměti L1 cache 32kB pro instrukce a 32kB pro data
- TCM (Tightly-Coupled Memory) 128kB

Kromě jader obsahuje RTPU pouze jednotku pro řízení přerušení GIC. Tato jednotka je stejná i pro varianty CG a EG.

3.3 Graphics Processing Unit

Grafickou jednotkou použitou pro tento systém je ARM Mali-400MP2. Tato jednotka je vybavena geometrickým procesorem, dvěma pixelovými procesory, jednotkou pro správu paměti MMU a pamětí L2 cache o velikosti 64kB. Stejnou jednotku nalezneme i u varianty EG, ale varianta CG je zcela bez grafické jednotky.

3.4 Periferní sběrnice

Aby byla platforma co nejvíce flexibilní, je výpočetní jednotka vybavena řadou periferních sběrnic. Jak si můžeme všimnout ve schématu 3.1, jsou tyto sběrnice rozděleny do dvou kategorií, a to na vysokorychlostní sběrnice a sběrnice pro obecnou konektivitu. Vysokorychlostní sběrnice dostupné na platformě jsou:

- DisplayPort v1.2a
- USB 3.0
- SATA 3.1
- PCIe 1.0/2.0
- PS-GTR

Dostupné obecné sběrnice jsou:

- GigE (Gigabit Ethernet)
- USB (Universal Serial Bus) 2.0
- CAN (Controller Area Network)
- UART (Universal Asynchronous Receiver-Transmitter)
- SPI (Serial Peripheral Interface)
- Quad SPI NOR
- NAND
- SD (Secure Digital)
- eMMC (Embedded MultiMediaCard)

3.5 Programovatelná logika

Varianta EV nabízí tři řady programovatelné logiky, které se liší především počtem logických prvků a velikostí pamětí. Varianty CG a EG nabízí i řady s většími počty logických prvků, ale nejsou vybaveny integrovaným kodekem H.264/H.265, který je u varianty EV ve všech třech řadách. Tento kodek je výhodný například pro enkódované streamování videa, a proto byla pro tuto práci použita varianta EV. Další vlastností čipu je, že programovatelná logika má samostatný napájecí okruh a je tak možné využít procesorovou část čipu samostatně.

Prostřednictvím programovatelné logiky by bylo možné velice efektivně implementovat testovací generátor obrazu a s využitím integrovaného kodeku obraz streamovat. Ovšem vzhledem k tomu, že cílový jazyk implementace je jazyk C++, nebyla tato možnost v rámci této práce využita.

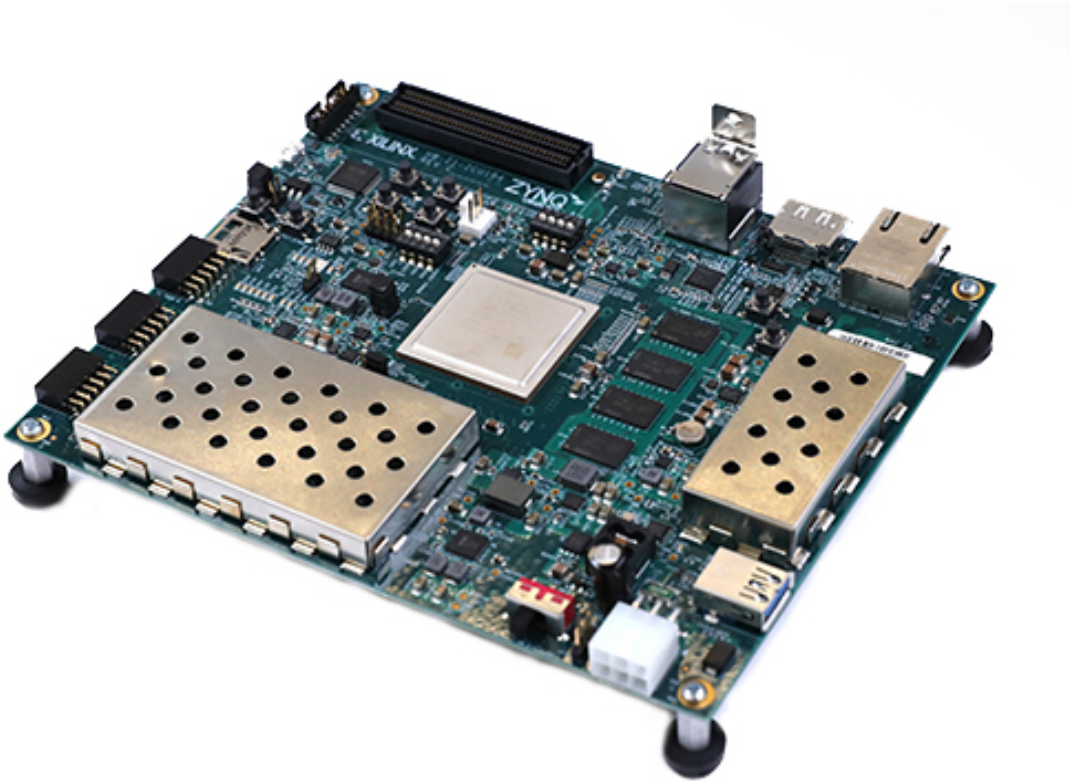
3.6 PetaLinux

Nástroje PetaLinux od společnosti Xilinx umožňují vytváření embedded Linuxových systémů pro platformy s čipy Xilinx [12]. Pomocí nástrojů PetaLinux lze snadno sestavit a nakonfigurovat systém. Sada nástrojů PetaLinux umožňuje snadné vytváření uživatelských aplikací a ovladačů pomocí standardních open-source komponent a knihoven, bez nutnosti ručního sestavování systému z jednotlivých komponent. Díky tomu mohou vývojáři efektivněji vyvíjet aplikace a ovladače pro embedded Linuxové systémy. Kromě toho nástroje PetaLinux umožňují vytvořit SDK (Software Development Kit), které obsahuje nástroje pro sestavování, ladění a profilování kódu.

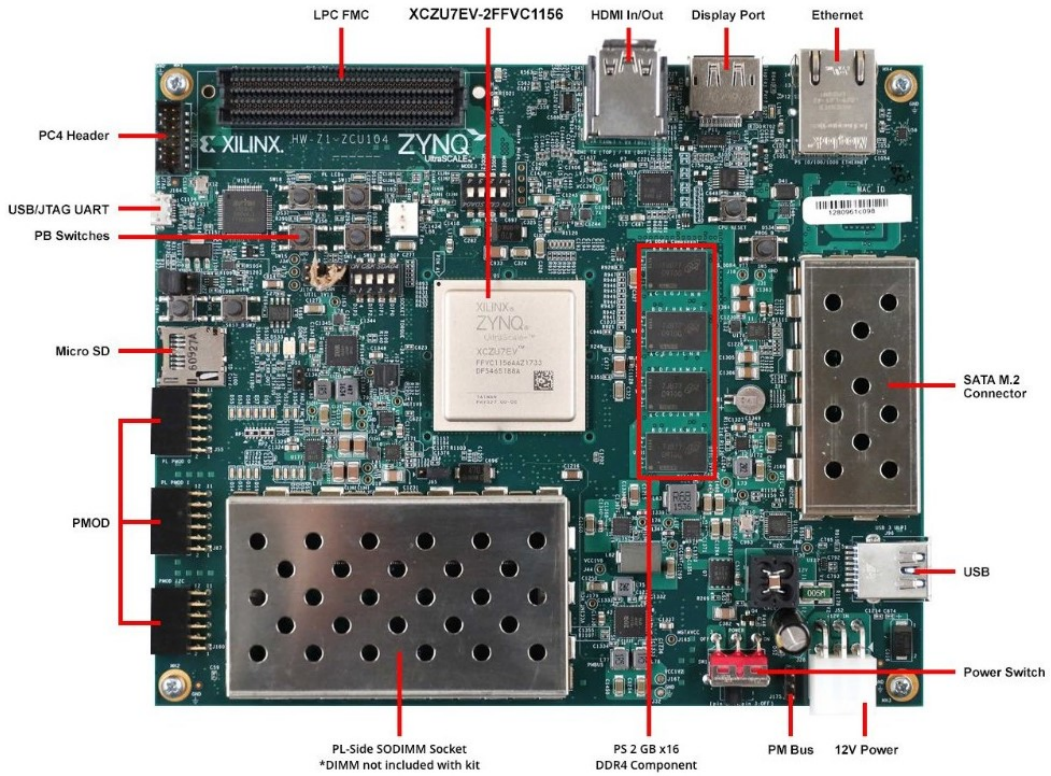
3.7 ZCU104

Deska ZCU104, kterou můžeme vidět na obrázku 3.3, je vysoce výkonná platforma pro vývoj a prototypování systémů založených na programovatelné logice. Deska je osazena čipem Zynq UltraScale+ XCZU7EV-2FFVC1156 MPSoC doplněným o 2GB (Gigabyte) paměti DDR4. To umožňuje uživatelům provádět rychlé prototypování, vývoj a ověřování výkonu různých aplikací v oblasti umělé inteligence, strojového učení, zpracování obrazu a videa, síťového zpracování a dalších.

Deska je vybavena rozhraními jako jsou GigE, HDMI (High-Definition Multimedia Interface), USB 3.0, JTAG (Joint Test Action Group), DisplayPort a SATA M.2, což umožňuje snadnou integraci s jinými zařízeními a systémy. Jak jsou jednotlivá rozhraní na desce situována ilustruje obrázek 3.4.



Obrázek 3.3. Deska ZCU104. Převzato z [11]



Obrázek 3.4. Rozhraní desky ZCU104. Převzato z [11]

Kapitola 4
Implementace

Tato kapitola se zaměřuje na popis vzniklé implementace. Kapitola je rozdělena do pěti částí. V první části je představena implementace ONVIF serveru, ve druhé části je popsáno zapouzdření serveru do knihovny, třetí část se věnuje aplikaci na otestování serveru a streamování videa pomocí protokolu RTSP a zbylé dvě části se věnují spuštění desky ZCU104 a křížovému překladu.

4.1 Implementace ONVIF serveru

No rozdíl od jazyků jako je Java nebo C# nemá jazyk C++ nativní podporu webových služeb ani protokolu SOAP. Existují zde však knihovny pro jazyk C++, které se zabývají touto tematikou.Mezi nejrozšířenější z nich patří například gSOAP od společnosti Genivia[1], KDSoap[2] a Axis[3]. Pro tuto práci byla použita knihovna gSOAP. Hlavním důvodem této volby bylo zpracování dokumentace, ale i to, že je knihovna gSOAP stále udržovaná.

Knihovna gSOAP [13] nabízí nástroje pro implementaci webových služeb využívajících protokol SOAP, a to jak ze strany klienta využívajícího služeb, tak i ze strany serveru poskytujícího služby. Tato knihovna je dostupná jak v komerční, tak i v bezplatné verzi pod licencí *GNU General Public License*. Základním prvkem knihovny jsou nástroje *wsdl2h* a *soapcpp2*, které umožňují převést definice webových služeb v jazyce WSDL do kódu v jazyce C++. Tyto nástroje mají více způsobů využití, jako například návrh webových služeb s využitím syntaxe jazyka C++ nebo implementace webových služeb definovaných v jazyce WSDL.

[1] `https://www.genivia.com/dev.html`
[2] `https://www.kdab.com/development-resources/qt-tools/kd-soap/`
[3] `https://axis.apache.org/axis/`

4.1.1 wsdl2h

Jak bylo popsáno v kapitole 2, webové služby, které zařízení vyhovující standardu ONVIF musí poskytovat, jsou definovány v jazyce WSDL. Tento popis služeb ale není v jazyce C++ příliš vhodný pro další práci. Nástroj *wsdl2h* však umožňuje popis převést do hlavičkového souboru jazyka C++. Příklad použití tohoto nástroje můžeme vidět v ukázce 4.1.

```
wsdl2h -O4 -P -x -o onvif.h devicemgmt.wsdl
```

Ukázka kódu 4.1. Příklad použití nástroje *wsdl2h*

Zde by nástroj měl definice webových služeb v souboru *devicemgmt.wsdl* převést do syntaxe jazyka C++ a uložit je do souboru *onvif.h*. Přepínače **-O4**, **-P** a **-x** slouží především k optimalizaci výstupu a odstranění nepotřebného kódu. Přepínač **-o** slouží ke specifikaci výstupního souboru. Užitečnou vlastností tohoto nástroje je také možnost jako vstupní soubor použít soubor umístění na internetu. Je tedy možné nahradit soubor *devicemgmt.wsdl* adresou *http://www.onvif.org/onvif/ver10/device/wsdl/devicemgmt.wsdl*. Je také možné použít více souborů s definicemi webových služeb v jazyce WSDL, což můžeme vidět v ukázce 4.2

```
wsdl2h -O4 -P -x -o onvif.h \
    http://www.onvif.org/onvif/ver10/device/wsdl/devicemgmt.wsdl \
    http://www.onvif.org/onvif/ver10/media/wsdl/media.wsdl
```

Ukázka kódu 4.2. Příklad použití nástroje *wsdl2h* s více WSDL soubory

Dále nástroj *wsdl2h* vyžaduje přítomnost souboru *typemap.dat* v aktuálním adresáři. Tento soubor umožňuje konfigurovat podobu výstupního kódu v jazyce C++. Najdeme v něm například nastavení mapování jmenných prostorů jazyka XML do jazyka C++ nebo nastavení typů některých proměnných. Spolu s knihovnou je dodávána i základní verze tohoto souboru, která lze beze změn využít i pro služby standardu ONVIF.

```
class _tds__GetDeviceInformationResponse
{
public:
    std::string Manufacturer;
    std::string Model;
    std::string FirmwareVersion;
    std::string SerialNumber;
    std::string HardwareId;
    struct soap *soap;
};
```

Ukázka kódu 4.3. Příklad výstupu z nátroje *wsdl2h*

Po použití nástroje *wsdl2h* tedy získáme hlavičkový soubor, ve kterém jsou převedené definice z jazyka WSDL do tříd, struktur a enumů. Jak by vypadala třída reprezentující odpověď služby *GetDeviceInformation* z ukázek 2.1 a 2.2 najdeme v ukázce 4.3.

Předpona *tds* před názvem třídy odkazuje na jmenný prostor jazyka XML, do kterého webová služba spadá. Jaký jmenný prostor patří k předponě je možné dohledat v souboru *typemap.dat*, zde konkrétně se jedná o *http://www.onvif.org/ver10/device/wsdl*. Ve vygenerované třídě poté najdeme členy pro jednotlivé parametry odpovědi. V ukázce jsou pro jednoduchost vynechány komentáře, které nástroj generuje. V komentářích jsou uvedeny speciální direktivy, které dále využívá nástroj *soapcpp2*. Zároveň je komentáři označeno, které informace ve třídách jsou povinné a které mohou být vynechány. Nepříjemnou vlastností hlavičkového souboru vytvořeného nástrojem *wsdl2h* je, že i pro poměrně malý počet webových služeb nabývá soubor velkého objemu a není příliš přehledný.

4.1.2 soapcpp2

Protokol SOAP přenáší informace prostřednictvím zpráv ve formátu XML. Zpráva má tedy po přijetí podobu formátovaného textu, který pouze s definicemi služeb nelze využít. Je zapotřebí převést data zprávy z textu do datových typů jazyka C++. Tomuto procesu se anglicky říká *demarshalling* a při převodu v obráceném směru *marshalling*. Nástroj *soapcpp2* umožňuje z definic služeb v hlavičkovém souboru vytvořit kostru kódu, která obsahuje funkce pro *marshalling* i *demarshalling* zpráv.

```
soapcpp2 -2 -j -L -S -a -x onvif.h
```

Ukázka kódu 4.4. Příklad použití nástroje *soapcpp2*

V ukázce 4.4 vidíme příklad volání nástroje *soapcpp2*. Vstupním souborem tohoto nástroje je hlavičkový soubor obsahující definice webových služeb. Zpravidla se jedná o soubor získaný nástrojem *wsdl2h*. Přepínač **-2** určuje, že chceme vygenerovat kód pro protokol SOAP verze 1.2, který je použit ve standardu ONVIF. Přepínač **-S** st vuje, že generovaný kód je dále určený pro návrh serveru poskytujícího služby. Přepínač **-x** zabrání vytváření vzorových zpráv v jazyce XML. Zbylé přepínače, tedy **-j**, **-L** a **-a**, dále optimalizují výstup nástroje. Dále je nutné pomocí přepínače **-I** doplnit cesty k použitým definicím. Výstupem nástroje *soapcpp2* je zpravidla několik souborů. Pokud by jsme použili nástroj *soapcpp2* podle ukázky 4.4 na hlavičkový soubor vytvořený v ukázce 4.1, jednalo by se o následující soubory:

- soapH.h
- soapC.cpp
- soapStub.h
- soapDeviceBindingService.h
- soapDeviceBindingService.cpp
- DeviceBinding.nsmap

První tři z uvedených souborů obsahují definice enumů, tříd a struktur, které slouží jako parametry a návratové hodnoty jednotlivých webových služeb, a dále také interní funkce knihovny, určené pro *marshalling* a *demarshalling* zpráv protokolu SOAP. V souboru *DeviceBinding.nsmap* nalezneme definici jmenných prostorů jazyka XML, které budou používány. Pro další implementaci je pro nás nejzajímavější soubor *soapDeviceBindingService.h*, ve kterém je definována třída *DeviceBindingService*. V této třídě najdeme několik interních metod knihovny, ale také především virtuální metody reprezentující konkrétní webové služby. Vlastní implementaci jednotlivých webových služeb

za nás nástroj *soapcpp2* však již doplnit nemůže. Podrobnější implementaci jednotlivých služeb popíši později. Tříd jako je *DeviceBindingService* může nástroj *soapcpp2* generovat více podle obsahu vstupního hlavičkového souboru. Pokud použijme nástroj *soapcpp2* na hlavičkový soubor z ukázky 4.2, získáme tedy také třídu *MediaBindingService*, která bude podobně jako třída *DeviceBindingService* obsahovat virtuální metody pro příslušné webové služby.

4.1.3 Poskytování služeb

Jelikož se knihovna gSOAP snaží podporovat co nejvíce různých aplikací protokolu SOAP, nabízí hned několik možností jak server vytvořit, ty se liší především podle počtu poskytovaných webových služeb a počtu klientů, kteří se serverem komunikují ve stejném čase. Pro potřeby této práce byla zvolena taková implementace, která počítá spíše s větším počtem poskytovaných služeb, ale s malým počtem klientů.

Nejdříve je nutné vytvořit instanci struktury *soap*, která slouží jako základní kontext při práci s knihovnou gSOAP. Příklad vytvoření struktury je v ukázce 4.5. Zde se kromě vytvoření také nastavují časové limity, které mají být aplikovány při nečinnosti serveru a odesílání či příjmu dat.

```
struct soap* soap = soap_new1();
// timeouts in seconds
soap->accept_timeout = 60;
soap->send_timeout = 10;
soap->recv_timeout = 10;
soap->transfer_timeout = 30;
```

Ukázka kódu 4.5. Inicializace struktury *soap*

Dalším krokem při implementaci serveru je vytvoření instancí tříd zajišťujících webové služby, jejichž vytvoření již bylo popsáno dříve. Pokud by taková třída byla pouze jedna, bylo by možné celý následující postup značně zjednodušit, nicméně v případě implementace standardu ONVIF bude těchto tříd více. Zde se pro jednoduchost omezím na dvě třídy, jejichž vytvoření je v ukázce 4.6. Všimněme si, že konstruktor tříd umožňuje vstupním parametrem třídy napojit na strukturu *soap*.

```
DeviceBindingService device(soap);
MediaBindingService media(soap);
```

Ukázka kódu 4.6. Vytvoření instancí tříd *DeviceBindingService* a *MediaBindingService*

Nyní již jsou inicializovány všechny proměnné a tak je možné pokročit k připojení kontextu na port, ze kterého bude jak server komunikovat s klienty. Připojení najdeme v ukázce 4.7, kde se také kontroluje, zda bylo připojení úspěšné. Kontext začne po připojení na daném portu poslouchat příchozí zprávy.

Posledním krokem implementace serveru je samotné poskytování služeb. Oproti předchozím krokům, které jsou univerzální, zde existuje více způsobů, jak služby poskytovat. Pro některé aplikace je zde vhodné volit vícevláknová řešení, a to zejména tam, kde se serverem bude komunikovat velký počet klientů najednou. Cílová platforma pro tuto práci však není příliš vhodná ke spouštění velkého počtu vláken a navíc lze předpokládat, že s vytvářeným serverem bude komunikovat pouze jeden klient. Proto bylo

```
SOAP_SOCKET socket = soap_bind(soap, NULL, 8080, 10);
if (!soap_valid_socket(m))
{
    soap_print_fault(soap, stderr);
    return 101;
}
```

Ukázka kódu 4.7. Připojení konextu *soap* na port

v ukázce 4.8 zvoleno pouze jednovláknové řešení. Jedná se o cyklus, který se opakuje pokud je stále validní připojení kontextu. V tomto cyklu se funkcí *soap_begin_serve* zahájí obsluha příchozího požadavku a poté se postupně volá metoda *dispatch* jednotlivých instancí tříd poskytujících služby. Tato metoda zajistí *demarshalling* příchozí zprávy a na jeho základě volá odpovídající službu. Po dokončení služby se provede *marshalling* a výsledky se odešlou klientovi, od kterého byla služba volána. Metoda *dispatch* může mít několik návratových hodnot, zde je však zásadní *SOAP_NO_METHOD*, která signalizuje, že požadovaná webová služba není třídou poskytována. V případě, kdy *dispatch* jedné třídy vrátí hodnotu *SOAP_NO_METHOD*, je volána metoda *dispatch* další třídy. Takto lze volání řetězit pro libovolný počet tříd poskytujících služby. Po vykonání služby se kontroluje, zda při jejím vykonání nedošlo k chybě a případné chyby se zasílají klientovi. Na konci cykly jsou volány dvě funkce, a to *soap_destroy*, která uvolní objekty vytvořené v kontextu během obsluhy požadavku, a *soap_end*, která ukončí obsluhu požadavku a umožní tak přesunout se na další.

```
while (soap_valid_socket(soap_accept(soap)))
{
    if (soap_begin_serve(soap) == SOAP_OK)
    {
        if (device.dispatch() == SOAP_NO_METHOD)
        {
            media.dispatch();
        }
        if (soap->error)
        {
            soap_send_fault(soap);
        }
    }
    soap_destroy(soap);
    soap_end(soap);
}
```

Ukázka kódu 4.8. Cyklus obsluhující příchozí požadavky na webové služby

Spojením předchozích ukázek kódu získáme kompletní implementaci serveru, což můžeme vidět v ukázce 4.9. Přibylo zde volání funkce *soap_free*, která zajistí uvolnění paměti zabrané kontextem *soap*.

```
int main (int argc, char* argv[])
{
    struct soap* soap = soap_new1();
    // timeouts in seconds
    soap->accept_timeout = 60;
    soap->send_timeout = 10;
    soap->recv_timeout = 10;
    soap->transfer_timeout = 30;

    DeviceBindingService device(soap);
    MediaBindingService media(soap);

    SOAP_SOCKET socket = soap_bind(soap, 10.15.0.165, 8080, 10);
    if (!soap_valid_socket(m))
    {
        soap_print_fault(soap, stderr);
        return 101;
    }

    while (soap_valid_socket(soap_accept(soap)))
    {
        if (soap_begin_serve(soap) == SOAP_OK)
        {
            if (device.dispatch() == SOAP_NO_METHOD)
            {
                media.dispatch();
            }
            if (soap->error)
            {
                soap_send_fault(soap);
            }
        }
        soap_destroy(soap);
        soap_end(soap);
    }

    soap_free(soap);

    return 0;
}
```

Ukázka kódu 4.9. Kompletní implementace serveru

4.1.4 WSDD

Protokol WSDD [4] je velmi často používaný pro vyhledávání zařízení, a proto je obsažen i v knihovně gSOAP. Nalezneme zde několik hlavičkových souborů obsahujících definice zpráv, nicméně všechny definují stejné zprávy a liší se pouze verzemi definic základních typů, na které odkazují. Pomocí nástroje *soapcpp2* lze z těchto hlavičkových souborů snadno vytvořit kostru kódu. Musíme však při volání nástroje odebrat přepínač **-S**, jelikož potřebujeme i kód generovaný pro klienta. Doplnění takto vniklého kódu je však mnohem jednodušší než v předchozích případech. Například pro odeslání zprávy *Hello* postačí zavolat funkci *soap_wsdd_Hello* a jako její parametry použít informace o zařízení, což můžeme vidět v ukázce 4.10.

```
soap_wsdd_Hello(soap,
                SOAP_WSDD_ADHOC,
                "soap.udp://239.255.255.250:3702",
                soap_wsa_rand_uuid(soap),
                NULL,
                "endpoint-id",
                "device-type",
                "scopes",
                NULL,
                "address",
                0)
```

Ukázka kódu 4.10. Příklad použití *soap_wsdd_Hello*

Aby bylo zařízení schopné odpovídat na příchozí multicastové požadavky od klientů, je nutné podobně jako v případě 4.9 vytvořit strukturu *soap*. Tentokrát je však nutné nastavit ji pomocí funkce *setsockopt* tak, aby přijímala zprávy na příslušné multicastové adrese. K přijetí a zpracování zpráv je možné použít funkci *soap_wsdd_listen*. Použití těchto funkcí k implementaci WSDD je v ukázce 4.11. Pokud je protokol WSDD využíván současně s poskytováním služeb, je nutné spustit rutinu obsluhující webové služby paralelně s rutinou zajišťující WSDD.

```
void main(int argc, char* argv[])
{
    struct soap* server = soap_new1(SOAP_IO_UDP);
    server->bind_flags = SO_REUSEADDR;
    SOAP_SOCKET socket = soap_bind(server, NULL, 3702, 1000);
    if (!soap_valid_socket(socket))
    {
        soap_print_fault(server, stderr);
        soap_destroy(server);
        soap_end(server);
        m_running = false;
        return;
    }

    ip_mreq mcast;
    inet_pton(AF_INET, "239.255.255.250",
                       &mcast.imr_multiaddr.s_addr);
    mcast.imr_interface.s_addr = htonl(INADDR_ANY);
    if (setsockopt(server->master,
                   IPPROTO_IP,
                   IP_ADD_MEMBERSHIP,
                   reinterpret_cast<const char*>(&mcast),
                   sizeof(mcast)) < 0)
    {
        soap_destroy(server);
        soap_end(server);
        m_running = false;
        return;
    }

    sendHello(server);
    while (running)
    {
        soap_wsdd_listen(server, -100);
    }
    sendBye(server);

    soap_destroy(server);
    soap_end(server);
}
```

Ukázka kódu 4.11. Spuštění WSDD

4.1.5 Zabezpečení služeb

Podobně jako u protokol WSDD, má knihovna gSOAP podporu pro oba protokoly ověřování uživatelů použité ve standardu ONVIF. Oproti WSDD zde však již není nutné používat nástroj *soapcpp2*, jelikož je tato funkcionalita zahrnuta v samostatných pluginech knihovny. Tyto pluginy je nutné při inicializaci struktury *soap* zaregistrovat pomocí funkce *soap_register_plugin*.

Protokol WSS-UsernameToken je obsažen v pluginu *wsseapi*. Hlavními funkcemi tohoto modulu jsou *soap_wsse_get_Username* a *soap_wsse_verify_Password*. Jak již název napovídá, funkce *soap_wsse_get_Username* slouží ke zjištění jména uživatele, aby mohl být dohledán v databázi uživatelů. Oproti tomu funkce *soap_wsse_verify_Password* slouží k ověření platnosti hesla uživatele. Příklad použití těchto funkcí je v ukázce 4.12.

```
const std::string username(soap_wsse_get_Username(soap));
auto userData = userDatabase.findUser(username);
if (!userData.hasValue())
{
    // user not in database
    return 401;
}

if (soap_wsse_verify_Password(soap, userData.password()))
{
    // invalid password
    return soap->error;
}
```

Ukázka kódu 4.12. Ověření WS-UsernameToken

Pro druhý z protokolů ověřování uživatelů používaný v rámci standardu ONVIF, HTTP Authentication, je určen plugin *httpda*. Při registraci tohoto pluginu je nutné specifikovat, že má být použito šifrovací algoritmus MD5, jak je definováno v rámci standardu ONVIF. Na rozdíl od pluginu *wsseapi* jsou v tomto případě iniciály uživatele přístupné přes členy struktury *soap*. Na základě toho, kteří členové struktury *soap* jsou platní, je nutné rozhodnout, jestli má být pro ověření použita *Basic Authentication* nebo *Digest Authentication*. Pokud je platný člen obsahující heslo uživatele, jedná se o variantu *Basic Authentication* a k ověření pak stačí údaje ve struktuře *soap* porovnat s údaji v databázi. V opačném případě se předpokládá, že má být použita metoda *Digest Authentication*, pro kterou nabízí plugin *httpda* několik různých funkcí na ověření uživatele. Každá z těchto funkcí je určena jedné metodě požadavku protokolu HTTP [8]. Pro účely této práce je však zajímavá především funkce *http_da_verify_post*, jelikož standard ONVIF využívá metodu POST. Jak by mohlo vypadat ověření HTTP Authentication najdeme v ukázce 4.13. V ukázce si můžeme všimnout, že kromě jména a hesla uživatele se zde kontroluje a nastavuje *authrealm*, což je unikátní identifikátor serveru, který se využívá při šifrování iniciálu uživatele ke zvýšení úrovně zabezpečení.

```
if (!soap->userid)
{
    // no authentication credentials received
    soap->authrealm = authrealm;
    return 401;
}

const std::string username(soap->userid);
auto userData = userDatabase.findUser(username);
if (!userData.hasValue())
{
    // user not in database
    soap->authrealm = authrealm;
    return 401;
}

if (soap->authrealm) // HTTP Digest Authentication
{
    if (!strcmp(soap->authrealm, authrealm))
    {
        if (!http_da_verify_post(soap, user.value().password().c_str()))
        {
            return SOAP_OK;
        }
    }
}
else if (soap->passwd) // HTTP Basic Authentication
{
    if (!strcmp(soap->passwd, user.value().password().c_str()))
    {
        return SOAP_OK;
    }
}

soap->authrealm = authrealm;
return 401;
```

Ukázka kódu 4.13. Ověření *HTTP Authentication*

4.2 Implementace knihovny

V kapitole 4.1 již bylo popsáno, jak vytvořit ONVIF server, který bude poskytovat dané webové služby. Takto vzniklý server však není příliš vhodný k integraci do dalších aplikací. Z tohoto důvodu byl server zapouzdřen do knihovny jazyka C++, které poskytuje rozhraní vhodnější pro integraci. Nyní se zaměříme na způsob, jakým bylo toto rozhraní navrženo.

Základním prvkem rozhraní knihovny je třída *Server*, která umožňuje pomocí metod *start()* a *stop()* spustit či zastavit server poskytující ONVIF webové služby běžící v samostatném vlákně. Definici této třídy najdeme v ukázce 4.14. Do konstruktoru je nutné vložit adresu, na které má server komunikovat, a sdílený ukazatel[4] na třídu *Context*. Tato třída dále slouží jako prostředník pro předávání informací mezi knihovnou a koncovou aplikací. Vlastní implementace serveru je skryta ve třídě *ServerInternal*.

```
class Server
{
public:
  Server(const std::shared_ptr<Context>& context,
         const std::string& ip,
         int port=8080);
  ~Server();

public:
  void start();
  void stop();

private:
  std::unique_ptr<ServerInternal> internal;
};
```

Ukázka kódu 4.14. Definice třídy *Server*

Třída *Context*, jejíž definice je v ukázce 4.15, předává reference na třídy, které se zabývají určitou částí rozhraní. Každá z těchto tříd má definované virtuální metody, jejichž implementaci je nutno doplnit podle konkrétních potřeb nadřazené aplikace.

```
class Context
{

public:
    virtual SystemContext& system() = 0;
    virtual SecurityContext& security() = 0;
    virtual NetworkContext& network() = 0;
    virtual MediaContext& media() = 0;
    virtual DiscoveryContext& discovery() = 0;
};
```

Ukázka kódu 4.15. Definice třídy *Context*

[4] https://en.cppreference.com/w/cpp/memory/shared_ptr

Příkladem třídy, která se již zabývá pouze určitou částí rozhraní knihovny je třída *SecurityContext*. Definici této třídy najdeme v ukázce 4.16. Pro tuto třídu je definováno pět virtuálních metod, které se týkají správy uživatel. Dále je v rámci třídy definován výčtový typ *ReturnValue*, kterým metody předávají případné chybové hlášky. Pro metodu *userByName* je zde navíc definovaný typ *Optional*, pomocí kterého je možné předat buď validní hodnotu, nebo chybu definovanou ve výčtu *ReturnValue*.

```
class SecurityContext
{
public:
  enum ReturnValue
  {
    RETURN_VALUE__BEGIN,
    OK = RETURN_VALUE__BEGIN,
    USER_ALREADY_EXISTS,
    PASSWORD_TOO_LONG,
    USERNAME_TOO_LONG,
    WEAK_PASSWORD,
    MAXIMUM_USERS_EXCEEDED,
    ANONYMOUS_USER_NOT_ALLOWED,
    USERNAME_TOO_SHORT,
    FIXED_USER,
    USER_NOT_RECOGNIZED,
    RETURN_VALUE__END = USER_NOT_RECOGNIZED
  };
  template <class T>
  using Optional = OptionalResult<T, ReturnValue>;

  virtual std::vector<User> users() const = 0;
  virtual Optional<User> userByName(const std::string& name) const = 0;

  virtual ReturnValue onCreateUsers(const std::vector<User>&) = 0;
  virtual ReturnValue onDeleteUsers(const std::vector<std::string>&) = 0;
  virtual ReturnValue onEditUsers(const std::vector<User>&) = 0;
};
```

Ukázka kódu 4.16. Definice třídy *SecurityContext*

Zbylé třídy rozhraní knihovny, které jsou *DiscoveryContext*, *MediaContext*, *NetworkContext* a *SystemContext*, jsou implementovány stejným způsobem jako třída *SecurityContext*. Při integraci knihovny je nutné vytvořit potomky těchto tříd a doplnit implementaci virtuálních metod podle potřeb konkrétní aplikace.

V rámci knihovny je implementováno celkem 46 webových služeb z definice ONVIF profilu S. Celkem tento profil definuje 65 webových služeb, jejichž implementace je pro zařízení povinná. Služby které nejsou implementovány se týkají především hlášení událostí. Důvodem vynechání implementace těchto služeb bylo, že se jedná o rozsáhlou funkci, která se netýká streamování videa. Přehled implementovaných služeb se nachází v tabulce 4.1.

Služba	Implementováno	Služba	Implementováno
GetCapabilities	ano	PullMessages	ne
GetWsdlUrl	ano	GetEventProperties	ne
GetDiscoveryMode	ano	TopicFilter	ne
SetDiscoveryMode	ano	MessageContentFilter	ne
GetScopes	ano	GetProfiles	ano
SetScopes	ano	GetStreamUri	ano
AddScopes	ano	GetVideoEncoderConfiguration	ano
RemoveScopes	ano	GetVideoEncoderConfigurations	ano
GetHostname	ano	AddVideoEncoderConfiguration	ano
SetHostname	ano	RemoveVideoEncoderConfiguration	ano
GetDNS	ano	SetVideoEncoderConfiguration	ano
SetDNS	ano	GetCompatibleVideoEncoderConfigurations	ano
GetNetworkInterfaces	ano	GetVideoEncoderConfigurationOptions	ne
SetNetworkInterfaces	ano	GetGuaranteedNumberOfVideoEncoderInstances	ano
GetNetworkProtocols	ano	GetProfile	ano
SetNetworkProtocols	ano	CreateProfile	ano
GetNetworkDefaultGateway	ano	DeleteProfile	ano
SetNetworkDefaultGateway	ano	GetVideoSources	ano
GetDeviceInformation	ano	GetVideoSourceConfiguration	ano
GetSystemDateAndTime	ano	GetVideoSourceConfigurations	ano
SetSystemDateAndTime	ano	AddVideoSourceConfiguration	ano
SetSystemFactoryDefault	ano	RemoveVideoSourceConfiguration	ano
SystemReboot	ano	SetVideoSourceConfiguration	ano
GetUsers	ano	GetCompatibleVideoSourceConfigurations	ano
CreateUsers	ano	GetVideoSourceConfigurationOptions	ne
DeleteUsers	ano	GetMetadataConfiguration	ne
SetUser	ano	GetMetadataConfigurations	ne
Notify	ne	AddMetadataConfiguration	ne
Subscribe	ne	RemoveMetadataConfiguration	ne
Renew	ne	SetMetadataConfiguration	ne
Unsubscribe	ne	GetCompatibleMetadataConfigurations	ne
SetSynchronizationPoint	ne	GetMetadataConfigurationOptions	ne
CreatePullPointSubscription	ne		

Tabulka 4.1. Stav implementace jednotlivých služeb

4.3 Testovací aplikace

Posledním částí implementace je testovací aplikace, která poslouží zároveň k demonstraci použití vzniklé knihovny.Jak vyplývá z kapitoly 4.2, v rámci testovací aplikace bylo nutné doplnit implementace třídy *Context* a i další třídy zabývající se dílčími částmi rozhraní. Dále také bylo nutné doplnit některé další elementy, jako je databáze uživatelů či seznam dostupných konfigurací RTSP (Real-Time Streaming Protocol) streamu.

4.3.1 RTSP stream

Součástí implementace testovací aplikace je také testovací generátor obrazu a jeho streamování prostřednictvím protokolu RTSP. Pro tyto účely byla využita knihovna GStreamer[5], která nabízí integrovaný RTSP server. Navíc knihovna nabízí také integrovaný generátor testovacích obrazů a možnost čtení videa ze souborů. Implementace streamovacího serveru je ve třídě *RTPSServer*, jejíž veřejné rozhraní je v ukázce 4.17.

```
class RTSPServer
{
...
    bool start();
    void stop();
    void addMount(const std::string& mount, const std::string& launch);
...
```

Ukázka kódu 4.17. Definice třídy *RTSPServer*

Třída *RTSPServer* nabízí tři metody. Metody *start* a *stop* slouží pro spuštění, respektive zastavení, činnosti serveru. Pomocí metody *addMount* je možné přidat do serveru stream dat, například z generátoru testovacích obrazců.

V ukázce 4.18 je příklad užití třídy *RTSPServer* s generátorem testovacích obrazců a streamem videa ze souboru. Vzniknou takto dva streamy. Jeden stream bude na adrese *rtsp://<ip>:8554/file* vysílat video ze souboru. Druhý stream bude vysílat výstup generátoru obrazců na adrese *rtsp://<ip>:8554/generator*. Dále si v ukázce lze všimnout, že u generátoru obrazců je možné pomocí parametrů *width* a *height* nastavit výstupní rozlišení a dále pomocí parametru *framerate* nastavit výstupní frekvenci. Pro tyto dva streamy jsou v rámci aplikace definovány dva fixní profily.

```
RTSPServer rtsp;
rtsp.addMount("file", "(filesrc location=\"./file.mp4\" ! qtdemux name=d"
              " d. ! queue ! rtph264pay pt=96 name=pay0"
              " d. ! queue ! rtpmp4apay pt=97 name=pay1)");
rtsp.addMount("generator", "( videotestsrc pattern=ball "
              "! video/x-raw,width=3840,height=2160,framerate=30/1 "
              "! omxh264enc ! h264parse ! rtph264pay name=pay0 pt=96 )");
rtsp.start();
```

Ukázka kódu 4.18. Použití třídy *RTSPServer*

[5] https://gstreamer.freedesktop.org/

4.4 Vytvoření obrazu systému pro ZCU104

Ke spuštění testovací desky ZCU104 je nejdříve nutné vytvořit obraz systému pomocí nástrojů PetaLinux. Tyto nástroje umožňují systém široce konfigurovat. Podrobný popis konfigurace a vytvoření obrazu systému je v návodu[6] od firmy Xilinx.

Prvním krokem k vytvoření obrazu systému je založení projektu pomocí nástroje *petalinux-create*. Při zakládání projektu je možné šablonu, nebo nahrát *Board support package* pro konkrétní desku. Příklad volání nástroje je v ukázce 4.19.

```
petalinux-create -t project -s ./bsp/xilinx-zcu104.bsp -n name
```

Ukázka kódu 4.19. Použití nástroje *petalinux-create*

V případě, že má na desce být využita programovatelná logika, je dalším krokem použití nástroje *petalinux-config* s argumentem **–get-hardware-description**, pomocí kterého se do projektu přidá hardwarová konfigurace programovatelné logiky.

```
petalinux-config -c rootfs
```

Ukázka kódu 4.20. Použití nástroje *petalinux-config*

Nástroj *petalinux-config* je možné také využít ke konfiguraci adresářového systému, k čemuž slouží příkaz v ukázce 4.20. Lze takto například přidávat do obrazu systému knihovny nebo aplikace.

Po dokončení konfigurace obrazu systému již můžeme přejít k sestavení obrazu. Pro tento účel slouží nástroj *petalinux-build*. Sestavený obraz systému již stačí jen zabalit pomocí nástroje *petalinux-package* a může být nahrán na desku.

Aby bylo možné provést křížový překlad, je nutné vytvořit pomocí nástroje *petalinux-build* také SDK, ve kterém je obsaženo vše, co je zapotřebí pro kžížový překlad. V nakonfigurovaném projektu k tomu postačí volání nástroje *petalinux-build* s přepínačem *-s*.

4.5 Křížový překlad pro Zynq UltraScale+ MPSoC

Knihovna i testovací aplikace jsou překládány pomocí multiplatformního softwaru CMake[7]. Tento nástroj umožňuje pomocí vlastní syntaxe popsat, jakým způsobem má být překlad proveden, a následně tento popis převést do podoby, která může být použita systémovým překladačem, jako je například *make*. Příklad použití CMake z příkazové řádky pro sestavení projektu je v ukázce 4.21.

```
cmake -S <project_directory> --B <build_directory>
```

Ukázka kódu 4.21. Příklad překladu pomocí CMake

[6] https://docs.xilinx.com/r/en-US/ug1144-petalinux-tools-reference-guide
[7] https://cmake.org/

```
. <sdk_directory>/environment-setup-cortexa72-cortexa53-xilinx-linux
cmake -S <project_directory> --B build
```

Ukázka kódu 4.22. Křížový překlad s použitím skriptu na nastavení systémových proměnných

Pro křížový překlad je nutné pomocí systémových proměnných nastavit, aby byly pro překlad použity překladače obsažené v SDK vytvořeném nástroji PetaLinux. SDK obsahuje také skript, který zajistí potřebná nastavení systémových proměnných. Po nastavení systémových proměnných je možné provést překlad, jímž vznikne binární soubor spustitelný na cílové platformě.

Kapitola 5
Testování

Součástí implementace je bezpochyby také testování. Existuje mnoho metod na testování aplikací, ale jelikož se tato práce zabývá implementací síťového protokolu, je samotné testování poněkud komplikované. Pro testování byl zvolen následující scénář. Testovací aplikace zmíněná v kapitole 4.3 byla spuštěna na desce ZCU104. Deska byla přes switch pomocí ethernetového kabelu připojena k PC, ze kterého se prováděly testy.

Fórum ONVIF pro účely testování poskytuje testovací nástroje, které jsou však přístupné pouze členům fóra. Jelikož je specifikace standardu ONVIF veřejně dostupná, vznikl zde i veřejně dostupný nástroj ONVIF Device Manager[1]. Nejedná se však o testovací nástroj, a proto bohužel v případě chyby neumožňuje diagnostikovat, proč k chybě dochází. V takovém případě bylo nutné použít aplikaci pro analýzu provozu v síti, jako je například Wireshark[2].

5.1 Testování pomocí ONVIF Device Manager

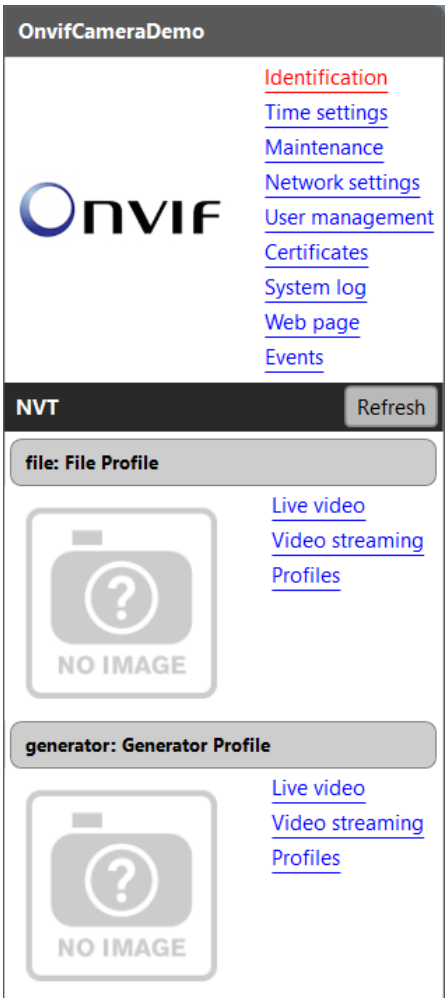
Aplikace ONVIF Device Manager neumožňuje ručně volat konkrétní webové služby, ale k jejich volání dochází na základě akcí uživatele v rámce aplikace. Pro otestování jednotlivých služeb tak bylo nutné také vypozorovat, kdy jsou volány jaké služby.

Obrázek 5.1. Hlavní okno programu ONVIF Device Manager

[1] https://sourceforge.net/projects/onvifdm/
[2] https://www.wireshark.org/

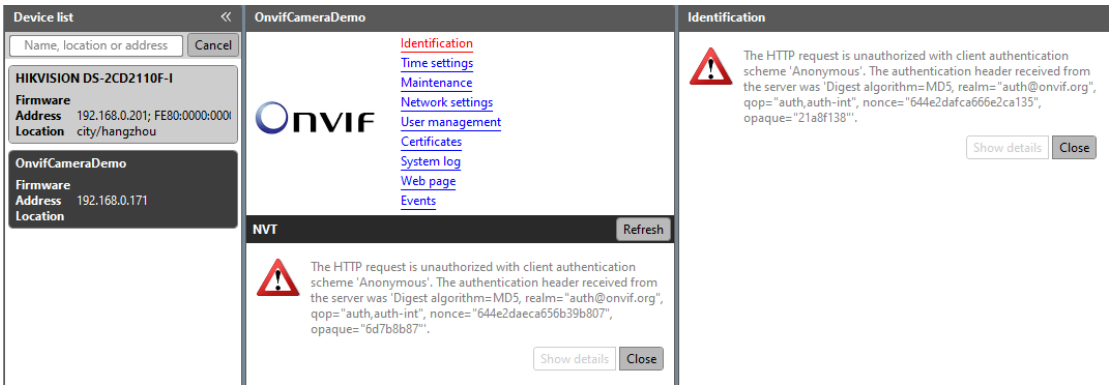
Na obrázku 5.1 vidíme okno aplikace ONVIF Device Manager. V levé části okna se nachází lišta obsahující seznam zařízení, kde se objevují zařízení automaticky vyhledaná pomocí protokolu WSDD, nebo zařízení přidaná manuálně. Popis jednotlivých položek je tvořen informacemi získanými pomocí WSDD a doplněn pomocí výsledků služeb *GetScopes* a *GetDeviceInformation*. Můžeme si povšimnout, že se zde nachází zařízení s názvem *OnvifCameraDemo*. Toto zařízení představuje testovací aplikaci spuštěnou na desce ZCU104.

Po zvolení zařízení v seznamu se objeví další panel viz obrázek 5.2. V horní části tohoto panelu se nalézá menu pro přístup k obecným nastavením zařízení. Pod tímto menu se může nacházet další menu, které však již obsahuje položky specifické pro konkrétní typ zařízení. V tomto případě se jedná o položky zaměřené na video, jelikož testovací aplikace simuluje funkci videokamery. Po kliknutí na některou z položek v menu se přes zbylou plochu objeví další panel, jehož obsah odpovídá položce z menu.



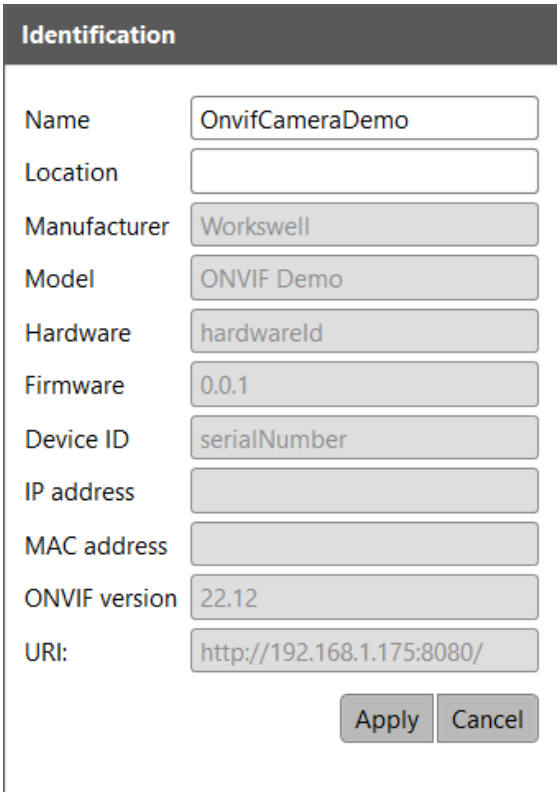
Obrázek 5.2. Menu zařízení OnvifCameraDemo

Nad seznamem zařízení se nalézají kolonky na jméno a heslo pro přihlášení, avšak k ověření údajů nedochází po stisknutí tlačítka *Log in*, ale během volání webových služeb při práci se zařízením. K ověření údajů uživatele je využíván pouze protokol WSS-UsernameToken. V případě, že není v aplikaci přihlášený uživatel anebo není uživatel v databázi na zařízení, objeví se v pravém panelu místo obsahu chybová hláška, kterou můžeme vidět na obrázku 5.3.



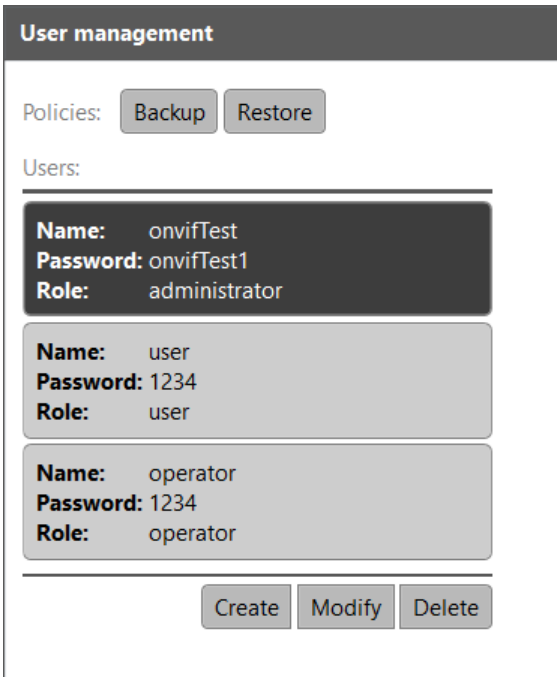
Obrázek 5.3. Chybová hláška v případě neautorizovaného přístupu

Na panelu *Identification* se nachází údaje o zařízení, což můžeme vidět na obrázku 5.4. Obsah by měl odpovídat informacím předaným webovou službou *GetDeviceInformation*.



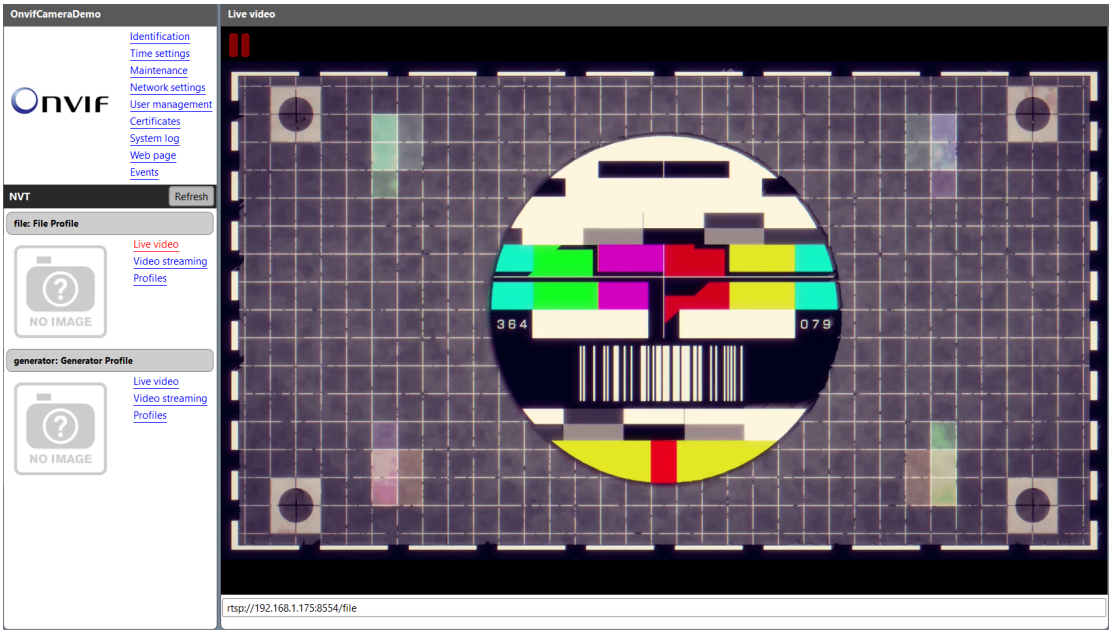
Obrázek 5.4. Panel identifikace zařízení

Panel *User management* obsahuje tabulku uživatelů, kterou můžeme vidět na obrázku 5.5. Obsah tabulky se načítá pomocí služby *GetUsers*. Pod tabulkou se nachází tlačítka *Create*, *Modify* a *Delete*, která v rámci své funkce volají služby *CreateUsers*, *SetUser* a *DeleteUser*.



Obrázek 5.5. Tabulka pro správu uživatelů

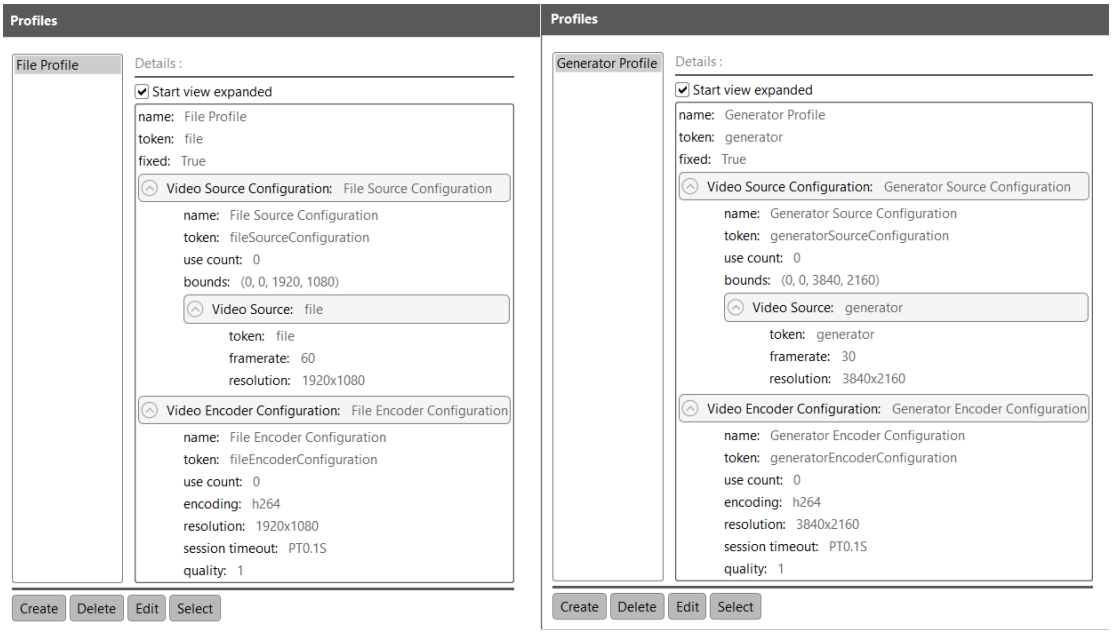
Pro každý zdroj videa jsou v menu položky *Live video*, *Video streaming* a *Profiles*. Pod možností *Live video* se nachází přehrávač RTSP streamu ze zařízení, který můžeme vidět v obrázku 5.6. Pro získání adresy streamu se využívá služba *GetStreamUri*.



Obrázek 5.6. Zobrazení streamu ze souboru

Pro přístup k nastavením enkodéru slouží panel *Video streaming*. Využívá se zde zejména služeb *GetVideoEncoderConfiguration* a *SetVideoEncoderConfiguration*, pomocí kterých je možné měnit parametry jako FPS (Frames Per Second), kvalitu videa, interval enkódování nebo bitrate.

Další konfiguraci nalezneme pod položkou *Profiles*. Zde je možné vytvářet a spravovat takzvané profily, k čemuž se využívají služby jako jsou například *GetProfiles*, *CreateProfile* nebo *DeleteProfile*. Profil obsahuje informace o tom, jakému zdroji videa přísluší, v jaké konfiguraci má být zdroj použit a nastavení enkodéru. V rámci testovací aplikace je pro každý zdroj videa definován jeden fixní profil, což můžeme vidět v ukázce 5.7.



Obrázek 5.7. Zobrazení profilů pro oba zdroje testovací aplikace

Kapitola 6
Závěr

V průběhu této bakalářské práce byla provedena analýza standardu ONVIF, která se zaměřovala na jeho popis a význam v kontextu bezpečnostních systémů. Na základě této analýzy byly identifikovány klíčové vlastnosti a funkce standardu ONVIF, které ho činí vhodným pro interoperabilitu mezi různými zařízeními v prostředí bezpečnostních systémů.

Druhá část práce byla věnována cílové platformě Zynq Ultrascale+ MPSoC. Byla zde podrobně představena její architektura. Důkladná analýza platformy umožnila identifikovat její přednosti a omezení, které byly důležité pro následnou implementaci standardu ONVIF.

Implementace standardu ONVIF byla hlavním zaměřením třetí části práce. Největším problémem zde byla implementace protokolu SOAP v jazyce C++, který pro něj na rozdíl od jiných jazyků, nemá nativní podporu. Tento problém se však povedlo vyřešit použitím knihovny gSOAP. Následně zde byli využity znalosti z první a druhé části k vlastní implementaci standardu ONVIF, která byla rozdělena na dvě části - knihovnu a testovací aplikaci. V rámci knihovny byl implementován server, který zprostředkovává komunikaci dle standardu ONVIF. Jelikož je však standard velmi rozsáhlý, pokrývá tato knihovna pouze jeho část. Testovací aplikace demonstruje použití knihovny a navíc je v rámci aplikace implementován RTSP server, který vysílá stream videa z generátoru obrazců. Dále se tato část zabývá také vytvořením obrazu systému pro testovací desku ZCU104 pomocí nástrojů PetaLinux a křížovým překladem pro cílovou platformu.

Poslední část práce se věnovala testování implementovaného standardu ONVIF. Pro testování byl použit program ONVIF Device manager, který slouží jako klient komunikující prostřednictvím standardu ONVIF. S pomocí tohoto programu byla otestována správná funkčnost jednotlivých služeb a jejich zabezpečení. Navíc bylo možné otestovat také funkčnost RTSP serveru a generátoru obrazců.

Výsledkem této práce je knihovna, která umožňuje jednoduchou integraci standardu ONVIF do systémů postavených na platformě Zynq Ultrascale+ MPSoC, ale i na jiných platformách. I když knihovna pokrývá pouze malou část standardu ONVIF, má tento výsledek potenciál přinést značné výhody v oblasti vývoje a implementace bezpečnostních systémů, kde standard ONVIF nachází široké uplatnění. V budoucnu by bylo vhodné pokračovat v dalším vývoji a optimalizaci této knihovny, aby byla její implementace efektivnější a pokrývala širší část standardu.

Literatura

[1] Vlk, Jakub. *Využití standardu ONVIF v kamerových systémech* [online]. [vid. 2022/06/29]. Dostupné na https://dspace.vsb.cz/handle/10084/93288.

[2] Christensen, Erik, Francisco Curbera, Greg Meredith a Sanjiva Weerawarana. *Web Services Description Language (WSDL) 1.1* [online]. [vid. 2023/04/03]. Dostupné na https://www.w3.org/TR/wsdl.html.

[3] Gudgin, Martin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar a Yves Lafon. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)* [online]. [vid. 2023/04/03]. Dostupné na https://www.w3.org/TR/soap12/.

[4] Nixon, Toby, Alain Regnier, Vipul Modi a Devon Kemp. *Web Services Dynamic Discovery (WS-Discovery) Version 1.1* [online]. [vid. 2023/04/04]. Dostupné na http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.html.

[5] Lawrence, Kelvin, Chris Kaler, Anthony Nadalin, Ronald Monzillo a Phillip Hallam-Baker. *Web Services Security UsernameToken Profile 1.1* [online]. [vid. 2023/04/07]. Dostupné na https://www.oasis-open.org/committees/download.php/13392/wss-v1.1-spec-pr-UsernameTokenProfile-01.htm.

[6] Hallam-Baker, Phillip M., Jeffery L. Hostetler, Scott D. Lawrence, Paul J. Leach, Ari Luotonen a Lawrence C. Stewart. *HTTP Authentication: Basic and Digest Access Authentication* [online]. [vid. 2023/04/07]. Dostupné na https://www.ietf.org/rfc/rfc2617.txt.

[7] Schulzrinne, Henning, Anup Rao a Robert Lanphier. *Real Time Streaming Protocol (RTSP)* [online]. [vid. 2023/04/21]. Dostupné na https://www.rfc-editor.org/rfc/rfc2326.html.

[8] *HTTP request methods* [online]. [vid. 2023/04/21]. Dostupné na https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.

[9] *ONVIF* [online]. [vid. 2022/06/29]. Dostupné na https://www.onvif.org/.

[10] Inc., Xilinx. *Zynq UltraScale+ MPSoC* [online]. [vid. 2023/04/03]. Dostupné na https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html.

[11] Xilinx, Inc. *Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit* [online]. [vid. 2023/05/09]. Dostupné na https://www.xilinx.com/products/boards-and-kits/zcu104.html.

[12] Xilinx, Inc. *PetaLinux SDK* [online]. [vid. 2023/03/12]. Dostupné na https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html.

[13] Genivia. *gSOAP User Guide* [online]. [vid. 2022/06/29]. Dostupné na https://www.genivia.com/doc/guide/html/index.html.

[14] *CMake Documentation* [online]. [vid. 2023/04/23]. Dostupné na `https://cmake.org/documentation/`.

[15] *GStreamer Documentation* [online]. [vid. 2023/05/11]. Dostupné na `https://gstreamer.freedesktop.org/documentation/?gi-language=c`.

Příloha A
Seznam použitých zkratek

APU	Application Processing Unit
CAN	Controller Area Network
CCI	Cache Coherent Interconnect
DDR4	Double Data Rate 4
eMMC	Embedded MultiMediaCard
ETM	Embedded Trace Macrocell
FPS	Frames Per Second
FPU	Floating Point Unit
GB	Gigabyte
GIC	Generic Interrupt Controller
GigE	Gigabit Ethernet
GPU	Graphics Processing Unit
HDMI	High-Definition Multimedia Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IoT	Internet of Things
IP	Internet Protocol
JTAG	Joint Test Action Group
MD5	Message Digest Algorithm 5
MMU	Memory Management Unit
MPSoC	Multiprocessor System on a Chip
MQTT	Message Queuing Telemetry Transport
ONVIF	Open Network Video Interface Forum
PCIe	Peripheral Component Interconnect Express
PTZ	Pan-Tilt-Zoom
RTPU	Real-Time Processing Unit
RTSP	Real-Time Streaming Protocil
SATA	Serial Advanced Technology Attachment
SCU	Snoop Control Unit
SD	Secure Digital
SDK	Software Development Kit
SHA1	Secure Hash Algorithm 1
SIMD	Single Instruction Multiple Data
SMMU	System Memory Management Unit
SOAP	Simple Object Access Protocol
SPI	Serial Peripheral Interface
TCM	Tightly-Coupled Memory
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
VFPU	Vector Floating Point Unit

WSDD	Web Service Dynamic Discovery
WSDL	Web Service Description Language
WSS-UsernameToken	Web Service Security Username Token
XML	Extensible Markup Language