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

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2015 **Radomír Lasota**

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

Zadání bakalářské práce

Student: **Radomír Lasota**

Studijní program: B2649 Elektrotechnika

Studijní obor: 2612R041 Řídicí a informační systémy

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: ATEsystem, s.r.o.
2. 2. Struktura závěrečné zprávy:
a. Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
b. Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
c. Zvolený postup řešení zadaných úkolů.
d. Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
e. Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
f. Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vedl odbornou praxi studenta.

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

Vedoucí bakalářské práce: **doc. Ing. Petr Bilík, Ph.D.**

Konzultant bakalářské práce: Ing. Michal Harhaj

Datum zadání: 01.09.2014
Datum odevzdání: 07.05.2015

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

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

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.
Dne: 6.5.2015

Radek Lastar

.....

Podpis

ATEsystem s.r.o.

Studentská 6202/17, Poruba

708 00, Ostrava 8

tel: +420 595 172 394

fax: +420 595 170 100

e-mail: atesystem@atesystem.cz

Radomír Lasota

Vendryně 752

739 94, Vendryně

Souhlasíme se zveřejněním bakalářské práce na téma „Absolvování individuální odborné praxe“ dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

Datum: 7. května 2015

Ing. Michal Harhaj

jednatel

ATEsystem s.r.o.
Studentská 6202/17
Ostrava-Poruba
708 00
IČ: 01392042 DIČ: CZ01392042

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce **doc. Ing. Petru Bilíkovi, Ph.D. a pracovníkům společnosti ATEsystem s.r.o.** za odbornou pomoc a konzultaci při vytváření této práce.

Abstrakt

Náplní této bakalářské praxe byla práce ve firmě ATEsystem s.r.o. Hlavním zaměřením byla práce v návrhovém prostření LabVIEW. Další prací v menší míře byla práce na přípravě PC pro zákazníka. Hlavními cíli v oblasti programování v LabVIEW bylo prozkoumání, pochopení a realizace programu, zaměřující se na webové služby a jejich využití ve spojení s prostředím UI Builder. V tomto prostředí byla vytvořena ukázková aplikace, která po navolení parametrů z webové stránky odešle data ke zpracování na serverovou aplikaci. Zpět jsou poslány výsledky a zobrazeny v grafu. Byl kladen důraz na zjištění maximálního objemu dat, který je možné přenést. Další zkoumanou oblastí byla realizace programu pro ztrátovou kompresi obrazu pomocí grafické karty s technologií CUDA. Pro vyzkoušení byl vytvořen program pro ověření práce funkce Call Library Function Node. Nedílnou součástí byla úprava knihovny GPUJPEG, vytvořená společností CESNET, z. s. p. o. Z časových důvodů a v důsledku možné chyby ve veřejně dostupné knihovně nebyla aplikace pro kompresi obrazu zcela dokončena.

Klíčová slova

LabVIEW, Webové služby, UI Builder, CUDA, GPU, ztrátová komprese, JPEG

Abstract

Center of this bachelor thesis was the individual professional practice in the ATEsystem s.r.o. with major focus on LabVIEW system design software. Minor part was working on PC for customer. The main objective of programming in the LabVIEW had two parts. The first one was researching, understanding the issues and realization the program using web services and User Interface Builder environment. In this environment was built demo application, which is able to send data to server application after selecting the options. Processed data are sent back and being viewed in the graph. The main objective was to find out how much data is possible to send. The second objective was making a program for lossy image compression using graphic card with CUDA. It was created a demo program for demonstration of functionality of the Call Library Function Node. An integral part of the adjustment library GPUJPEG developed by CESNET, z. s. p. o. Application for image compression was not completed because of lack of time and because of possible bug in the public available library.

Key words

LabVIEW, Web services, UI Builder, CUDA, GPU, lossy compression, JPEG

Seznam použitých symbolů a zkratek

LabVIEW	Laboratory Virtual Instruments Engineering Bench
UI Builder	User Interface Builder
NI	National Instruments
VI	Virtual Instrument
HTTP	Hypertext Transfer Protocol
GUI	Grafické uživatelské rozhraní
CPU	Central Processing Unit
GPU	Graphic Processing Unit
GPUJPEG	Knihovna určená pro kompresi a dekompresi obrazu s použitím grafické karty
VS	Visual Studio
HD	High Definition
4K	Standard pro rozlišení obrazu

Obsah

1. Úvod

Hlavním zaměřením činností ve společnosti ATEsystem s.r.o. je práce v software LabVIEW za účelem návrhu aplikací pro inspekci zboží a výrobků pomocí kamerových systémů, instalace na místě provozu a různé drobnější úkony, související s funkčností a údržbou těchto stanovišť. Obsahem zde popisované praxe bylo programování ve vývojovém prostředí LabVIEW a příprava PC použitého v průmyslovém testeru.

2. Seznam vykonávaných činností

2.1. Příprava průmyslového PC

Úkolem byla příprava PC po softwarové stránce před tím, než mělo být nasazeno k použití. Bylo zapotřebí provést požadované nastavení operačního systému. Dále instalace potřebného software, nezbytného k chodu a požadavkům pracoviště. Jedním z požadavků byla vytvoření obrazu systémového disku a následné uchování na serverovém úložišti spolu s dokumentací SW a HW. Posledním krokem bylo provedení zátěžového testu, zdali je PC schopno trvale pracovat při plném zatížení.

Požadované úkony:

- Instalace zákazníkem požadovaných programů, a nastavení licenčních klíčů
- Nastavení samotného operačního systému Windows 7
- Vytvoření obrazu systémového disku a následná záloha na server
- Pořízení dokumentace SW a HW

2.2. Programování ve vývojovém prostředí LabVIEW společnosti National Instruments

2.2.1. Webové služby a UI Builder

Zadáním bylo prozkoumat funkce Webových služeb v LabVIEW a jejich možné využití v případě vytvořených webových stránek nástrojem UI Builder. Dále zjistit, zdali a jak funguje zobrazování dat v standardním internetovém prohlížeči, pakliže jsou dané webové stránky vytvořeny v nástroji User Interface Builder a jak velké množství dat je možné přenést a zobrazit.

2.2.2. Komprese obrazu pomocí GPU

Požadavkem bylo vytvořit program v LabVIEW, který bude komprimovat obraz pomocí grafické karty. Obraz bude snímán průmyslovou kamerou a díky vysokému výkonu grafické karty NVIDIA bude prováděna ztrátová komprese obrazu, například do formátu JPEG.

3. Webové služby

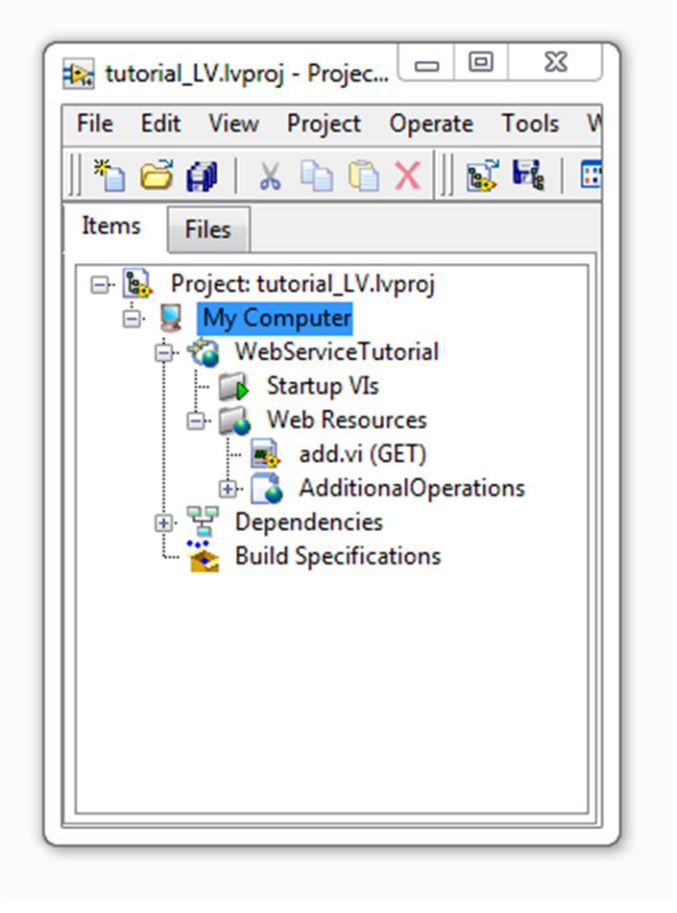
Při programování šlo o to, prozkoumat a vhodně využít webové služby LabVIEW. Jedním z dlouhodobějších cílů při této praxi bylo programování aplikací, pomocí nichž dokážeme přistupovat k databázovým údajům po síti. K tomu je možné využít široké škály možností, avšak zde byl kladen požadavek na využití webového nástroje UI Builder. Idea je taková, že se jednoduše odkudkoliv může přistupovat k vytvořené aplikaci pomocí webového prohlížeče. K tomu se snažíme dojít, skrze několik menších úkolů, které s touto problematickou úzce souvisí. Prvním z nich bylo pochopení takzvaných webových služeb.

3.1. Webové služby obecně

Síla tohoto nástroje spočívá v univerzálnosti. Jde o to, že klientská aplikace může přistupovat k datům pomocí samostatně běžící aplikace pomocí sítě. To je možné ať už po vnitřní oddělené síti na provoze, nebo přes půl světa po internetové síti. Na straně serveru běží spuštěná aplikace – webová služba. Ta může obsahovat i větší množství VI, proměnných a dalších souborů, které zajišťují chod aplikace. Komunikačním protokolem je velmi běžné a známé HTTP.
Možné využití služeb: uživatel může vzdáleně přistupovat, ovládat a nastavovat své aplikace a přístroje (které jsou v této službě zahrnuty) velmi jednoduše, bez nutnosti instalace velkých vývojových prostředí a balíků. Stačí k tomu pouze standardní webový prohlížeč, podporující HTTP protokol. Aplikace mohou komunikovat prostřednictvím metod POST, GET, nebo Stream. Platí zde jedno omezení, a to, že při spuštění aplikace přes webový prohlížeč se stáhne zásuvný modul (plugin) Microsoft SilverLight. Tento nástroj je pro přistupování k aplikacím přes webový prohlížeč nutný – to s sebou nese omezení použití pouze na operační systém Microsoft Windows. Je možné spustit SilverLight i na jiných operačních systémech, ale není zde zcela zaručen bezchybný a spolehlivý chod. Tím není řečeno, že by to vůbec nebylo možné, jen je zde menší šance na optimalizaci a správnou činnost. Obecně vzato, prohlížeč, na který je LabVIEW a jejich služby odladěn, je MS Internet Explorer [1] [2].

3.2. Postup zprovoznění webové služby

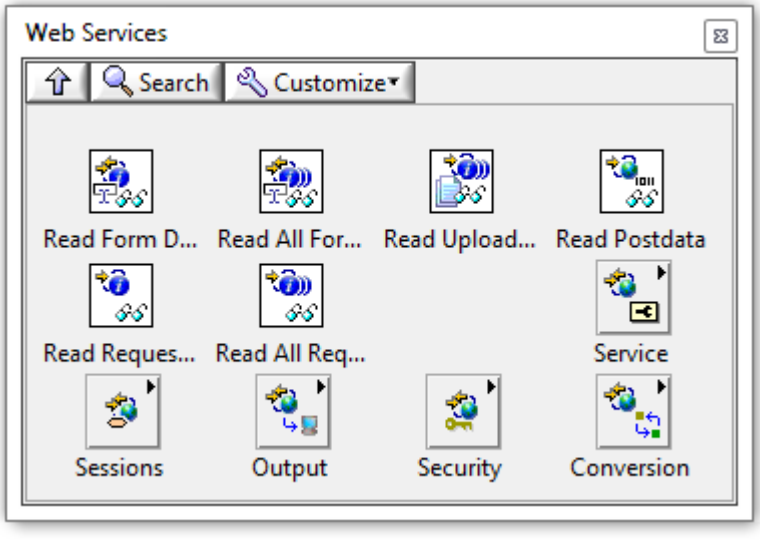
V okně projektu v LabVIEW se vloží novou webovou službu (New – Web Service). Do podsložky Web Resources se vloží nové VI (klikneme pravým tlačítkem na Web Resources – New VI). Pokud se bude využívat využívat Globální proměnné, v okně projektu se automaticky vytvoří podsložky s příponou nazev.lvlib., přičemž název knihovny lze nastavit [3].



Obr. 1 - Okno projektu

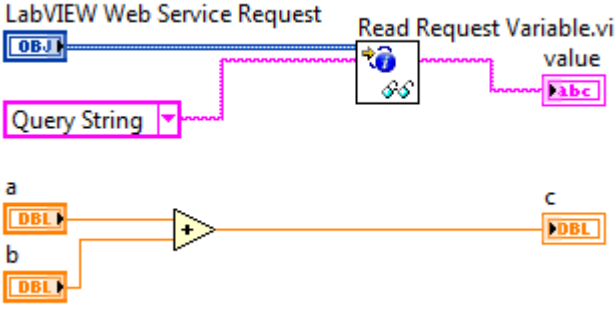
Na čelní panel se umisťují prvky, které mají být zobrazeny v naší aplikaci. Blokový diagram, který bude vykonávat, co klient potřebuje, musí být doplněn o funkce, umožňující fungování webové služby. K tomu slouží paleta Web Services, která je podskupinou nabídky Connectivity. Jsou zde funkce pro vyčtení konkrétních dat, vyčtení všech dat, funkce pro vyčtení konkrétní proměnné, nebo všech, dále funkce pro získání statusu, nebo možnost šifrování a dešifrování přenášených dat. V tomto případě bude použita ikona Read Request Variable. Ke konektoru LabVIEW Web Service Request se připojí stejnojmenný prvek, který se již vytvořil s novým VI. Ke konektoru Variable name je třeba vložit konstantu, a v nabídce

navolit možnost Query String. Tato část zajistí, že webová služby bude publikovat všechny hodnoty, které budou na čelním panelu připojeny ke konektoru.



Obr. 2 - Paleta Web Services

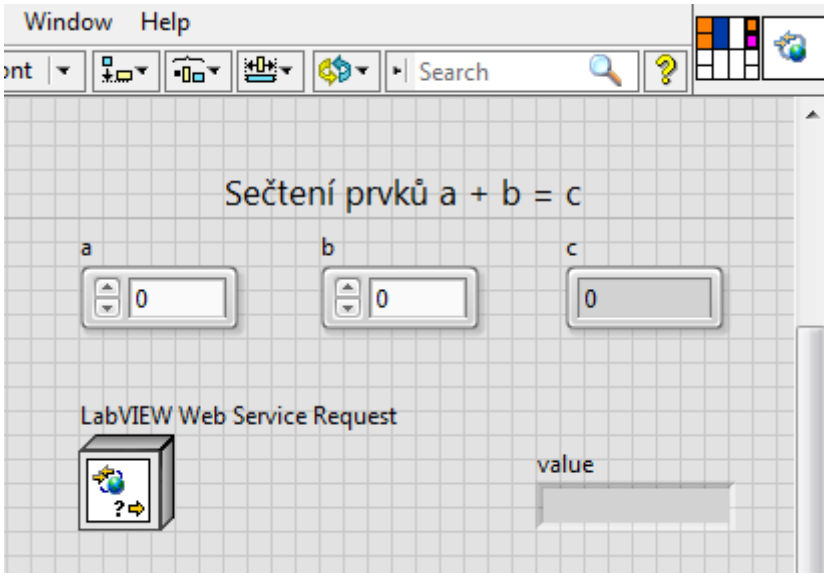
Nyní je třeba v blokovém diagramu sestavit program, který bude vykonávat, co požadujeme. Pro příklad je zde ukázka nejjednoduššího programu, který se postará o sečtení dvou čísel.



Obr. 3 - Ukázka webové služby

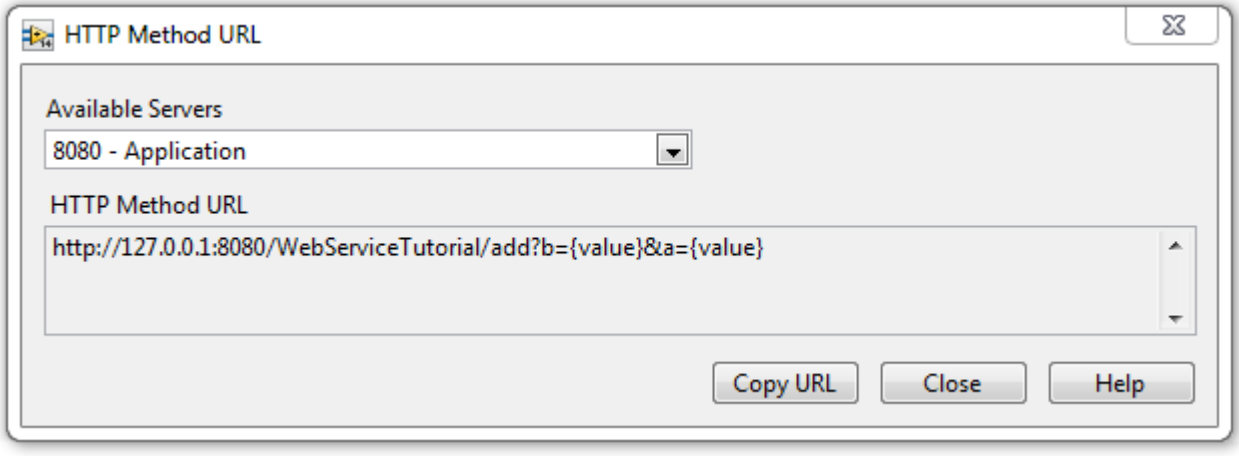
Jak už bylo výše zmíněno, platí zde jedna zásada – vše co má být publikováno musí být připojeno ke konektoru na čelním panelu. To se provádí při vybraném nástroji: Tools – Connect Wire tak, že se vybere prvek, jež se má připojit, a pak se klikne na daný volný slot konektoru. I zde je doporučeno pro přehlednost dodržovat zásadu, že vstupy se připojují nalevo, výstupy pak na pravou stranu. Pro chod musí být rovněž připojen i prvek LabVIEW Web Service Request. Při sestavování prvků na blokovém diagramu není zapotřebí starat se o rozložení, velikost a tvar jednotlivých prvků, protože to se musí nastavit v nástroji UI Builder. Zde se musí dbát pouze jednoho, a to, že název proměnné nesmí

obsahovat české znaky, mezery a jiné speciální znaky, které by byly v rozporu s fungováním samotného protokolu HTTP (jako například znak lomítka). Pokud je v názvu proměnné zapomenut nepovolený znak, aplikace při spuštění skončí chybou 67301 – Invalid Request Variable.



Obr. 4 - Rozložení čelního panelu a konektoru

Když jsou prvky umístěny na čelním panelu, a připojeny ke konektoru, je možné publikovat danou webovou službu. V okně projektu, se pravým tlačítkem klikne na webovou službu, a v možnosti Application Web Server se zvolí Publish. Po publikování se v PC vytvoří proces, který bude službu zajišťovat. O běhu aplikace se lze přesvědčit, když se zvolí Application Web Server – Manage Web Server. Otevře se okno webového prohlížeče, kde je možné nastavit některé důležité parametry. Prvním z nich je, v záložce Application Web Server HTTP port, na kterém bude aplikace komunikovat – zde se nechává port 8080. Důležité je však správně nastavit verzi 32/64bit. Pokud se nenastaví správná verze šířky sběrnice (v mém případě 64-bit), aplikace nebude pracovat. Ve čtvrté záložce jsou vypsány aktivní webové služby, na daném PC. Pokud tam není ta, kterou jsme právě vytvořili, je třeba znovu zkontrolovat nastavení a zkusit znovu publikovat službu. Pokud je aktivní, můžeme okno zavřít.
Pokud se v okně projektu klikne pravým tlačítkem myši na VI, které obsahuje webovou službu (v tomto případě add.vi(GET) – Show Method URL. Ukáže se následující okno:



Obr. 5 - výstup URL

Na výběr jsou dvě možnosti Available Servers: 8080 – Application a 8001 – Local Debugging. Pokud je nechán nastavený port 8001, nebude se využívat port, který je pro to určen, ale jiný, který je určen pro odladění aplikace. Proto se nastaví port 8080. (Pro srovnání, při prohlížení běžných webových stránek je využíván port 80). Níže je řádek HTTP Method URL. Po kliknutí na tlačítko Copy URL se uloží se aktuální URL adresa do schránky. Ze schránky pak daný řádek:
`http://127.0.0.1:8080/WebServiceTutorial/add?b={value}&a={value}`
vložit do internetového prohlížeče. Než se však navigaci potvrdí, je potřeba za volitelné parametry nastavit konkrétní hodnoty. Při této aplikaci si lze ověřit funkčnost na té nejnižší úrovni. Vysvětlení parametrů:

Tab. 1 - Význam položek URL adresy

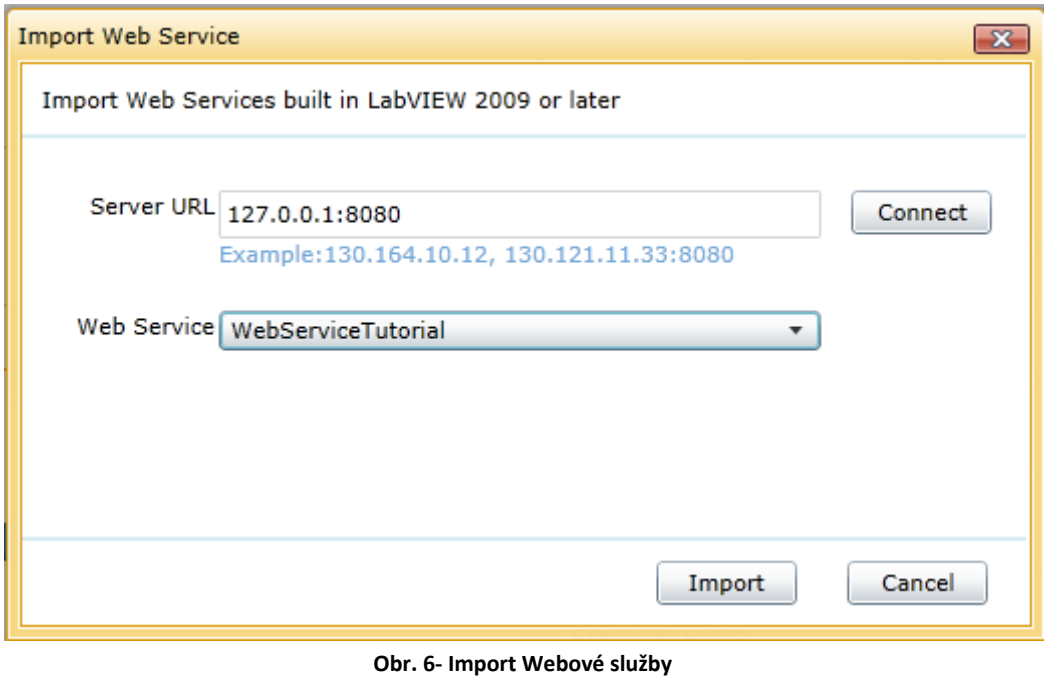
`http`	Použitý protokol
`127.0.0.1:8080`	IP adresa a číslo portu
`WebServiceTutorial`	Název projektu
`add`	VI s webovou službou
`b={value}&a={value}`	Proměnné

Místo textu *{value}* se vkládá libovolná povolená hodnota, která poslouží jako vstup proměnné. Pro ukázku jsem vložil za b číslo 5 a za a číslo 7. URL pak bude vypadat takto:
`http://127.0.0.1:8080/WebServiceTutorial/add?b=5&a=3`
Výsledkem bude výstup ve formátu XML, čili textová forma:
`<Response>`

```
<Terminal>
<Name>value</Name>
<Value>b=5&a=3</Value>
</Terminal>
<Terminal>
<Name>c</Name>
<Value>8</Value>
</Terminal>
</Response>
```

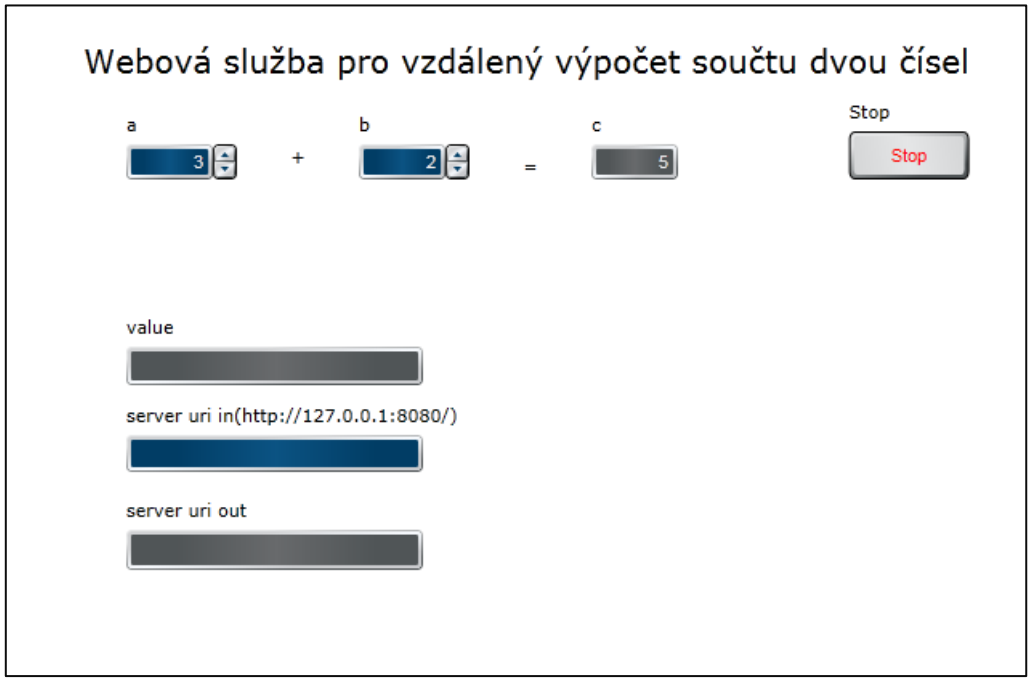
V tomto výpisu je vidět, že proměnná c, tedy výsledek je roven 8. Webová služba funguje a je možné přejít k práci v nástroji User Interface Builder. Ten je přístupný z webových stránek National Instruments pod adresou: `http://uibuilder.niwsc.com/webuibuilder/editor.htm`.

V okně projektu se zvolí Import Web Service. Zde se importuje publikovaná webová služba do projektu. Do položky Server URL je třeba vložit IP adresu a číslo portu ve tvaru, jak je vidět na níže uvedeném obrázku. Po kliknutí na tlačítko Connect se vyhledají webové služby na dané adrese a vybere se tu správnou, v tomto případě WebServiceTutorial. Po stisku tlačítka Import se vloží do projektu.



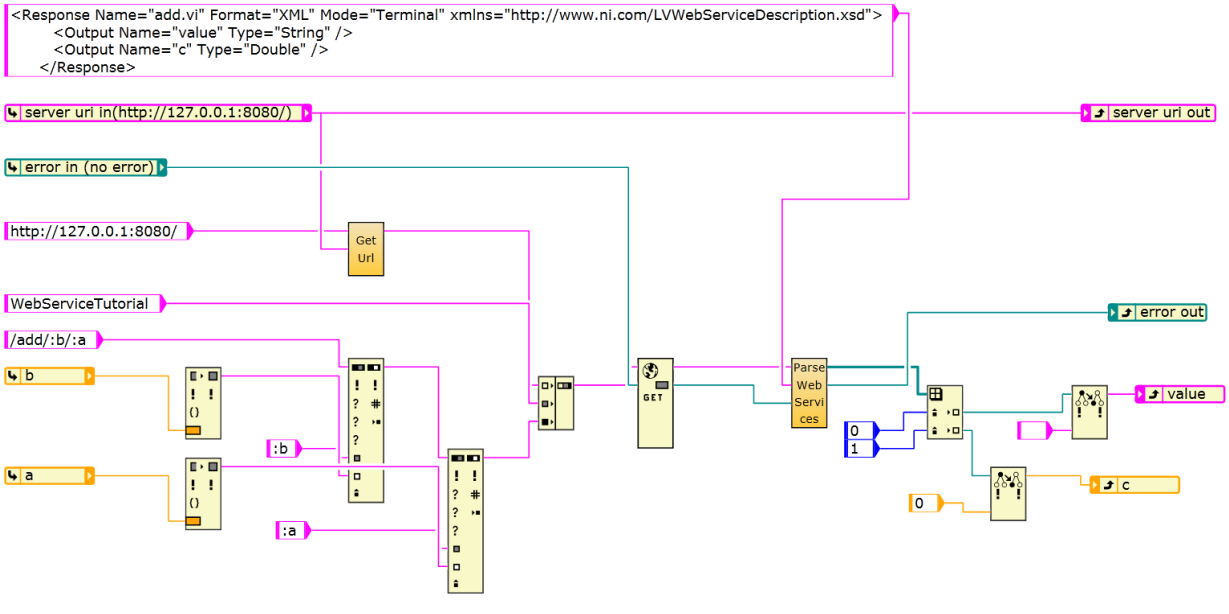
Obr. 6- Import Webové služby

V okně dole vlevo v záložce Project se vybere Vi s názvem add.vix. Nyní je třeba sestavit čelní panel. K připojení jsou zde dva vstupy: a, b a výstup c. Dále se můžou zobrazit i jiné položky jako value, server uri in a server uri out. Ty jsou nutné pro chod aplikace, ale v tomto případě nebudou využity. Základní popis prostředí je uveden v kapitole 3.3.



Obr. 7 - Ukázka čelního panelu

Druhá část programu je pak blokové schéma. Tam je podpora a funkce pro fungování webové služby. V tomto případě tam není tunto nic doplňovat. Pokud je požadováno, aby program běžel stále, umístí vše z blokového diagramu do nekonečné smyčky while.



Obr. 8 - Ukázka blokového diagramu

Když je aplikace spuštěna pomocí tlačítka Run, webová služba pošle požadavek pomocí protokolu HTTP přes port 8080 metodou GET na server. Ten zpracuje požadavek a pošle zpět výsledek. V okně je zobrazen výsledek sčítání.

3.3. Popis nástroje UI Builder

Je to nástroj, vyvinutý společností National Instruments pro naprogramování aplikace, sloužící k publikaci uživatelských dat. Jeho nespornou výhodou je fakt, že je celý přístupný skrz prostředí běžného internetového prohlížeče. Platí zde však omezení na operační systém Microsoft Windows, jelikož podmínkou pro chod je přítomnost volně stažitelného zásuvného modulu SilverLight. Hlavní myšlenka je v univerzálnosti spuštění – pro chod stačí pouze webový prohlížeč [4] [5].

Prostředí k programování aplikace je velmi podobné tomu, na co je uživatel zvyklý u klasické vývojářské verze LabVIEW. Má jednotný moderní design, který je podobný stylu Silver v LabVIEW 2014. Jelikož je však cílen jako prostředí GUI, je oproti plné verzi LabVIEW zjednodušen. Nepodporuje klávesové zkratky, a na čelním panelu chybí některé prvky.

Tab. 2 - LabVIEW a nástroje UI Builder

Funkce	LabVIEW	UI Builder
For loop	✓	✓
While loop	✓	✓
Case Structure	✓	✓
Flat Sequence	✓	✓
Timed Structures	✓	
Formula Node	✓	
Diagram Disable Structure	✓	
Event Structure	✓	

Dále UI Builder neobsahuje datové typy Cluster a speciální datové typy mezi grafy, jako např. výstupy expresních funkcí, TDMS (speciální formát dat LabVIEW). Neumožňuje Breakpointy a sondy (nástroje pro ladění programu v LabVIEW), ale jinak je velmi podobný LabVIEW.
Obsahuje čelní panel a blokový diagram. Design a rozvržení blokového diagramu se již na první pohled liší. Po importování webové služby obsahuje prvky nutné pro běh webové služby. Princip však zůstává stejný.
Dle druhu přenášených dat se volí způsob přenosu. Pro tyto potřeby jsou vyčleněny dvě hlavní skupiny.

Buffer interfaces je určen pro přenos větších objemů dat. Jeho výhodou je, že se postará o bezztrátový přenos všech hodnot. Je tedy určen například pro přenos všech naměřených hodnot. Dále je touto metodou možné poslat například i obrázek, nebo celé video.

Variable interfaces – používá se pro malé objemy dat. Jsou určeny pro přenos několika desítek (až stovek) hodnot, např. informace o stavu, zapnuto/vypnuto, nastavení několika parametrů. Jeho nevýhodou je fakt, že při přetížení sítě, nebo při zahlcené frontě dat může některé hodnoty zahodit.

3.4. Ukázka vzdáleného generátoru signálu

Jako výstupní aplikací pro vyzkoušení fungování Webových služeb a prostředí UI Builder byla vytvořena aplikace s názvem Generátor signálu pomocí Webové služby. Ukázka programu v LabVIEW, čelního panelu z prostředí UI Builder a blokové schéma jsou uvedeny v přílohách: Příloha 1, příloha 2 a příloha 3.

V internetovém prohlížeči se načte stránka, vygenerovaná UI Builderem. Na ní uživatel navolí čtyři základní parametry – Počet vzorků, který se má vygenerovat, frekvence signálu, tvar signálu, kdy je možno volit mezi sinusovým, pilovým a obdélníkovým signálem, a poslední možností je amplituda.

Data se odešlou na serverovou část, v tomto případě je to PC, kde je spuštěná daná webová služba v LabVIEW. Tam se data zpracují, vypočte se výstupní pole hodnot a to je posláno prostřednictvím Variable interface a portu 8080 zpět do webové stránky, vygenerované UI Builderem. Na grafu se zobrazí požadovaný průběh.

Vzhledem k velkému počtu hodnot, které se posílají, tím je na mysli pole vypočtených hodnot, to není typické užití, na které bylo toto prostředí navrženo, ale aplikace sloužila k vyzkoušení, jak si poradí právě s větším balíkem dat. Hranice velikosti pole byla odhadnuta až na velikost 100 000 prvků. Vzhledem k použitému datovému typu double, je celkový objem dat roven přibližně 800kB. To je i na samotné webové stránky hodně. Doba přenosu mezi serverem a webovou stránkou se pak pohybovala přibližně v řádu desítek sekund, což už je téměř nepoužitelné.

Správnější použití by samozřejmě bylo počet posílaných dat zmenšit na nutné minimum, které by zajistilo použitelné zobrazení v grafu. Je vhodné zmínit, že během programování nebylo nutné dělat žádné zásahy do samotného blokového diagramu v nástroji UI Builder, práce se sestávala pouze v navržení čelního panelu. Schéma v LabVIEW, schéma z prostředí UI Builder a čelní panel z UI Builder jsou uvedeny v přílohách.

3.5. Zhodnocení

Po naprogramování aplikací, jež ukazují jednoduchý součet čísel, a vzdálený generátor čísel lze tvrdit, že systém webových služeb a hlavně nástroj UI Builder je použitelný pro mnoho aplikací. Výpočty náročnější na výkon jsou prováděny na serveru, takže uživatelský počítač není zatěžován. Tento systém Variable interfaces však má své limity, a to hlavně v maximálním počtu posílaných dat. V aplikaci generátoru signálu došlo k netypickému využití tohoto rozhraní. Pro jednodušší vizualizaci nebude nastávat problém, pokud se bude posílat pouze redukované množství dat, potřebné k vizualizaci.

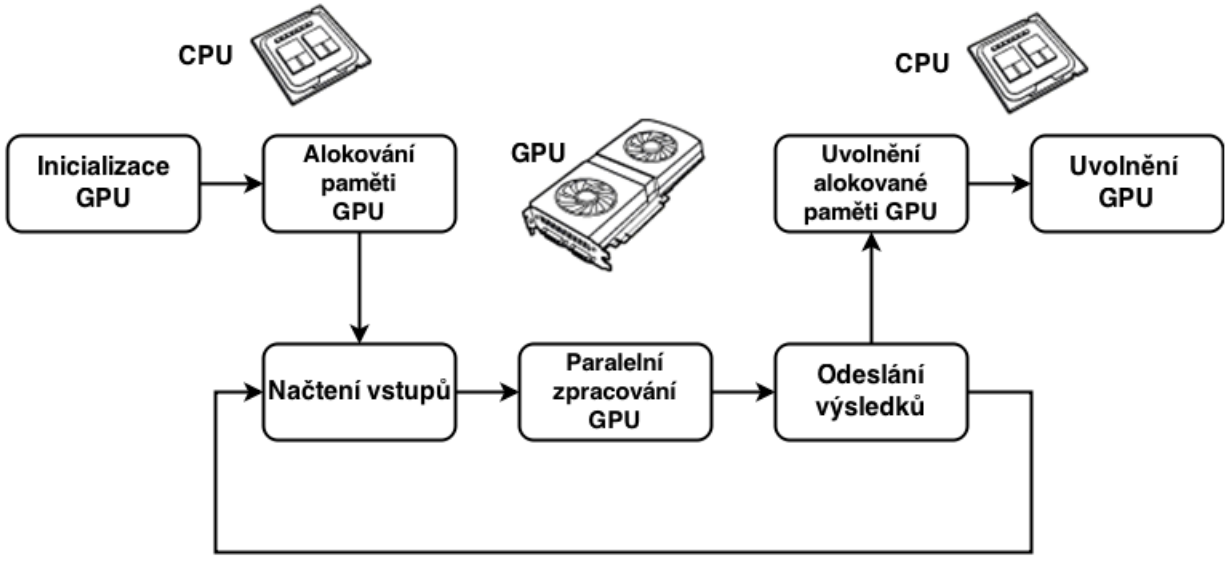
4. Komprese obrazu pomocí GPU v LabVIEW

4.1. Zadání

Další částí zadání během mé bakalářské praxe bylo využívat grafickou kartu počítače ke ztrátové kompresi obrazu.
Z vysokorychlostní kamery se stahují surová data (jednotlivé snímky), a ty se budou komprimovat se ztrátovou kompresí, například do formátu JPG díky vysokému výkonu grafické karty. Jelikož celá aplikace poběží přes LabVIEW, bylo velmi vhodné vybrat grafickou kartu od výrobce NVIDIA, podporující technologii CUDA, jelikož samo LabVIEW má pro toto rozhraní podporu.

4.2. Princip využití grafické karty

Obecný princip využití grafické karty je takový, že se hlavní část výpočtů převede z procesoru na grafickou kartu. Nejprve se alokují prostředky pro grafickou kartu, pak se do grafické karty načtou data pro výpočty. Spustí se výpočty na GPU. Grafická karta má oproti procesoru tu výhodu, že disponuje mnohem větším počtem paralelních vláken. Po vypočítání načteného balíku dat předá GPU výsledky, a buďto se načtou další data ke zpracování, nebo se uvolní prostředky GPU a proces je ukončen [6].



Obr. 9 - Schéma využití grafické karty

4.3. Důvod využití grafické karty

Důvodem využívání GPU je samozřejmě výkon, respektive čas, za který se provedou požadované výpočty. Grafická karty má oproti klasickým procesorům výhodu v mnohem větším počtu paralelních vláken, a taky se obvykle vykazují větší datovou propustností a větší pamětí. Běžný počet jader procesoru dosahuje v lepším případě čísla 8, obvykle jsou však plně samostatná paralelní vlákna pouze čtyři. Využívá se technologie Hyper-Threading, avšak vzhledem k nutnosti sdílet některé systémové prostředky se nedosahuje očekávaného dvojnásobného výkonu. Naproti tomu, grafické karty jsou sice principielně jednodušší, ale zato obsahují mnohem většího počtu jader, čili samostatných paralelních jednotek. U dnešních GPU se pohybujeme okolo stovek paralelních jader.
Pokud splníme dvě zásadní podmínky, že úlohu pro výpočty lze počítat nezávisle, a zároveň se jedná o tutéž operaci, jen mnohokrát opakovanou, dosáhneme díky využití GPU mnohem rychlejšího zpracování [7].
Přesně tímto případem je ztrátová komprese obrazu. Proto byl zadán požadavek právě na zpracování pomocí GPU.

4.4. Využití GPU v LabVIEW

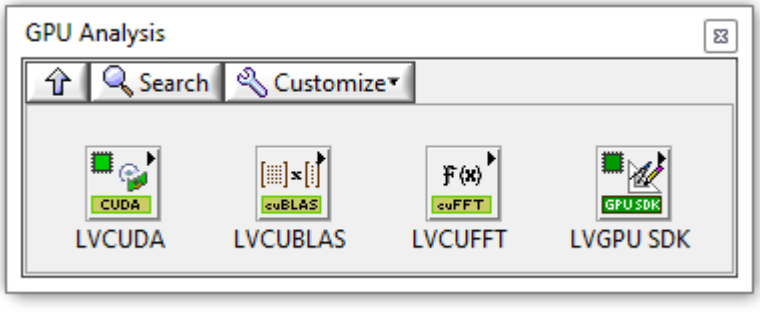
Aby bylo možné využívat nástroje pro práci s grafickou kartou, je nutné stáhnout ze stránek National Instruments balík pro LabVIEW, nesoucí název NI LabVIEW GPU Analysis Toolkit [8]. Poté se zpřístupní v High Performance Analysis – GPU Analysis čtyři palety:

LVCUDA - Obsahuje funkce, jež přímo korespondují s režií obsluhy a využívání GPU. Umožňuje práci se zařízeními, jež podporují NVIDIA CUDA Toolkit interface. Jsou vhodné, pokud chceme využívat grafickou kartu s technologií CUDA přímo prostřednictvím LabVIEW.

LVCUBLAS - Basic Linear Algebra Subprograms, čili podprogramy pro výpočty základní lineární algebry. Představují základní kameny pro operace se skaláry, vektory a maticemi. Umožňuje práci pouze se zařízeními s technologií CUDA.

LVCUFFT – funkce pro výpočty Rychlé Fourieriho transformace

LVGPU SDK – balík dalších funkcí pro operace s PGU jako alokace paměti, čtení dat a nastavování parametrů



Obr. 10 - Paleta nástrojů GPU Analysis

4.5. Ztrátová komprese obrazu

Ztrátová komprese (nejen obrazu) se používá všude tam, kde se třeba snížit objem užitečných dat a zároveň není nutné z komprimovaných dat absolutně přesně rekonstruovat původní informace. Jako příklad, ztrátová komprese u textového dokumentu by se nedala použít, jelikož by se tím mohla zásadně změnit textová část a tím i celá vypovídací hodnota dokumentu. Tímto omezením nejsme vázáni nejen například u digitálních fotografií, ale i u hudby a videí [9].

Posloupnost operací při ztrátové kompresi do formátu JPEG:

- Transformace barev např. z RGB do barevného prostoru YC_bC_r (tato část je bezztrátová)
- Podvzorkování barevných složek C_b a C_r, beze změny zůstane pouze složka Y (jasová složka)
- Aplikuje se diskrétní kosinová transformace na bloky 8×8 hodnot
- Bloky jsou kvantovány. Jelikož mnoho hodnot bývá nulových, dochází k velkému snížení datového toku

- Kvantované koeficienty jsou kódovány aritmetickým, nebo Huffmanovým kódováním. Huffmanovo kódování je přibližně o 10% méně účinné, ale je méně náročné a proto se více vyžívá
- Uložení dat např. do formátu JPEG [10].

4.6. Analýza současných řešení dané problematiky

Samotná problematika ztrátové komprese obrazu např. do formátu jpg je velice rozsáhlá. Prostředí LabVIEW obsahuje nástroj pro jednoduchou kompresi obrazu pro ukládání, ale tento nástroj není vhodný pro řešení této aplikace pro svou nemožnost detailního nastavení. Tvorba celého programu čistě z univerzálních bloků, jež jsou obsaženy v GPU Analysis Toolkit by bylo velmi náročné a přesáhlo by rozsah tvorby bakalářské práce. Proto byla analyzována různá řešení, která se touto problematikou zabývají. Jeden ze zdrojů různých řešení je na stránkách NVIDIA [11], odkud je možné stáhnout různé projekty. Ani jeden z uvedených zdrojových kódů však nebyl příliš použitelný pro kompresi obrazu. Další možností, která byla zvolena je projekt uvedený na stránkách sourceforge.net [12], s názvem GPUJPEG. Jedná se o knihovnu, určenou pro kompresi a dekompresi obrazu s vysokým rozlišením s využitím grafické karty. Knihovna byla vyvinuta společností CESNET, z. s. p. o. [13], a vztahuje se na ní BSD 2 licence [14].

5. GPUJPEG knihovna

5.1. Popis knihovny GPUJPEG

GPUJPEG je knihovna, jejíž autorem je Martin Šrom, CESNET, z. s. p. o. Je určena pro kompresi a dekompresi obrazu do formátu JPEG a při výpočtech je využívána grafická karta NVIDIA. Byla vyvíjena pro real time přenos obrazu ve vysokém rozlišení (testována na rozlišení 4K a HD). V popisu je uvedeno, že se jedná o první implementaci, a zatím není tato knihovna optimalizovaná. Využívá Huffmanovo kódování, výstupním formátem je JPEG. Je možné nastavovat výstupní kvalitu (0 – 100%), a dále lze nastavit i mnoho dalších parametrů, které jsou spojeny s principem této komprese. Ze stránek sourceforge.net je možné stáhnout celý projekt, který byl tvořen v prostředí Visual Studio v jazyku C.

5.2. Propojení s LabVIEW

Postup použití knihovny v LabVIEW byl následující – Pomocí prostředí Visual Studio (dále jen VS) provést nutné úpravy a vygenerovat knihovnu s příponou dll, která se následně importuje do programu v LabVIEW. Jelikož je tato knihovna psána jako konzolová aplikace, bylo třeba vyvést potřebné prvky pro nastavení komprimace do LabVIEW. Finální verze bude vypadat tak, že data z kamery budou načítána do prostředí LabVIEW, kde se bude volat uvedená knihovna GPUJPEG. Data budou zpracována, a buď se jednotlivé snímky uloží na disk, nebo budou použita k dalšímu zpracování v LabVIEW.

5.3. Průběh úprav knihovny GPUJPEG

Jak se později ukázalo, celý projekt byl napsán pod operačním systémem Linux. Proto bylo třeba jej upravit pro systém Windows, jelikož drtivá většina počítačů, na kterých je spouštěno LabVIEW je vybaveno právě systémem Windows. Při kompilaci v prostředí VS se ukázalo velké množství chyb, které byly způsobeny absencí knihoven v Linuxu. Proto byla odmazána část kódu, která se odkazovala na chybějící knihovny, jež nebylo možné nahradit ekvivalentem pro Windows. Většinou se jednalo o kód, který zajišťoval styčné rozhraní pro příkazovou řádku Linuxu pro nastavení parametrů.

Dalším krokem bylo napsání rozhraní mezi touto knihovní funkcí a LabVIEW. Dle přiloženého souboru readme bylo taktéž zapotřebí napsat funkce nutné pro obsluhu knihovny. Vše v jazyku C, kód se nachází v souboru main.cpp. Pro použití knihovny GPUJPEG pro komprimaci bylo třeba nastavit parametry vstupního obrazu a nastavení komprese, nastavení barevného prostoru a vzorkovacího faktoru. Dále se inicializuje CUDA zařízení, vytvoří se enkodér, dále se skladují hotová data. Nyní je možné při načtení dalšího obrázku celý postup opakovat.
Pro ověření základní funkčnosti využití externích knihoven v LabVIEW byla vytvořena ukázková aplikace popsaná v kapitole 6.

Při implementaci knihovní funkce do LabVIEW nastaly problémy. Po propojení zkompilované knihovny GPUJPEG, kde již byla napsána i funkce main, zajišťující propojení s LabVIEW, skončilo samotné LabVIEW neočekávanou chybou. Další navrhovaný postup byl napsat další obslužnou aplikaci v jazyku C, ve které se bude daná zkompilovaná knihovna volat místo LabVIEW, neboť zde nebyla kvůli pádům LabVIEW žádná možnost knihovnu ani program ladit.

Proto byl vytvořen další projekt ve Visual Studiu v jazyku C, který měl za úkol dočasně přebrat funkci LabVIEW a umožnit ladění samotné GPUJPEG knihovny. Zde však nastal problém, protože mé znalosti jazyka C nejsou na takové úrovni, abych byl schopen efektivně ladit cizí knihovnu. Při psaní obslužného programu se ukazovalo několik chyb při kompilaci. V jisté chvíli, kdy byla většina parametrů nastavena jako výchozí, nastal jiný problém, a to při samotném užití upravené knihovny GPUJPEG. Vzhledem k dodržení uvedeného postupu od tvůrců knihovny jsem došel k závěru, že buďto je knihovna napsána s nějakou chybou, nebo jsem se při úpravách dopustil chyby já, a program se nechová tak, jak by měl.

5.4. Současný stav

Vzhledem k již věnovanému času a rozsahu problematiky nebyla úprava a implementace knihovny GPUJPEG dokončena. Funkčnost samotné aplikace v LabVEIW nemůže být ověřena, neboť není jisté, zdali je knihovna GPUJPEG správně upravená, nebo celkově funkční. Jak již bylo v popisu projektu GPUJPEG uvedeno, jedná se o první použití, a knihovna byla pravděpodobně odladěna pouze pro použití v příkazové řádce Linuxu.

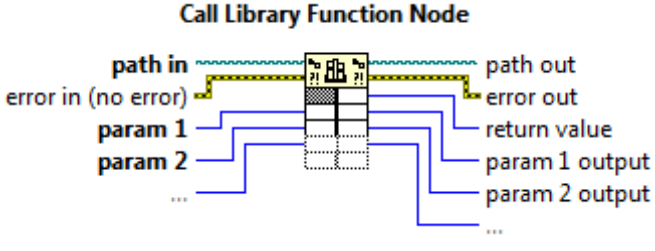
6. Ukázka volání externí knihovny

Pro ověření základní funkčnosti využití externích knihoven v LabVIEW byla vytvořena ukázková aplikace. Byla vyzkoušena práce s funkcí Library Function Node, dále pak ověření, které datové typy lze Library Function Node

6.1. Call Library Function Node

Je to funkce v LabVIEW, umožňující volání externí knihovny dll. Představuje univerzální a obecný nástroj k využívání funkcí a prostředků, které nejsou v LabVIEW přístupné, nebo by se v tomto prostředí realizovaly příliš obtížně (je to obdobný způsob, jako když se pro vytvoření matematických operací využívá struktury Formula Node namísto tvoření výpočtů za pomocí jednotlivých komponent z palety Numeric). Nachází se v paletě Connectivity – Libraries & Executables – Call Library Function Node [16].

Při použití je nutné nastavit cestu ke knihovně dll, která se bude využívat, a dále v záložce Parameters navolit jednotlivé vstupy a výstupy. Navolit lze různé datové typy, mezi něž patří například Numeric, Array, String, Waweform, Digital Waweform. Jedna z možností je i Adapt to Type, když je třeba připojit celou strukturu neboli Cluster. V připojené knihovně je pak třeba se vstupy a výstupy pracovat jako s ukazateli.



Obr. 11 - Funkce pro volání externí knihovny

6.2. Popis ukázkové aplikace

Uvedený program za pomocí externí knihovní funkce k číslu I32 přičte jedničku, pomocí enumerátoru zamění hodnoty „kocka“ a „pes“, a dále sečte dva prvky a,b jež jsou ve formátu Int32. Program je doplněn o výstup z error clusteru, který byl použit hlavně ve fázi ladění.

6.3. Část na straně LabVIEW

Vstupními parametry jsou dva clustery: Data a Soucet. Výstupem (z Call Library Function Node) jsou opět dva clustery: Data 3 a Vysledek. Pro svou potřebu při upravování hotové knihovny jsem potřeboval jako datový typ použít string, ale při pokusu o spuštění hotové knihovny vždy LabVIEW zhavarovalo, musel jsem místo toho použít datový typ Enum, který posílá číselnou hodnotu int, a pak ve volané knihovně v jazyku C napsat konverzi z čísla na daný string.
Položka „kocka“ má hodnotu 222 a položka pes se rovná hodnotě 1. Ve stejném clusteru je i prvek numeric ve formátu Int32. Volaná knihovna provede přičtení jedničky k tomuto číslu. Obě položky (I32 i I32_2) jsou zabaleny v clusteru se jménem Data. Call Library Function Node je nastaven v režimu Adapt to Type, proto lze použít celý cluster. Druhým clusterem je vstup Soucet a Vysledek. Zde se provede sečtení dvou prvků a + b do výsledku c.

6.4. Část na straně knihovny v jazyku C

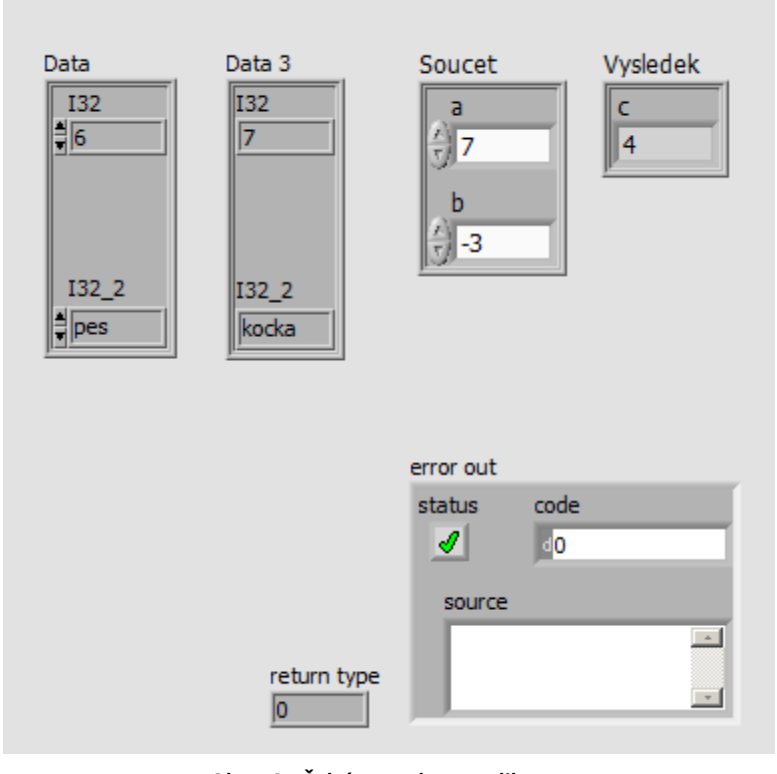
Program je vytvořen v prostředí Microsoft Visual Studio 2013. Pro funkčnost je nutno mít nainstalovaný podpůrný balík CUDA Toolkit 6.5, který je možné stáhnout ze stránek vývojářů NVIDIA [16]. Obsahuje dvě základní části – hlavičkový soubor predani_str.h a samotný program predani_str.cpp.
V hlavičkovém souboru je zajištěno styčné rozhraní mezi LabVIEW a touto knihovnou. Jsou zde vytvořeny požadované clustery (zde jsou to struktury) a také konverze mezi string a int hodnotou v enumerátoru I32_2.
V souboru s příponou cpp již dochází k požadovaným výpočtům a logickým operacím.
Když byl program hotov, byla vygenerována knihovna dll. Cesta k tomuto souboru se zadá v LabVIEW do funkce Call Library Function Node a při spuštění programu v LabVIEW je již využívána knihovní funkce. Pokud je nainstalovaná verze LabVIEW 64-bitová, je nutno při kompilaci knihovny nastavit, že se má kompilovat pro 64bitový systém, jinak zkompilovaná knihovna použít.

6.5. Ukázka kódu v jazyce C

```
#include "stdafx.h"
#include "predani_str.h"
int PREDANI_STR_API struktura(Data * str_in, Data * str_out, Soucet * soucet_in, Vysledek
* vysledek_out)
{
	MyEnum konstanta = pes;
	//soucet
	vysledek_out->c = soucet_in->a + soucet_in->b;
	//inkrementace
	str_out->I32 = str_in->I32 + 1;
```

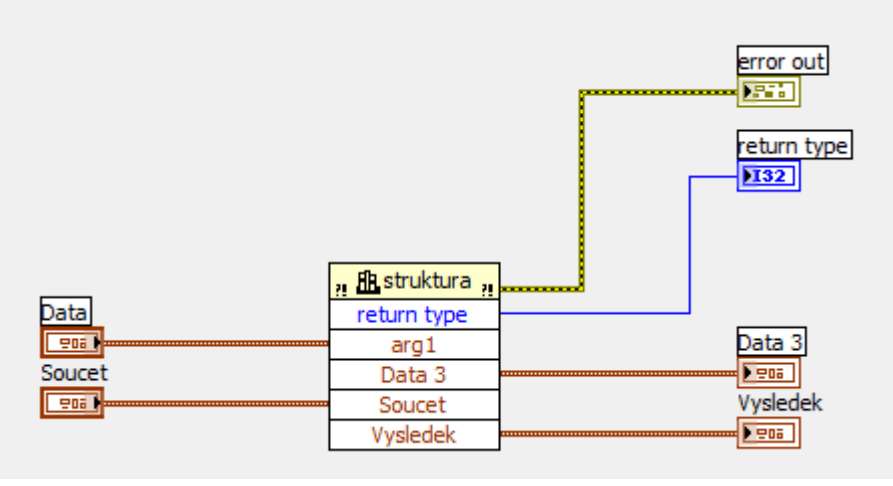
```
        //záměna prvků
        if (str_in->I32_2 ==  pes)
        {
                str_out->I32_2 = kocka;
        }
        else
        {
                str_out->I32_2 = pes;
        }
        return 0;
}
```

6.6. Ukázka programu v LabVIEW



Obr. 12 - Čelní pane demo aplikace

Na uvedeném obrázku je zachycen čelní panel za běhu, kde je ukázána funkčnost využívání jak externích knihoven, tak správnost kódu v jazyku C.



Obr. 13 - Blokový diagram demo aplikace

Zde je ukázka blokového diagramu, kde je vidět použití datového typu Cluster na vstupu i výstupu funkce Call Library Function Node.

7. Závěr

V průběhu odporné praxe jsem se setkal s různorodými úkoly, které vyplynuly z potřeby řešit úkoly ve společnosti ATEsystem s.r.o. Zprvu jsem pracoval na přípravě PC na tester. Byly splněny požadavky zadavatele, a byl úspěšně dokončen i test stability. Ohledně programování v LabVIEW jsem splnil zadání prozkoumat fungování Webových služeb v LabVIEW. Byla vytvořena ukázková aplikace pro součet dvou čísel a jednoduchý generátor signálu, jež dokáže zobrazit data ve webovém prohlížeči s pomocí prostředí User Interface Builder. Ohledně možností komprese obrazu díky výkonu grafických karet jsem nastudoval problematiku komprese a možností využití GPU. Byla vytvořena aplikace k ukázce využívání externích knihoven. Úpravy knihovny GPUJPEG, která měla být určena pro rychlou kompresi obrazu, však dokončeny nebyly kvůli možné chybě v samotné knihovně, možné chybě při rozsáhlých úpravách této knihovny, nebo z důvodu špatné implementace do LabVIEW. Co se týče přínosu této praxe do mých osobních zkušeností, mohu říct, že jsem porozuměl fungování a možností využití webových služeb v LabVIEW. Dále již vím, jak využívat externí knihovny pro využití procesoru i grafické karty, a díky mnoha hodinám, strávených při programování v prostředí LabVIEW jsem si zlepšil techniku efektivnější práce a osvojil používání klávesových zkratek v tomto prostředí. V neposlední řadě jsem si zopakoval programování v jazyku C.

Seznam použité literatury

[1] Zone.ni.com. *Overview: Web-based Communication with a LabVIEW Application (Real-Time, Windows)* [online]. 2013 [cit. 2015-04-23]. Dostupné z: http://zone.ni.com/reference/en-XX/help/371361K-01/lvconcepts/webservices/

[2] Ni.com. *LabVIEW Web Services FAQ* [online]. 2014, 10.9.2014 [cit. 2015-04-22]. Dostupné z: http://www.ni.com/white-paper/7747/en/

[3] Zone.ni.com. *Tutorial: Creating and Accessing a LabVIEW Web Service (Real-Time, Windows)* [online]. 2013 [cit. 2015-04-22]. Dostupné z: http://zone.ni.com/reference/en-XX/help/371361K-01/lvhowto/build_web_service/

[4] Ni.com. *LabVIEW Web UI Builder Overview* [online]. 2010 [cit. 2015-04-22]. Dostupné z: http://www.ni.com/white-paper/11602/en/

[5] Ni.com. *LabVIEW Web UI Builder FAQ* [online]. 2013 [cit. 2015-04-22]. Dostupné z: http://www.ni.com/white-paper/11604/en/

[6] Ni.com. *Introduction to GPU Computing with LabVIEW* [online]. 2013 [cit. 2015-04-22]. Dostupné z: http://www.ni.com/white-paper/14077/en/

[7] Totaj.com. *Výpočty pomocí grafických procesorů GPU – GPGPU* [online]. 2009 [cit. 2015-04-23]. Dostupné z: http://tojaj.com/vypocty-pomoci-gpu/

[8] Sine.ni.com. *NI LabVIEW GPU Analysis Toolkit* [online]. [cit. 2015-04-23]. Dostupné z: http://sine.ni.com/nips/cds/view/p/lang/cs/nid/210829

[9] Elektrorevue: Srovnání metod pro ztrátovou kompresi obrazu. *Elektrorevue* [online]. 25.10.2006 [cit. 2015-04-22]. Dostupné z: http://www.elektrorevue.cz/clanky/06042/index.html

[10] Root.cz. TIŠNOVSKÝ, Pavel. *Ztrátová komprese obrazových dat pomocí JPEG* [online]. 14.12.2006. 2006 [cit. 2015-04-22]. Dostupné z: http://www.root.cz/clanky/ztratova-komprese-obrazovych-dat-pomoci-jpeg/

[11] Nvidia.com. *CUDA zone* [online]. 2009, 15.6.2009 [cit. 2015-04-22]. Dostupné z: http://www.nvidia.com/content/cudazone/cuda_sdk/Image_Video_Processing_and_Data_Compression.html

[12] Sourceforge.net. *GPUJPEG: JPEG compression and decompression accelerated on GPU* [online]. 2013, 20.12.2013 [cit. 2015-04-22]. Dostupné z: http://sourceforge.net/projects/gpujpeg/

[13] CESNET.cz. *CESNET* [online]. 1996, 2015 [cit. 2015-04-22]. Dostupné z: http://www.cesnet.cz

[14] Opensource.org. *The BSD 2-Clause License* [online]. [cit. 2015-04-22]. Dostupné z: http://opensource.org/licenses/BSD-2-Clause

[15] Zone.ni.com. *Call Library Function Node* [online]. 2011 [cit. 2015-04-23]. Dostupné z: http://zone.ni.com/reference/en-XX/help/371361H-01/glang/call_library_function/

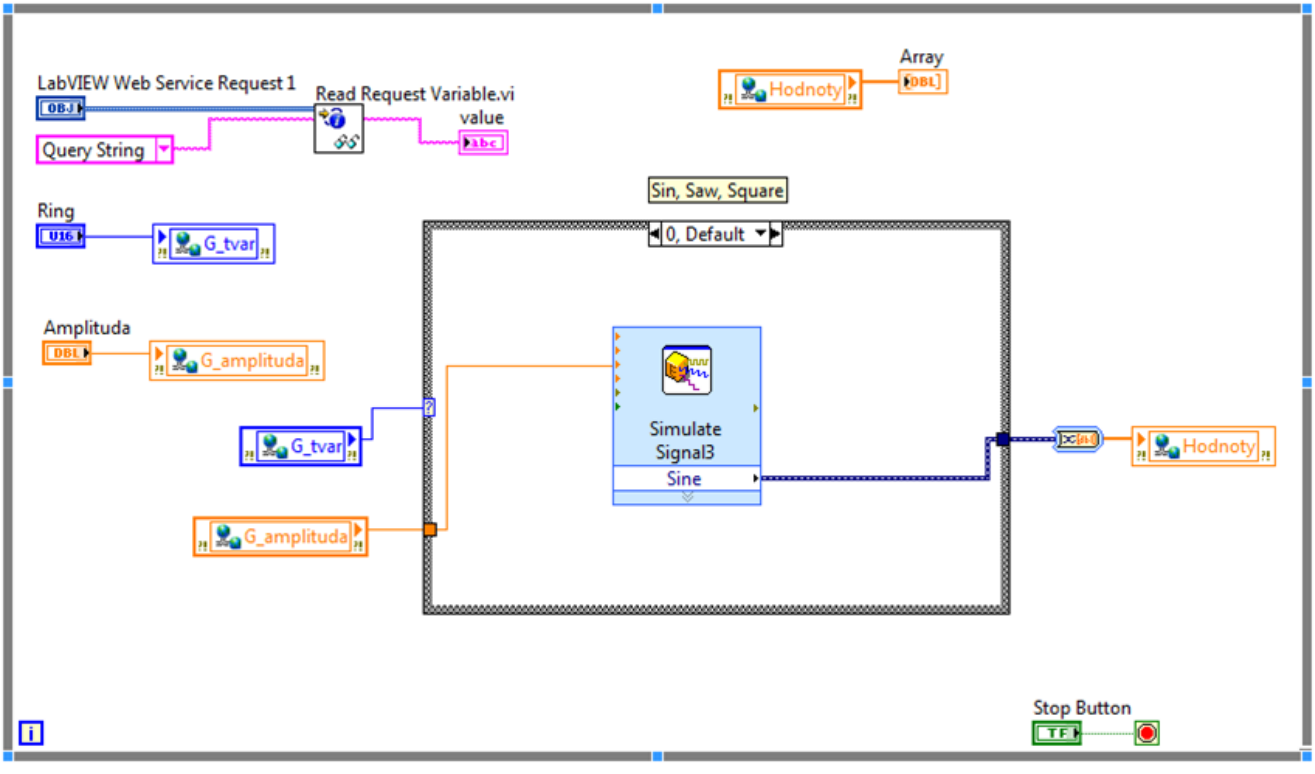
[16] Developer.nvidia.com. *CUDA Toolkit* [online]. 2015 [cit. 2015-04-23]. Dostupné z: https://developer.nvidia.com/cuda-toolkit

Seznam příloh

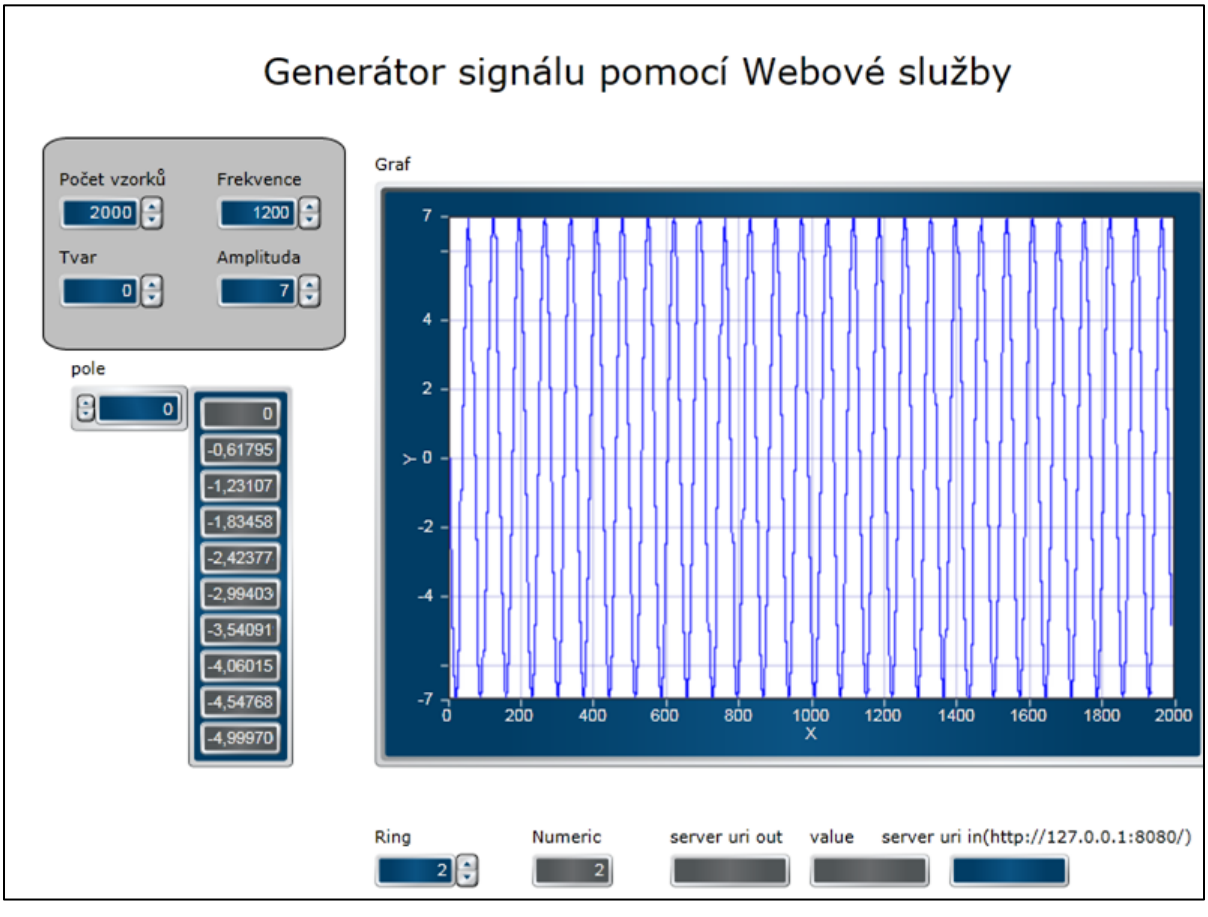
Obsah přiloženého CD

- \GPU\KompreseCUDA\
  obsahuje projekt upravované knihovny pro kompresi obrazu ve Visual Studiu a program vytvořený v LabVIEW
- \GPU\SoucetCUDA\
  obsahuje projekt pro ukázku volání externí knihovny ve Visual Studiu a program v LabVIEW
- \WeboveSluzby\GeneratorFunkci\
  obsahuje serverovou aplikaci v LabVIEW, poskytující data generátoru funkcí
- \WeboveSluzby\Scitani_WeboveSluzby\
  obsahuje serverovou aplikaci v LabVIEW, která umožňuje po síti sčítat dva čísla
- \WeboveSluzby\generator.vix
  exportovaný projekt generátoru funkcí z prostředí UI Builder
- \WeboveSluzby\scitani.vix
  exportovaný projekt aplikace sčítání dvou čísel z prostředí UI Builder

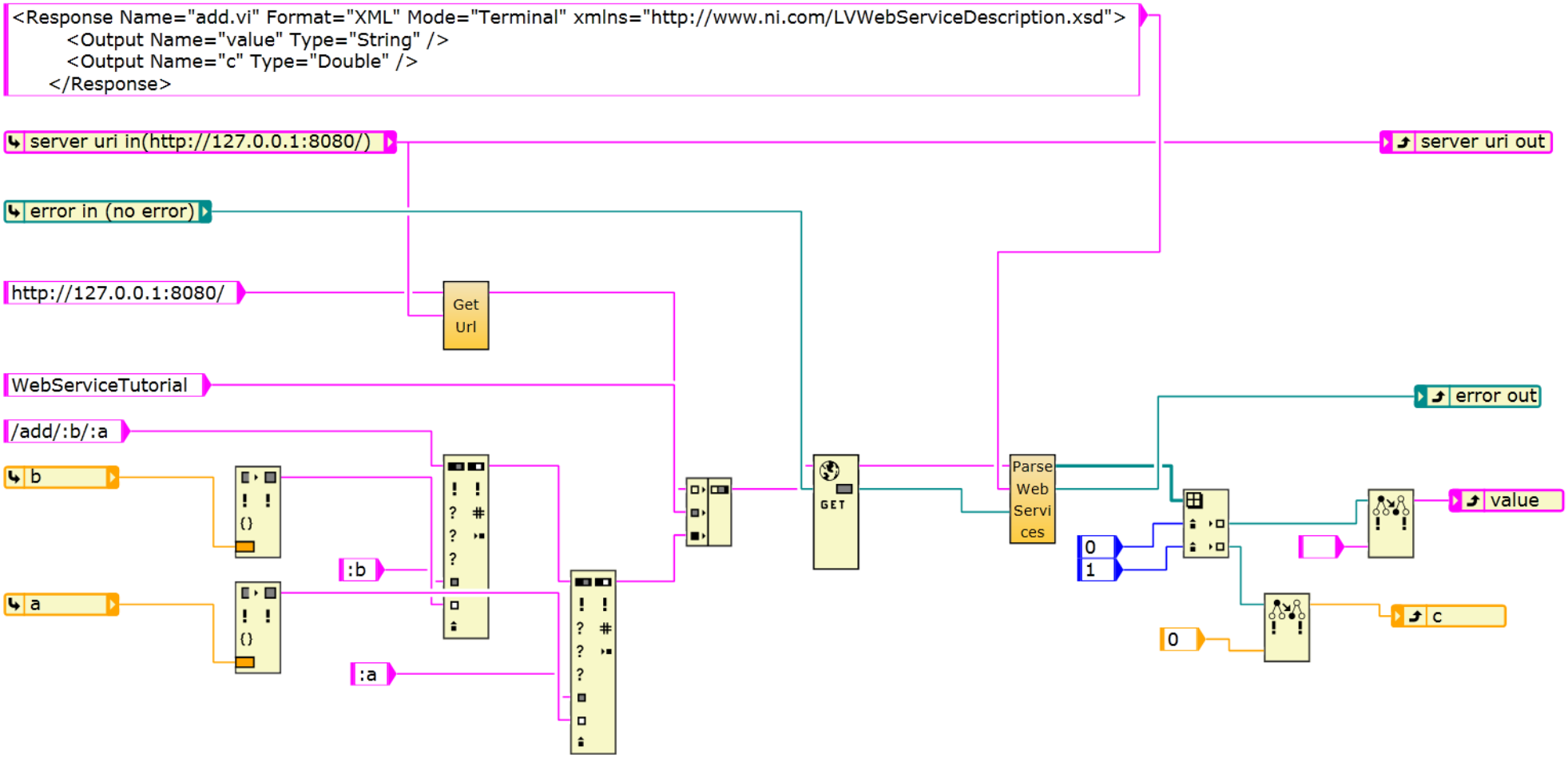
Příloha 1: Schéma serverové aplikace v LabVIEW



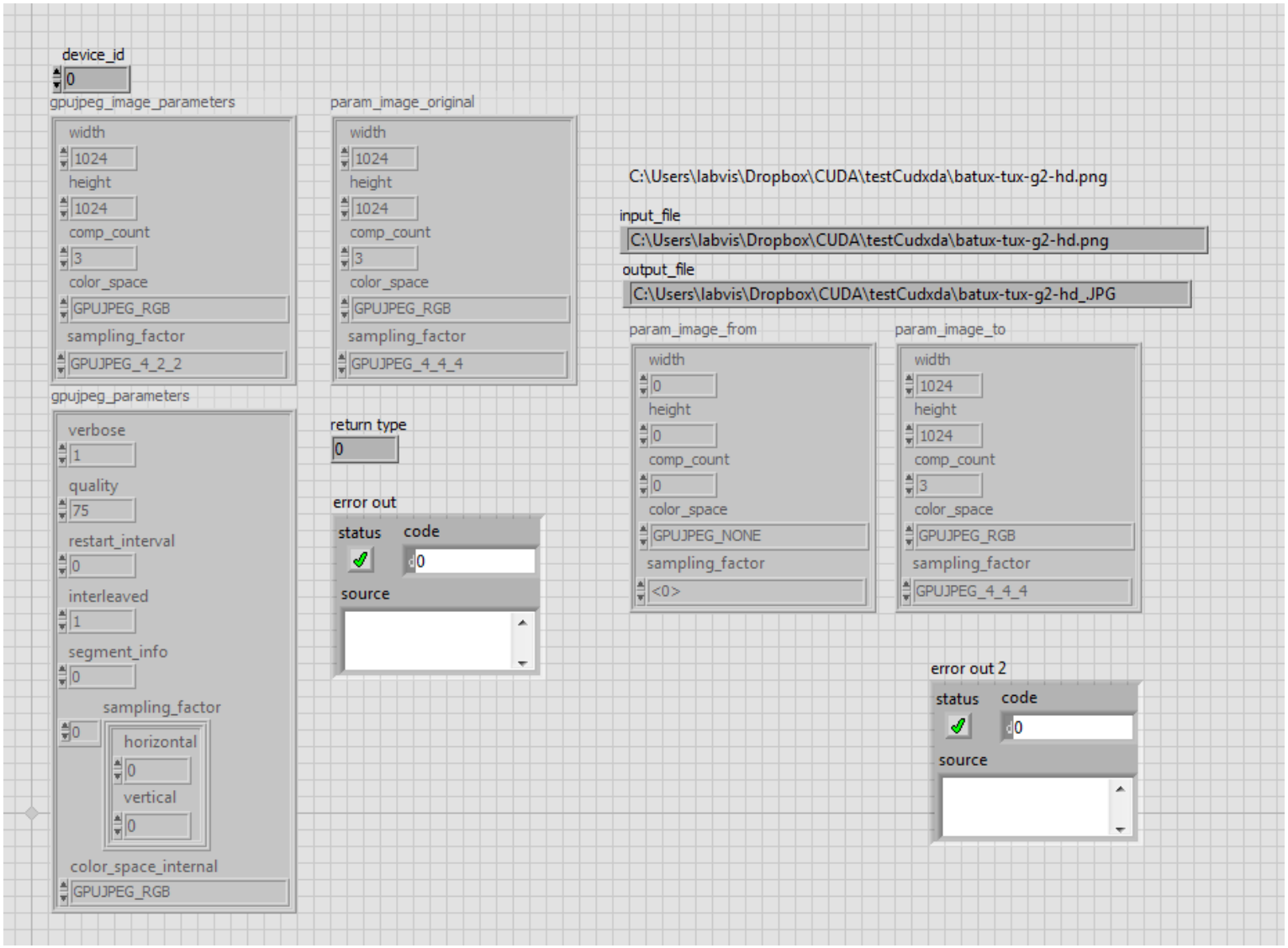
Příloha 2: Čelní panel z UI Builder



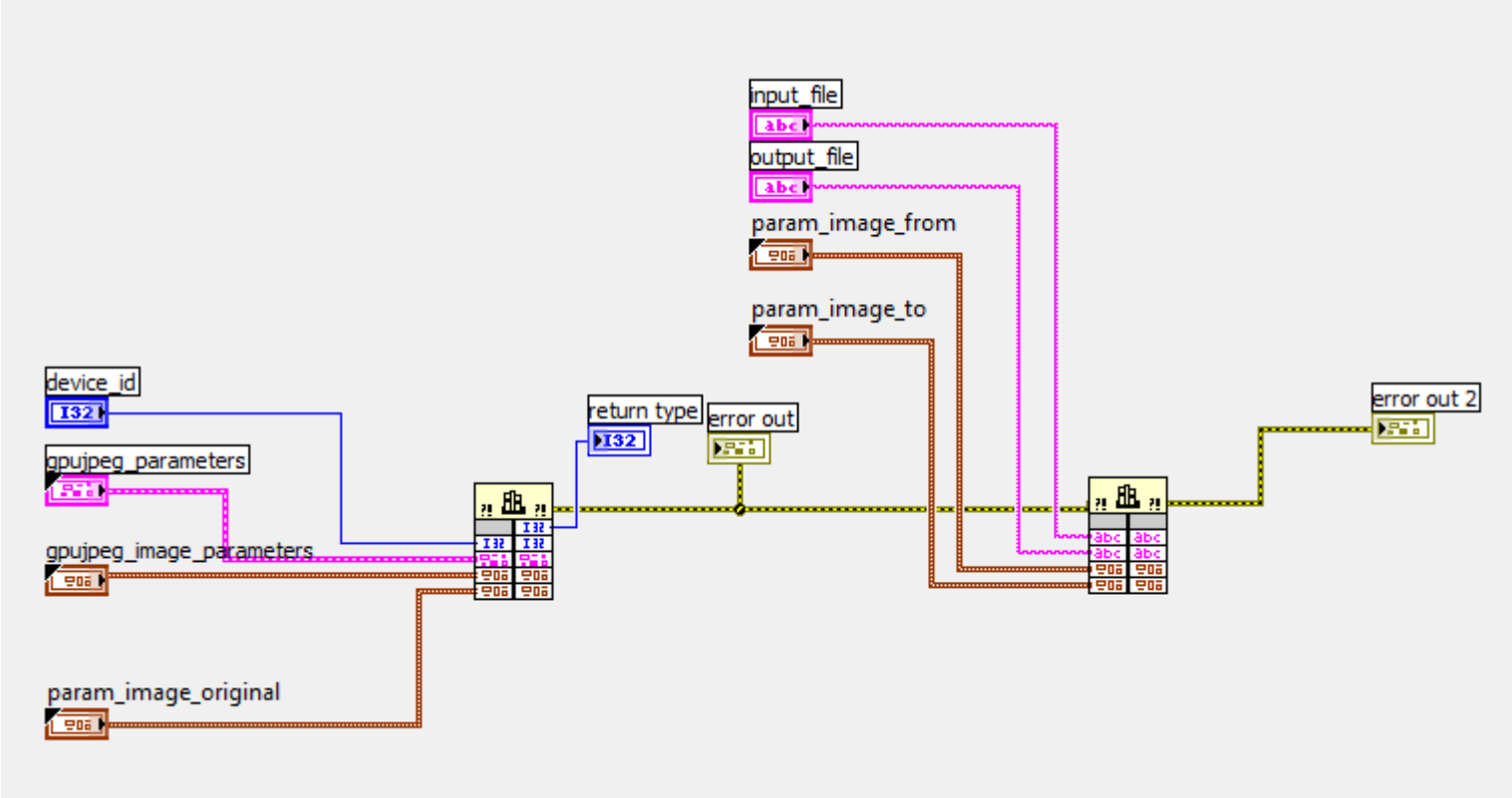
Příloha 3: Schéma z prostředí UI Builder



Příloha 4: Ukázka čelního panelu LabVIEW pro volání GPUJPEG knihovny



Příloha 5: Ukázka blokového diagramu LabVIEW pro volání GPUJPEG knihovny



Příloha 6: Ukázka funkce my_cuda_dll z prostředí Visual Studio

```
// my_cuda_dll.cpp : Defines the exported functions for the DLL application.

#include "stdafx.h"
#include "my_cuda_dll.h"
#define SNIMEK "C:\Users\labvis\Desktop\my_cuda_dll\data.rgb" //treba doplnit na disku
#define SNIMEK "C:\cppworkspace\pic.png"

	gpujpeg_parameters param;
	gpujpeg_image_parameters param_image;
	int device_id = 0;
	int image_size = 0 ;
	uint8_t* image = NULL;

// This is an example of an exported variable
MY_CUDA_DLL_API int nmy_cuda_dll=0;

// This is an example of an exported function.

 MY_CUDA_DLL_API void nacti_obr()
{
	//nacte obrazek


	 if (gpujpeg_image_load_from_file("C:\Users\labvis\Desktop\my_cuda_dll\data.rgb",
&image, &image_size) != 0)
		printf("ERROR");

		//return 23 ;
}
 MY_CUDA_DLL_API int set(void)
{
	gpujpeg_image_set_default_parameters(&param_image);
	gpujpeg_set_default_parameters(&param);
	gpujpeg_parameters_chroma_subsampling(&param);

	if (gpujpeg_init_device(device_id, GPUJPEG_VERBOSE)) // pouzije se vystup
		return -1;

	// vytvoreni encoderu
	struct gpujpeg_encoder* encoder = gpujpeg_encoder_create(&param, &param_image);
	if (encoder == NULL)
		return -1;

	return 42;
}
```