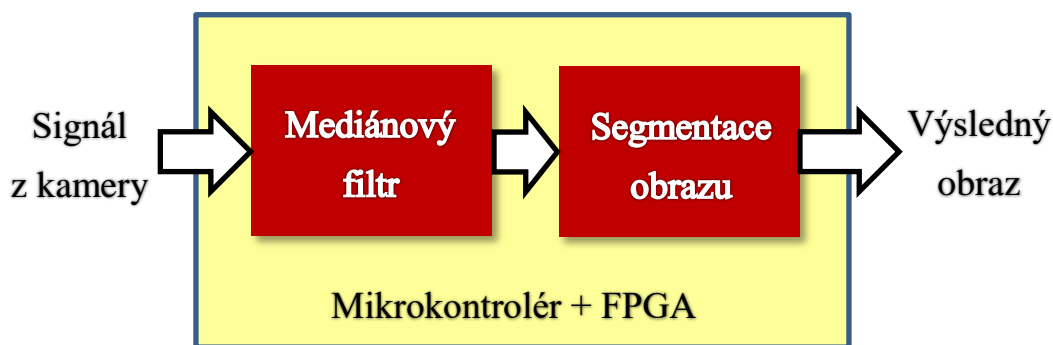


Vestavěný systém pro filtraci a segmentaci obrazu

Cílem projektu je vytvořit vestavěný systém, který s využitím kombinace mikrokontroléru a programovatelného obvodu typu FPGA vykonává filtraci a segmentaci obrazu z kamery (viz obrázek). Filtrace bude realizována skrze mediánový filtr. Segmentace obrazu pak bude implementována technikou prahování, kde hodnota prahu bude vypočtena z histogramu hodnot jednotlivých pixelů snímku metodou Otsu.



Jako vstup budete mít k dispozici kompletní algoritmus pro filtraci a segmentaci obrazu zapsaný v programovacím jazyce C, který lze zkompilovat a spustit na konvenčním procesoru. Vaším úkolem bude tento algoritmus přenést na platformu FITkit a vhodným způsobem jej rozdělit mezi dostupný mikrokontrolér MSP430 a FPGA čip s technologií SPARTAN3. S ohledem na omezené množství výpočetních zdrojů na platformě FITkit je požadováno, aby výsledná aplikace byla schopna zpracovat obraz s rozlišením 320x240 pixelů, který bude generován s rychlostí 60 snímků za vteřinu (více informací o omezeních platformy FITkit bude uvedeno níže).

Postup práce

1. V informačního systému si z adresáře *Projekty* stáhněte archiv s názvem *projekt.zip* a seznámte se s jeho obsahem (viz kapitola [Struktura vstupních souborů](#)).
2. Seznámte se vstupním algoritmem pro filtraci a segmentaci obrazu, který je umístěn v souborech *cpu/cpu.c* a *cpu/program.c*. Pozornost zaměřte na pochopení principu filtrace obrazu s použitím mediánového filtru, způsobu [Zpracování vstupních pixelů](#) a segmentace obrazu s použitím prahování, kde je práh vypočten dynamicky na základě histogramu metodou Otsu (viz samostatný dokument v IS). Všimněte si prosím, že vstupní obraz je v souboru generován s rozlišením 320x240 pixelů, kde každý pixel reprezentuje jeden z osmi možných odstínů šedi. Toto omezení vyplývá z vlastností přípravku FITkit (více informací bude uvedeno níže), které však nemají vliv na obecnost této úlohy. Dále si všimněte, že vstupní obraz je v algoritmu generován a parametrizován na základě loginu – **jeho hodnotu prosím změňte na Váš vlastní login v souboru *cpu/common.h***. Na standardní výstup vypisuje program hodnoty histogramů vybraných 5-ti snímků a také vypočtené hodnoty prahu metodou Otsu. Konkrétněji, program počítá histogram a novou hodnotu prahu u každého desátého snímku a tyto informace zobrazuje jednou za 100 snímků. Pozn.: pokud nepoužíváte na svém osobním počítači operační systém typu linux, můžete pro kompilaci a spuštění využít např. stroj *merlin.fit.vutbr.cz*.
3. Vytvořený algoritmus přeneste na mikrokontrolér MSP430 (do souboru *mcu/main_sw.c*) a vytvořte tak čistě softwarovou implementaci algoritmu (bez účasti FPGA). Ověřte funkčnost této implementace přímo na přípravku FITkit (postupujte dle návodu uvedeného v kapitole [Ověření aplikace na přípravku FITkit](#)). **Výstup generovaný na terminálu FITkitu musí být stejný jako výstup generovaný programem *cpu/program.c*. Důležitá poznámka:** Doba zpracování jednotlivých pixelů je na mikrokontroléru výrazně delší než na CPU. Abyste se dočkali výsledků v rozumném čase, je pro vás připravena upravená funkce generátoru pixelů, která dokáže posouvat generátor o několik snímků vpřed. Základní myšlenkou pro získání stejných hodnot histogramů jako na CPU je spustit výpočet na daném snímku a následně posunout generátor o 99 snímků vpřed, spočítat další snímek, posunout generátor o 99 snímků, atd. Na závěr tohoto bodu změňte

průměrnou dobu zpracování jednoho pixelu snímku. Pro změření tohoto času využijte předpřipravenou funkci `get_time()`. Přepínač *PROFILE* využijte pro výběr mezi režimem měření času a režimem výpisu histogramů a vypočteného prahu na terminál.

4. Z předchozího bodu by Vám mělo vyplynout, že samotný mikrokontrolér MSP430 nemá dostatečný výkon pro zpracování obrazu s rozlišením 320x240 pixelů, 60 snímků za vteřinu. Analyzujte proto časově kritické části algoritmu s využitím volně dostupného programu [gprof](#), který je schopen změřit procentuální zastoupení jednotlivých funkcí algoritmu v rámci přiděleného procesorového času. Pro získání přesnějších údajů proveďte měření alespoň desetkrát a použijte průměrnou hodnotu z naměřených výsledků. Jelikož program *gprof* není snadné aplikovat přímo na mikrokontrolér MSP430, proveďte tuto profilaci s původním algoritmem v souboru *cpu/program.c* a na běžném procesoru. Pozn.: pokud nepoužíváte na svém osobním počítači operační systém typu linux, můžete pro profilaci použít např. stroj [merlin.fit.vutbr.cz](#). Při profilaci aktivujte přepínač *PROFILE*, který zajistí, že se do naměřených hodnot nebude započítávat tisk histogramů a vypočtených prahů na standardní výstup.
5. Seznamte se s [Architektura systému na přípravku FITkit](#) a ověřte si funkčnost předpřipravené demo aplikace pro komunikaci mezi mikrokontrolérem a FPGA čipem. Pro vyzkoušení této demo aplikace postačí změnit v souboru *project.xml* zdrojový kód pro mikrokontrolér z *mcu/main_sw.c* na *mcu/main_swhw.c* a opět zkompileovat a naprogramovat přípravek FITkit. Jakmile se Vám podaří úspěšný test této demo aplikace, vyzkoušejte si dále postup, jakým lze pomocí nástroje Catapult C získat ze zdrojových souborů v jazyce C výstupní soubory v jazyce VHDL a Verilog. Postupujte dle návodu v kapitole [Syntéza souborů s využitím nástroje Catapult C](#) a aplikujte jej na soubor *fpga/src_filter/filter.cpp*. Na závěr si tímto postupem vygenerujte také nový VHDL soubor pro generátor pixelů (soubor *fpga/src_genpix/genpix.cpp*), který bude využívat **Vámi nastavený login ze soboru *cpu/common.h***.
6. Seznamte se s navrhovaným způsobem rozdělení algoritmu mezi hardware a software, který je uveden v kapitole [Rozdělení algoritmu mezi hardware a software](#). Proveďte implementaci navrženého systému a jeho **funkčnost ověřte nejprve simulací** v prostředí Catapult C. Pro účely simulace je pro Vás připraven testbench soubor *fpga/src_filter/tb_filter.cpp*.
7. Jakmile bude Vaše simulace vykazovat bezchybné výsledky, přistupte k ověření aplikace na přípravku FITkit. **Zkontrolujte, zda výstup generovaný na terminálu FITkitu se shoduje s výstupem generovaným původním programem *cpu/program.c***. Pozn.: Při implementaci prosím dbejte na efektivní využití zdrojů v FPGA čipu a jednotlivé datové typy proměnných optimalizujte na nezbytně nutnou datovou šířku. Ve vstupním zdrojovém kódu je pro vás připravena modifikovaná verze funkce pro výpočet mediánu, která je vhodnější pro hardwarovou realizaci.
8. Vytvořte technickou zprávu dle pokynů uvedených v kapitole [Pokyny pro vypracování technické zprávy](#).
9. Vytvořte výstupní archiv souborů *projekt.zip* dle pokynů v kapitole [Výstupy projektu](#) a **tento archiv si před odevzdáním otestujte skrze připravenou sadu skriptů *student_test.zip*, kterou naleznete v IS**. Podrobný popis pro spuštění těchto skriptů je uveden v příloženém README souboru.

Výstupy projektu

Prostřednictvím informačního systému budete odevzdávat archiv *projekt.zip*, který musí obsahovat následující soubory:

- Implementace aplikace běžící pouze na mikrokontroléru (bez FPGA)
 - `mcu/main_sw.c` – kód pro MSP430
- Implementace aplikace rozdělené mezi HW a SW
 - `mcu/main_swhw.c` – kód pro MSP430
 - `fpga/src_filter/filter.cpp`, `filter.h` – kód pro FPGA
 - `fpga/src_filter/directives.tcl` – skript, kterým byla provedena syntéza souboru `filter.cpp` do jazyka VHDL a Verilog
 - `build/med_filtr.bin` – binární soubor s konfigurací pro FPGA vytvořený v aplikaci QDevKit
- Technická zpráva vypracovaná přesně dle pokynů
 - `report.pdf`

Důležité poznámky:

- **Dodržujte prosím předepsanou adresářovou strukturu i pojmenování souborů. Jakékoliv odchylky mohou vést k chybám při opravování projektu a následně k udělení nižšího bodového hodnocení.**
- **Neupravujte soubory, které nejsou explicitně zmíněny v zadání projektu např. `cpu.h`, `cpu.c`, `program.c` apod. Některé z nich se používají při verifikaci, a proto by jejich úprava mohla vést k chybám při opravování projektu.**
- **Pozor, v případě opisování budou pachatelé předvedeni před disciplinární komisi a projekt bude hodnocen nula body!**

Doplňující informace

Struktura vstupních souborů

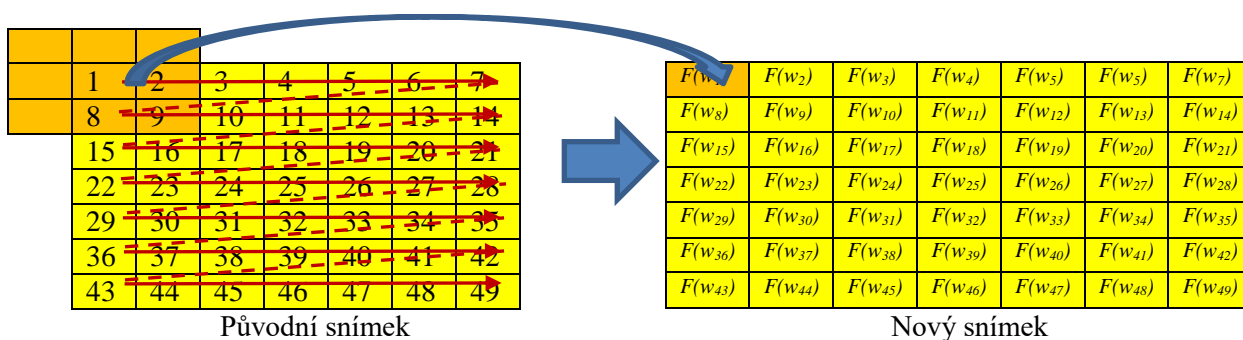
Archiv *projekt.zip* obsahuje adresář s názvem *segmentace* obsahující následující části:

- Vstupní algoritmus pro filtraci a segmentaci obrazu určený pro běžný procesor
 - `cpu/` – zdrojové kódy a Makefile
- Předpřipravený soubor pro aplikaci běžící pouze na mikrokontroléru (bez FPGA)
 - `mcu/main_sw.c` – kód pro MSP430
- Ukázková aplikace pro komunikaci mezi mikrokontrolérem a FPGA, která současně slouží jako šablona pro výsledné řešení
 - `mcu/main_swhw.c` – kód pro MSP430
 - `fpga/src_filter/` – zdrojový kód, testbench soubor a překladové skripty pro filtr
 - `fpga/src_genpix/` – zdrojový kód, testbench soubor a překladové skripty pro generátor pixelů
 - `fpga/src_video/` – zdrojový kód, testbench soubor a překladové skripty pro video buffer
 - `fpga/top_level.vhd` – VHDL soubor propojující všechny komponenty uvnitř FPGA
 - `fpga/sim/` – testbench soubor a fdo skript pro ModelSIM umožňující spustit simulaci všech komponent v FPGA
- Konfigurační soubor pro překlad a spuštění aplikace v prostředí QDevKit
 - `project.xml`

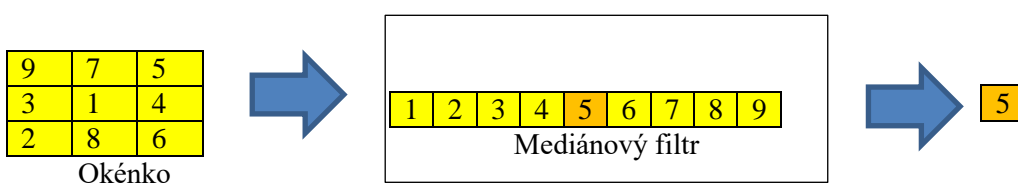
Více informací o úloze jednotlivých komponent v FPGA a jejich propojení naleznete níže v kapitole [Architektura systému na přípravku FITkit](#).

Zpracování vstupních pixelů

Celá řada obrazových operátorů v počítačové grafice používá posuvné okénko 3x3 pixelů, kterým postupně prochází celý snímek a na jejímž základě vypočítává hodnoty pixelů nového snímku (viz obrázek).

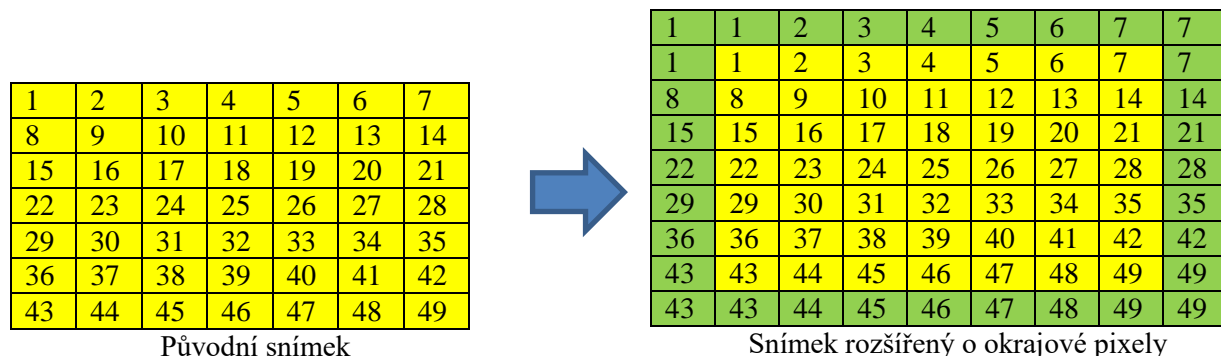


Mezi tyto operátory patří i mediánový filtr, který nové hodnoty pixelů (vyfiltrovaného snímku) vypočítává na základě okolních bodů původního snímku. Samotnou funkci mediánového filtru $F(w)$ lze popsat následovně: filtr hodnoty pixelů v posuvném okénku nejprve seřadí, a potom na svůj výstup vydá prostřední hodnotu této seřazené posloupnosti (viz obrázek).



Při aplikaci posuvného okénka vzniká problém, jak se zachovat k pixelům na okrajích snímku, ke kterým nemáme k dispozici okolní body. Jedním z nejjednodušších způsobů řešení tohoto problému je replikace okrajových bodů

(někdy označovaná také jako *window clipping*). Názorně je tato operace zobrazena na následujícím obrázku. Na takto rozšířený snímek již lze snadno aplikovat posuvné okénko i pro okrajové body. Samotné rozšíření snímku je možné realizovat buď *přímo* - skutečným zvětšením snímku; nebo *nepřímo* (až v průběhu výpočtu) – velikost snímku zůstává stejná, avšak v okamžiku zpracování některého z okrajových pixelů je okénko 3x3 vytvořeno replikací vnitřních bodů. V praktických aplikacích je častěji využíván tento druhý způsob, který nevyžaduje skutečné zvýšení rozlišení snímku.



Pokud je snímek zpracováván na běžném počítači, je obvykle k dispozici dostatek paměti pro jeho uložení. Pokud je však zpracování obrazu přesunuto na některé vestavěné zařízení nebo přímo do specializovaného čipu, potom je nezbytné s paměťovým prostorem co nejvíce šetřit. Vstupní obraz je do těchto zařízení zasílán obvykle ve formě sériového proudu dat (pixel po pixelu). Pokud si uvědomíme, jakým způsobem je posuvné okénko postupně aplikováno na celý snímek, zjistíme, že pro zpracování libovolného pixelu nám postačí paměť o velikosti nejvýše dvou řádků a tří pixelů (viz obrázek).

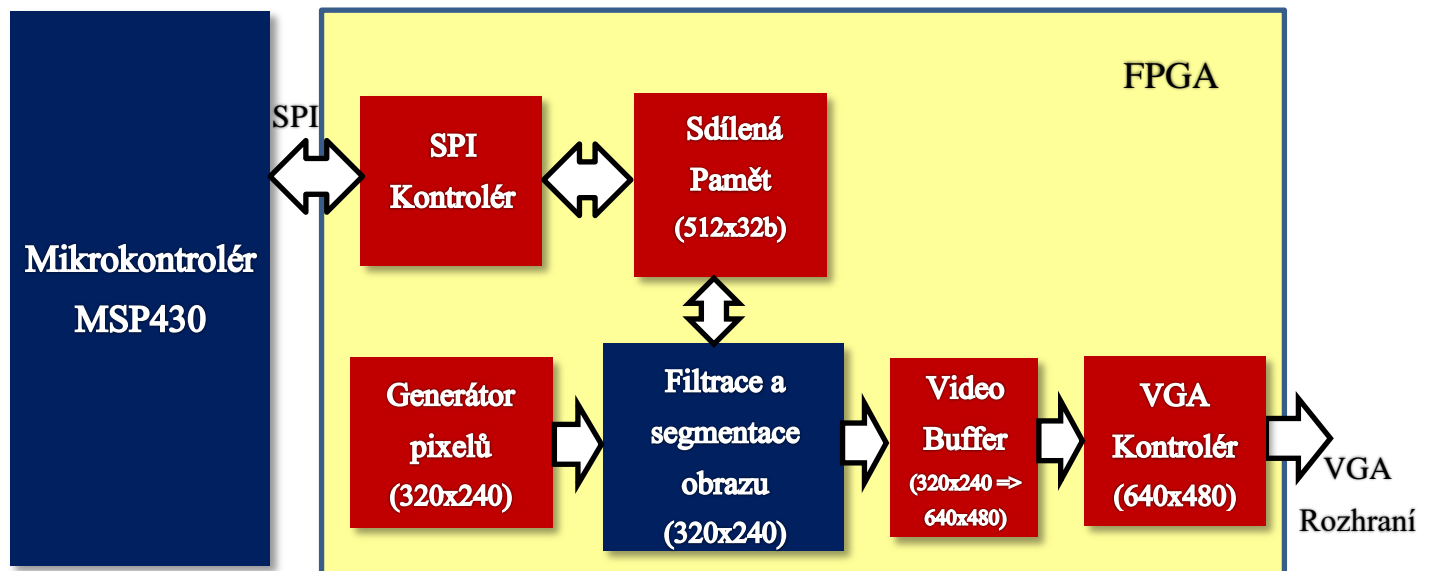
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49

Zvýrazněné pixely označují oblast, která musí být uložena v pomocné paměti

Architektura systému na přípravku FITkit

Předpřipravená architektura systému pro filtraci a segmentaci obrazu na přípravku FITkit je znázorněna na následujícím obrázku. Tento systém zahrnuje mikrokontrolér MSP430 a FPGA čip s technologií SPARTAN3, které jsou propojeny skrze standardní sériové rozhraní SPI. Propustnost tohoto rozhraní je pouze 460 kB/s a je navíc sdílena pro oba směry komunikace. Data jsou mezi mikrokontrolérem a FPGA čipem vyměňována skrze sdílenou paměť o velikosti 512x32b, která je umístěna na FPGA čipu (realizováno jako dvou-portová paměť typu BlockRAM). **Obsah této sdílené paměti je po spuštění aplikace vynulován.** Zatímco mikrokontrolér k obsahu této sdílené paměti přistupuje skrze SPI rozhraní a SPI kontrolér, FPGA čip může přistupovat přímo. V ukázkovém souboru pro mikrokontrolér jsou pro Vás připraveny pomocné funkce `fpga_read(addr)` a `fpga_write(addr, data)`, skrze které lze snadno z této paměti číst nebo do ní zapisovat.

Přípravek FITkit nemá bohužel k dispozici rozhraní pro vstup video signálu z kamery, a proto musí být tento vstup generován uvnitř systému. Jelikož je propustnost mezi mikrokontrolérem a FPGA čipem příliš nízká, není možné pixely generovat v mikrokontroléru a přenášet do FPGA a musí tak být generovány přímo uvnitř FPGA (komponenta *Generátor pixelů*). Jednotlivé pixely jsou z generátoru zasílány komponentě *Filtrace a segmentace obrazu*, která je ústřední částí celé architektury. Jednotlivé pixely vyfiltrovaného a segmentovaného obrazu jsou následně zasílány do *Video bufferu* a *VGA kontroléru*. Tyto komponenty zajišťují správné zobrazení výsledného snímku na VGA konektoru přípravku FITkit. Korektní chování Vaší aplikace bude tak možné ověřit na kterémkoliv monitoru podporujícím režim 640x480, 60 snímků za vteřinu.



Přípravek FITkit obsahuje několik omezení, které je potřeba vzít při řešení této úlohy v úvahu. Mezi tato omezení patří:

- Počet odstínů zobrazovaných na VGA rozhraní je pouze 8. Přesněji řečeno, pro každou barevnou složku (RGB) jsou na VGA rozhraní přípravku rezervovány pouze 3 bity. Jelikož tento projekt pracuje pouze s odstíny šedi, je k dispozici pouze 8 těchto odstínů.
- S ohledem na omezené výpočetní zdroje přípravku FITkit pracuje *Generátor pixelů* a obvod pro *Filtraci a segmentaci obrazu* pouze s polovičním rozlišením obrazu tj. 320x240, 60 snímků za vteřinu. Teprve až komponenta *Video buffer* provede transformaci tohoto obrazu na rozlišení 640x480 tak, aby jej bylo možné sledovat na monitoru. Pozn.: *VGA Kontrolér* nepodporuje nižší rozlišení než 640x480 pixelů.
- Obvod umístěný v FPGA pracuje na frekvenci 25MHz, která odpovídá frekvenci, s jakou jsou generovány jednotlivé pixely na VGA rozhraní s rozlišením 640x480. Jelikož filtrace a segmentace obrazu pracuje s polovičním rozlišením 320x240, je počet pixelů čtvrtinový oproti rozlišení 640x480 ($320 \times 240 = 640 \times 480 / 4$). Jednotka pro filtraci a segmentaci má proto k dispozici nejvýše 4 takty hodinového signálu mezi zpracováním jednotlivých pixelů. Jinými slovy každé 4 takty hodinového signálu musí začít zpracování nového pixelu (tzv. inicializační interval smyčky), délka tohoto zpracování (tj. latence) však může být libovolná.
- Výpočet histogramu a odvození nového prahu metodou Otsu se provede pouze jednou za 10 snímků obrazu (častější aktualizace není potřeba). Jelikož výpočet nového prahu metodou Otsu trvá určitou dobu, není možné jej aplikovat ihned na následující snímek obrazu, ale až o snímek později. Jinými slovy, pokud je vypočten histogram ze snímku s indexem i , potom v průběhu snímku $i+1$ je vypočtena nová hodnota prahu a tuto novou hodnotu lze aplikovat až od prvního pixelu snímku $i+2$.

V průběhu řešení tohoto projektu budete modifikovat pouze kód mikrokontroléru a komponenty *Filtrace a segmentace obrazu* (komponenty systému označeny modře). Všechny ostatní komponenty (označeny červeně) jsou pro vás již připraveny a nebudete je tak muset vytvářet.

Rozhraní komponenty pro filtraci a segmentaci obrazu

Pro správnou implementaci komponenty *Filtrace a segmentace obrazu* je potřeba pochopit rozhraní této komponenty jak na úrovni jazyka C, tak na úrovni výsledného obvodu v jazyce VHDL a Verilog.

Rozhraní na úrovni jazyka C je následující:

Název	Typ	Orientace	Popis
in_data	t_pixel	Vstup	Vstupní pixel
in_data_vld	bool	Vstup	Informace o tom, zda je vstupní pixel platný
out_data	t_pixel	Výstup	Výstupní pixel
mcu_data	t_mcu_data[512]	Vstup/Výstup	Sdílená paměť pro výměnu dat s MCU

Na úrovni výsledného obvodu v jazyce VHDL je potřeba, aby se jednotlivé parametry chovaly následovně:

Název	Rozhraní - Catapult	Rozhraní - VHDL
in_data	mgc_in_wire_en	in_data_rsc_z : IN std_logic_vector (2 DOWNT0 0); in_data_rsc_lz : OUT std_logic;
Pozn.: tento typ rozhraní přidá k samotné datové sběrnici <i>in_data</i> i jednobitový signál (<i>_lz</i>), jímž si komponenta žádá o nová data. Jinými slovy, tento signál je aktivován, kdykoliv je uvnitř kódu v jazyce C přečtena proměnná <i>in_data</i> .		
in_data_vld	mgc_in_wire	in_data_vld_rsc_z : IN std_logic;
Pozn.: jedná se pouze o jednobitový vodič informující o platnosti dat.		
out_data	mgc_out_stdreg_en	out_data_rsc_z : OUT std_logic_vector (2 DOWNT0 0); out_data_rsc_lz : OUT std_logic ;
Pozn.: Tento typ rozhraní přidá k samotné datové sběrnici <i>out_data</i> i jednobitový signál (<i>_lz</i>), jímž komponenta potvrzuje platnost výstupních dat. Jinými slovy, tento signál je aktivován, kdykoliv je uvnitř kódu v jazyce C zapsána hodnota do proměnné <i>out_data</i> .		
mcu_data	Xilinx_RAMs. BLOCK_SPRAM_RBW	mcu_data_rsc_d : OUT std_logic_vector (31 DOWNT0 0); mcu_data_rsc_adr : OUT std_logic_vector (8 DOWNT0 0); mcu_data_rsc_we : OUT std_logic; mcu_data_rsc_q : IN std_logic_vector (31 DOWNT0 0);
Pozn.: Tento typ rozhraní znamená, že uvedené pole bude umístěno v externí paměti typu RAM a rozhraní komponenty bude doplněno o běžné paměťové rozhraní. Kdykoliv je uvnitř kódu v jazyce C čtena nebo zapisována položka z/do tohoto pole, je vygenerována příslušná čtecí nebo zápisová transakce na tomto rozhraní.		

Důležitá poznámka: Typy jednotlivých rozhraní nemusíte v nástroji Catapult ručně nastavovat. Pokud pro překlad použijete předpřipravené skripty *directives.tcl* (viz níže), udělají to za vás.

Rozdělení algoritmu mezi hardware a software

Vzhledem k některým omezením architektury FITkit (popsaným v předchozí kapitole) je potřeba provést rozdělení algoritmu mezi hardware a software následujícím způsobem:

- Hardware (FPGA) bude realizovat:
 - Generování pixelu – funkce *gen_pixel()*, *update_base_position()*
 - Zpracování a uložení pixelu – funkce *pixel_processing()*, *system_input()*, *clip_windows()*, *shift_window()*, *buffer()*
 - Filtraci obrazu – funkce *median()*
 - Segmentaci obrazu – funkce *thresholding()*
- Software (MCU) bude realizovat:
 - Čekání na 10. snímek obrazu
 - Čtení histogramu a výpočet nového prahu metodou Otsu - funkce *otsu()*
 - Vymazání histogramu – funkce *histogram_clean()*
 - Tisk výsledného histogramu a nově vypočteného prahu na terminál – funkce *print_results()*

Komunikace mezi FPGA a MCU

FPGA a MCU si budou veškerá data předávat skrze sdílenou paměť umístěnou v FPGA a zpřístupněnou MCU skrze SPI rozhraní. Podrobný popis paměťového prostoru této sdílené paměti je uveden níže.

FPGA a MCU budou komunikovat následujícím způsobem:

- FPGA bude každý 10. snímek předávat MCU vypočtený histogram pixelů skrze položku sdílené paměti *FPGA_HISTOGRAM*.
- Informaci o aktuálně zpracovaném snímku předá FPGA směrem k MCU skrze počítadlo snímků *FPGA_FRAME_CNT*.
- MCU následně vypočte nový práh pro segmentaci obrazu a tento práh předá zpět do FPGA skrze položku *FPGA_THRESHOLD*.
- MCU také vynuluje obsah zpracovaného histogramu a připraví jej pro další 10. snímek.

Synchronizace FPGA a MCU

- Před začátkem komunikace je nezbytné, aby se MCU a FPGA vzájemně informovali o své připravenosti zpracovat danou úlohu. V případě platformy FITkit dochází k tomu, že FPGA je naprogramováno/spuštěno jako první a až následně je naprogramováno/spuštěno MCU. Pokud by FPGA část nepočkala na spuštění kódu na MCU a začala zpracovávat vstupní pixely, potom by mohlo dojít k různým chybám. Například FPGA očekává, že po prvních 10-ti snímcích bude ve sdílené paměti k dispozici hodnota nově vypočteného prahu metodou Otsu, ale tato hodnota tam ve skutečnosti ještě nebude. Dále MCU bude chtít vytisknout histogram pro snímek č. 100, ale nestihne jej zachytit, atd.
- Pro účely synchronizace byla ve sdíleném paměťovém prostoru vyčleněna položka *FPGA_MCU_READY*, která je inicializována na nulu. Jakmile je MCU připraveno, zapíše do této položky jedničku. FPGA část na tento okamžik čeká a jako potvrzení připravenosti obou stran zapíše nazpět do této položky hodnotu 2. Tímto je synchronizace ukončena a může začít generování/zpracování vstupní pixelů a generování histogramů.

Popis paměťového prostoru sdílené paměti:

Index	Název	Popis
0-7	FPGA_HISTOGRAM	Hodnoty histogramu jednotlivých pixelů <ul style="list-style-type: none">FPGA: inkrementuje počty pixelů s příslušným odstínem šediMCU: čte každý 10. snímek a nuluje
8	FPGA_MCU_READY	Příznak o připravenosti MCU <ul style="list-style-type: none">MCU: zapisuje 1 jako příznak připravenostiFPGA: čte a čeká, dokud nebude zapsána 1. Následně jako znamení připravenosti obou stran zapíše hodnotu 2.
9	FPGA_THRESHOLD	Práh pro segmentaci obrazu vypočtený metodou Otsu <ul style="list-style-type: none">MCU: zapisuje nově vypočtený práhFPGA: čte nově vypočtený práh a aplikuje jej při segmentaci
10	FPGA_FRAME_CNT	Číslo zpracovávaného snímku <ul style="list-style-type: none">FPGA: inkrementuje číslo zpracovaného snímkuMCU: čte a čeká na každý 10. Snímek

Ověření aplikace na přípravku FITkit

Vývojové nástroje

Pro vypracování projektu a jeho ověření na přípravku FITkit jsou potřeba následující nástroje:

- MSPGCC – kompilátor zdrojových souborů pro mikrokontrolér MSP430
- Xilinx ISE – vývojové prostředí pro syntézu obvodů v jazyce VHDL a Verilog do čipu FPGA
- ModelSIM – simulátor obvodů v jazyce VHDL a Verilog
- QDevKit – prostředí pro práci s přípravkem FITkit

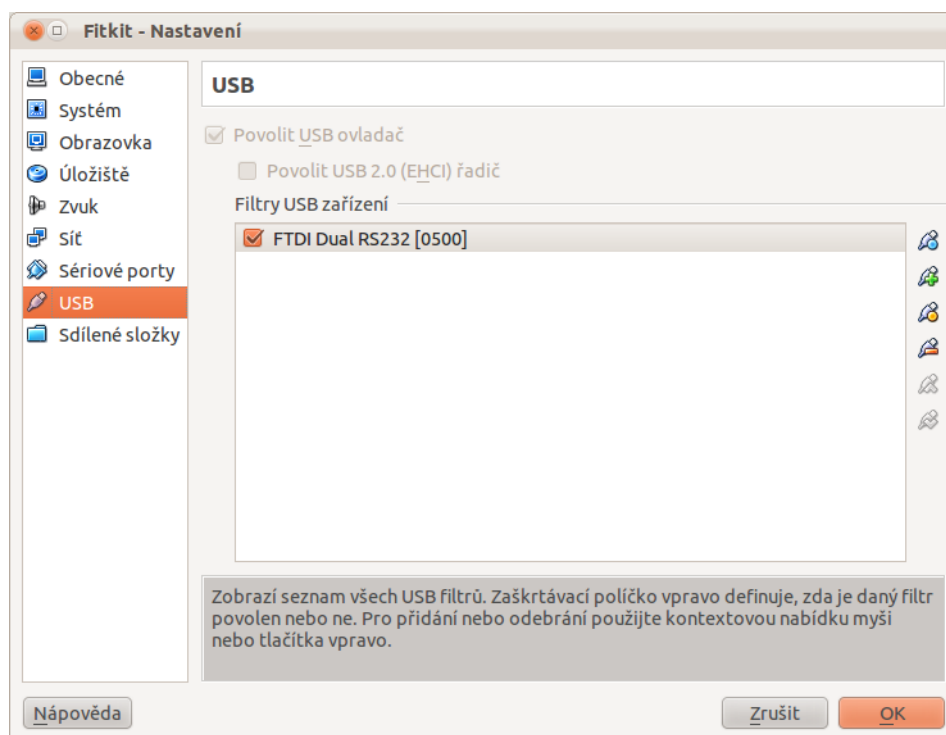
- Catapult C – vývojové prostředí pro syntézu algoritmů z jazyka C++ do jazyka VHDL a Verilog
- Microsoft Visual C++ 2008 – kompilátor zdrojových souborů v jazyce C++

S ohledem na náročnost instalace všech těchto nástrojů a skutečnost, že ne všechny instalační soubory jsou volně ke stažení, byl pro Vás připraven obraz virtuálního stroje, kde jsou všechny tyto nástroje již připraveny. Obraz tohoto virtuálního stroje si můžete stáhnout z [privátních stránek FITkitu](#). Pro spuštění tohoto stroje využijte volně dostupného programu VirtualBox. Pro pohodlnou práci je vhodné tomuto stroji vyčlenit 4GB paměti RAM. Pokud na svém počítači nemáte k dispozici dostatečné množství paměti, je možné využít počítače v CVT.

Spuštění aplikace na přípravku FITkit

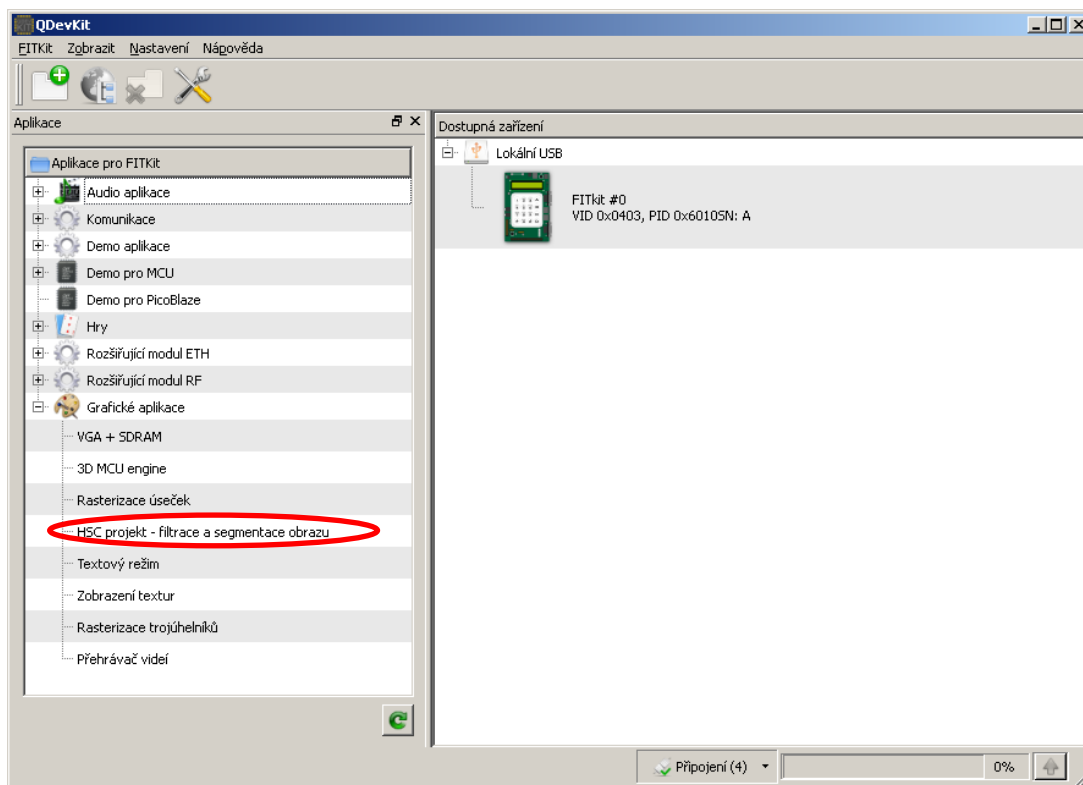
Postup pro spuštění aplikace na přípravku FITkit je následující:

1. Připojte FITkit k počítači.
2. Spusťte program VirtualBox (nebo VMware) a připojte obraz virtuálního stroje. V nastavení USB zařízení povolte FTDI Dual RS232 (viz obrázek) a virtuální stroj spusťte.



Důležitá poznámka: Používejte VirtualBox verze 5 a vyšší. Starší verze nepodporují USB 3.0, a proto pokud máte na svém počítači pouze porty USB 3.0, potom se FITkit nebude korektně propagovat z vašeho systému až do virtuálního stroje.

3. Do adresáře `C:\FitkitSVM\apps\vga\` nakopírujte rozbalený obsah archivu *projekt.zip*.
4. Spusťte si nástroj QDevKit. V záložce aplikace by se Vám měla objevit nová položka s názvem *HSC projekt – filtrace a segmentace obrazu* (viz obrázek).
5. Nyní můžete aplikaci přeložit, simulovat, naprogramovat do přípravku FITkit a následně spustit.



Poznámka k výpisu textu na terminál FITkitu

Pro výpis textu na terminál FITkitu ze strany mikrokontroléru se používají funkce `term_send_str()`, `term_send_num()`, apod. Někdy se vám může stát, že vypisovaný text nevypadá dle vašich představ. Například se vytiskne zalomení řádku navíc nebo se dokonce na konci zpracování vypíše obsah nápovědy. Jedná se o chyby, které prozatím nejsou v aplikaci QDevKit odladěny. Jako alternativu můžete použít ovládání FITkitu z příkazové řádky skrze příkazy (spuštěné v adresáři, kde je umístěn soubor *project.xml*):

`gmake load` – zajišťuje naprogramování přípravku FITkit

`gmake term` – zajišťuje otevření terminálu a spuštění aplikace

Spojení s terminálem na závěr ukončíte kombinací klávesy CTRL+D.

Při spuštění těchto příkazů je potřeba mít program QDevKit uzavřen (v opačném případě dochází k soupeření o přípravek a možným chybám).

Další užitečné příkazy z příkazové řádky, skrze které lze např. řídit překlad nebo spouštět simulace lze nalézt na [webových stránkách Fitkitu](#). Mezi ty nejčastěji používané patří:

`fcmake` – vytvoření Makefile souborů pro překlad projektu (stačí provést pouze na začátku nebo při změně souboru *project.xml*)

`gmake` – překlad projektu tj. MCU i FPGA části

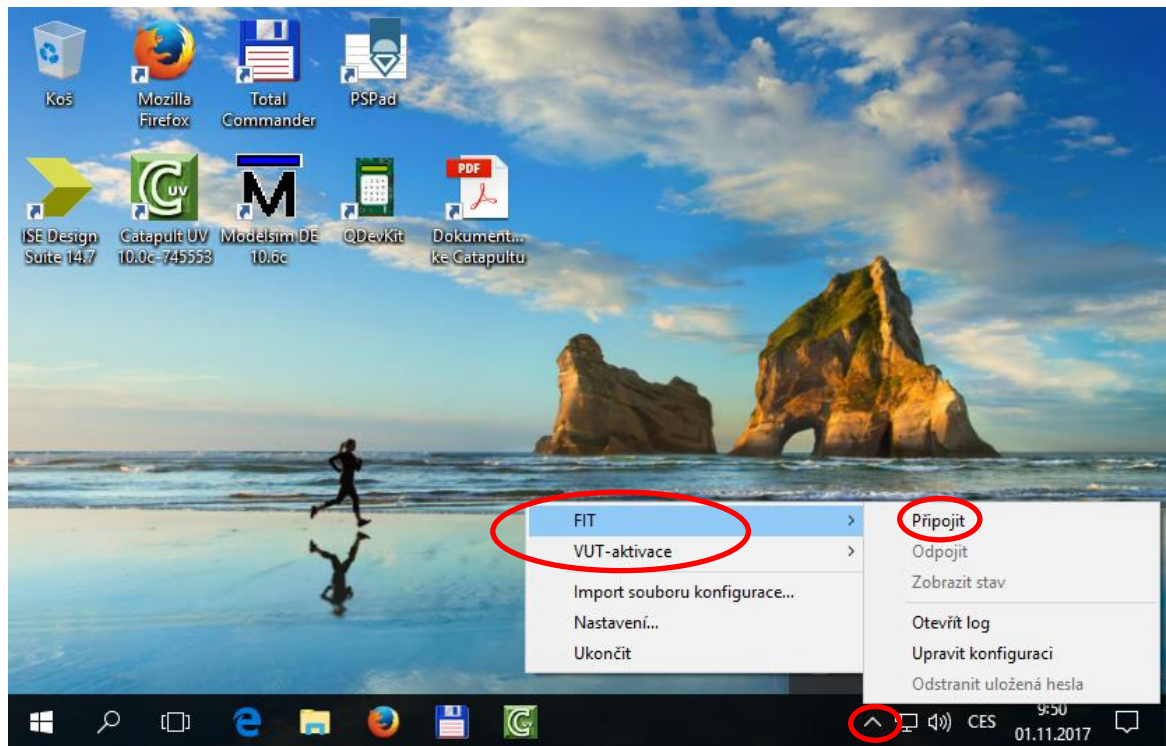
`gmake clean` – vyčištění přeložených souborů projektu

Syntéza souborů s využitím nástroje Catapult C

Překlad souborů v prostředí Catapult C s využitím skriptu *directives.tcl*

Pro syntézu souborů z jazyka C do jazyka VHDL a Verilog pomocí nástroje Catapult C postupujte následovně:

1. Pokud projekt řešíte z domova nebo mimo síť FIT VUT, potom je potřeba, abyste se nejprve přihlásili do sítě FIT skrze OpenVPN klienta. Tento klient se spustí automaticky po startu virtuálního stroje a najdete jej minimalizovaný v liště (viz obrázek). Pokud se vám nepodaří přihlásit přímo do sítě FIT, vyzkoušejte alespoň připojení do sítě VUT (volba VUT-aktivace). Připojení vašeho počítače do privátní sítě FIT nebo VUT je nezbytné pro spouštění programů typu Catapult C, nebo ModelSIM, které vyžadují připojení k licenčnímu serveru *semik.fit.vutbr.cz* a ten není mimo síť VUT dostupný.



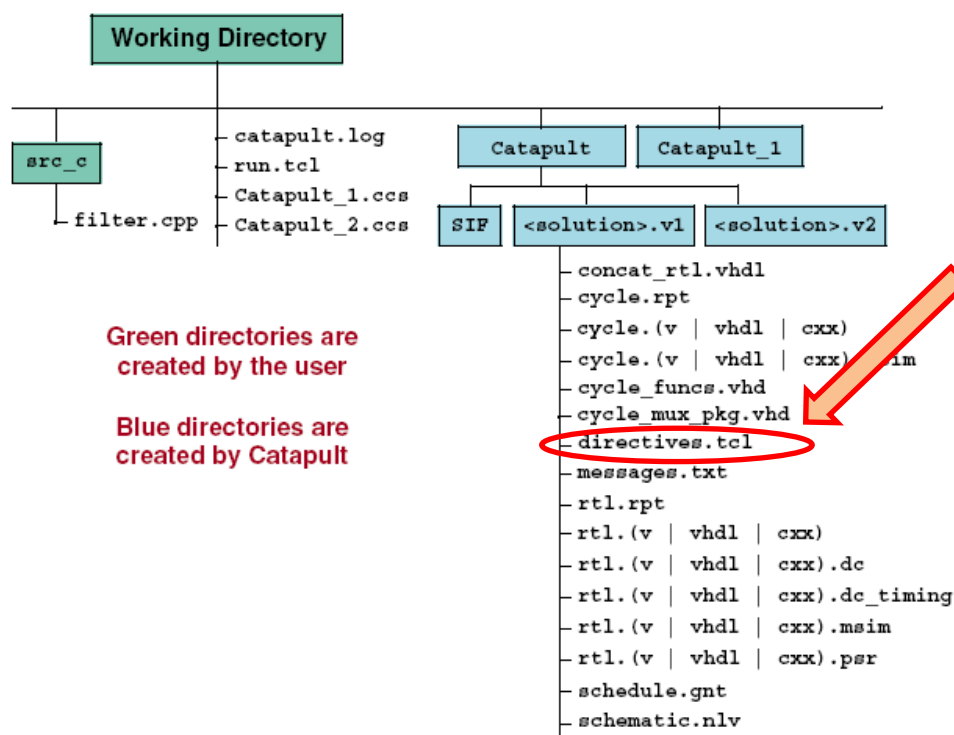
2. Spusťte si nástroj Catapult C. V *Task Baru* zvolte volbu *Set Working Directory* a nastavte jej na adresář *C:\FitkitSVN\apps\vga\segmentace\fpga*. Zde se vám budou vytvářet podadresáře s názvem CatapultX a v nich ukládat všechny výstupní soubory překladu.
3. V hlavním menu zvolte možnost *File → Run Script...* a v příslušném podadresáři se zdrojovými kódy (např. *src_filter/*) spusťte předpřipravený skript s názvem *directives.tcl*, který zajistí kompletní překlad zdrojových souborů a vytvoří výstupní VHDL a Verilog soubory s názvy *rtl.vhdl*, *rtl.v*, *concat_rtl.vhdl* a *concat_rtl.v*. Přesné umístění těchto výstupních souborů vám program vypíše na posledních řádcích okna transkriptu. Soubor *concat_rtl.v* si zkopírujte do adresáře k původním zdrojovým kódům (např. *src_filter/*).
Poznámka: soubor *concat_rtl.v* reprezentuje výsledný kód obvodu v jazyce Verilog, který současně zahrnuje i všechny knihovní komponenty potřebné pro následnou syntézu obvodu s využitím nástroje Xilinx ISE (odtud předpona *concat_** – konkatenace). Ekvivalentní soubor s názvem *concat_rtl.vhdl* bohužel nelze použít, jelikož obsahuje některé generické konstrukce jazyka VHDL, které nepodporuje nástroj Xilinx ISE.
4. Nyní lze v prostředí QDevKit spustit překlad aplikace a navazující fáze pro ověření aplikace na FITkitu.

Struktura výstupních souborů

Výstupní soubory Catapult ukládá do pracovního adresáře, kde si navíc vytváří podadresáře podle názvu projektu, obvykle ve tvaru Catapult_X (pokud není zadáno jinak).

Při každé změně zdrojových souborů nebo parametrů obvodu vytvoří Catapult nové řešení a nový podadresář `<solution>.vX`, kde ukládá výstupní soubory (viz obrázek). Poznámka: pokud se vás Catapult po změně vstupních souborů zeptá, zda vytvořit nové řešení, potom tuto volbu potvrďte. Podobně při každé změně parametrů architektury (rozbalení smyčky apod.) vytvoří Catapult nové řešení. Názvy těchto řešení zahrnují číslo verze a mezi těmito verzemi lze v Catapultu snadno přepínat skrze záložku *Table*.

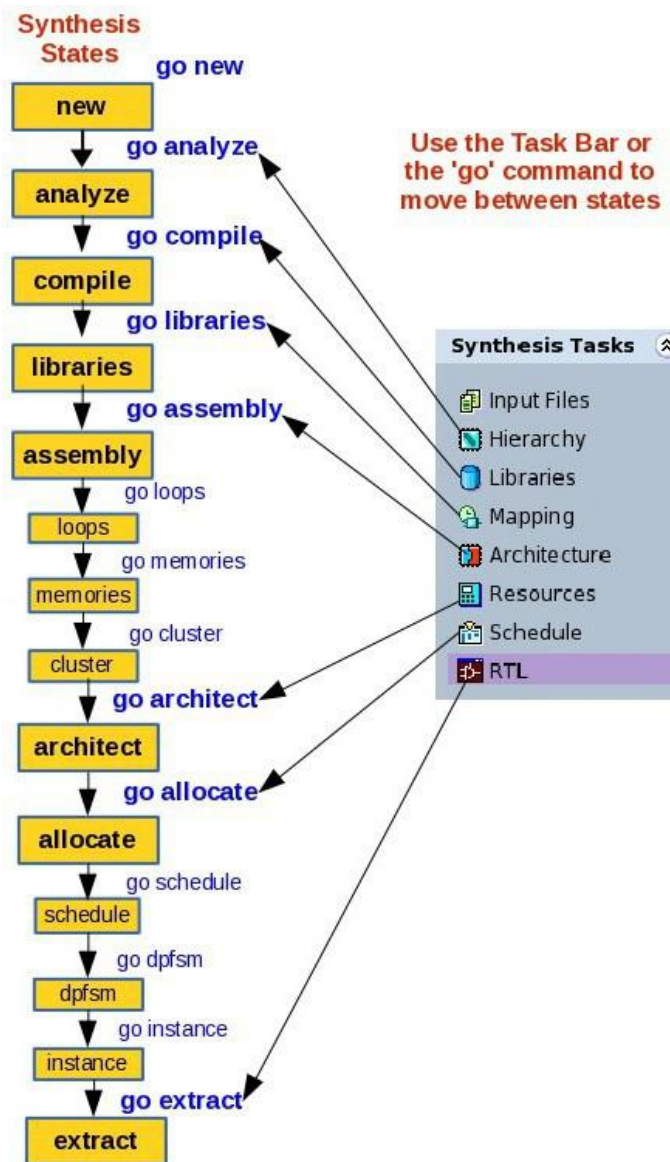
V rámci výstupních souborů vytváří Catapult také soubor s názvem *directives.tcl* obsahující všechny doposud provedené akce a nastavení v GUI, který může být dále použit jako šablona překladového skriptu (více informací viz následující kapitola).



Struktura souboru *directives.tcl*

Překlad souborů z jazyka C/C++ do jazyka VHDL a Verilog lze realizovat buď skrze grafické uživatelské rozhraní nebo skrze překladové skripty (obvykle) pojmenované jako *directives.tcl*. Překlad skrze GUI je výhodný zejména v první fázi, kdy návrhář zakládá nový projekt a nastavuje v něm celou řadu parametrů. Opakovaný překlad již nastaveného projektu je naopak výhodnější realizovat skrze překladový skript *directives.tcl* (není potřeba opakovaně klikat v GUI).

Soubor *directives.tcl* je rozdělen do několika sekcí, které začínají klíčovým slovem *go* a tyto sekce odpovídají jednotlivým fázím překladu spouštěným v GUI (viz obrázek). Do jednotlivých sekcí se následně vkládají příkazy a nastavují parametry, které jsou typické pro danou fázi překladu a jejich ekvivalent bychom našli i v některém z dialogových okének v GUI.



Kompletní seznam a popis příkazů a direktiv pro jednotlivé sekce lze nalézt v dokumentu *Catapult Synthesis User and Reference Manual*. Naštěstí není potřeba všechny příkazy a direktivy explicitně studovat, protože jedním z výstupních souborů Catapultu je i soubor *directives.tcl*, který odpovídá všem akcím, které jste v rámci projektu a konkrétní vývojové větve provedli (naklikali v GUI) a můžete jej použít jako základní šablonu pro váš vlastní skript. Tento skript pak můžete libovolně upravovat a spouštět skrze menu *File* → *Run Script...*

Simulace a verifikace komponent

Pro jednotlivé komponenty systému jsou pro vás v projektu předpřipraveny testbench soubory, které ověřují jejich základní funkčnost. Současně s testbench soubory v adresářích také naleznete soubory *directives_sim.tcl*, které slouží pro překlad komponenty a její následnou simulaci. Tento skript kromě přípravy souborů a jejich překladu také aktivuje prostředí *SCVerify* a spustí simulaci. Konkrétně se jedná o simulaci originálního kódu oproti modifikovanému kódu, která ověří funkčnost vašeho řešení na úrovni C/C++ kódu. V rámci prostředí Catapult lze provést také simulaci/verifikaci modifikovaného kódu oproti RTL schématu, které ověřuje funkčnost vašeho řešení na úrovni VHDL kódu. Tento typ simulace však pro vás není explicitně připraven a věřím, že jej nebudete ani potřebovat, jelikož simulace/verifikace na úrovni C/C++ kódu je obvykle dostačující.

Výstupy simulace na úrovni C/C++ kódu se vám budou zobrazovat přímo v terminálovém okně nástroje Catapult. Pokud v rámci ladění vašeho kódu budete potřebovat upravit vaše kódy a simulaci opět spustit, potom to můžete provést jedním z následujících způsobů:

- Opětovně spustit skript *directives_sim.tcl* skrze menu *File* → *Run Script...*

- Přeložit modifikované kódy do fáze *Libraries* a následně spustit simulaci výběrem volby *Project Files* → *Verification* → *MS Visual C++ 9.0* → *Original Design + Testbench*. Poznámka: spouštění dalších fází *Mapping*, *Architecture*, *Schedule* a *RTL* by vás zbytečně zdržovalo.

V průběhu testování vašeho modifikovaného kódu oproti originálnímu kódu, můžete pocítit potřebu vypisovat si určité ladící informace ze strany modifikovaného kódu. Obvykle toto nelze přímo realizovat, neboť modifikovaný kód je součástí kompilace výsledného obvodu a na úrovni obvodu nelze používat funkce typu *printf* nebo *cout*. Tento problém lze však elegantně vyřešit skrze podmíněný překlad a direktivu kompilátoru *#ifdef* (viz následující příklad):

```
#ifdef CCS_DUT_SYSC
    cout << "Zprava pouze pro ucely verifikace" << endl;
#endif
```

Neúspěšný překlad aplikace v prostředí QDevKit

V některých případech se vám v prostředí QDevKit nemusí podařit aplikaci úspěšně přeložit. Pokud v terminálu FITkitu objevíte hlášení typu „*WARNING: design did not meet timing*“, potom to znamená, že se nepodařilo rozmístit jednotlivé elementy obvodu tak, aby dodrželi frekvenci 25MHz (jedná se o fázi překladu *Place and Route*). Jedním z důvodů může být také to, že nástroj Catapult C neodhadl správně zpoždění jednotlivých operací v CDF grafu (princiálně to ani přesně udělat nelze) a naplánoval jich do jednoho C-stepu příliš mnoho. Vyřešit tento problém lze například tak, že v nastavení Catapultu zvýšíme procento režie pro C-step. Toto lze nejlépe provést úpravou následujícího řádku v souboru *directives.tcl*:

```
directive set -CLOCK_OVERHEAD 40.000000
```

Výchozí hodnota je u jednotlivých komponent nastavena na 40%, což znamená, že 40% z periody hodinového signálu (tj. 40% z 40ns pro frekvenci 25MHz) bude rezervováno pro případné zpoždění vzniklé na vodičích při rozmísťování hradel na FPGA technologii a 60% z periody hodinového signálu bude použito pro umístění operací z CDF grafu a jejich mapování do C-steps.

Pokyny pro vypracování technické zprávy

Technická zpráva musí obsahovat následující části:

1. **Hlavička** obsahující *název předmětu, školní rok, jméno, příjmení a login*.
2. **Tabulka a graf výsledků analýzy algoritmu z programu gprof**. V tabulce pro jednotlivé funkce uveďte, kolik procent času strávil procesor ve funkci z celkové doby výpočtu. Hodnoty z tabulky vynesete do sloupcového grafu, kde každá funkce bude reprezentována jedním sloupcem.
3. **Tabulka shrnující vlastnosti obvodu uvnitř FPGA**.
 - a. Uveďte vlastnosti vaší komponenty pro filtraci a segmentaci obrazu (soubor `fpga/src_filter/filter.cpp`), zejména inicializační interval hlavní smyčky a latenci obvodu (lze odečíst z Ganttova diagramu).
 - b. Na základě výsledků syntézy uveďte množství spotřebovaných zdrojů FPGA čipu v počtech *Flip Flops*, *LUTs* a *Slices*. Tyto informace najdete v souboru `build/fpga/med_filtr.map.mrp`.
4. **Tabulka porovnávající vlastnosti čistě softwarové implementace na mikrokontroléru MSP430 a implementace rozdělené mezi hardware a software**. V této tabulce uveďte pro obě realizace:
 - a. Průměrnou dobu pro zpracování jednoho pixelu. Pozn.: U hardwarového řešení lze tuto hodnotu odvodit přesně z hodnoty latence obvodu (viz Catapult) a frekvence 25MHz.
 - b. Počet bodů zpracovaných za vteřinu. Pozn.: U hardwarového řešení lze tuto hodnotu odvodit z hodnoty propustnosti obvodu (viz Catapult) a frekvence 25 MHz.
 - c. Hodnotu zrychlení, která vyjadřuje kolikrát je řešení využívající hardware rychlejší oproti čistě softwarovému řešení. Ve sloupci u softwarového řešení uveďte hodnotu 1.
5. **Shrnutí** – zhruba pětirádkový komentář k dosaženým výsledkům. Při hodnocení systému berte v úvahu nejen rychlost zpracování, ale i potřebnou velikost čipu, cenu nebo předpokládanou spotřebu výsledného řešení. Zamyslete se, co je úzkým místem navrženého systému, a jakým způsobem by bylo možné zpracování ještě více urychlit, případně za jakou cenu. Pokud se vám nepodaří zadání zcela splnit, uveďte prosím, kterých bodů se to týkalo.

Celá technická zpráva by svým rozsahem neměla překročit 2 strany formátu A4.