Univerzitet u Beogradu - Elektrotehnički fakultet
Digitalni VLSI sistemi (13M041DVS)

Ubrzanje skaliranja slike

Projektni zadatak

Student:
Lazar Premović 2023/3020

Beograd, februar 2024.

Sadržaj

1 Arhitektura sistema

Na najvišem nivou, implementirano rešenje se sastoji od *System on Chip*-a koji se pokreće na **FPGA** čipu, eksternih periferija i softvera koji se izvršava na *soft* procesoru unutar njega. Od eksternih periferija korišćena je **SDRAM** memorija i jedan taster. Taster se koristi za ručno resetovanje sistema, dok se u **SDRAM** memoriji nalazi programski kod i podaci.

1.1 Arhitektura SoC-a

Jezgro SoC-a je dizajnirano korišćenjem *Platform Designer* alata koji je deo *Quartus* softverskog paketa i omogućava lako instanciranje i povezivanje gotovih i ručno napravljenih **IP** blokova.

Dizajn koristi sledeće IP blokove:

- *Clock Source*: Predstavlja ulaznu tačku za eksterni signal takta (50 MHz) i reset signal.
- *PLL Intel FPGA IP*: Sintetiše signale takta za sistem i **SDRAM** čip korišćenjem fazno zaključanih petlji.
  I sistem i **SDRAM** koriste signal takta frekvencije (100 MHz) radi bojlih performansi a nakon provere maksimalne frekvencije sistema vremenskom analizom i **SDRAM** čipa uvidom u dokumentaciju. Dodatno signal takta za **SDRAM** prednjači 3ns (vrednost je uzeta iz dokumentacije razvojne ploče).
- *SDRAM Controller Intel FPGA IP*: Omogućava pristup eksternoj **SDRAM** memoriji preko interne *Avalon* magistrale.
  SDRAM čip ima širinu reči od 16 bita, 4 banke sa po 8192 redova i 1024 kolone po redu. Vremenski parametri **SDRAM** modula su uzeti iz dokumentacije umesto sa materijala sa vežbi, što je doprinelo dodatnom poboljšanju performansi.
- *Nios II Processor*: Izvršava kod koji omogućava tražene funkcionalnosti sistema.
  U pitanju je *Nios II/e* varijanta kojoj su vektori za reset i obradu izuzetka podešeni da se nalaze na **SDRAM** čipu.
- *JTAG UART Intel FPGA IP*: Omogućava pristup **UART** konzoli procesora preko **JTAG** konekcije korišćene sa spuštanje **FPGA bitstream**-a kao i programa i njegovo debagovanje.
- *Performance Counter Unit Intel FPGA IP*: Omogućava precizno merenje vremena izvršavanja više segmenata programa.
- *Scatter-Gather DMA Controller Intel FPGA IP*: Ovaj **DMA** kontroller se nalazi u *Memory to Stream* modu i koristi se za prenošenje podataka iz memorije u akcelerator.
  Scatter-Gather **DMA** kontroller je neophodan jer se prilikom skaliranja segmenta slike susedni redovi ne nalaze na kontinualnim memorijskim lokacijama. Širina podataka *streaming* interfejsa je 8 bita kako bi bila uparena sa širinom *sink streaming* interfejsa akceleratora.
- *Scatter-Gather DMA Controller Intel FPGA IP*: Ovaj **DMA** kontroller se nalazi u *Stream to Memory* modu i koristi se za prenošenje podataka iz akceleratora u memoriju.
  Scatter-Gather **DMA** kontroller je poželjan jer maksimalna veličina slike može prevazići maksimalnu veličinu **DMA** prenosa. Širina podataka *streaming* interfejsa je 8 bita kako bi bila uparena sa širinom *source streaming* interfejsa akceleratora.

- *acc_scale*: Predstavlja sam akcelerator koji ubrzava operaciju skaliranja slike. O arhitekturi samog akceleratora će biti reči kasnije.

Više detalja o sistemu je moguće videti u priloženom `.qsys` fajlu.
Opisan sistem je potom instanciran u *top-level* shemi koja pinove sistema direktno povezuje sa odgovarajućim pinovima samog sistema.

1.2 Arhitektura akceleratora

Pored reset signala i signala takta, interfejs akceleratora čini jedan generički parametar i tri *Avalon* interfejsa.
Generički parametar određuje maksimalnu širinu ulazne slike i ima podrazumevanu vrednost 1024. Treba napomenuti da softver nema način da vrednost ovog parametra sazna za vreme izvršavanja ali da je odgovornost na softveru da poštuje ograničenja veličine slike koju šalje akceleratoru. Ovo se potencijalno može rešiti dodavanjem *read-only* polja u kontrolni i statusni registar koje sadrži ovu informaciju.

Prenos podataka u i iz akceleratora se vrši putem dva *Avalon Streaming* interfejsa (jedan *sink* i jedan *source*), širina linije podataka oba interfejsa je 8 bita što je ekvivalentno širini jednog simbola (piksela), te *start of packet* i *end of packet* signali nemaju praktično značenje i mogu biti ignorisani.

Akcelerator takođe poseduje i *Avalon memory-mapped* interfejs koji omogućava pristup konfiguracionim registrima akceleratora.
Registarska mapa je prikazana ispod:

Addr	31..16	15..6	5	4	3	2	1	0
0x0	Reserved		yUS	ySc		xUS	xSc	
0x4	height	width						

Polja *height* i **width** sadrže visinu i širinu ulazne slike respektivno, ovo predstavlja još jedno ograničenje, te ulazna slika ne može biti veća od 65535 x 65535 nezavisno od generičkog bafera.

Polja *yUS* i *xUS* označavaju da li še vrši smanjivanje (vrednost 0) ili povećavanje (vrednost 1) slike po y i x osi respektivno.

Polja *ySc* i *xSc* sadrže enkodovan faktor skaliranja po z i x osi respektivno.
Faktori skaliranja se enkoduju na sledeći način:

ySc/xSc		Faktor Skaliranja
1	0	
0	0	1
0	1	2
1	0	3
1	1	4

Iako nije tražena u postavci projekta, mogućnost nezavisnog definisanja faktora skaliranja po x i y osi omogućava dodatne softverske optimizacije o kojima će biti više reči u sekciji o softveru.

1.3 Arhitektura softvera

Softver se primarno sastoji od *Board Support Package*-a i same aplikacije koja sadrži poslovnu logiku. Kako je *Board Support Package* automatski generisan na osnovu definicije sistema, ovaj izveštaj neće ulaziti u detalje o njemu.

Aplikacija se sastoji od 4 komponente (svaka odvojena u zaseban `.c` fajl sa opcionim `.h` fajlom), to su:

- `main`: Sadrži glavni program i funkcije koje se tiču parsiranja komandi, učitavanja slika, priprema komandi i izvršavanja komandi pokretanjem odgovarajućih implementacija.
- `benchmark_utils`: Sadrži pomoćne funkcionalnosti za automatizovano i polu-automatizovanu evaluaciju performansi i korektnosti.
- `sv_impl`: Sadrži implementaciju skaliranja slike isključivo korišćenjem softvera.
- `hw_impl`: Sadrži implementaciju skaliranja slike korišćenjem hardverskog akceleratora.

`main` i `benchmark_utils` su specifični za konkretnu implementaciju te će o njima biti više reči u sekciji o realizaciji softvera, dok će u nastavku biti opisan interfejs kojim druge aplikacije mogu da inkorporiraju ove metode skaliranja slika.

Sve funkcije za skaliranje imaju veoma sličan potpis koji će najpre biti opisan a potom će biti skrenuta pažnja na specifičnosti pojedinih funkcija.

```
#ifndef SW_IMPL_H_
#define SW_IMPL_H_

void scaleSW(unsigned char* source, unsigned char* destination, int
   sourceWidth, int sourceHeight, int x, int y, int width, int
   height, int destinationWidth, int destinationHeight, int xScale,
   int yScale);

#endif /* SW_IMPL_H_ */
```

```
#ifndef HW_IMPL_H_
#define HW_IMPL_H_

#include <altera_avalon_sgdma.h>

typedef struct
{
        int status;
        alt_sgdma_dev* txHandle;
        alt_sgdma_dev* rxHandle;
        alt_sgdma_descriptor* mallocPtr;
        alt_sgdma_descriptor* descPtr;
        volatile alt_32 txDone;
        volatile alt_32 rxDone;
} HWContext;

void printHWError(HWContext* ctx);
```

```
void cleanupHW(HWContext* ctx);
int checkHW(HWContext* ctx);
void initHW(HWContext* ctx);
void scaleHW(HWContext* ctx, unsigned char* source, unsigned char*
    destination, int sourceWidth, int sourceHeight, int x, int y, int
     width, int height, int destinationWidth, int destinationHeight,
    int xScale, int yScale);
void scaleHSCD(HWContext* ctx, unsigned char* source, unsigned char*
     destination, int sourceWidth, int sourceHeight, int x, int y,
    int width, int height, int destinationWidth, int
    destinationHeight, int xScale, int yScale);

#endif /* HW_IMPL_H_ */
```

Argumenti zajednički za sve funkcije skaliranja slike su:

- `unsigned char* source`: Pokazivač na početak izvorišne slike
- `unsigned char* destination`: Pokazivač na početak memorije alocirane za rezultujuću sliku
- `int sourceWidth`: Širina izvorišne slike
- `int sourceHeight`: Visina izvorišne slike
- `int x`: X koordinata gornjeg levog ugla segmenta slike za skaliranje
- `int y`: Y koordinata gornjeg levog ugla segmenta slike za skaliranje
- `int width`: Širina segmenta slike za skaliranje
- `int height`: Visina segmenta slike za skaliranje
- `int destinationWidth`: Širina rezultujuće slike
- `int destinationHeight`: Visina rezultujuće slike
- `int xScale`: Faktor skaliranja po X osi
- `int yScale`: Faktor skaliranja po Y osi

Faktori skaliranja moraju biti iz opsega {-4, -3, -2, -1, 1, 2, 3, 4} gde negativni brojevi označavaju umanjenje a pozitivni uvećanje.

Softversko skaliranje je moguće pokrenuti samo korišćenjem ovih argumenata, dok pokretanje hardverskog skaliranja zahteba i pokazivač na `HWContext` kao argument.

`HWContext` sadrži sve deljene podatke koji nisu u direktnoj vezi sa nekom konkretnom operacijom hardverskog skaliranja i potrebno ga je inicijalizovati pozivanjem funkcije `initHW`. Nakon završetka kompletne obrade (ne nakon obrade svake slike) potrebno je osloboditi resurse `HWContext`-a pozivanjem funkcije `cleanupHW`.

Hardversko skaliranje je moguće pokrenuti u dve varijante `scaleHW` i `scaleHSCD` koje koriste isključivo hardver za skaliranje ili koriste i hardver i softver za skaliranje kako bi dodatno poboljšali performanse.
Nakon poziva `scaleHW`, `scaleHSCD` ili `initHW`, neophodno je proveriti da li je došlo do greške prilikom njihovog izvršavanja pozivom funkcije `checkHW` koja će proveriti da li je do greške došlo i ukoliko jeste ispisati opis greške, osloboditi odgovarajuće resurce i vratiti povratnu vrednost 1, u suprotnom funkcija vraća 0 i nema druge efekte.

2 Realizacija hardvera

Kako je struktura celog sistema već opisana, ova sekcija će se baviti isključivo detaljima implementacije samog akceleratora.

Kompletan akcelerator se sastoji od 3 komponente i jednog pomoćnog fajla koji sadrži definicije komponenti i nekoliko pomoćnih funkcija.

2.1 image_counter

`image_counter` se sastoji od 4 registra koji pamte poziciju piksela, kao i broj ponavljanja po obe ose. Za to su neophodne informacije o faktoru skaliranja po obe ose, da li je u pitanju umanjenje ili uvećanje i dimenzije ulazne slike.

Kada dobije signal za pomeranje na sledeći piksel komponenta inkrementira broj ponavljanja piksela i pri uvećanju resetuje broj ponavljanja a inkrementira broj piksela ukoliko je broj ponavljanja dostigao faktor skaliranja, dok pri umanjenju uvek resetuje broj ponavljanja i inkrementira broj piksela za faktor skaliranja.

Slična logika sa inkrementiranjem i resetovanjem se primenjuje i ukoliko je broj piksela veći ili jednak širini ulazne slike.

Ova komponenta se koristi na dva mesta u akceleratoru, gde odbrojava trenutni piksel koji se šalje na izlazni tok (i tom prilikom koristi informacije o faktoru skaliranja iz konfiguracionih registara) i gde odbrojava trenutni piksel koji se učitava sa ulaznog toka (u tom slučaju faktori skaliranja su fiksirani na 1).

2.2 line_buffer

`line_buffer` predstavlja prostu implementaciju **RAM** memorije sa dva porta (jedan za čitanje i jedan za pisanje) u kojoj se čuvaju vrednost piksela trenutnog reda. Veličina memorije je specifikovana generičkim parametrom. `line_buffer` dva ulazna porta za adrese čitanja i pisanja, ulazni i izlazni port za podatke, kao i kontrolni signal za upis.

2.3 acc_scale

`acc_scale` je *top-level* komponenta koja sadrži dve gorenavedene instance `image_counter`-a i `line_buffer`. Pored toga, tu su implementirane sve potrebne konverzije tipova, konverzija iz enkodovanog u prirodni oblik faktora skaliranja, memorijski mapirani registri (na način sličan onom prikazanom na vežbama), kao i signali koji inkrementiraju i resetuju brojače piksela.

Brojači piksela se resetuju u trenutku kada su oba brojača van opsega slike, pri bilo kom upisu u konfiguracione registre ili reset signalom cele komponente. Signali za inkrementiranje su nešto kompleksniji ali se svode na inkrementiranje odgovarajućeg brojača kada su i *ready* i *valid* signali tog interfejsa aktivni.

Ready i *valid* signali ulaznog i izlaznog *streaming* porta respektivno se formiraju na osnovu toga da li se sme izvršiti upis ili čitanje piksela određenog brojača u bafer.

Pravila za čitanje su nešto prostija i glase:

$$Or < Ir || (Or == Ir \&\& Op < Ip)$$

Gde se O odnosi na brojač za čitanje, I na brojač za pisanje r na red piksela i p na poziciju piksela unutar reda.
Pravila za pisanje mogu biti podjednako prosta:

$$Or > Ir || (Or == Ir \&\& Op >= Ip)$$

I mogu se dodatno uprostiti jer ukoliko je $Or == Ir$, možemo slobodno učitati dati red do kraja bez brige da ćemo prepisati neki podatak koji je potreban. Pa dobijamo:

$$Or > Ir || Or == Ir$$

Dodatno je radi poboljšanja performansi dodat uslov koji omogućava upisivanje u bafer ukoliko je $Or < Ir$, ali samo ukoliko se upisuje neposredno sledeći red i taj red više neće biti potreban (vrši se umanjenje ili je brojač ponavljanja jednak faktoru skaliranja - 1) i taj piksel više nije potreban ($Op > Ip$).
Kompletan izraz za ove signali kao i još detalja implementacije se mogu naći u priloženom **VHDL** kodu.

3 Realizacija softvera

Arhitektura softvera je već prikazana te se u ovom delu nalaze samo detalji implementacije pojedinačnih funkcija.

3.1 main

`main` pre svega sadrži definiciju `Command` strukture koja sadrži sve informacije potrebne za izvršavanje jedne operacije skaliranja, to su ukratko: faktori skaliranja po x i z osi, koordinate i dimenzije segmenta koji se skalira, dimenzije i veličina ulazne i izlazne slike i tri pokazivača na ulazni i izlazne bafere za softversku i hardversku implementaciju.

Funkcija `printHelp` prosto ispisuje uputstvo za upotrebu progrma koje sadrži opis formata u kome se zadaju komande, dok funkcija `printError` ispisuje opis greške koja se dogodila na osnovu vrednosti `status` promenljive. `cleanup` oslobađa sve resurse koje je trenutna komanda alocirala i koristi se kao deo `checkCommand` koja proverava status i ukoliko je došlo do greške ispisuje opis greške i oslobađa resurse. Takođe povratna vrednost `checkCommand` funkcije je 1 ukoliko je došlo do greške što uz upotrebu `CCC` makroa omogućava automatsko preskakanje ostatka obrade komande, čim dođe do greške.

`parseCommand` učitava tekst komande koji je korisnik uneo i na osnovu njega popunjava neka od sledećih polja (u zavisnosti od unete komande): putanju do ulazne slike, poziciju i veličinu

segmenta za skaliranje, faktore skaliranja po x i y osi. Potom `loadImage` učitava ulaznu sliku prvo čitajući njene dimenzije i alocirajući bafer dimenzija same slike i učitavajući same vrednosti u taj bafer.

Nakon `parseCommand` komanda sadrži samo informacije koje je korisnik direktno uneo i to u formatu u kom su unete (koji ne mora biti u potpunosti validan), `prepareCommand` vrši te provere i popunjava ostatak informacija. Provere koje se vrše su: unet je validan faktor skaliranja, segment za skaliranje ne prelazi granice slike, izlazni baferi su uspešno alocirani. Polja koja se pritom popunjavaju su: koordinate segmenta ukoliko se koristi podrazumevani, koordinate donjeg desnog ugla segmenta, dimenzije i veličina izlazne slike, pokazivači na izlazne bafere koji su alocirani na osnovu veličine izlazne slike.

`resizeImage` vrši skaliranje na tri načina navedena u arhitekturi softvera, proverava da li je rezultat hardverskog skaliranja identičan rezultatu softverskog skaliranja i meri vreme svakog izvršavanja kako bi te informacije mogao da predstavi korisniku. Za menje vremena se koristi *performance counter* **IP** blok i pre svakog pokretanja skaliranja se prazni keš sa podacima. `saveImage` upisuje rezultujuću sliku na disk korišćenjem odgovarajuće pomoćne funkcije čija će implementacija biti detaljnije objašnjena kasnije.

`main` funkcija inicijalizuje *performance counter* kako bi njegova vrednost mogla da se koristi za konfigurisanje generatora slučajnih brojeva, inicijalizuje deljene resurse za hardversko skaliranje i ispisuje uputstvo za upotrebu. Potom se unutar `while` petlje ponavlja učitavanje komande, učitavanje slike, pripremaKomande, skaliranje slike, čuvanje rezultujuće slike i oslobađanje odgovarajućih resursa. Ukoliko se radi o automatskoj evaluaciji performansi, priprema komande, skaliranje slike i čuvanje slike su zamenjeni pozivom odgovarajuće komande koja vrši tu evaluaciju.

3.2 benchmark_utils

`benchmark_utils` sadrži funkcionalnosti za automatizovanu evaluaciju performansi, ona je realizovana izvršavanjem predefinisanih i nasumično generisanih operacija nad jednom učitanom slikom i proveru korektnosti skaliranja, kao i vremena izvršavanja svih verzija algoritma za skaliranje.

struktura `TestCase` sadrži polja za sve parametre koji definišu jednu komandu (faktori skaliranja i segment slike koji se skalira), kao i po tri polja za svako ponavljanje testa (broj ponavljanja je definisan makroom `TEST_REPEATS`) u kojima se beleži korektnost te operacije skaliranja i njeno vreme izvršavanja. Funkcija `verify` poredi `size` bajtova na koje pokazuju `reference` i `target` pokazivači i vraća broj različitih bajtova.

`generateTests` generiše predefinisane i nasumično generisane testove. Prvih `BENCH_CASES` testova je predefinisano i oni skaliraju celu sliku svim mogućim faktorima skaliranja, dok je sledećih `TEST_CASES` testova nasumično generisano i u tom slučaju svi parametri uzimaju u potpunosti nasumične validne vrednosti. Cilj predefinisanih testova je formiranje standardne baze testova za evaluaciju vremena izvršavanja, dok nasumični testovi proveravaju korektnost algoritama poređenjem sa softverskom implementacijom.

Funkcija `submitResult` vrši proveru korektnosti rezultata vodeći računa da pri prvom pokretanju softverskog skaliranja ne postoji slika sa kojom se može uporediti rezultat i taj rezulat upisuje u `TestCase` strukturu. Ova funckija takođe ispisuje mali indikator koji signalizira koji

testovi su do sada izvršeni. `submitTimes` ima sličnu ulogu u tome što beleži vremena izvršavanja ali sa razlikom da beleži vreme izvršavanja sve tri verzije odjednom. To je urađeno kako bi izbegli zaustavljanje *performance counter*-a za vreme testiranja do kojeg dolazi kada koristimo funkciju `perf_get_section_time` za dohvatanje vremena izvršavanja.

`writeImage` čuva prosleđenu sliku na disku zamenom ekstenzije prosleđenog imena u ".out"upisivanjem širine i visine prosleđene slike i konačno upisivanjem samih vrednosti piksela. `writeResult` koristi ovu funkciju kako bi upisao rezultat testa na disk, nakon formiranja imena izlazne datoteke uklanjanjem ekstenzije, dodavanjem parametara testa i ".out"ekstenzije.

`runTests` izvršava generisane testove, ali pre pokretanja testova alocira dva bafera (po jedan za hardversko i softversko skaliranje) maksimalne moguće veličine kako bi se uštedelo na vremenu koje bi bilo potrebno da se bafer alocira za svako pokretanje testa (iz istog razloga, kao i zbog ograničenog protoka **JTAG** interfejsa, svi testovi se izvršavaju nad istom slikom). Potom se redom izvršavaju svi testovi, kako bi bilo moguće pozvati funkcije za skaliranje, za svaki test je potrebno izračunati dimenzije izlazne slike. Svaki test se izvršava `TEST_REPEATS` puta za svaju verziju i svi rezultati i vremena se pamte. Ovaj segment koda je poprilično sličan funkciji `resizeImage` te neće biti detaljnije obrađivan. Nakon pokretanja svih verzija i ponavljanja testova, konačna rezultujuća slika može biti (u zavisnosti od vrednosti `WRITE_RESULT` makroa) sačuvana na disku za ručnu proveru.

Nakon izvršavanja svih testova, funkcija `writeResults` čuva parametre testova, vremena izvršavanja i rezultat provere korektnosti rezultata za svaki test u jednom `.csv` fajlu na disku.

3.3 sw_impl

`sw_impl` sadrži implementaciju softverskog skaliranja slike koja se sastoji od dve funkcije `scaleLineSW` i `scaleSW` koje skaliraju jednu liniju i celu sliku respektivno.
Sama implementacija je poprilično prosta, sa nekoliko detalja koji poboljšavaju performanse i koji će ovde biti detaljnije opisani.

`scaleLineSW` koristi odmotanu petlju umesto `for` petlje ili `memset` funkcije za ponavljanje piksela `xScale` puta. Iako bi bilo intuitivno da `memset` bude brža metora, upoređivanjem vremena izvršavanja je primećeno da se za ovako mali broj ponavljanja odmotana petlja brže izvršava čak i od `for` petlje, što znači da kompajler nije uspeo da je odmota na najbolji način.

Slična metoda se koristi i za ponavljanje redova, gde se red skalira samo prvi put korišćenjem `scaleLineSW` funkcije, dok se ostala ponavljanja dobijaju kopiranjem tog reda, slično odmotana kao i gore.

Sve petlje koje prolaze kroz redove ili piksele koriste više brojača kako bi se smanjio potreban broj operacija množenja i deljenja, što je imalo značajan uticaj na performanse. Treba na pomenuti da i dalje postoji nekoliko operacija množenja (u ekspanziji makroa `PIXEL`) ali da je njihovo uklanjanje rezultovalo u pogoršanim performansama.

3.4 hw_impl

`hw_impl` sadrži implementaciju hardveskog skaliranja slike koja se sastoji od tri glavne funkcije `initHW`, `scaleHW` i `scaleHSCD` koje incijalizuju deljenje resurse i skaliraju sliku korišćenjem samo hardvera ili i hardvera i softvera. Ostale pomoćne funkcije obuhvataju prekidne rutine

i pomoćne funkcije za oslobađanje resursa i proveru i ispis grešaka. Kako su one dosta slične onima objašnjenim u `main` podsekcicji, one ove neće biti detaljnije objašnjivane.

Funkcija `initHW` inicijalizuje deljene reusrse tako što popunjava sva polja `HWContext` strukture. Prvo se dohvataju ručke ka dva **DMA** kontrolera koja će biti korišćena i čuvaju u strukturi. Potom se alocira bafer koji će čuvati sve **DMA** deskriptore, ovaj bafer se alocira tako da u njega mogu stati svi deskpriptori potrebni za sliku maksimalne veličine (što je oko `5 * (BUFFER_SIZE + 1)` bafera), time se poboljšavaju performanse izbegavanjem alokacije bafera za svaku sliku. Nakon alokacije je neophodno poravnati bafer na veličinu jednog deskriptora. To je realizovano postavljanjem najnižih `log2(ALTERA_AVALON_SGDMA_DESCRIPTOR_SIZE)` bita na 0, ukoliko je bafer već bio poravnat pokazivač će ostati isti, međutim ukoliko nije bio poravnat novi pokazivač će biti manji i samim tim pokazivati van alocirane memorije, te se ukoliko je pokazivač manji on inkrementira za veličinu deskriptora. Na kraju incijalizacije se registruju prekidne rutine za oba **DMA** kontrolera.

Funkcija `scaleHW` vrši skaliranje isključivo koristeći hardver na očekivani način. Funkcija započinje proverom dimenzija slike, enkodovanjem faktora skaliranja i upisivanjem tih vrednosti u memorijski mapirane registre. Potom se počevši od početka bafera za deskriptore konstruišu deskriptori koji prenose po jedan red ulazne slike u akcelerator i nakon toga se setuje stop deskriptor koji zaustavlja **DMA** prenos. Odmah se u nastavku bafera na isti način konstruišu i deskriptori za prenos skalirane slike u memoriju. Nakon kreiranja deskriptora resetuju se indikatori koji označavaju kraj **DMA** prenosa i pokreću se oba **DMA** kontrolera. Program potom čeka da se oba prenosa završe i zaustavlja **DMA** kontrolere.

Pri umanjivanju slika ova funkcija se može dodatno ubrzati zato što **DMA** kontroleri prenose veliki broj piksela koji nisu deo izlazne slike. Kako sistem koristi *Scatter-Gather* **DMA** kontrolere, lako je preneti nekontinualne segmente memorje. Ovo ipak nije praktično primeniti na nivou piksela jer bi to rezultovalo u kreiranju po jednog deskriptora za svaki piksel rezultujuće slike, ali je perfektno za primenu na redove, te se kreiraju deskriptori samo za redove koji učestvuju u rezultujućoj slici. Što ne samo da smanjuje količinu prenetih podataka već smanjuje i broj deskriptora.

Upravo ova metoda je primenjena u funkciji `scaleHSCD` koja se na samo nekoliko mesta razlikuje od `scaleHW`. Prvo jeste kreiranje deskriptora za prenos iz memorije gde se konstruiše samo svaki `yScale`-ti deskriptor ukoliko je u pitanju umanjenje (za uvećanje je ponašanje isto kao `scaleHW`). Druga razlika je u vrednostima koje se upisuju u memorijski mapirane registre, kako se šalju samo redovi koji učestvuju u rezultujućoj slici, iz perspektive akceleratora to znači da je faktor skaliranja sada 1 a visina slike ista kao visina rezultujuće slike, te ove izmenjene vrednosti treba upisati u memorijski mapirane registre.

4 Ostali implementacioni detalji

Projekat sadrži još nekoliko detalja koji nisu direktno deo implementacije akceleratora niti softvera ali su bitni za rad projekta i njegovo uspešno reprodukovanje. Kako su ti fajlovi svakako priloženi, samo neki od njih će ovde biti ukratno opisani.

To su:

- *top.cdf*: Fajl koji sadrži definiciju **JTAG** lanca za korišćenu razvojnu ploču.
- *top.qsf*: Fajl sa podešavanjima *Quartus* softvera za sintezu.

- *DVSProj.sdc*: Fajl sa vremenskim parametrima koje *Quartus* koristi pri sintezi.
- *stp1.stp*: Fajl sa definicijom *SignalTap* integrisanog logičkog analizatora.
- *imshow.py*: Pajtoj skripta za pregled i konverziju slika sačuvanih na disku od strane soft procesora.

4.1 top.qsf

top.qsf sadrži veliki broj podešavanja, nemali broj kojih se odnosi na alokaciju pinova. Bitna podešavanja koja imaju uticaj na rezultujući hardver a nemaju podrazumevane vrednosti su: `OPTIMIZATION_MODE "AGGRESSIVE PERFORMANCE"` i `VHDL_INPUT_VERSION VHDL_2008` koje poboljšavaju vremenske karakteristike dizanja po ceni dužeg vremena sinteze i omogućavaju korićenje specifikacije **VHDL** jezika iz 2008. godine.

4.2 DVSProj.sdc

DVSProj.sdc sadrži vremenske parametre koji daju indikaciju *Quartus* softveru kako da rasporedi dizajn na **FPGA** čipu. Za ovaj projekat je definisan period eksternog signala takta, na osnovu kog *Quartus* sam određuje frekvenije generisanih signala takta. Za komunikaciju sa **SDRAM** memorijom je važno da kašnjenja između određenih signala budu kontrolisana te su i ti parametri uneti u `.sdc` fajl, vrednosti ovih parametara su dobijene iz dokumentacije korišćene razvojne ploče. Na kraju je reset signal koji je povezan na taster na razvojnoj ploči okarakterisan kao asinhroni signal kako ne bi učestvovao u evaluaciji vremenskih ograničenja.

4.3 imshow.py

Ova skripta omogućava pregled slika koje se šalju ili primaju sa soft procesora implementiranog u ovom projektu.

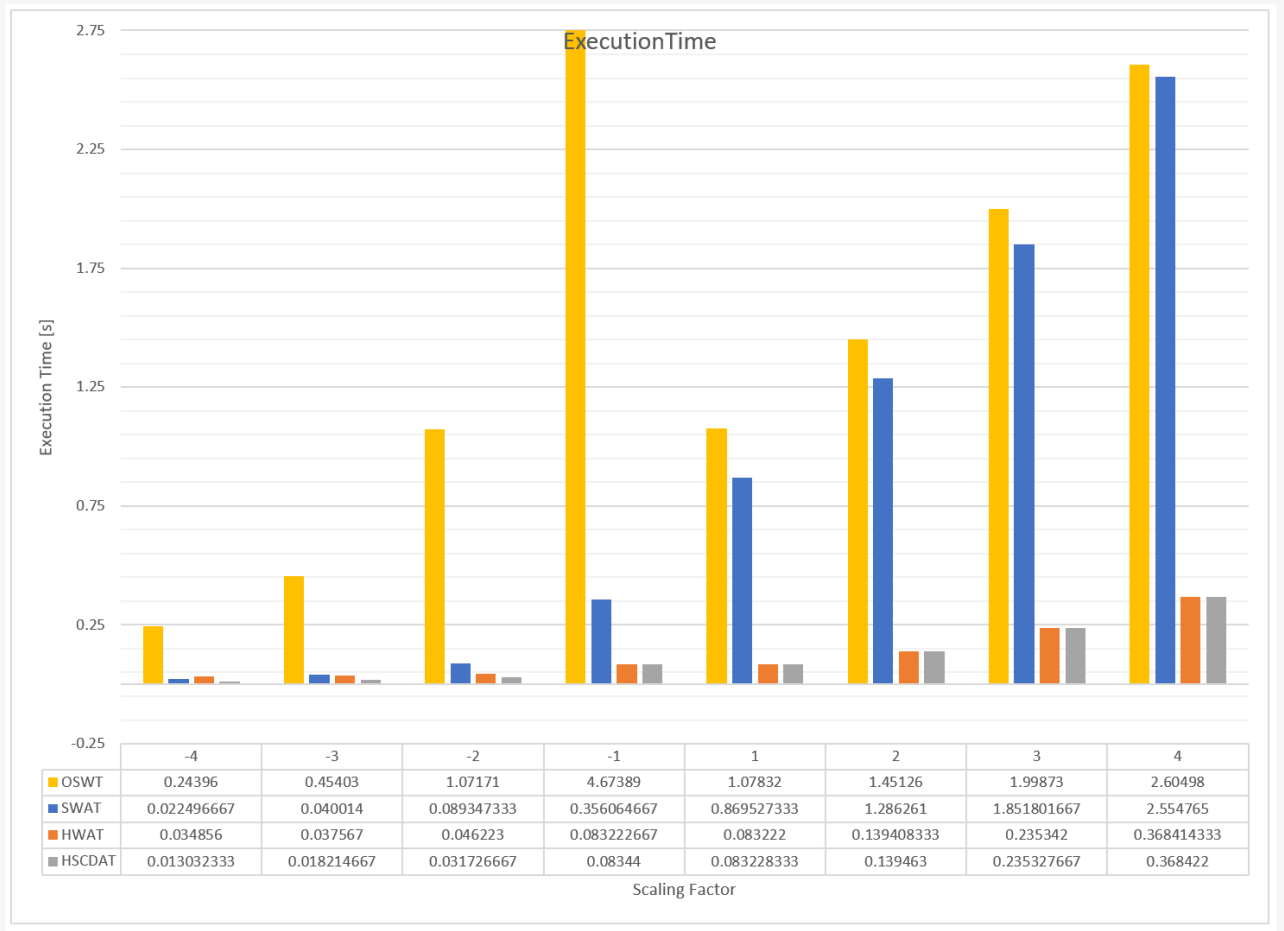
Ukoliko se skripta pokrene bez argumenata konvertovaće sve `.bin` i `.out` fajlove u `.png` slike po formatu u postavci projekta.
Ukoliko se skripti prosledi jedan argument skripta će prikazati sliku koja ja prosleđena kao argument i ukoliko postoji fajl sa istim nazivom ali `.out` ekstenzijom i on će biti prikazan pored prosleđenog fajla.

Dodatno je potrebno napomenuti da su i aplikacija i **BSP** projekat prevođeni sa najvišim nivoom optimizacije (`-O3`). Povećavanje nivoa optimizacije aplikacije je poboljšalo performanse i hardverske i softverske implementacije ali je imalo značajniji uticaj na softversku implementaciju, dok je povećanje nivoa optimizacije **BSP** projekta imalo uticaj isključivo na performanse hardverske implementacije.

5 Rezultati

Rezultati su prikupljeni korišćenjem funkcionalnosti automatske evaluacije performansi i potom su upoređeni sa nekoliko ranije obavljenih ručnih testova.
Pokrenuto je 8 predefinisanih i 50 nasumično generisanih testova nad datom slikom i svaki test je pokrenut nad svim varijantama (`scaleSW`, `scaleHW` i `scaleHSCD`) tri puta.
Detalji testa se mogu videti u priloženom `Excel` dokumentu a ovde će biti samo sumirani.

Svi testovi su izvršeni bez grešaka i sa rezultatom identičnim kao referentna softverska implementacija. Maksimalna devijacija vremena izvršavanja je tek nešto veća od 300 mikrosekundi.



	-4	-3	-2	-1	1	2	3	4
OSWT	0.24396	0.45403	1.07171	4.67389	1.07832	1.45126	1.99873	2.60498
SWAT	0.022496667	0.040014	0.089347333	0.356064667	0.869527333	1.286261	1.851801667	2.554765
HWAT	0.034856	0.037567	0.046223	0.083222667	0.083222	0.139408333	0.235342	0.368414333
HSCDAT	0.013032333	0.018214667	0.031726667	0.08344	0.083228333	0.139463	0.235327667	0.368422

Na grafiku se porede performanse jedne manje optimizovane softverske implementacije (žuto), optimitzovane softverske implementacije (plavo), hardverske implementacije (naranždasto) i hardversko/softverske implementacije (sivo) na 8 predefinisanih testova.

Sa grafika se može videti da za razliku od manje optimizovane verzije (koja ima veoma veliki skok za faktor skaliranja -1) optimizovana softverska verzija ima monotono lošije performanse što je faktor skaliranja veći i da je za faktor skaliranja -4 čak i brža od isključivo hardverske implementacije. Takođe je zanimljiva i velika razlika u performansama između faktora skaliranja -1 i 1 iako je rezultujuća slika identična.

Isključivo hardverska implementacija ima nešto lošije performanse od softverske za faktor skaliranja -4, skoro identične performanse za faktor skaliranja -3 i 2 do 4 puta bolje performanse za faktore skaliranja -2 i -1. Za pozitivne faktore skaliranja isključivo hardverska implementacija je 6 - 10 puta brža u odnosu na softversku i vreme izvršavanja takođe monotono raste. Takođe se može videti da je vreme izvršavanja za faktore -1 i 1 identično što je i očekivano.

Za negativne faktore skaliranja je hardversko/softverska implementacija ubedljivo najbrža gde je 40 - 25% brža od prve sledeće implementacije, dok za pozitivne faktore skaliranja ima identične performanse kao isključivo hardverska implementacija, što je i očekivano. I opet su performanse za faktore skaliranja -1 i 1 identične.