UNIVERZITET U BEOGRADU, ELEKTROTEHNIČKI FAKULTET, KATEDRA ZA ELEKTRONIKU

Avalon interfejsi i DMA prenos

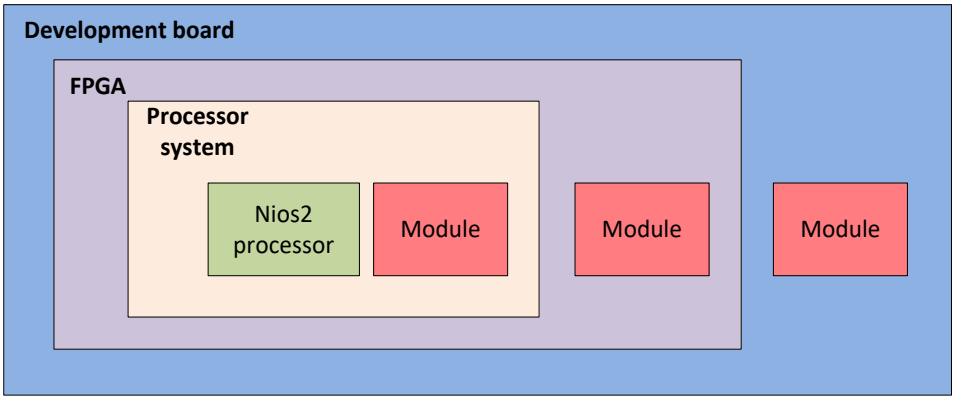
DIGITALNI VLSI SISTEMI

V2.0

BEOGRAD, 2018

Uvod

Na Slici 1. je prikazana blok šema namenskog sistema koji u sebi sadrži FPGA čip sa procesorskim sistemom na njemu. Hardverski moduli unutar ovog namenskog sistema mogu imati različite stepene integracije kao što je prikazano na Slici 1.



Slika 1. Integracija hardverskog modula u sistem

Hardverski modul može biti posebna diskretna komponenta koja se nalazi van samog FPGA čipa u okviru projektovanog namenskog sistema. Komunikacija između procesora i ove periferije se odvija preko IO pinova FPGA čipa. Drugi stepen integracije je da se hardverski modul nalazi unutar FPGA čipa ali izvan procesorskog sistema. U ovom slučaju nije potrebno koristiti IO pinove FPGA čipa ali je neophodno obezbediti interfejs iz procesorskog sistema (poput PIO modula) ka posmatranom hardverskom modulu s obzirom da procesor nema način da direktno pristupi tom modulu. Treći stepen integracije je da se hardverski modul nalazi unutar procesorskog sistema i da je povezan na magistralu podataka procesora. U ovom slučaju odgovarajući registri posmatranog modula se nalaze u okviru memorijske mape procesora i može im se direktno pristupiti.

Kako su slučajevi modula van procesorskog sistema bili pokriveni ranije, u ovom dokumentu akcenat je stavljen na integraciju hardverskog modula unutar procesorskog sistema.

Interfejsi Avalon magistrale

Avalon predstavlja magistralu koja se koristi u Alterinim procesorskim sistemima na čipu. Avalon magistrala sadrži različite tipove interfejsa koji omogućavaju upisivanje i čitanje kontrolnih registara i memorije, brz prenos velike količine podataka kao i kontrolu modula koji se nalaze van čipa. Ovi standardni interfejsi su već implementirani unutar komponenti koje se nalaze u Platfrom Designer-u tako da njihovo korišćenje u korisničkim periferijama omogućava njihovo jednostavno povezivanje i uklapanje u procesorski sistem. Sam Platform Designer alat je dizajniran tako da maksimalno automatizuje postupak integracije različitih komponenti u sistem automatskim generisanjem adresnih dekodera, arbitara magistrale, dinamičke promene širine magistrale i sl. U okviru ovog dokumenta će biti opisane ključne osobine 2 osnovna tipa interfejsa Avalon magistrale:

1) **Avalon Memory Mapped (Avalon-MM)** interfejs – koji se koristi za pristupanje memorijski mapiranim registrima i memorijskim modulima u sistemu
2) **Avalon Streaming (Avalon-ST)** interfejs – koji se koristi za brz prenos velike količine podataka

Detaljniji opis ovih interfejsa kao i opis ostalih interfejsa koji nisu spomenuti ovde može se pronaći u okviru dokumenta *Avalon Interface Specification* https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl_avalon_spec.pdf.

Avalon Memory Mapped

Avalon Memory Mapped (Avalon-MM) interfejs predstavlja interfejs *master-slave* tipa. Master moduli iniciraju cikluse čitanja ili upisa na magistrali dok slave moduli reaguju samo u slučaju da su adresirani od strane nekog master modula. Svaki slave modul povezan na master magistralu ima jedinstveni opseg adresa kojem master modul može pristupiti. U Tabeli 1. je data lista osnovnih signala Avalon-MM magistrale. Ova magistrala podržava i prenos podataka u burst režimu međutim ti signali nisu navedeni u ovoj listi i neće biti predmet razmatranja u ovom dokumentu.

Tabela 1. Lista osnovnih signala Avalon-MM magistrale

Naziv signala	Širina	Smer	Opis
address	1-64	Master→Slave	Master: Adresa (u bajtovima) kojoj se pristupa. Adresa mora predstavljati celobrojni umnožak dužine reči podataka. Za pristup pojedinačnim bajtovima koristi se ***byteenable*** signal. Slave: Sa strane periferije kojoj se pristupa adresa se prevodi u adresu (u rečima podataka) određenog registra unutar periferije.
byteenable	2,4,8,16, 32,64,128	Master→Slave	Omogućava selekciju samo određenih bajtova sa magistrale veće širine (koristi se za dinamičko prilagođavanje šrine magistrale). Svaki bit unutar ovog signala odgovara jednom bajtu reči podataka sa magistrale. Kada se aktivira više od jednog bita signala byteenable svi aktivirani biti moraju biti susedni i njihov broj mora biti stepen broja 2. Primer korišćenja ovog signala na 32-bitnoj magistrali: 1111 – upisuje se cela 32-bitna reč 0011 – upisuje se donjih 16-bita 1100 – upisuje se gornjih 16-bita 0001 – upisuje sa samo bajt 0 0010 – upisuje se samo bajt 1 0100 – upisuje se samo bajt 2 1000 – upisuje se samo bajt 3
read	1	Master→Slave	Aktiviranje ovog signala označava proces čitanja.
readdata	8,16,32,64,128 256,512,1024	Slave→Master	Podaci koje slave modul šalje master modulu prilikom procesa čitanja.
write	1	Master→Slave	Aktiviranje ovog signala označava proces upisa.
writedata	8,16,32,64,128 256,512,1024	Master→Slave	Podaci koje master modul šalje slave modulu prilikom procesa upisa.
waitrequest	1	Slave→Master	Signal kojim se kontroliše brzina komunikacije. U slučaju da slave nije u mogućnosti da trenutno obradi zahtev za upis ili čitanje, postavljen od strane mastera, aktivira ovaj signal i na taj način signalizira master modulu da čeka deaktiviranje ovog signala kako bi nastavio dalje sa radom.

Avalon Streaming

Avalon Streaming (Avalon-ST) interfejs predstavlja jednosmeran interfejs *point-to-point* tipa. Ovaj interfejs omogućava efikasan i brz prenos velike količine podataka između dva hardverska modula. Svaka veza između dva modula definiše izvorište (*source*) i odredište (*sink*) tako da protok podataka uvek ide od izvorišta ka odredištu.

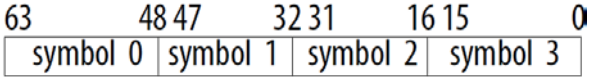
Tabela 2. Lista osnovnih signala Avalon-ST magistrale

Naziv signala	Širina	Smer	Opis
data	1-4096	Source→Sink	Podatak koji se prenosi od izvorišta ka odredištu.
ready	1	Sink→Source	Aktiviranje ovog signala označava da je odredište spremno da prihvati podatak. U slučaju aktivacije signala ready u toku ciklusa <n> odredište spremno da prihvati podatak u toku ciklusa <n+readyLatency>. Parametar readyLatency se podešava prilikom generisanja interfejsa.
valid	1	Source→Sink	Aktiviranje ovog signala označava da se izvorište postavilo validan podatak na data liniju i da je spremno za transfer. Prenos podataka između izvorišta i odredišta se odvija na svaki takt u kom je valid signal aktivan i odredište spremno da prihvati podatak (određeno ready signalom).
empty	1-5	Source→Sink	Označava broj nevalidnih simbola u trenutoj reči podataka. U slučaju da se u okviru reči podataka nalazi jedan simbol signal empty se ne koristi.
startofpacket	1	Source→Sink	Aktiviranje ovog signala označava početak prenosa paketa podataka.
endofpacket	1	Source→Sink	Aktiviranje ovog signala označava kraj prenosa paketa podataka.

Tabela 3. Lista osnovnih parametara Avalon-ST magistrale

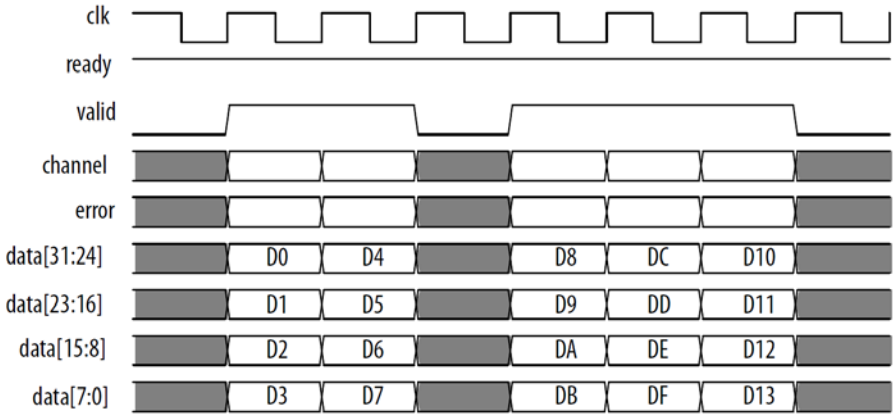
Naziv parametra	Podrazumevano	Opseg	Opis
symbolsPerBeat	1	1-32	Broj simbola koji se prenosi u svakom validnom ciklusu.
dataBitsPerSymbol	8	1-512	Veličina simbola koji se prenose u bitima.
firstSymbolInHighOrderBits	true	true, false	U slučaju da se prenosi više simbola određuje da li se prvi simbol smešta u više ili niže bite. Na primer ako se prenose 8 bitni simboli preko magistrale širine 32-bita onda se u slučaju postavljanja ove vrednosti na true, prvi simbol D0 smešta na linije data[31..24] dok se u suprotnom smešta na linije data[7..0].
readyLatency	0	0-8	U slučaju aktivacije signala ready u toku ciklusa <n> odredište spremno da prihvati podatak u toku ciklusa <n+readyLatency>.

Širina linije data određuje koliko će bita biti preneseno od izvorišta kao odredištu u svakom validnom ciklusu pri čemu su ti podaci organizovani u logičke celine koji se nazivaju simboli. Ovde je potrebno voditi računa da je Avalon-ST magistrala big endian tipa odnosno da će pri podeli veće reči na manje simbole prvi simbol predstavljati najviših N bita. Na Slici 2. je prikazana podela 64-bitne reči na 4 16-bitna simbola.



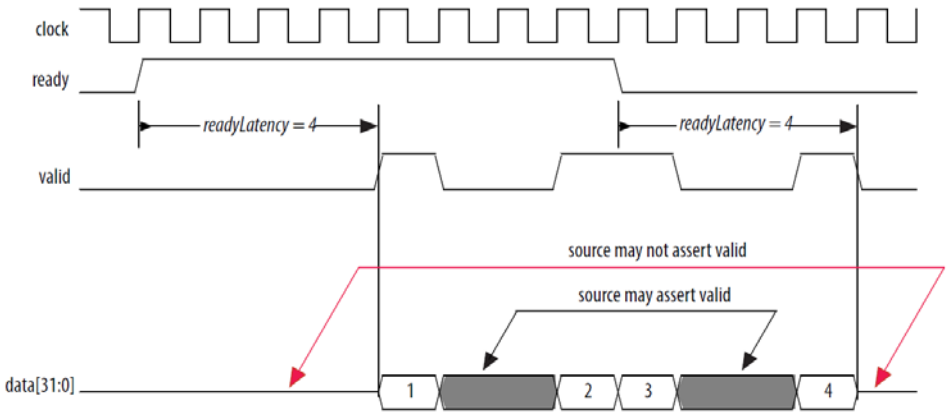
Slika 2. Podela 64-bitne reči na 4 16-bitna simbola

Prilikom prenosa podataka po Avalon-ST magistrali definišu se *ready* ciklusi kao oni ciklusi u kojima je odredišna periferija spremna da prihvati podatak sa magistrale podataka. Izvorišna periferija može za vreme *ready* ciklusa da aktivira *valid* signal i na taj način označi prenos podatka od izvorišta ka odredištu. Na Slici 3. je prikazan tipičan dijagram prenosa 8-bitnih simbola po 32-bitnoj magistrali podataka pri čemu je odredište konstantno spremno da prihvati podatak (*ready* je stalno na 1). U ovom primeru 4 simbola se prenose od izvorišta ka odredištu na svaku uzlaznu ivicu signala takta kada je signal *valid* aktiviran.

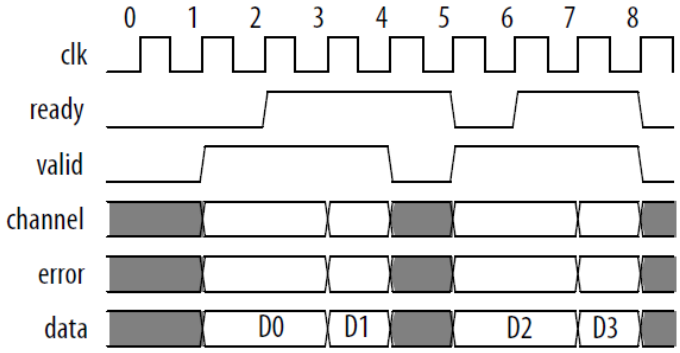


Slika 3. Tipični dijagram prenosa u slučaju 32-bitne magistrale podataka sa 8-bitnim simbolima

U slučaju da odredište ne može da prihvati pristižuće podatke potrebno je da deaktivira *ready* signal i na taj način signalizira izvorištu da se prenos pauzira. Vrednost parametra *readyLatency* označava broj taktova od aktiviranja/deaktiviranja signala *ready* do važenja nove vrednosti tog signala. Na Slici 4. je prikazan tipičan dijagram prenosa u slučaju kada je readyLatency podešen na 4. Izvorište bi trebalo da aktivira signal *valid* samo u okviru ready ciklusa dok u suprotnom treba da drži signal *valid* na 0. Ovo međutim ne važi u slučaju kada je readyLatency podešen na 0 kao što je prikazano na Slici 5. U tom slučaju izvorište postavlja vrednost valid kad god je spremno da pošalje neki podatak odredištu. Prenos se dešava na svaku ivicu signala takta u kojoj su i *valid* i *ready* na aktivnoj vrednosti.

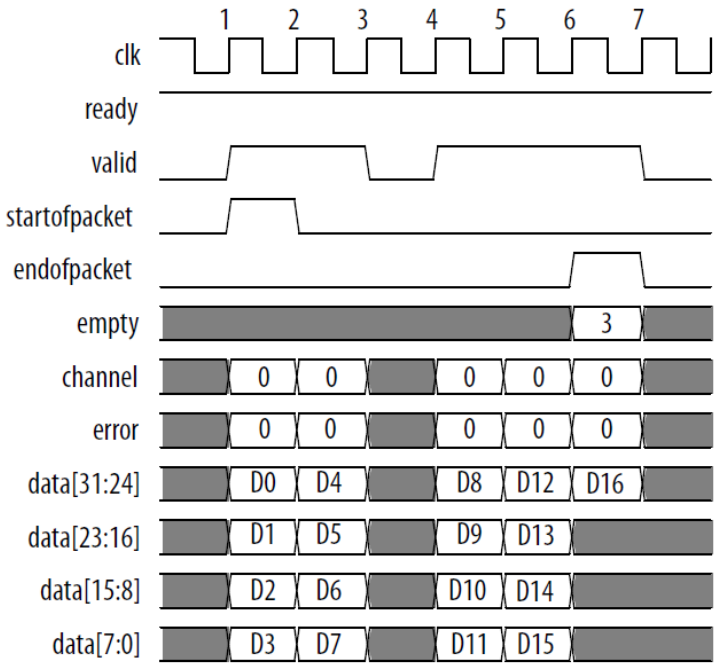


Slika 3. Tipični dijagram prenosa za readyLatency=4



Slika 4. Tipični dijagram prenosa za readyLatency=0

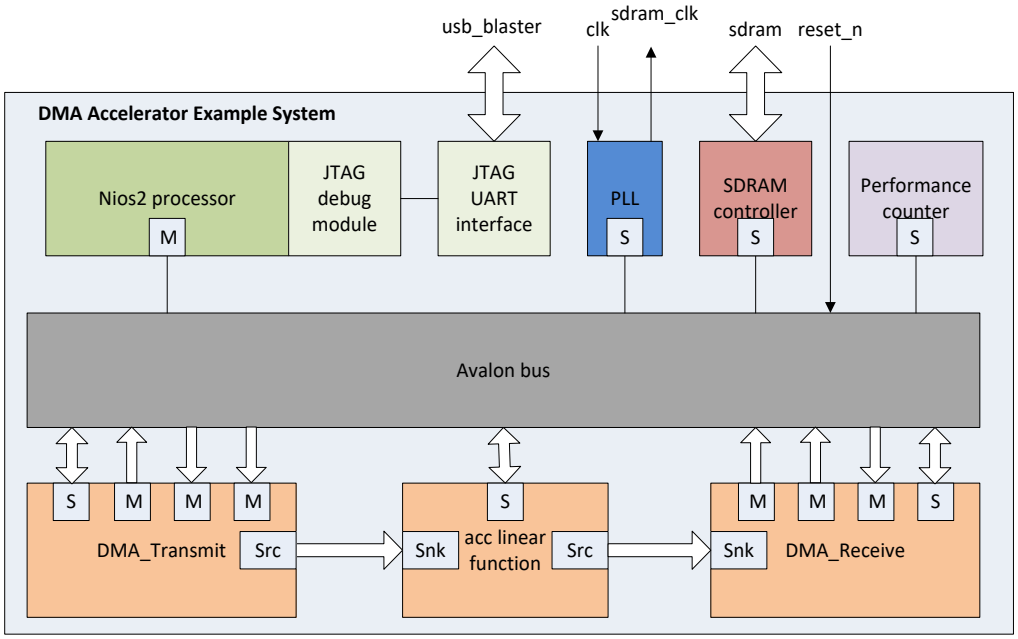
U slučaju kada se prenosi više povezanih simbola oni se mogu organizovati u okviru paketa. Početak i kraj paketa se označavaju posebnim signalima *startofpacket* i *endofpacket*. Vrednosti signala *startofpacket* i *endofpacket* se proveravaju jedino u trenucima kada je signal *valid* aktiviran. U slučaju da broj simbola koji se prenosi u trenutnom ciklusu manji od broja simbola po reči to se može naznačiti postavljanjem *empty* signala na odgovarajuću vrednost. Signal *empty* ima uticaja jedino u trenucima kada su aktivirani signali *valid* i *endofpacket*. Na Slici 5. je prikazan prenos paketa od 17 bajtova preko 32-bitne magistrale podataka. Kako se u poslednjem transferu prenosi samo jedan bajt signal *empty* je postavljen na vrednost 3 da označi da je od 4 simbola u okviru 32 bitne reči podataka samo 1 simbol validan.



Slika 5. Tipični dijagram prenosa za prenos jednog paketa od 17 simbola veličine 8 bita

Opis sistema

U svrhu demonstracije rada različitih interfejsa Avalon magistrale i integrisanja hardverskih modula unutar procesorskog sistema biće razmatrano ubrzanje sistema koji obavlja računanje linearne funkcije, Y = AX+B, velike količine podataka. Podaci se nalaze u eksternoj SDRAM memoriji u koju se smeštaju i rezultati izračunavanja. Linearna funkcija ulaznih podataka se može računati pomoću Nios2 procesora softverskim putem tako što je procesor zadužen za dohvatanje ulaznih podataka iz memorije, računanje izlaznih podataka primenom linearne funkcije i smeštanje izlaznih podataka na odredišnu adresu u SDRAM memoriji. Kako bi se ubrzalo procesiranja i procesor oslobodio za obavljanje drugih operacije, računanje linearne funkcije ulaznih odbiraka se može premestiti u hardver. Hardverski blok za računanje linearne funkcije mora imati 2 registra A i B koji čuvaju informaciju o koeficijentima linearne funkcije koja se računa, pri čemu je potrebno obezbediti programabilnost odnosno pristup ovim registrima sa procesora uvođenjem Avalon-MM slave porta. Dalje postavlja se pitanje na koji način dostavljati podatke hardverskom modulu. DMA kontroler može direktno pristupiti memorji preko Avalon-MM master porta i preneti te podatke ka hardverskom modulu za računanje linearne funkcije. Najefikasniji interfejs za prenos velike količine podataka između DMA i hardverskog modula je Avalon-ST. Na Slici 6. je prikazana blok šema sistema sa hardverskim modulom za ubrzanje linearne funkcije. Sa slike se vidi da su pored navedenog hardverskog modula u sistema uvedena i dva DMA kontrolera za prenos podataka od memorije ka hardverskom modulu kao i za prenos izlaznih podataka od hardverskog modula kako memoriji. Hardverski modul je povezan sa DMA modulima pomoću Avalon-ST interfejsa koji omogućavaju brz prenos podataka dok samo DMA moduli pomoću Avalon-MM master interfejsa pristupaju SDRAM kontroleru a preko njega i SDRAM memorijskom modulu. DMA kontroleri imaju još 2 Avalon-MM master porta za čitanje i upis deskriptora koji sadrže odgovarajuće informacije o podacima koje je potrebno preneti kao i po jedan Avalon-MM slave port preko koga procesor može pristupiti kontrolnim registrima DMA modula. Modul brojača u sistemu služi za merenje vremena rada odgovarajućih modula u cilju određivanja ubrzanja hardverske u odnosu na softversku implementaciju.



Slika 6. Blok šema sistema za ubrzanje računanja linearne funkcije

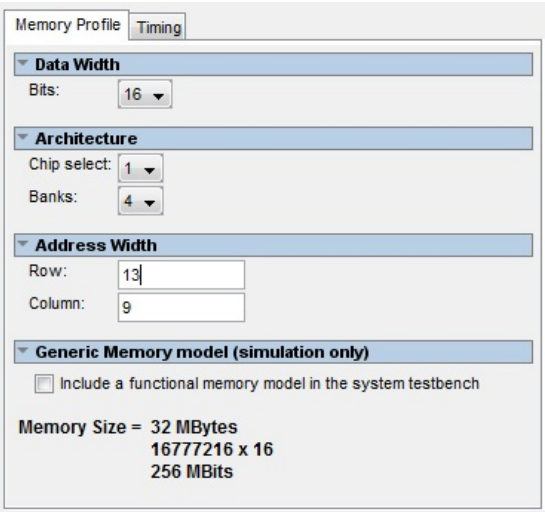
Realizacija sistema

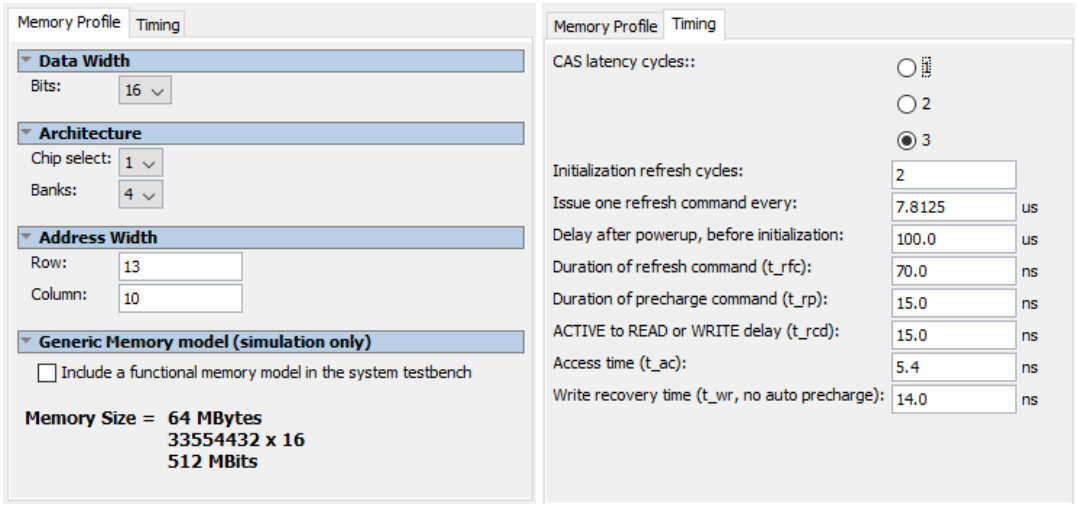
Za realizaciju ovog sistema na raspolaganju su dve platforme DE1-SoC sa **Cyclone V 5CSEMA5F31C6** čipom i DE0-Nano sa **Cyclone IVE EP4CE22F17C6** čipom.

Pokrenuti novi projekat i nazvati ga DMA_Accelerator_Example. Prilikom kreiranja podesiti odgovarajući FPGA čip u zavisnosti od toga koja od dve navedene platforme se koristi. Nakon kreiranja projekta pokrenuti Qsys kreirati sistem prikazan na Slici 6. i sačuvati ga pod imenom DMA_Accelerator_Example_System.

Da bi sistem ispravno radio neophodno je podesiti da takt koji se dovodi na procesorski deo sistema fazno prednjači 3ns nad taktom koji se dovodi do SDRAM modula. Ovo se postiže tako što se u polje phase shift kod izlaza PLL-a koji se vodi kao procesorski takt podesi vrednost 3ns.

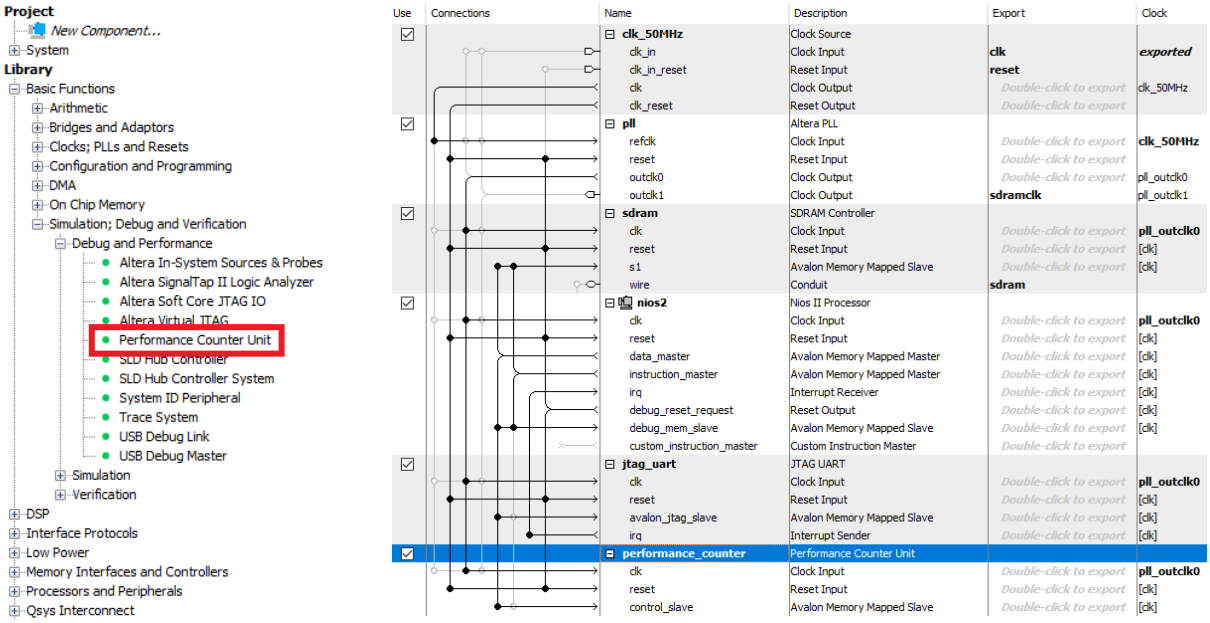
Razlika između ove dve platforme je u organizaciji SDRAM modula. Podešavanja SDRAM kontrolera za obe platforme su prikazana na Slici 7. Za DE0-Nano ostaviti podrazumevana vremenska podešavanja.





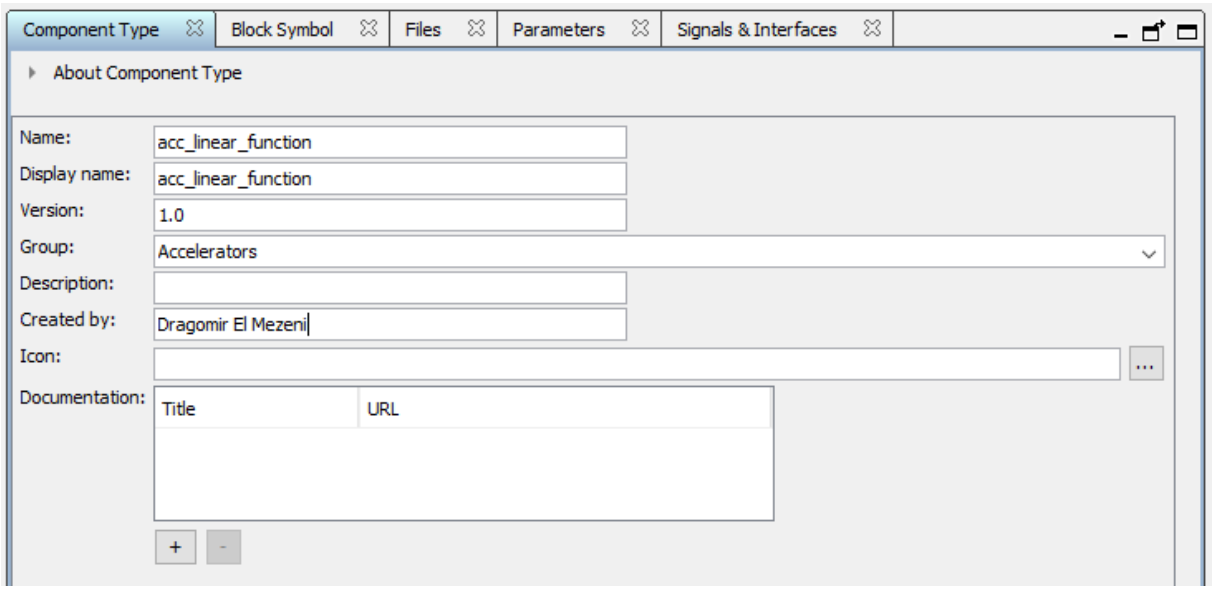
Slika 7. Podešavanje SDRAM kontrolera za DE1-SoC (dole) i DE0-Nano (gore)

Izgled osnovnog procesorskog sistema pre dodavanja hardverskog modula za ubrzanje rada linearne funkcije prikazan je na Slici 8.



Slika 8. Osnovni procesori sistem pre dodavanja hardverskog modula za računanje linearne funkcije

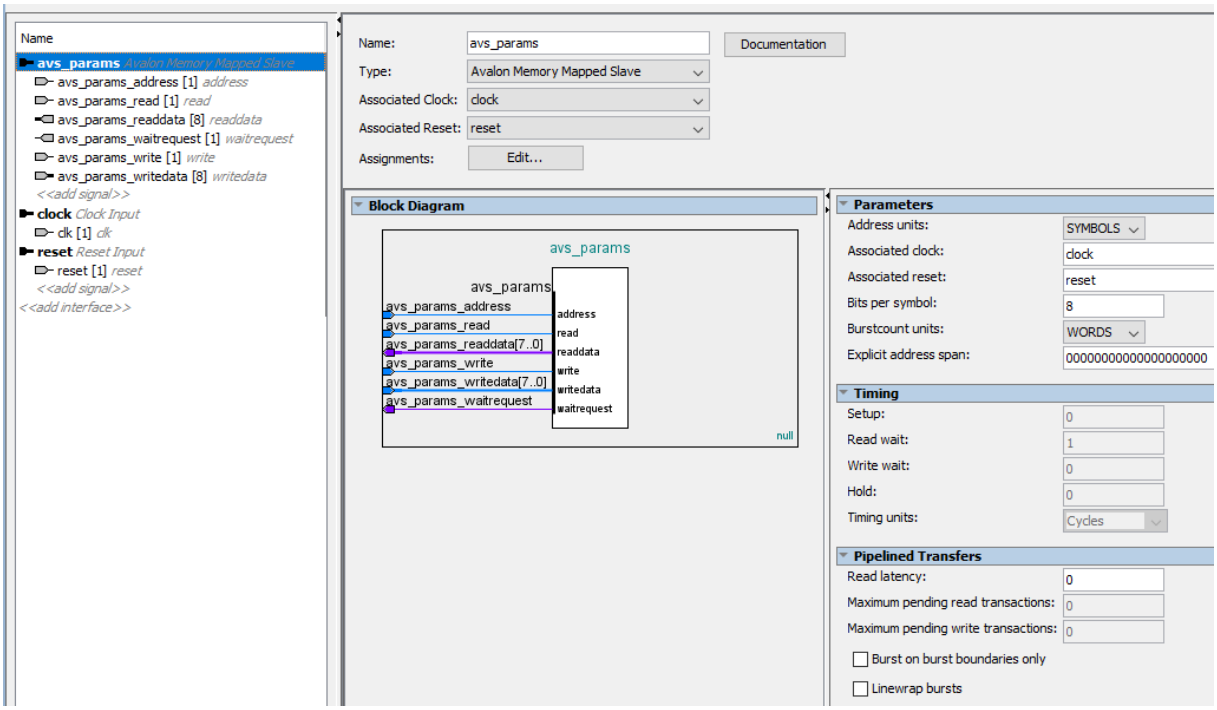
Biranjem opcije ***File → New Component*** u okviru Qsys-a započinje se procedura dodavanja nove komponente u procesorski sistem. Nakon čega se otvara dijalog u okviru kojeg je potrebno najpre upisati naziv i kategoriju kreirane komponente kao što je prikazano na Slici 9.



Slika 9. Upisivanje naziva i kategorije nove komponente

Polja sa signalima i interfejsima su trenutno prazna. Iz prethodnog razmatranja funkcionalnosti komponente zaključujemo da je potrebno dodati jedan Avalon-MM slave interfejs kako bi procesor mogao konfigurisati koeficijente linearne funkcije. Takođe potrebna su nam dva Avalon-ST interfejsa, jedan za ulazni i drugi za izlazni tok podataka. Interfejsi se mogu dodavati ručno korišćenjem tastera *Add Interface*. Jednostavniji način je korišćenje gotovih obrazaca kojim se može pristupiti preko menija *Templates*. Prednost ovih obrazaca je što pored zahtevanog interfejsa, dodaju i prateće interfejse poput takta i reseta kao i podrazumevani set signala.

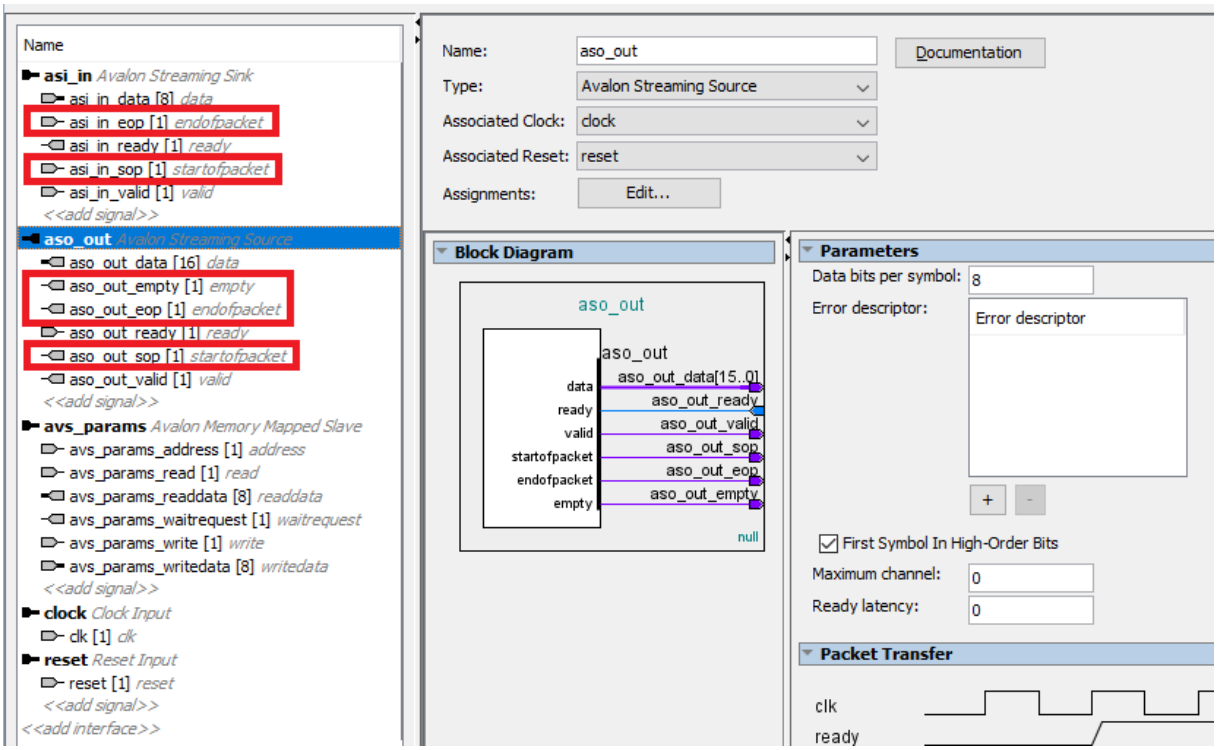
Korišćenjem obrasca Add Avalon-MM Simple Slave dodati novi slave interfejs. Kako je Avalon-MM sinhron interfejs on zahteva da mu se pridruže odgovarajući signali za reset i takt. Naziv ovog novog interfejsa je s0. Kako će se ovaj interfejs koristiti za pristupanje parametrima linearne funkcije zgodno je da mu se naziv promeni u **params**. Sledeći parametar koji je potrebno podesiti je način adresiranja. Adresiranje u okviru slave modula moguće je obavljati u jedinicama memorijskih reči (WORDS) procesora 32-bita ili u jedinicama simbola (SYMBOLS) određene periferije. Kako ćemo u okviru ovog primera koristiti 8-bitne registre zgodno je da adresiranje bude po simbolima, pri čemu svaki simbol sadrži 8 bita. U levom delu prozora može se uočiti da su postavljeni neki signali karakteristični za Avalon-MM slave interfejs. Konvencija naziva signala je avs od (Avalon Slave), zatim naziv interfejsa i potom uloga signala. Kako smo promenili naziv interfejsa potrebno je da se prilagode i nazivi ovih signala. Širina magistrale podataka se može smanjiti na 8 bita pošto se prenose 8-bitne reči, dok se širina adresne magistrale može smanjiti na 1 bit s obzirom da u periferiji postoje samo 2 registra. Izgled kartice *Signals* nakon dodavanja Avalon-MM slave interfejsa je prikazan na Slici 10.



Slika 10. Izgled kartice Signals nakon dodavanja Avalon-MM slave interfejsa

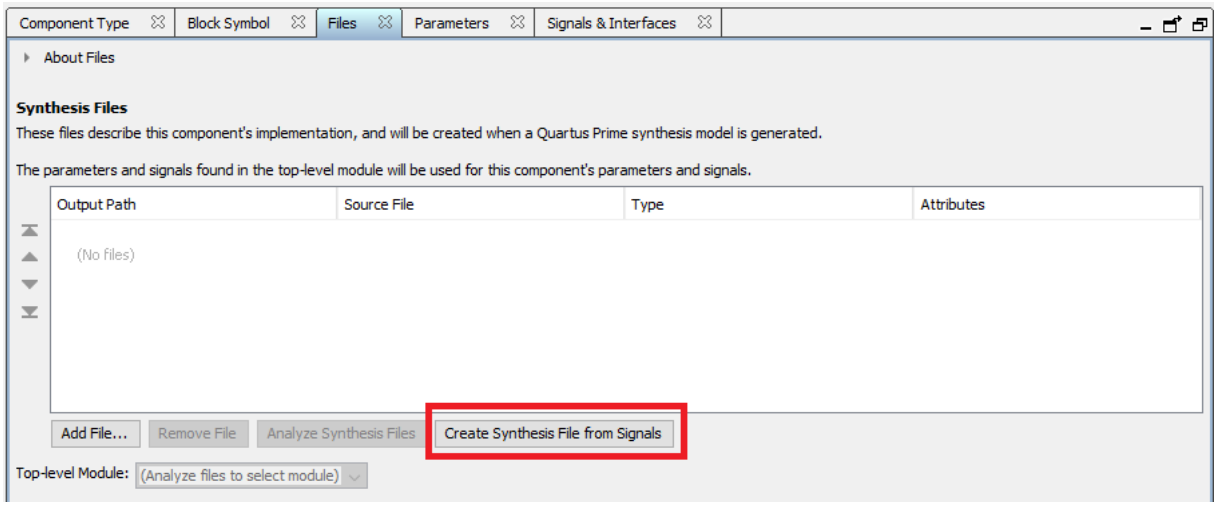
Zatim je potrebno dodati 2 Avalon-ST interfejsa i to Sink interfejs za ulazni tok podataka i Source interfejs za ulazni tok podataka. Ovo se može postići korišćenjem obrazaca: Add Typical Avalon-ST Source i Add Typical Avalon-ST Sink. Potrebno je Sink interfejs preimenovati u **in** a Source interfejs preimenovati u **out**. Iako su izlazni podaci 16-bita nije dozvoljeno menjati broj bita po simbolu. Razlog za ovo je što DMA periferija koju ćemo koristiti za prenos podataka uvek podrazumeva 8-bitne simbole. Reči koje imaju više bita, npr 16 bita kao u našem slučaju biće prenete kao više simbola (u našem slučaju 2). Takođe potrebno je da parametar *Ready latency* bude podešen na 0.

Prelaskom na karticu *Signals* primećuje se da su dodati novi signali koji odgovaraju upravo dodatim Avalon-ST interfejsima. Prilagoditi imena ovih signala tako da odgovaraju imenima odgovarajućih interfejsa pri čemu je prva reč uvek oznaka tipa interfejsa asi – Avalon-ST sink (input) i aso – Avalon-ST source (output). Takođe je potrebno podesiti širine odgovarajućih magistrala podataka. Akcelerator za računanje linearne funkcije na ulazu prihvata 8-bitne podatke dok na izlazu vraća 16-bitne podatke. Dalje DMA kontroleri na koje će se povezivati ovi interfejsi u svojim streaming interfejsima imaju signale ***start_of_packet*** i ***end_of_packet***. Kako bi se obezbedila kompatibilnost ovih interfejsa potrebno je dodati i ovde te iste signale. Kako izlazni interfejs ima 2 simbola po jednoj izlaznoj reči potrebno je dodati i ***empty*** signal. Kompletan izgled signala nove periferije je prikazan na Slici 11.



Slika 11. Kompletna lista signala nove komponente

VHDL fajl sa interfejsom upravo specificirane komponente se može automatski kreirati sa kartice ***Files*** klikom na taster ***Create Synthesis File from Signals*** kao što je prikazano na Slici 12.



Slika 12. Automatsko kreiranje VHDL fajla sa interfejsom komponente na osnovu specificiranih signala

Komponenta će biti sačuvana pod imenom ***new_component***. Kreirani fajl se može otvoriti direktno iz Quartus-a. Promeniti naziv fajla i entity-a u ***acc_linear_function*** i učitati ovaj fajl u komponentu u Qsys-u pomoću opcije Add File u okviru kartice Files (Slika 12). Dalje je potrebno kreirati odgovarajuću funkcionalnost.

Najpre je potrebno definisati dva 8-bitna registra **a_reg** i **b_reg** u kojima se čuva trenutna konfiguracija linearne funkcije. Ovim registrima se može pristupiti sa procesora pomoću Avalon-MM slave interfejsa tako da je potrebno definisati odgovarajuće adrese (u simbolima) za svaki od njih. Kako postoji samo dva registra, adresna magistrala je 1-bitna pa su i adrese koje je potrebno definisati 1-

btine (**A_ADDR**, **B_ADDR**). Signali ***a_strobe*** i ***b_strobe*** se aktiviraju kada master pokuša da upiše u odgovarajući registar projektovane periferije.

```
signal a_reg : std_logic_vector(7 downto 0);
signal b_reg : std_logic_vector(7 downto 0);

constant A_ADDR : std_logic := '0';
constant B_ADDR : std_logic := '1';

signal a_strobe : std_logic;
signal b_strobe : std_logic;

signal read_out_mux : std_logic_vector(7 downto 0);
```

Slika 13. Registri parametara linearne funkcije i odgovarajući signali za pristup

Dakle ***a_strobe*** se aktivira ako je signal ***avs_params_write*** aktivan i ako je adresa ***avs_params_address*** jednaka **A_ADDR**, odnosno adresi registra a. Signal ***avs_params_readdata*** treba uvek da sadrži trenutnu vrednost registra čija se adresa nalazi na adresnoj magistrali. Ovo se postiže korišćenjem jednostavnog multipleksera čiji je izlaz ***read_out_mux***. Za svaku od akcija po Avalon-MM slave interfejsu (upis u konfiguracione registre i čitanje) kreiran je poseban sekvencijalan proces.

```
a_strobe <= '1' when (avs_params_write = '1') and (avs_params_address = A_ADDR) else '0';
b_strobe <= '1' when (avs_params_write = '1') and (avs_params_address = B_ADDR) else '0';

read_out_mux <= a_reg when (avs_params_address = A_ADDR) else
                b_reg when (avs_params_address = B_ADDR) else
                "00000000";

write_reg_a: process(clk, reset)
begin
   if (reset = '1') then
      a_reg <= "00000010";
   elsif (rising_edge(clk)) then
      if (a_strobe = '1') then
         a_reg <= avs_params_writedata;
      end if;
   end if;
end process;

write_reg_b: process(clk, reset)
begin
   if (reset = '1') then
      b_reg <= "00000011";
   elsif (rising_edge(clk)) then
      if (b_strobe = '1') then
         b_reg <= avs_params_writedata;
      end if;
   end if;
end process;

read_regs: process(clk, reset)
begin
   if (reset = '1') then
      avs_params_readdata <= "00000000";
   elsif (rising_edge(clk)) then
      avs_params_readdata <= read_out_mux;
   end if;
end process;
```

Slika 14. Implementacija Avalon-MM slave interfejsa u okviru projektovanog modula

Avalon-ST interfejs je složeniji za implementaciju i zahteva projektovanje posebne mašine stanja koja će obavljati kontrolu protoka po ovom interfejsu. Modul za računanje linearne funkcije ima dva registra ***input_sample*** i ***output_sample***. Podaci se sa linije podataka Avalon-ST sink interfejsa smeštaju u ***input_sample*** registar i nakon jedne periode takta rezultat je dostupan u ***output_sample*** registru koji se postavlja na liniju podataka Avalon-ST source interfejsa. U zavisnosti od dostupnosti podataka na ulaznom interfejsu odnosno od spremnosti izlaznog interfejsa da prihvati novi podatak zavisiće i trenutno stanje projektovane mašine stanja. Proučavajući Avalon-ST protokol i način funkcionisanja modula za računanje linearne funkcije dolazi se do mašine stanja od 5 stanja prikazanih u Tabeli 4.

Tabela 4. Spisak stanja mašine stanja koja kontroliše Avalon-ST interfejs projektovanog modula

Naziv stanja	Opis
wait_input	Modul je spreman da prihvati podatak, čeka se ulazni interfejs da označi validnost ulaznog podatka. Vrednost u izlaznom registru je već pročitana i više nije validna.
process_input	Pristigao je novi ulazni podatak. Potrebno je da prođe jedna perioda takta da se obrađena vrednost pojavi u izlaznom registru. Na narednu ivicu signala takta u izlaznom registru će se nalaziti validan izlazni podatak.

wait_output	U izlaznom registru se nalazi validan izlazni podatak. Čeka se izlazni interfejs da postane spreman i prihvati ovaj podatak. Podatak koji se nalazi u ulaznom registru je već obrađen.
wait_output_and_process	U izlaznom registru se nalazi validan izlazni podatak. Čeka se izlazni interfejs da postane spreman i prihvati ovaj podatak. Podatak koji se nalazi u ulaznom registru još uvek nije obrađen.
full_process	Ovo stanje znači da se istovremeno na ulazu nalazi validan podatak i da je izlazni interfejs spreman da prihvati nove podatke. Za vreme ovog stanja obavlja se protočna obrada odnosno na svaku uzlaznu ivicu signala takta u ulazni registar se upisuje novi podatak iz izlaznog registra se čita prethodno obrađeni podatak i u isto vreme se upisuje trenutno obrađeni podatak.

```
process_sample : process(clk, reset)
begin
   if (reset = '1') then
      output_sample <= x"BEEF";
   elsif (rising_edge(clk)) then
      if ((current_state = process_input) or ((current_state = full_process or current_state = wait_output_and_process) and aso_out_ready = '1')) then
         output_sample <= signed(a_reg)*input_sample + signed(b_reg);
      end if;
   end if;
end process;

aso_out_data <= std_logic_vector(output_sample);

read_sample : process(clk, reset)
begin
   if (reset = '1') then
      input_sample <= x"00";
   elsif (rising_edge(clk)) then
      if (int_asi_in_ready = '1' and asi_in_valid = '1') then
         input_sample <= signed(asi_in_data);
      end if;
   end if;
end process;
```

Slika 15. Kontrola učitavanja i procesiranja podataka u zavisnosti od stanja Avalon-ST mašine stanja

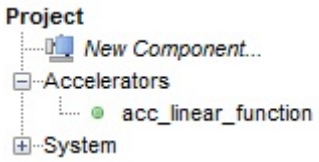
Na Slici 15. su prikazani procesi kojim se kontroliše učitavanje i procesiranje podataka. Učitavanje se obavlja u svakom taktu kada je naš modul spreman da prihvati podatke (aktiviran signal ***int_asi_in_ready***) i kada je postavljeni podatak validan (aktiviran signal ***asi_in_valid***). Interni *ready* signal je kreiran iz prostog razloga što alat ne dozvoljava očitavanje izlaznih signala. Obratiti pažnju da se procesiranje (osvežavanje izlaznog registra) ne sme obavljati u svakom stanju jer to može dovesti do brisanja podatka pre nego što se uspešno prosledi na izlazni interfejs. Ulazni podatak za procesiranje je dostupan u stanjima ***process_input***, ***wait_output_and_process*** i ***full_process***. U stanju ***process_input*** vrednost izlaznog registra nije validna i može se slobodno prebrisati. S druge strane u stanjima ***wait_output_and_process*** i ***full_process*** vrednost izlaznog registra je validna i potrebno je čekati izlazni interfejs da je pročita aktiviranjem signala ***aso_out_ready*** pre nego što se upiše nova vrednost.

```
streaming_protocol: process(current_state, asi_in_valid, aso_out_ready)
begin
   case current_state is

      when wait_input =>
         int_asi_in_ready <= '1';
         aso_out_valid <= '0';

         if (asi_in_valid = '1') then
            next_state <= process_input;
         else
            next_state <= wait_input;
         end if;

      when process_input =>
         int_asi_in_ready <= '1';
         aso_out_valid <= '0';

         if (asi_in_valid = '1') then
            next_state <= full_process;
         else
            next_state <= wait_output;
         end if;

      when wait_output =>
         int_asi_in_ready <= '1';
         aso_out_valid <= '1';

         if (aso_out_ready = '1') then
            if (asi_in_valid = '1') then
               next_state <= process_input;
            else
               next_state <= wait_input;
            end if;
         else
            if (asi_in_valid = '1') then
               next_state <= wait_output_and_process;
            else
               next_state <= wait_output;
            end if;
         end if;

      when full_process =>
         aso_out_valid <= '1';
         int_asi_in_ready <= '1';

         if (aso_out_ready = '1' and asi_in_valid = '1') then
            next_state <= full_process;
         elsif (aso_out_ready = '1' and asi_in_valid = '0') then
            next_state <= wait_output;
         else
            int_asi_in_ready <= '0';
            next_state <= wait_output_and_process;
         end if;

      when wait_output_and_process =>
         int_asi_in_ready <= '0';
         aso_out_valid <= '1';

         if (aso_out_ready = '1') then
            if (asi_in_valid = '1') then
               int_asi_in_ready <= '1';
               next_state <= full_process;
            else
               next_state <= wait_output;
            end if;
         else
            next_state <= wait_output_and_process;
         end if;


   end case;
end process;
```

Slika 16. Logika za prolazak kroz stanja mašine stanja koja kontroliše protok podataka

Kompletan kod komponente **acc_linear_function.vhd** se nalazi u arhivi **dma_accelerator_example_files.zip**.

Nova komponenta se nalazi u sada u biblioteci komponenata sa leve strane radnog prozora, kao što je prikazano na Slici 17. i može se dodati u procesorski sistem dvostrukim klikom.



Slika 17. Biblioteka komponenti sa projektovanom komponentom

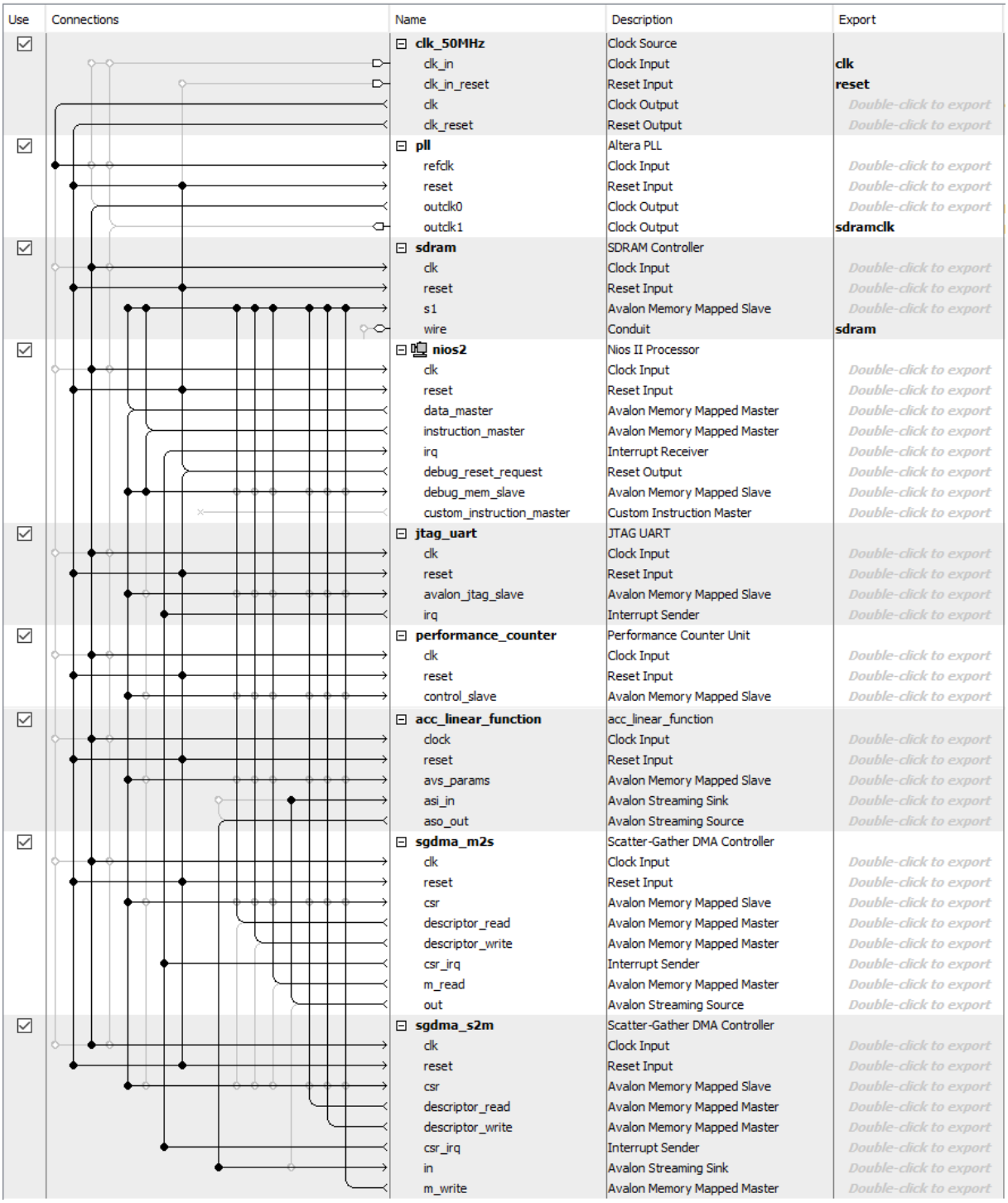
Po dodavanju nove komponente u sistem potrebno je njen takt povezati na takt ***outclk0*** na koji su povezane i sve ostale komponente u sistemu. Kako bi se obezbedila konfigurabilnost sa procesora potrebno je Avalon-MM slave interfejs ***params*** povezati na ***data_master*** magistralu procesora.

Podaci koje naš novi modul treba da obrađuje nalaze se u ekesternoj SDRAM memoriji a i rezultat procesiranja treba smestiti u istu tu memoriju. Modul koji obezbeđuje direktan pristup podacima u memoriji naziva se DMA. Ovi moduli se nalaze u biblioteci komponenti pod odeljkom ***Basic Functions→DMA***. Za potrebe ovog projekta se koristi ***Scatter-Gather DMA***. Ovaj modul za razliku od njegove jednostavnije varijante omogućava efikasan prenos podataka sa memorijskih lokacija koje ne moraju biti sukcesivne korišćenjem ulančane liste deskriptora. Takođe ovaj modul se može konfigurisati da podatke koje čita iz memorije šalje kao tok (*stream*) nekom modulu korišćenjem Avalon-ST source interfejsa. Takođe ima mogućnost da prihvata tok podataka na Avalon-ST sink interfejsu i da te podatke smešta na odgovarajuću lokaciju u memoriji. Kao što vidimo ovo je idealni modul za potrebe našeg sistema.

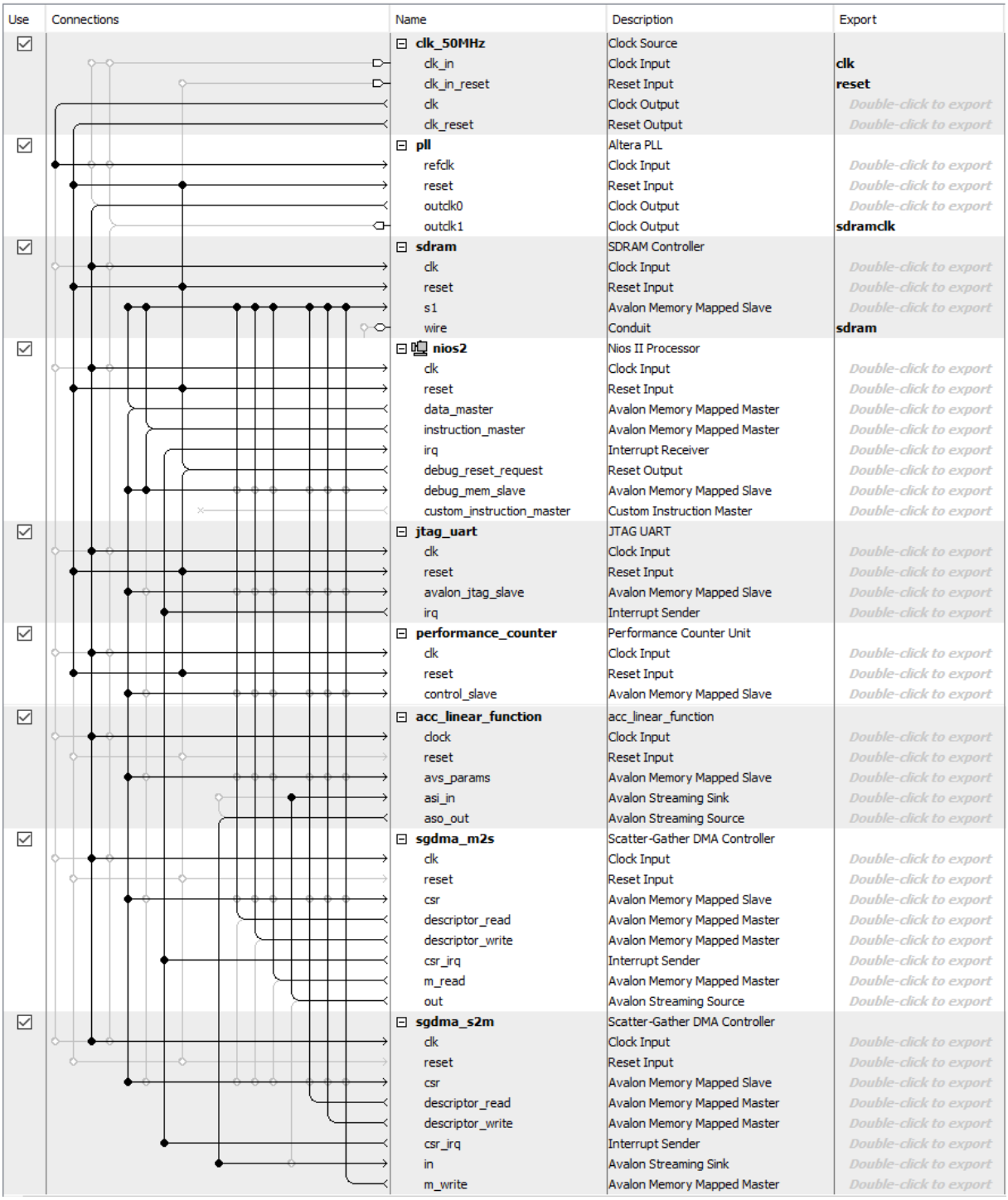
Sa Slike 6. se može uočiti da su u ovom sistemu potrebna dva DMA kontrolera. Jedan je zadužen za čitanje ulaznih podataka iz memorije i slanje toga podatka projektovanom modulu. Nakon obrade rezultati se šalju kao tok podataka drugom DMA modulu koji ih prikuplja i smeša na odgovarajuće mesto u memoriji. Širine Avalon-ST interfejsa je potrebno podesiti tako da odgovaraju projektovanom modulu. Voditi računa da prilikom prenosa više bajtova dolazi do obrtanja redosleda, pa je u slučaju višebajtnih reči potrebno uključiti opciju ***Avalon mm byte reorder mode: Byte Swap***.

DMA kontroleri imaju dva Avalon-MM master porta ***descriptor_read***, ***descriptor_write*** koji obezbeđuju pristup deskriptorima kojima se definišu parametri aktuelnih transfera. Takođe imaju jedan ili dva master porta ***m_read*** i ***m_write*** kojima čitaju ili upisuju podatke u odgovarajuću memoriju. Kako su u našem primeru svi podaci (i ulazni i izlazni) kao i deskriptori u eksternoj SDRAM memoriji, sve ove portove je potrebno povezati na slave ulaz SDRAM kontrolera. Takođe kako bi se obezbedila kontrola DMA modula sa procesora slave port ***csr*** je potrebno povezati na ***data_master*** magistralu procesora. Odgovarajući Avalon-ST portovi se povezuju sa Avalon-ST portovima ***acc_linear_function*** modula.

Nakon svih podešavanja novi deo Nios2 sistema izgleda kao na Slici 18.



Use	Name	Description	Export
☑	**clk_50MHz**	Clock Source	
	clk_in	Clock Input	**clk**
	clk_in_reset	Reset Input	**reset**
	clk	Clock Output	*Double-click to export*
	clk_reset	Reset Output	*Double-click to export*
☑	**pll**	Altera PLL	
	refclk	Clock Input	*Double-click to export*
	reset	Reset Input	*Double-click to export*
	outclk0	Clock Output	*Double-click to export*
	outclk1	Clock Output	**sdramclk**
☑	**sdram**	SDRAM Controller	
	clk	Clock Input	*Double-click to export*
	reset	Reset Input	*Double-click to export*
	s1	Avalon Memory Mapped Slave	*Double-click to export*
	wire	Conduit	**sdram**
☑	**nios2**	Nios II Processor	
	clk	Clock Input	*Double-click to export*
	reset	Reset Input	*Double-click to export*
	data_master	Avalon Memory Mapped Master	*Double-click to export*
	instruction_master	Avalon Memory Mapped Master	*Double-click to export*
	irq	Interrupt Receiver	*Double-click to export*
	debug_reset_request	Reset Output	*Double-click to export*
	debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to export*
	custom_instruction_master	Custom Instruction Master	*Double-click to export*
☑	**jtag_uart**	JTAG UART	
	clk	Clock Input	*Double-click to export*
	reset	Reset Input	*Double-click to export*
	avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*
	irq	Interrupt Sender	*Double-click to export*
☑	**performance_counter**	Performance Counter Unit	
	clk	Clock Input	*Double-click to export*
	reset	Reset Input	*Double-click to export*
	control_slave	Avalon Memory Mapped Slave	*Double-click to export*
☑	**acc_linear_function**	acc_linear_function	
	clock	Clock Input	*Double-click to export*
	reset	Reset Input	*Double-click to export*
	avs_params	Avalon Memory Mapped Slave	*Double-click to export*
	asi_in	Avalon Streaming Sink	*Double-click to export*
	aso_out	Avalon Streaming Source	*Double-click to export*
☑	**sgdma_m2s**	Scatter-Gather DMA Controller	
	clk	Clock Input	*Double-click to export*
	reset	Reset Input	*Double-click to export*
	csr	Avalon Memory Mapped Slave	*Double-click to export*
	descriptor_read	Avalon Memory Mapped Master	*Double-click to export*
	descriptor_write	Avalon Memory Mapped Master	*Double-click to export*
	csr_irq	Interrupt Sender	*Double-click to export*
	m_read	Avalon Memory Mapped Master	*Double-click to export*
	out	Avalon Streaming Source	*Double-click to export*
☑	**sgdma_s2m**	Scatter-Gather DMA Controller	
	clk	Clock Input	*Double-click to export*
	reset	Reset Input	*Double-click to export*
	csr	Avalon Memory Mapped Slave	*Double-click to export*
	descriptor_read	Avalon Memory Mapped Master	*Double-click to export*
	descriptor_write	Avalon Memory Mapped Master	*Double-click to export*
	csr_irq	Interrupt Sender	*Double-click to export*
	in	Avalon Streaming Sink	*Double-click to export*
	m_write	Avalon Memory Mapped Master	*Double-click to export*



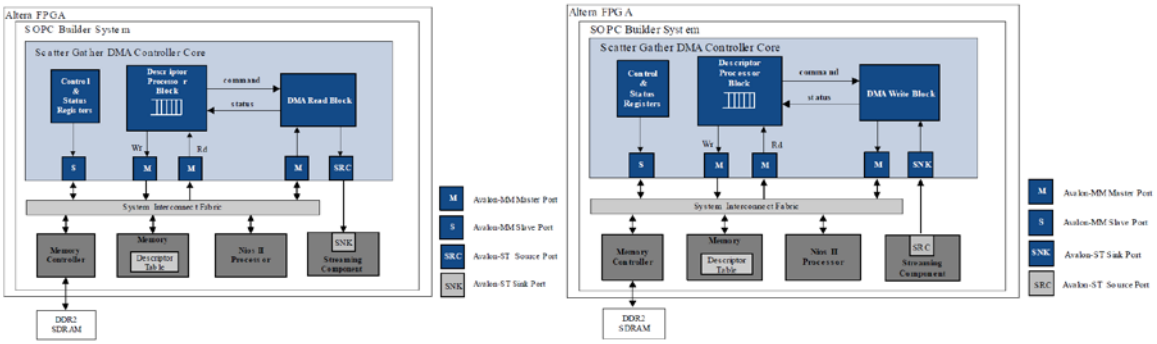
Slika 18. Kompletan DMA_Accelerator_Example_System

DMA kontroler

Za prenos podataka iz memorije do hardverskog akceleratora kao i za prihvatanje obrađenih rezultata i smeštanje u memoriju koriste se Scatter-Gather DMA komponente. Ova komponenta omogućava brz prenos velike količine podataka i može se konfigurisati za jedan od tri načina prenosa podataka:

1) Iz memorije u tok podataka (Memory-to-Stream)
2) Iz toga podataka u memoriju (Stream-to-Memory)
3) Iz memorije u memoriju (Memory-to-Memory)

U ovom primeru koriste se prva dva načina konfiguracije za ulazni izlazni DMA kontroler. Na Slici 19. je prikazan primer sistema za ove dve konfiguracije.



Slika 19. Konfiguracija ulaznog DMA kontrolera (levo) i izlaznog DMA kontrolera (desno)

Sa Slike 19. se može uočiti da svaki DMA kontroler sadrži kontrolni blok i blok za procesiranje deskriptora. U okviru kontrolnog bloka se nalaze kontrolni i statusni registri DMA kontrolera. Kontrolni blok ima Avalon-MM slave interfejs preko koga procesor može pristupiti odgovarajućim registrima DMA kontrolera.

Parametri svakog transfera koji je potrebno obaviti se opisuju deskriptorima. Deskriptori sadrže sve neophodne informacije poput izvorišne i odredišne adrese, količine podataka koje je potrebno preneti i sl. Kako se pomoću deskriptora može definisati samo prenos podataka na memorijski sukcesivnim lokacijama za prenos podataka koji se ne nalaze na susednim memorijskim lokacijama potrebno je više deskriptora. Standardni DMA kontroler zahteva da se za svaki novi transfer, novu grupu podataka na sukcesivnim lokacijama, ponovo startuje DMA prenos. Ovo podrazumeva značajno gubljenje vremena oko obrade prekida, ponovnog konfigurisanja i startovanja DMA prenosa. Scatter-Gather DMA sa druge strane omogućava obradu ulančane liste deskriptora. Svaki deskriptor osim osnovnih parametara tekućeg transfera sadrži i pokazivač na sledeći deskriptor koji opisuje naredni transfer. Na taj način se omogućava prenos podataka koji ne moraju biti na sukcesivnih memorijskim lokacijama bez intenzivnog učešća procesora. Dovoljno je samo kreirati ulančanu listu deskriptora, konfigurisati DMA sa adresom prvog deskriptora u listi i započeti transfer. Nakon svakog obrađenog deskriptora, popunjava se broj prenetih podataka i signalizira se da je deskriptor obrađen brisanjem odgovarajućeg bita u okviru deskriptora. Ovo čitanje i pisanje deskriptora obavlja poseban blok za procesiranje deskriptora korišćenjem dva Avalon-MM master porta. Ovi portovi treba da budu vezani za memoriju u kojoj se nalaze deskriptori za DMA prenos. Blok za obradu deskriptora čita odgovarajuće deskriptore iz memorije, izdvaja odgovarajuće komanda za čitanje i upis i prosleđuje ih blokovima za čitanje i upis. Nakon uspešno obavljene transakcije blokovi za čitanje i upis vraćaju povratnu informaciju koliko je podataka zaista pročitano/upisano na osnovu čega osvežavaju

informaciju u trenutnim deskriptorima i upisuju ih nazad u memoriju. Kako se obrada deskriptora i prenos podataka obavljaju u zasebnim blokovima omogućena je paralelizacija ovih proces čime se značajno ubrzava ceo proces.

Registarska mapa DMA kontrolera je prikazana u Tabeli 3. dok su u Tabelama 4. i 5. prikazane strukture kontrolnog i statusnog registra.

Tabela 3. Registarska mapa DMA kontrolera

Ofset (32-bit)	Naziv registra	Opis
0	**status**	Trenutni status DMA kontrolera. Sadrži informacije da li je u toku procesiranje, šta je prouzrokovalo poslednji prekid i sl
1	**version**	Informacija o verziji hardvera. Može se koristiti za proveru validnosti dostupnih drajvera.
4	**control**	Specificira ponašanje DMA kontrolera. U okviru ovog registra se definišu uslovi pri kojima će se generisati prekid. Takođe postavljanjem odgovarajućih bita ovog registra se pokreće i zaustavlja DMA kontroler.
8	**next_descriptor_pointer**	Sadrži adresu prvog narednog deskriptora za procesiranje. Preporuka je da se pre čitanja ovog registra RUN bit u kontrolnom registru postavi na 0 i da se sačeka da BUSY bit statusnog registra postane 0.

Tabela 4. Mapa kontrolnog registra

Bit	Naziv	Pristup	Opis
0	**IE_ERROR**	R/W	Postavljanje ovog bita omogućava generisanje prekida u slučaju greške po Avalon-ST magistrali.
1	**IE_EOP_ENCOUNTERED**	R/W	Postavljanje ovog bita omogućava generisanje prekida pri pojavi kraja paketa (EOP).
2	**IE_DESCRIPTOR_COMPLETED**	R/W	Postavljanje ovog bita omogućava generisanje prekida nakon obrade svakog pojedinačnog deskriptora.
3	**IE_CHAIN_COMPLETED**	R/W	Postavljanje ovog bita omogućava generisanje prekida nakon što se svi deskriptori ulančane liste obrade, odnosno nakon nailaska na deskriptor čiji je OWNED_BY_HW bit postavljen na 0.
4	**IE_GLOBAL**	R/W	Postavljanje ovog bita omogućava DMA kontroleru da generiše prekide.
5	**RUN**	R/W	Postavljanje ovog bita započinje učitavanje i obrada deskriptora čija se adresa nalazi u *next_descriptor_pointer* registru. Procesiranje se nastavlja sve dok se ne naiđe na deskriptor čiji je OWNED_BY_HW bit postavljen na 0. DMA kontroler se zaustavlja postavljanjem ovog bita na 0. Ukoliko se DMA zaustavi u sred procesiranja nekog deskriptora, najpre se

			završava procesiranje tekućeg deskriptora i tek nakon toga se DMA zaustavlja.
6	**STOP_DMA_ER**	R/W	Postavljanje ovog bita omogućava da se DMA kontroler zaustavi nakon pojave greške na Avalon-ST magistrali.
7	**IE_MAX_DESC_PROCESSED**	R/W	Postavljanje ovog bita omogućava generisanje prekida nakon obrade određenog broja deskriptora. Broj deskriptora, MAX_DESC_PROCESSED, nakon kojih se generiše prekih specificiran je u okviru narednih 8 bita kontrolnog registra.
8..15	**MAX_DESC_PROCESSED**	R/W	
16	**SW_RESET**	R/W	DMA kontroler se može resetovati dvostrukim upisom u ovaj bit. Nakon drugog upisa DMA se resetuje. Ako se DMA resetuje u toku prenosa može doći do zaključavanja magistrale koja se ne može otključati do prvog narednog reseta celog sistema.
17	**PARK**	R/W	Ukoliko je ovaj bit postavljen na 0 DMA kontroler nakon obrade svakog deskriptora postavlja OWNED_BY_HW bit na 0 čime signalizira da je određeni deskriptor obrađen. U slučaju da se isti set deskriptora koristi za višestruku obradu, nekada je dobro da DMA ne resetuje OWNED_BY_HW bit. Ovo se obezbeđuje postavljanjem PARK bita kontrolnog registra na 1.
18	**DESC_POLL_EN**	R/W	Postavljanje ovog bita na 1 omogućava se poliranje OWNED_BY_HW bita deskriptora čija adresa se nalazi u next_descriptor_address registru.
19	**Rezervisano**		
20..30	**TIMEOUT_COUNTER**	R/W	Definiše broj perioda signala takta između dva intervala poliranja. Validan opseg je 1-255.
31	**CLEAR_INTERRUPT**	R/W	Postavljanje ovog bita čisti sve prekide koji su na čekanju.

Tabela 5. Mapa statusnog registra

Bit	Naziv	Pristup	Opis
0	**ERROR**	R/C	Označava pojavu greške na Avalon-ST magistrali. Potrebno je obrisati bit nakon čitanja upisom 1.
1	**EOP_ENCOUNTERED**	R/C	Vrednost 1 ovog bita označava da je tekući transfer završen kao posledica pojave kraja paketa na Avalon-ST interfejsu.
2	**DESCRIPTOR_COMPLETED**	R/C	Vrednost 1 označava da je završeno procesiranje tekućeg deskriptora.

			Potrebno je obrisati bit nakon čitanja upisom 1.
3	**CHAIN_COMPLETED**	R/C	Vrednost 1 označava da je završeno procesiranje celog lanca deskriptora. Potrebno je obrisati bit nakon čitanja upisom 1.
4	**BUSY**	R	Vrednost 1 označava da je u toku procesiranje deskriptora. Ovaj bit se postavlja na 1 u prvom narednom ciklusu takta nakon postavljanja bita RUN kontrolnog registra i ostaje postavljen na 1 dok god se ne desi neki od uslova: 1) Završi se procesiranje svih deskriptora i resetuje RUN bit kontrolnog registra 2) Nastane greška na Avalon-ST magistrali i bit STOP_DMA_ER je postavljen na 1
5..31	**Rezervisano**		

Tabela 6. Struktura DMA deskriptora

Byte Offset	Field Names															
	31				24	23			16	15			8	7		0
base	source															
base + 4	Reserved															
base + 8	destination															
base + 12	Reserved															
base + 16	next_desc_ptr															
base + 20	Reserved															
base + 24	Reserved									bytes_to_transfer						
base + 28	desc_control					desc_status				actual_bytes_transferred						

Tabela 7. Opis registara DMA deskriptora

Naziv registra	Pristup	Opis
source	R/W	Adresa sa koje se čitaju podaci. Ova adresa se postavlja na 0 ako je ulaz tok podataka sa Avalon-ST interfejsa.
destination	R/W	Adresa na koju se upisuju podaci. Ova adresa se postavlja na 0 ako je izlaz tok podataka na Avalon-ST interfejs.
next_desc_ptr	R/W	Adresa narednog deskriptora ulančane liste.
bytes_to_transfer	R/W	Broj bajtova za tekući prenos. Ako je ovaj registar postavljen na 0, DMA kontroler nastavlja transfer dok god se ne pojavi kraj paketa.
actual_bytes_transferred	R	Broj bajtova koji je uspešno prenet. Ovaj podatak postavlja DMA kontroler nakon obrade odgovarajućeg deskriptora.

desc_status	R/W	Ovo polje postavlja DMA kontroler nakon obrade deskriptora.
desc_control	R/W	Definiše ponašanje kontrolera. Osvežava se nakon obrade deskriptora.

Tabela 8. Mapa DESC_CONTROL registra

Bit	Naziv	Pristup	Opis
0	**GENERATE_EOP**	W	Kada je ovaj bit postavljen na 1, DMA blok za čitanje iz memorije postavlja signal za kraj paketa po Avalon-ST interfejsu nakon čitanja poslednje reči iz memorije.
1	**READ_FIXED_ADDRESS**	R/W	Kada je ovaj bit postavljen na 1, DMA ne inkrementira adrese prilikom čitanja iz memorije već sva čitanja obavlja sa iste memorijske adrese.
2	**WRITE_FIXED_ADDRESS**	R/W	Kada je ovaj bit postavljen na 1, DMA ne inkrementira adrese prilikom upisa u memoriju već sve upise obavlja na istu memorijsku adresu.
3..6	**AVALON-ST_CHANNEL_NUMBER**	R/W	Identifikacija kanala koji će DMA kontorler koristiti po Avalon-ST magistrali.
7	**OWNED_BY_HW**	R/W	Ovaj bit označava da li hardver ili softver imaju pristup ovom deskriptoru. Ako deskriptor nije još uvek obrađen njegov OWNED_BY_HW bit je postavljen na 1 i softver ne bi trebalo da menja vrednost tog deskriptora kako ne bi došlo do problema trke. U slučaju da je ovaj bit postavljen na 0 to znači da je hardver završio sa procesiranjem deskriptora i da mu se slobodno može pristupiti iz softvera.

Softverska kontrola DMA modula

Osnovne strukture podataka i funkcije za kontrolu i pristup registrima nalaze se u BSP-u u sledećim fajlovima koje obavezno treba uključiti u funkciju koja manipuliše DMA kontrolerom:

- **altera_avalon_sgdma_regs.h**
- **altera_avalon_sgdma.h**
- **altera_avalon_sgdma_descriptor.h**

Struktura podataka **alt_sgdma_dev** sadrži informacije određenom DMA modulu. Pokazivač na ovu strukturu se koristi u svim funkcijama za pristup i čitanje određenih registara DMA kako bi se naznačilo na koji tačno DMA kontroler se konkretni poziv odnosi. Deskriptori su opisani strukturom **alt_sgdma_descriptor**. Ova struktura zauzima 32 bajta i sadrži sve potrebne informacije kao što su adresa i broj bajtova za čitanje i pisanje, adresa narednog deskriptora u listi, bajt ***control*** koji predstavlja 8 bita kontrolnog registra DESC_CONTROL kao i bajt ***status*** koji sadrži informaciju o eventualnoj grešci. Podatak *actual_bytes_trasferred* postavlja DMA nakon uspešno obavljene transakcije i predstavlja broj bajtova koji je uspešno prenet tokom te transakcije. U nastavku će ukratko biti opisane osnovne funkcije korišćene u primeru sa časa. Za potpuniji opis svih funkcija DMA

kontrolera pogledati dokument: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_embedded_ip.pdf.

U sistemu postoje dva DMA kontrolera **sgdma_m2s** (memory-to-stream) i **sgdma_s2m** (stream-to-memory). Zbog toga je potrebno kreirati pokazivače na dve ***alt_sgdma_dev*** strukture i dodeliti im vrednosti pozivom funkcije ***alt_avalon_sgdma_open*** čiji je jedini parametar naziv odgovarajućeg kontrolera. Naziv kontrolera se može pronaći u fajlu **system.h**. Primer za naš sistem je prikazan na Slici 20.

```
alt_sgdma_dev * sgdma_m2s = alt_avalon_sgdma_open("/dev/sgdma_m2s");
alt_sgdma_dev * sgdma_s2m = alt_avalon_sgdma_open("/dev/sgdma_s2m");
```

Slika 20. Inicijalizacija pokazivača na dva DMA kontrolera prisutna u sistemu

Lista deskriptora se kreira pozivom odgovarajućih funkcija:

- **alt_avalon_sgdma_construct_mem_to_stream_desc**
- **alt_avalon_sgdma_construct_stream_to_mem_desc**
- **alt_avalon_sgdma_construct_mem_to_mem_desc**

Kako su različiti podaci potrebni za svaki od ovih transfera to su i različiti interfejsi ovih funkcija. Pre kreiranja liste deskriptora potrebno je alocirati odgovarajući prostor u memoriji u slučaju da se deskriptori čuvaju na heap-u. U slučaju da su deskriptori u nekoj on-chip memoriji dovoljno je postaviti odgovarajuću vrednost pokazivača, pri čemu je potrebno voditi računa da je na raspolaganju dovoljna količina memorije. Prilikom alociranja memorije za deskriptore potrebno je voditi računa o sledećim stvarima:

1) **Adrese deskriptora moraju biti na memorijskim lokacijama koje su celobrojni umnošci veličine deskriptora ALTERA_AVALON_SGDMA_DESCRIPTOR_SIZE, odnosno 32 bajta.**
2) **Poslednji deskriptor u nizu se koristi za zaustavljanje DMA prenosa, nema validne podatke za prenos i OWNED_BY_HW bit mu je postavljen na 0.**

Zbog ovog razloga je u primeru za prenos N bafera alocirana memorija za N+2 deskriptora. N deskriptora sadrži informaciju o validnim transferima, poslednji deskriptor se koristi za zaustavljanje. Dodatni prostor je alociran kako bi se obezbedilo kreiranje lanca deskriptora na poravnatim memorijskim lokacijama. Naime, pokazivač koji se dobije nakon alokacije memorije ne mora se nalaziti na poravnatoj memorijskoj lokaciji. Zato se nakon alokacije traži prva naredna veća memorijska lokacija koja predstavlja celobrojni umnožak veličine deskriptora. Ovaj novi pokazivač se dalje koristi za kreiranje liste deskriptora. Prvobitni pokazivač je potrebno sačuvati kako bi se na kraju procesiranja ispravno oslobodila alocirana memorija. **OWNED BY HW (ILI CEO CONTROL BAJT) BIT POSLEDNJEG DESKRIPTORA SE MORA POSTAVITI NA 0 KAKO BI SE OBEZBEDILO ZAUSTAVLJANJE DMA KONTROLERA.**

Ulančana lista deskriptora se kreira u petlji kao što je to prikazano na Slici 21.

```
/* Clear out the null descriptor owned by hardware bit.  These locations
 * came from the heap so we don't know what state the bytes are in (owned bit could be high).*/
receive_descriptors[number_of_buffers].control = 0;


for(buffer_counter = 0; buffer_counter < number_of_buffers; buffer_counter++)
{
    /* This will create a descriptor that is capable of transmitting data from an Avalon-MM buffer
     * to a packet enabled Avalon-ST FIFO component */
    alt_avalon_sgdma_construct_mem_to_stream_desc(&transmit_descriptors[buffer_counter],  // current descriptor pointer
                                                  &transmit_descriptors[buffer_counter+1], // next descriptor pointer
                                                  (alt_u32*)input_buffers[buffer_counter],  // read buffer location
                                                  (alt_u16)buffer_lengths[buffer_counter],  // length of the buffer
                                                  0, // reads are not from a fixed location
                                                  0, // start of packet is disabled for the Avalon-ST interfaces
                                                  0, // end of packet is disabled for the Avalon-ST interfaces,
                                                  0);  // there is only one channel

    /* This will create a descriptor that is capable of transmitting data from an Avalon-ST FIFO
     * to an Avalon-MM buffer */
    alt_avalon_sgdma_construct_stream_to_mem_desc(&receive_descriptors[buffer_counter],  // current descriptor pointer
                                                  &receive_descriptors[buffer_counter+1], // next descriptor pointer
                                                  (alt_u32*)output_buffers[buffer_counter],  // write buffer location
                                                  (alt_u16)buffer_lengths[buffer_counter]*sizeof(alt_u16),  // length of the buffer
                                                  0); // writes are not to a fixed location
}
```

Slika 21. Kreiranje ulančane liste deskriptora

Obratite pažnju da se količina podataka uvek specificira brojem bajtova i da je tip podataka kojim se specificira broj bajtova alt_u16. Kako su rezultati procesiranja 16-bita odnosno dva bajta, količina podataka koja se prenosi od izlaza akceleratora do memorije jednaka je veličini bafera pomnoženoj sa veličinom jednog podatka u bajtovima. Voditi računa da tip podataka alt_u16 ograničava maksimalan broj podataka koji se može preneti pomoću jednog deskriptora na 65535 bajtova. Ako bafer sadrži više podataka potrebno je kreirati više deskriptora bez obzira što su svi podaci na sukcesivnim memorijskim lokacijama.

Kako DMA kontroler nakon događaja koji su podešeni u kontrolnom registru generiše prekid potrebno je registrovati odgovarajuće prekidne rutine koje će se pozvati kao posledica ovih prekida. Za registrovanje prekidnih rutina DMA kontrolera se koriste posebne funkcije

alt_avalon_sgdma_register_callback

U okviru ove funkcije se pored pokazivača na DMA kontroler za koji se registruje funkcija, pokazivača na prekidnu rutinu i argumenta prekidne rutine prosleđuje i sadržaj kontrolnog registra DMA kontrolera kojim se definišu uslovi pri kojima se generiše prekid i ponašanje DMA kontrolera nakon obrade deskriptora. Registrovanje prekidnih rutina u korišćenom primeru je prikazano na Slici 22. DMA kontroleri u ovom primeru su podešeni da generišu prekid nakon što se kompletna lista deskriptora obradi, pri čemu je bit PARK postavljen na 1 što znači da DMA kontroler neće resetovati OWNED_BY_HW bit nakon uspešno obrađenog deskriptora.

```
/*****************************************************
   * Register the ISRs that will get called when each (full)  *
   * transfer completes. When park bit is set, processed      *
   * descriptors are not invalidated (OWNED_BY_HW bit stays 1)*
   * meaning that the same descriptors can be used for new    *
   * transfers.                                               *
   *****************************************************/
  alt_avalon_sgdma_register_callback(sgdma_m2s,
                                     &transmit_callback_function,
                                     (ALTERA_AVALON_SGDMA_CONTROL_IE_GLOBAL_MSK |
                                      ALTERA_AVALON_SGDMA_CONTROL_IE_CHAIN_COMPLETED_MSK |
                                      ALTERA_AVALON_SGDMA_CONTROL_PARK_MSK),
                                     &rx_done);

  alt_avalon_sgdma_register_callback(sgdma_s2m,
                                     &receive_callback_function,
                                     (ALTERA_AVALON_SGDMA_CONTROL_IE_GLOBAL_MSK |
                                      ALTERA_AVALON_SGDMA_CONTROL_IE_CHAIN_COMPLETED_MSK |
                                      ALTERA_AVALON_SGDMA_CONTROL_PARK_MSK),
                                     &tx_done);
/*****************************************************/
```

Slika 22. Registoravanje prekidnih rutina DMA kontrolera

Kako se podaci pomoću DMA prenose direktno iz memorije nezavisno od procesora, u sistemima u kojima postoji keš za podatke potrebno je osigurati koherenciju, odnosno obezbediti potpuno poklapanje sadržaja keša i eksterne memorije. Ovo se obezbeđuje pozivom komande **alt_dcache_flush_all()** pre početka DMA transfera. Da bi se pozvala ova funkcija potrebno je uključiti fajl **sys/alt_cache.h.**

DMA prenos se pokreće pozivom neke od funkcija:

- **alt_avalon_sgdma_do_sync_transfer**
- **alt_avalon_sgdma_do_async_transfer**

Obe funkcije kao argumente prihvataju pokazivač na DMA kontroler koji se startuje kao i pokazivač na glavu ulančane liste deskriptora. Ove funkcije upisuju odgovarajuću adresu prvog deskriptora u lancu u **next_descriptor_address** registar DMA kontrolera i brišu i postavljaju RUN bit u kontrolnom registru čime se otpočinje novi transfer. Razlika između ove dve funkcije je što je prva blokirajuća, odnosno procesor se zaustavlja sve dok se ne izvrši kompletan DMA transfer. Druga funkcija je neblokirajuća, tj nakon pokretanja transfera kontrola toka se vraća procesoru koji može obavljati neke druge instrukcije. Nakon završetka procesiranja, odnosno nakon ispunjenja uslova definisanih u kontrolnom registru DMA kontroler generiše prekid pri čemu se poziva odgovarajuća prekidna rutina. U slučaju da više različitih događaja može generisati prekid tačna vrsta prekida se može saznati čitanjem sadržaja statusnog registra. Primer pokretanja DMA kontrolera je prikazan na Slici 23.

```
/* Start non blocking transfer with DMA modules. */
if(alt_avalon_sgdma_do_async_transfer(transmit_DMA, &transmit_descriptors[0]) != 0)
{
  printf("Writing the head of the transmit descriptor list to the DMA failed\n");
  return 1;
}
if(alt_avalon_sgdma_do_async_transfer(receive_DMA, &receive_descriptors[0]) != 0)
{
  printf("Writing the head of the receive descriptor list to the DMA failed\n");
  return 1;
}
```

Slika 23. Neblokirajuće pokretanje transfera ulaznog i izlaznog DMA kontrolera

Kompletan softverski deo projekta koji generiše nekoliko (slučajan broj) bafera, različitih dimenzija popunjenih uzastopnim brojevima i obrađuje ih softverski i hardverski linearnom funkcijom korišćenjem različitih parametara dat je u okviru direktorijuma **code** u arhivi **dma_accelerator_example_files.zip**.

Profilisanje sistema

Najjednostavniji način profilisanja sistema je korišćenjem modula **Performance counter**-a. Ova hardverska komponenta sadrži 64-bitne brojače koji mere broj taktova utoršenih na izvršavanje određene sekcije koda. Maksimalan broj sekcija se definiše prilikom instanciranja hardverske komponente. Kako bi se koristile funkcije ovog brojača potrebno je uključiti sledeći fajl:

- **altera_avalon_performance_counter.h**

Profilisanje sistema je veoma jednostavno i obavlja se pozivanjem sledećih funkcija:

- **PERF_RESET(base_address) –** svi brojači se resetuju i pripremaju za početak merenja
- **PERF_START_MEASURING(base_address)** – započinje se merenje performansi sistema
- **PERF_BEGIN(base_address, section_num)** – označava početak sekcije za koju se meri vreme, broj sekcije ide od 1 do maksimalnog broja sekcija za odgovarajući brojač
- **PERF_END(base_address, section_num)** – označava kraj sekcije za koju se meri vreme

Kompletan, formatiran izveštaj se može dobiti pozivom funkcije:

- **perf_print_formatted_report**

Argumenti ove funkcije su: bazna adresa modula brojača, učestanost takta, broj sekcija za koje se štampa izveštaj i po jedan string za svaku sekciju sa nazivom te sekcije. Primer poziva funkcije za štampanje izveštaja prikazan je na Slici 24. dok su rezultati ispisa prikazani na Slici 25.

```
perf_print_formatted_report(PERFORMANCE_COUNTER_BASE,
                            alt_get_cpu_freq(),
                                          2,
                          "linear_function_hw",
                          "linear_function_sw");
```

Slika 24. Primer poziva funkcije za ispis izveštaja profilisanja za dve sekcije koda

```
--Performance Counter Report--
Total Time: 22.1186 seconds  (1105928755 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|linear_function_hw|0.303|    0.06702|        3351193|          7|
+---------------+-----+-----------+---------------+-----------+
|linear_function_sw| 51.4|   11.35867|      567933364|          7|
+---------------+-----+-----------+---------------+-----------+
```

Slika 25. Primer izveštaja profilisanja za dve sekcije koda