Sadržaj

Analiza performansi

Za procenu performansi izabrano je sintetizovati proces relaksacija korišćenjem Vivado HLS alata. S obzirom na to da je particionisanje izvršeno i postoji jasno definisan hardver, potrebno je napisati C++ izvorni fajl koji modeluje proces relaksacija Dajkstrinim algoritmom uz uvažavanje izvršenog particionisanja. Deklaracija funkcije koja će se sintetizovati data je kod listingom 1.

```
void top_function(
    ap_uint<8> *cfg_val,
    ap_uint<32> bram[49208],
    bool *ip_interrupt,
    ap_uint<32> *dma_out
);
```

Kod listing 1 Deklaracija funkcije koja će se sintetizovati HLS alatom

`cfg_val` je vrednost koja će se upisivati u promenljivu koja će modelovati IP_CFG_REG, a `bram[49208]` modeluje bram memoriju. `ip_interrupt` i `dma_out` modeluju prekid i izlaz kroz koji se šalju relaksacije DMA modulu. Unutar glavne funkcije za sintezu, instancira se objekat klase `DijkstraCore`. Ova klasa sadrži sve potrebne podatke kojima se definiše stanje u kojem se IP modul nalazi u svakom trenutku.

```
//stanje IP modula
enum ip_state_t {RESET, DMA_STREAM, DO_RELAXATION, END, IDLE};

class DijkstraCore {
    public:
        ip_state_t state;
        ap_uint<32> relaxation_buffer[50];
        ap_uint<6> buffer_index;
        ap_uint<6> dma_index;
        bool ip_interrupt;
        ap_uint<8> ip_cfg_reg;
        ap_uint<32> dma_out;

        DijkstraCore();
        void setCfgReg(ap_uint<8> cfg_val);
        ap_uint<8> getCfgReg();
        bool getInterruptVal();
        ap_uint<32> getDMAOut();
        ap_uint<6> getBufferIndex();
};
```

Kod listing 2 Deklaracija klase `DijkstraCore` i stanja IP modula

Stanja IP modula su prikazana kod listingom 2, uz deklaraciju `DijkstraCore` klase. `state` polje predstavlja stanje u kom se IP modul nalazi. Stanje u kome se modul nalazi zavisi od trenutne vrednosti u polju `ip_cfg_reg`, koje modeluje IP_CFG_REG. Moguća stanja su:

1. *Reset modula* (`state == RESET`), tokom kojeg se vrši postavljanje svih 32-bitnih vrednosti `relaxation_buffer`-a na nulu. `relaxation_buffer` modeluje interni bafer od 50 32-bitnih lokacija na kojima se čuvaju poslednjih 50 izvršenih relaksacija. Kroz bafer se prolazi pomoću indeksa `dma_index` i `buffer_index`. Petlja koja obavlja reset je označena labelom `reset_l`.
2. *Rad relaksacija* (`state == DO_RELAXATION`), sastoji se iz tri glavne petlje, označene labelama `l_1`, `l_2` i `l_3`.

   - l_1: Vrši se pronalaženje neposećenog temena sa minimalnom udaljenošću od početnog temena. Prolazi se kroz sva temena grafa, odnosno kroz 9122 lokacije BRAM-a. Primenjene su tri optimizacione direktive nad ovom petljom: #PRAGMA HLS LOOP_TRIPCOUNT, #PRAGMA HLS LOOP_UNROLL factor = 2 i #PRAGMA HLS PIPELINE.

   - l_2: prolazak kroz lokacije u BRAM-u na kojima su `tail` vrednosti (izvorna temena ivica grafa), u potrazi za ID vrednosti neposećenog temena sa trenutno minimalnom udaljenošću od početnog temena (određenog u petlji `l_1`). Broj iteracija nije unapred određen i zbog toga se koristi optimizaciona direktiva #PRAGMA HLS LOOP_TRIPCOUNT.

   - l_3: Ukoliko je ID vrednost pronađena u `tail` vrednostima, ispituju se sva susedna temena pretragom `head` vrednosti (ponorna temena ivica grafa) u BRAM-u, u potrazi za potencijalnom relaksacijom. Izvršena relaksacija se skladišti u pomoćni bafer (`relaxation_buffer`). Kada se bafer napuni dešava se prekid, što se modeluje postavljanjem `ip_interrupt` izlaza na `true`. Vrednost u promenljivoj `ip_cfg_reg` se takođe menja, u skladu sa objašnjenjem datim u dokumentu Particionisanje i registarska mapa. Broj iteracija nije unapred određen i zbog toga se koristi optimizaciona direktiva #PRAGMA HLS LOOP_TRIPCOUNT.
3. *Slanje relaksacija DMA modulu*, (`state == DMA_STREAM`), pomoću `dma_index`-a se indeksira `relaxation_buffer` niz i na `dma_out` izlaz se postavljaju relaksacije.
4. *Kraj procesa* (`state == END`), na `ip_interrupt` izlaz se postavlja `true` i u `ip_cfg_reg` se na četvrti bit postavlja jedinica.
5. *Stanje mirovanja* (`state == IDLE`) u kome modul ne radi ništa.

Telo funkcije `top_function` se izvršava u testbenč-u (`main` funkcija) sve dok se ne dođe do kraja procesa relaksacija, `state == END`. Za test ulaze, koji su korišćeni i u Analizi pre particionisanja pri određivanju vremenskog intervala izvršavanja procesa relaksacija, telo funkcije se izvršava 18412 puta. Treba uzeti u obzir da je 9400 iteracija samo ispis relaksacija na `dma_out`, kada se IP modul nalazi u stanju `DMA_STREAM`. Za procenu performansi će se uzeti da IP modul u jednom ciklusu šalje jednu vrednost DMA modulu.

Nakon uspešne C simulacije i sinteze dobija se izveštaj o sintetizovanom rešenju za funkciju `top_function`.

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.585 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
1	4575	10.000 ns	45.750 us	1	4575	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- reset_l	50	50	1	-	-	50	no
- l_1	4562	4562	3	1	1	4561	yes
- l_2	0	1	2	-	-	0	no
- l_3	0	3	4	-	-	0	no

Slika 1 Izveštaj o sintetizovanom rešenju, procena performansi

Slika 1 prikazuje procenu performansi top_function sintetizovane funkcije. Na osnovu izveštaja se može zaključiti sledeće:

6. Mininalna perioda takt signala iznosi 8.585 ns, uz nesigurnost od 1.25 ns. Ciljana perioda takt signala je ostala podrazumevana, 10 ns.
7. Broj ciklusa neophodnih da se izvrši jedan prolaz kroz implemetiranu funkciju je maksimalno 4575, odnosno, vreme potrebno da se jedan prolaz izvrši iznosi maksimalno 45.750 μs.
8. Prikaz performansi po petljama je prikazan na slici 1 u poslednjoj tabeli. Najkritičnija je petlja sa labelom `l_1`.

Vreme izvršavanja pretrage optimalne putanje, korišćenjem ove implementacije, iznosi (18412 - 9400) × 45.750 μs + 9400 × 10 ns ≈ 0.412393 s. Vreme potrebno za izvršavanje istog upita host platformi, opisanoj u dokumentu Analiza pre particionisanja, je 0.390622 s [7].

Procena površine

Uređaj je mapiran na Zynq-7000 SoC, koji na Zybo Z7 10 pločici dolazi u CLG400 BGA pakovanju, površine 289 mm^2. Softver se pokreće na ARM Cortex-A9 procesoru, fiksne površine, dok površina hardvera zavisi od potrošenih hardverskih resursa. Slika 2 daje procenu potrošenih hardverskih resursa potrebnih za implementaciju funkcije koja opisuje IP modul. Potrebno je 805 lookup tabela (LUT) za implementaciju osnovnih aritmetičkih, logičkih i relacionih operacija u implementiranom modulu. Dodatnih 408 lookup tabela iskorišćeno je za implementaciju multipleksera koji se koriste za kontrolu toka podataka. Pošto unutar modula postoji jedan pomoćni bafer sa 50 elemenata, izdvojena je jedna BRAM ćelija za njegovu implementaciju. 507 flipflopova je iskorišćeno za implementaciju registara koji se koriste za skladištenje sadržaja promenljivih funkcije koja opisuje IP modul.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	805	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	0	0	0
Multiplexer	-	-	-	408	-
Register	-	-	507	-	-
Total	1	0	507	1213	0
Available	120	80	35200	17600	0
Utilization (%)	~0	0	1	6	0

Slika 2 Procena potrošenih hardverskih resursa sintetizovanog rešenja

Pored IP modula, na FPGA resurse se mapiraju i Xilinx-ove standardne komponente: AXI DMA kontroler, AXI interconnect komponenta i zauzima se 49208 32-bitnih BRAM lokacija. One se smeštaju na 86 BRAM ćelija (BRAM_18K). DMA kontroler u režimu direktnog adresiranja, AXI interconnect i BRAM kontroler ne doprinose značajno ovim vrednostima i nisu uzeti u razmatranje u ovoj proceni [5][6].

Analiza potrošnje i interfejsa

U ovoj proceni za potrošnju, u obzir će biti uzeto samo pomeranje podataka od i ka DRAM memoriji – izbrojane su TLM transakcije unutar virtuelne platforme. Procesor i DMA modul izvrše ukupno 55376 TLM transakcija. Podaci koji su se prenosili su tipa `uint32_t` (izvršene relaksacije) i `float` (geografske koordinate). Veličina podataka ovog tipa je 4 B.

U ovom sistemu bezbednost nije kritična i neće se koristiti nestandardni interfejsi (kakvi bi bili u slučaju da je sistem dizajniran npr. u vojne svrhe). Koristiće se standardni AXI interfejsi, kako bi se IP modul lako povezao sa Xilinx-ovim AXI DMA kontrolerom, BRAM kontrolerom i AXI interconnect komponentom.

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	top_function	return value
ap_rst	in	1	ap_ctrl_hs	top_function	return value
ap_start	in	1	ap_ctrl_hs	top_function	return value
ap_done	out	1	ap_ctrl_hs	top_function	return value
ap_idle	out	1	ap_ctrl_hs	top_function	return value
ap_ready	out	1	ap_ctrl_hs	top_function	return value
cfg_val_V_i	in	8	ap_ovld	cfg_val_V	pointer
cfg_val_V_o	out	8	ap_ovld	cfg_val_V	pointer
cfg_val_V_o_ap_vld	out	1	ap_ovld	cfg_val_V	pointer
bram_V_address0	out	16	ap_memory	bram_V	array
bram_V_ce0	out	1	ap_memory	bram_V	array
bram_V_we0	out	1	ap_memory	bram_V	array
bram_V_d0	out	32	ap_memory	bram_V	array
bram_V_q0	in	32	ap_memory	bram_V	array
bram_V_address1	out	16	ap_memory	bram_V	array
bram_V_ce1	out	1	ap_memory	bram_V	array
bram_V_we1	out	1	ap_memory	bram_V	array
bram_V_d1	out	32	ap_memory	bram_V	array
bram_V_q1	in	32	ap_memory	bram_V	array
ip_interrupt	out	1	ap_vld	ip_interrupt	pointer
ip_interrupt_ap_vld	out	1	ap_vld	ip_interrupt	pointer
dma_out_V	out	32	ap_vld	dma_out_V	pointer
dma_out_V_ap_vld	out	1	ap_vld	dma_out_V	pointer

Slika 3 Korišćeni interfejsi u sintetizovanom rešenju

Na slici 3 su prikazani korišćeni interfejsi u HLS implementaciji. Ukoliko bi se ova implementacija koristila kao finalna, dovoljno bi bilo eksportovati generisani RTL model u formatu sintetizovanog čekpionta u Vivado. Potom, u Vivado Design Suite alatu, pomoću IP Packager alata moguće je automatski generisati ulazno izlazne kontrolere za AXI-Full, AXI-Lite i AXI-Stream protokole i upakovati sve potrebne fajlove koji čine jedno IP jezgro u standardni format [7].

Literatura

[1] Zybo Z7 10: https://digilent.com/reference/_media/reference/programmable-logic/zybo-z7/zybo-z7_rm.pdf [jun 2025]

[2] Vranjković, V. : Projektovanje elektronskih uređaja na sistemskom nivou, Novi Sad: Fakultet tehničkih nauka, 2020.

[3] Grant Martin, Brian Bailey, Andrew Piziali: ESL Design and Verification: A Prescription for Electronic System Level Methodology (Systems on Silicon)

[4] Brian Bailey, Grant Martin: ESL Models and their Application: Electronic System Level Design and Verification in Practice

[5] Više o Xilinx AXI DMA kontroleru: link [jun 2025]

[6] Više o Xilinx AXI interconnect komponenti: link [jun 2025]

[7] Struharik, R. :Projektovanje složenih digitalnih sistema, materijal za predavanja i vežbe: link [jun 2025]