Implementacija in optimizacija 256-točkovne FFT v Vitis HLS

Simon Kaurin

7. januar 2026

1 Uvod

Hitri Fourierjev transform (FFT) je eden izmed temeljnih algoritmov digitalne obdelave signalov (DSP), saj omogoča učinkovito pretvorbo signala iz časovne v frekvenčno domeno. Zaradi svoje računske zahtevnosti je FFT zelo primerna za implementacijo na programabilni logiki (FPGA), kjer lahko izkoristimo paralelizacijo in cevovodno obdelavo.

V tem projektu je bil razvit 256-točkovni kompleksni FFT algoritem v okolju *Vitis HLS*. Algoritem je bil optimiziran z uporabo HLS pragm ter sintetiziran v obliki IP jedra, ki je bilo nato integrirano in verificirano v okolju Vivado.

Cilj projekta ni bil zgolj implementirati FFT, temveč razumeti vpliv posameznih pragm na arhitekturo vezja, časovno obnašanje (timing) in porabo virov.

2 Opis FFT algoritma

Uporabljen je bil radix-2 decimation-in-time (DIT) FFT algoritem. Za FFT dolžine $N = 256$ je potrebnih:

- 8 FFT stopenj,
- kompleksne butterfly operacije,
- kompleksni twiddle faktorji W_N^k.

Algoritem poteka v naslednjih korakih:

1. bit-reversal permutacija vhodnega zaporedja,
2. zaporedna izvedba osmih FFT stopenj,
3. zapis rezultatov v izhodni pomnilnik.

Butterfly operacija je definirana kot:

$$X[k] = a + W_N^k \cdot b, \qquad X[k + N/2] = a - W_N^k \cdot b$$

3 Predstavitev podatkov

Vhodni in izhodni podatki so predstavljeni kot kompleksna števila s fiksno vejico:

```
typedef ap_fixed<16,8, AP_RND_CONV, AP_SAT> fx_t;
```

Uporabljena je Q8.8 predstavitev, ki predstavlja kompromis med natančnostjo in porabo virov. Zaokroževanje in saturacija zmanjšujeta numerične napake in preprečujeta prelive.

4 Struktura HLS implementacije

FFT je implementiran z uporabo lokalnega delovnega buffra, v katerem se izvede celotna FFT transformacija. Implementacija vključuje:

- bit-reversal fazo,
- osem FFT stopenj,
- zaključni zapis rezultatov v izhodni pomnilnik.

Takšna zasnova omogoča enostavno krmiljenje in dobro optimizacijo s strani HLS orodja.

5 Uporaba HLS pragm

5.1 BRAM vmesnik

```
#pragma HLS INTERFACE bram port=in
#pragma HLS INTERFACE bram port=out
```

Ta pragma določa, da se vhodni in izhodni podatki mapirajo na blokovni RAM (BRAM), kar omogoča visoko prepustnost in enostavno integracijo v Vivado.

5.2 Cevovodna obdelava (PIPELINE)

```
#pragma HLS PIPELINE II=1
```

Pipeline omogoča začetek nove iteracije v vsakem ciklu ure, s čimer se zmanjša latenca in poveča prepustnost FFT algoritma.

5.3 Paralelizacija (UNROLL)

```
#pragma HLS UNROLL factor=2
```

Unroll pragma omogoča paralelno izvajanje več butterfly operacij, kar poveča zmogljivost na račun večje porabe virov.

5.4 Particioniranje polj

```
#pragma HLS ARRAY_PARTITION variable=buf cyclic factor=2
```

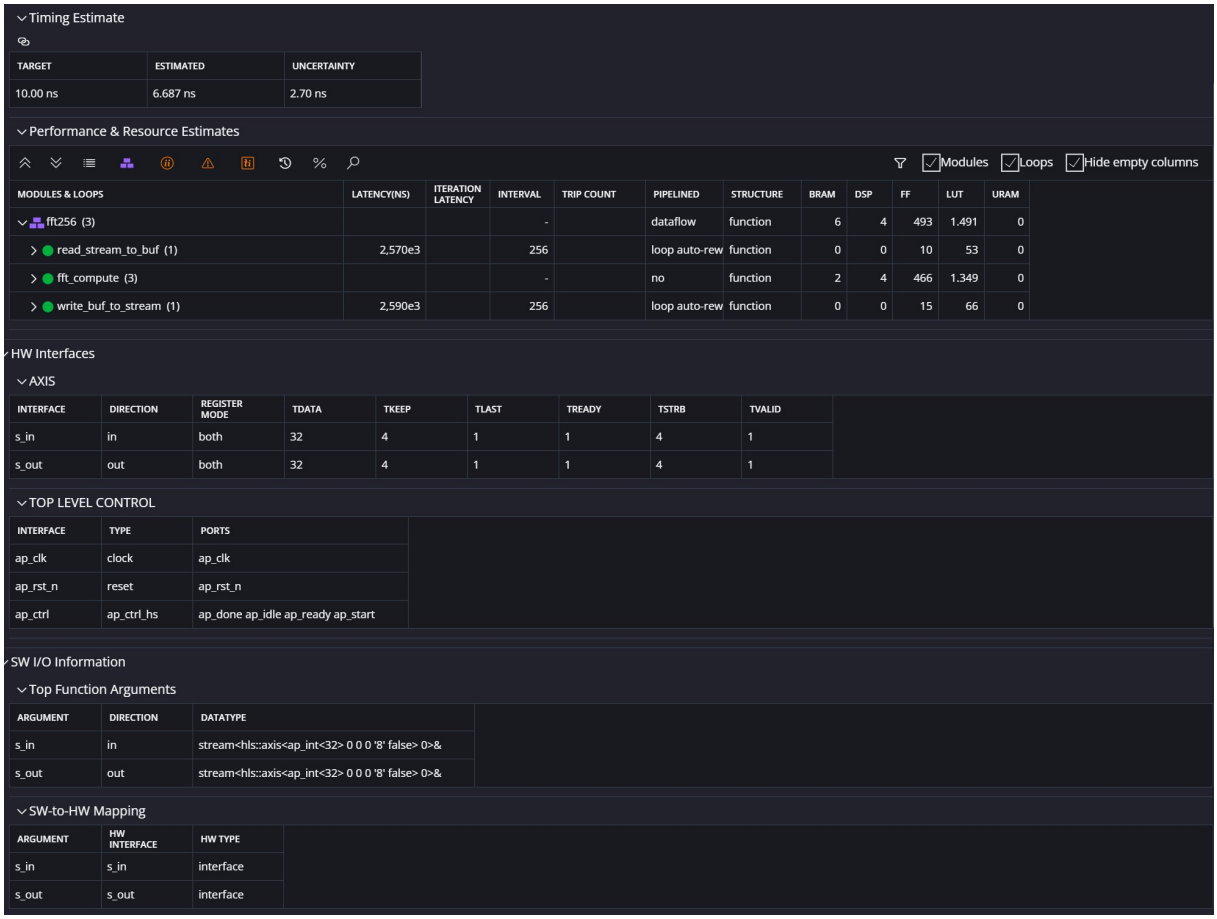
S tem se delovni buffer razdeli v več pomnilniških bank, kar zmanjša konflikte pri sočasnem dostopu.

5.5 Uporaba DSP blokov

```
#pragma HLS BIND_OP variable=ac op=mul impl=dsp
```

Ta pragma zagotovi, da se kompleksna množenja izvajajo v DSP48 blokih, kar izboljša časovno obnašanje in zmanjša porabo LUT virov.

6 Rezultati sinteze in časovna analiza



Timing Estimate

TARGET	ESTIMATED	UNCERTAINTY
10.00 ns	6.687 ns	2.70 ns

Performance & Resource Estimates

Modules Loops Hide empty columns

MODULES & LOOPS	LATENCY(NS)	ITERATION LATENCY	INTERVAL	TRIP COUNT	PIPELINED	STRUCTURE	BRAM	DSP	FF	LUT	URAM
fft256 (3)			-		dataflow	function	6	4	493	1,491	0
read_stream_to_buf (1)	2,570e3		256		loop auto-rew	function	0	0	10	53	0
fft_compute (3)			-		no	function	2	4	466	1,349	0
write_buf_to_stream (1)	2,590e3		256		loop auto-rew	function	0	0	15	66	0

HW Interfaces

AXIS

INTERFACE	DIRECTION	REGISTER MODE	TDATA	TKEEP	TLAST	TREADY	TSTRB	TVALID
s_in	in	both	32	4	1	1	4	1
s_out	out	both	32	4	1	1	4	1

TOP LEVEL CONTROL

INTERFACE	TYPE	PORTS
ap_clk	clock	ap_clk
ap_rst_n	reset	ap_rst_n
ap_ctrl	ap_ctrl_hs	ap_done ap_idle ap_ready ap_start

SW I/O Information

Top Function Arguments

ARGUMENT	DIRECTION	DATATYPE
s_in	in	stream<hls::axis<ap_int<32> 0 0 0 '8' false> 0>&
s_out	out	stream<hls::axis<ap_int<32> 0 0 0 '8' false> 0>&

SW-to-HW Mapping

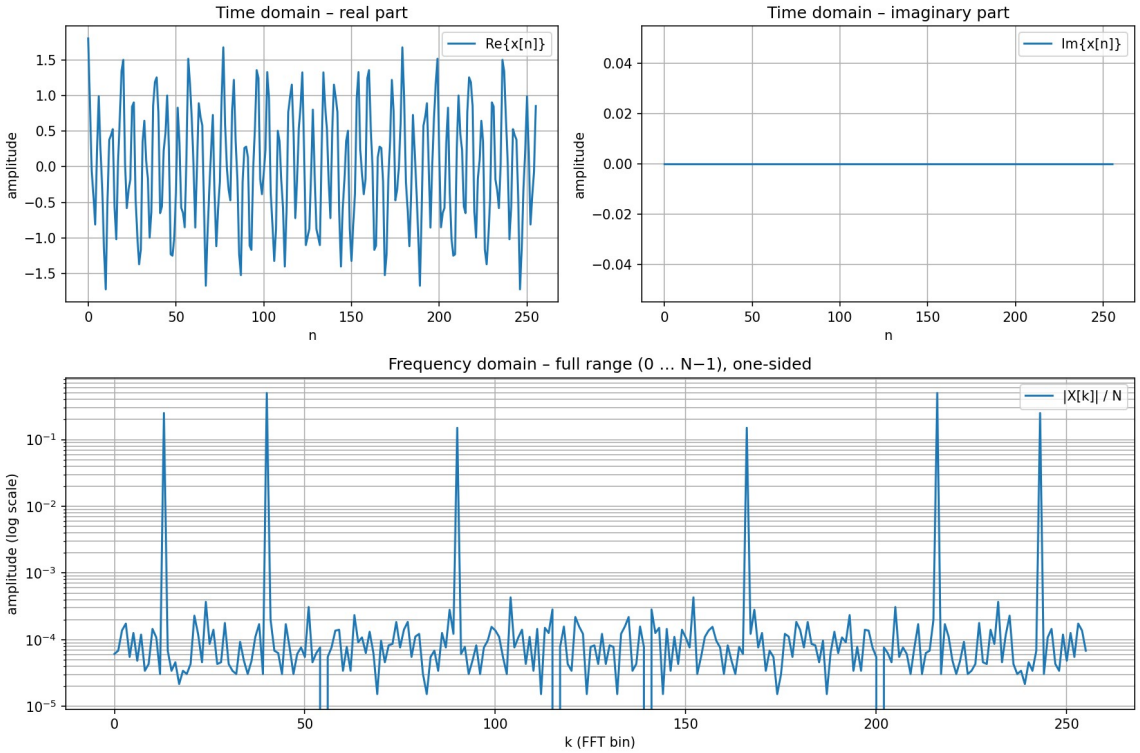
ARGUMENT	HW INTERFACE	HW TYPE
s_in	s_in	interface
s_out	s_out	interface

Slika 1: Vitis HLS poročilo: časovna ocena in poraba virov (BRAM, DSP, FF, LUT).

Poročilo sinteze pokaže porabljene vire. FFT algoritmi zahtevajo veliko računskih operacij zato porabijo precej več virov kot razni digitalni filtri. Prikazana je tudi velika časovna zakasnitev, ampak to je najmanjša zakasnitev ki je bila še sintezitabilna. Poraba virov je skladna z uporabo pipelininga, unroll pragm in DSP blokov.

7 Verifikacija delovanja

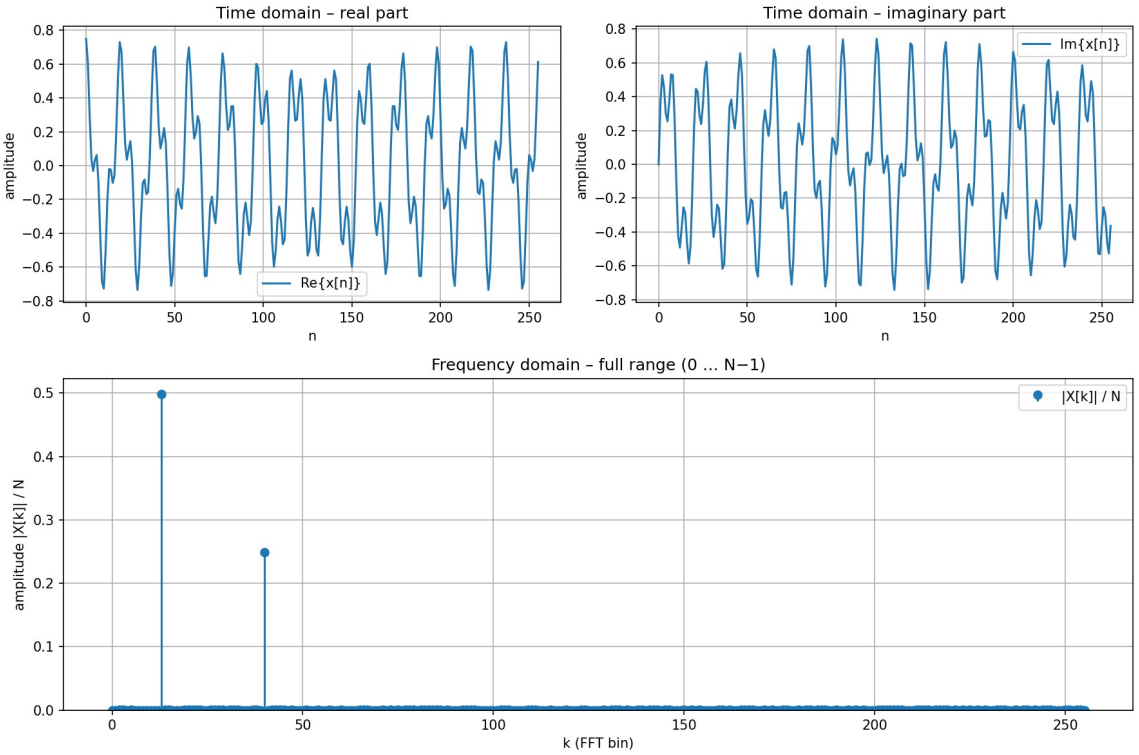
7.1 Vitis testbench (realni vhod)



Slika 2: Rezultat FFT v Vitis testbenchu: realni vhodni signal in pripadajoči spekter.

V tem primeru je imaginarni del vhodnega signala enak nič. V frekvenčni domeni se pojavijo zrcalni vrhovi pri pričakovanih bin-ih, kar potrjuje pravilnost implementacije.

7.2 Vivado testbench (kompleksni vhod)

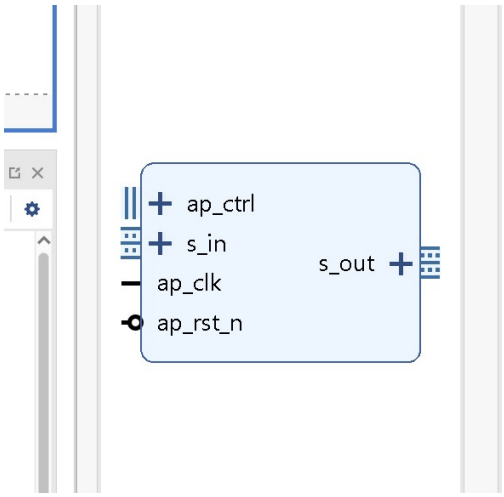


Slika 3: Rezultat FFT v Vivado testbenchu za kompleksni vhodni signal.

Vivado simulacija potrjuje pravilno delovanje kompleksne aritmetike in pravilno generiranje spektra tudi v RTL nivoju.

8 Integracija FFT IP jedra v Vivado

Takšna zasnova omogoča neposredno povezavo FFT jedra z drugimi AXI-stream kompatibilnimi IP bloki, npr. DMA enotami, filtrirnimi bloki ali procesorskim sistemom.



Slika 4: FFT256 IP jedro v Vivado Block Design okolju.

8.1 Primer povezave signalov v RTL

V testbenchu ali zgornjem (wrapper) modulu so signali FFT IP jedra povezani na naslednji način:

```
fft256_1 dut (
  .ap_clk       (ap_clk),
  .ap_rst_n     (ap_rst_n),
  .ap_start     (ap_start),
  .ap_done      (ap_done),
  .ap_idle      (ap_idle),
  .ap_ready     (ap_ready),

  .s_in_TDATA   (s_in_tdata),
  .s_in_TVALID  (s_in_tvalid),
  .s_in_TREADY  (s_in_tready),
  .s_in_TLAST   (s_in_tlast),

  .s_out_TDATA  (s_out_tdata),
  .s_out_TVALID (s_out_tvalid),
  .s_out_TREADY (s_out_tready),
  .s_out_TLAST  (s_out_tlast)
);
```

FFT jedro uporablja standardni HLS kontrolni vmesnik `ap_ctrl` ter AXI4-Stream vmesnika `s_in` in `s_out`. Takšna arhitektura omogoča jasno ločitev med krmiljenjem in podatkovnim tokom ter enostavno integracijo v večje sisteme.

9 Zaključek

Projekt je pokazal, da Vitis HLS omogoča učinkovito implementacijo kompleksnih DSP algoritmov, kot je FFT. Pravilna uporaba pragm (PIPELINE, UNROLL, ARRAY_PARTITION, DSP) bistveno vpliva na zmogljivost in porabo virov.

Rezultati simulacij v Vitis in Vivado okolju potrjujejo pravilno delovanje FFT algoritma, sintezno poročilo pa kaže, da zasnova izpolnjuje časovne zahteve in je primerna za nadaljnjo integracijo v realne sisteme.

10 Povezava do projekta

Github : Github repozitorij (link)