

# **16-pt FFT 결과보고서**

전기전자공학부  
201810528 고려욱  
201810845 박종혁

### <SDK 출력 결과>

```
<16-point Fourier Transform>
<Number of Iterations : 256>
```

Measured Accuracy (vs Floating Point): NSR(dB) = -64.206

```
Measured Accuracy (vs RTL): NSR(dB) = -inf
```

```
-----Benchmarking Start-----
```

### Case 0: FFT Reference

Nr,	Max,	Min,	Average,	Fltr Avg,	Fltr_Avg(us)
5,	254966,	254539,	254743,	254737,	764.211

## Case 1: FFT Optimization

Nr,	Max,	Min,	Average,	Fltr Avg,	Fltr_Avg(us)
5,	141989,	141835,	141899,	141890,	425.670

```

-----Benchmarking Complete-----

```

HW FFT is x1.80 faster than Reference

▶ ILA 파형 - 처음 Input data가 FFT module에 들어가는 부분

AXI Status: Idle

### ▶ ILA 파형 - FFT module의 마지막 출력 부분

LA Status: Idle

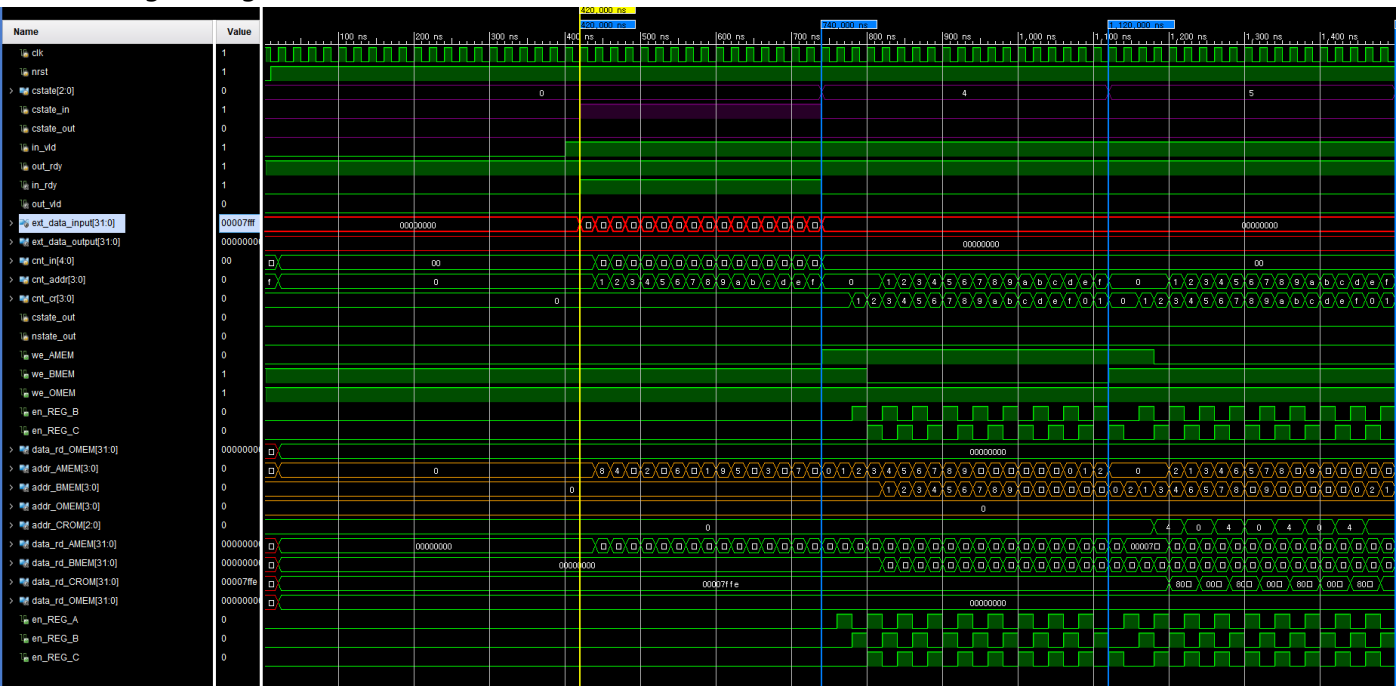
The timing diagram illustrates the LA status signals over time, showing the interaction between the LA and the system. The signals are categorized into three main sections:

- Slot 4: axi\_interconnect\_0\_M00\_AXI**
  - Write Transactions 0: Shows a single write transaction with a data length of 100.
  - Read Transactions 0: Shows a single read transaction with a data length of 100.
  - Channel Status: Includes signals like `axi_interconnect_0_M00_AXI: WVALID`, `axi_interconnect_0_M00_AXI: WREADY`, `axi_interconnect_0_M00_AXI: WLAST`, and `axi_interconnect_0_M00_AXI: WDATA`.
  - Channel Type: `axi_interconnect_0_M00_AXI: WSTRB`.
  - Channel Data: `axi_interconnect_0_M00_AXI: B Channel`.
- Slot 3: axi\_dma\_0\_M\_AXI\_S2MM**
  - Interface: Shows a single transaction with a data length of 100.
  - Channel Status: Includes signals like `axi_dma_0_M_AXI_S2MM: WVALID`, `axi_dma_0_M_AXI_S2MM: WREADY`, `axi_dma_0_M_AXI_S2MM: WLAST`, and `axi_dma_0_M_AXI_S2MM: WDATA`.
  - Channel Type: `axi_dma_0_M_AXI_S2MM: WSTRB`.
  - Channel Data: `axi_dma_0_M_AXI_S2MM: B Channel`.
- Slot 0: axi\_dma\_0\_M\_AXIS\_MM2S**
  - Interface: Shows a single transaction with a data length of 100.
  - Channel Status: Includes signals like `axi_dma_0_M_AXIS_MM2S: TVALID`, `axi_dma_0_M_AXIS_MM2S: TREADY`, `axi_dma_0_M_AXIS_MM2S: TLAST`, and `axi_dma_0_M_AXIS_MM2S: TDATA`.
  - Channel Type: `axi_dma_0_M_AXIS_MM2S: TKEEP`.
  - Channel Data: `axi_dma_0_M_AXIS_MM2S: T Channel`.

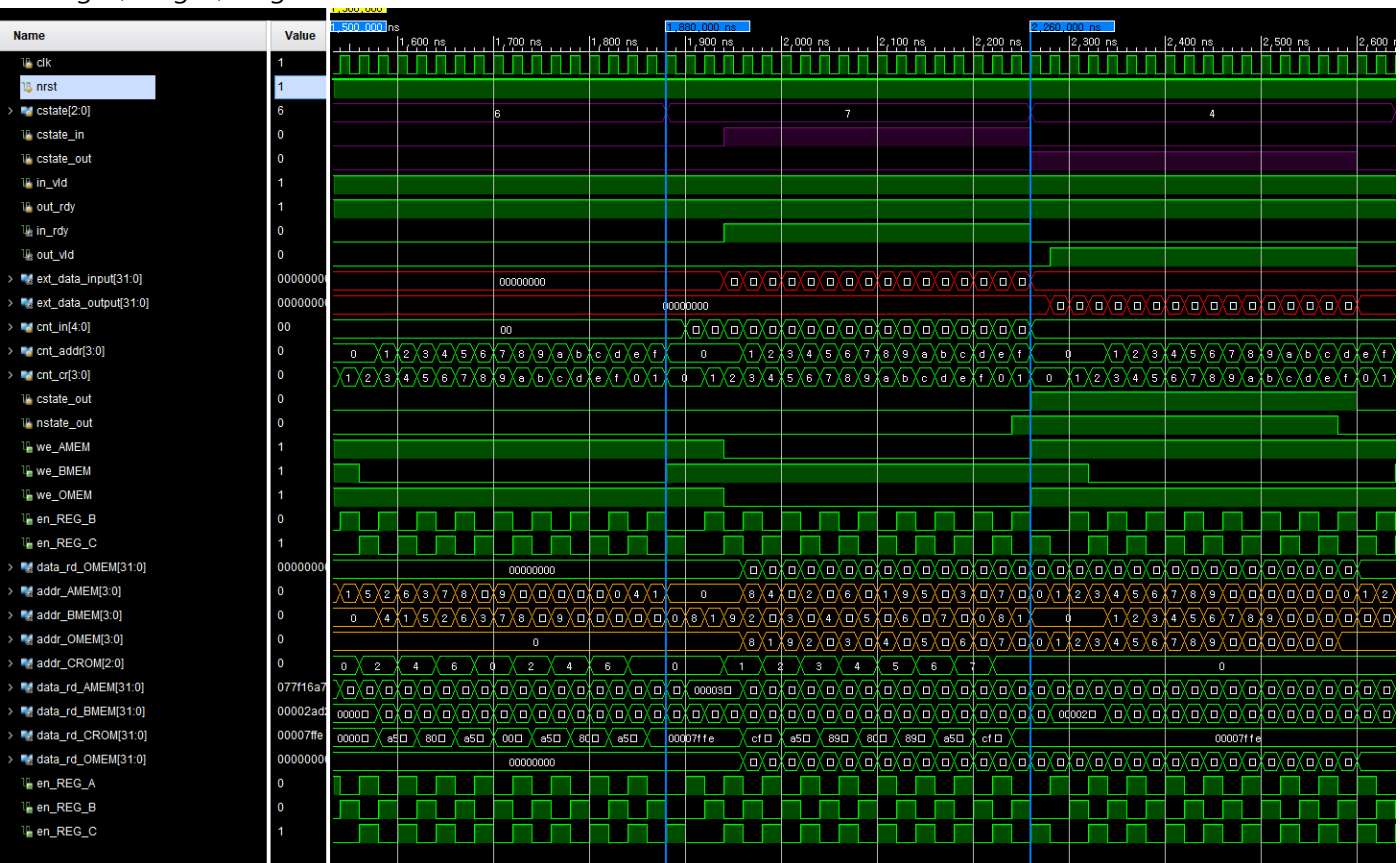
The diagram also shows the LA status signals, including `axi_dma_0_M_AXI_S2MM: TVALID`, `axi_dma_0_M_AXI_S2MM: TREADY`, `axi_dma_0_M_AXI_S2MM: TLAST`, and `axi_dma_0_M_AXI_S2MM: TDATA`. The LA status signals are shown as a series of pulses, indicating the state of the LA at different points in time.

### <Behavioral Simulation>

► IDLE, Stage1, Stage2



► Stage3, Stage4, Stage1



```
4095 : signal : 00d83051a13c6 Temp : 00000ff581cb1
```

```
noise : 00000000000000-dec :      0, signal : 00d83051a13c6-dec :   148563777471942
```

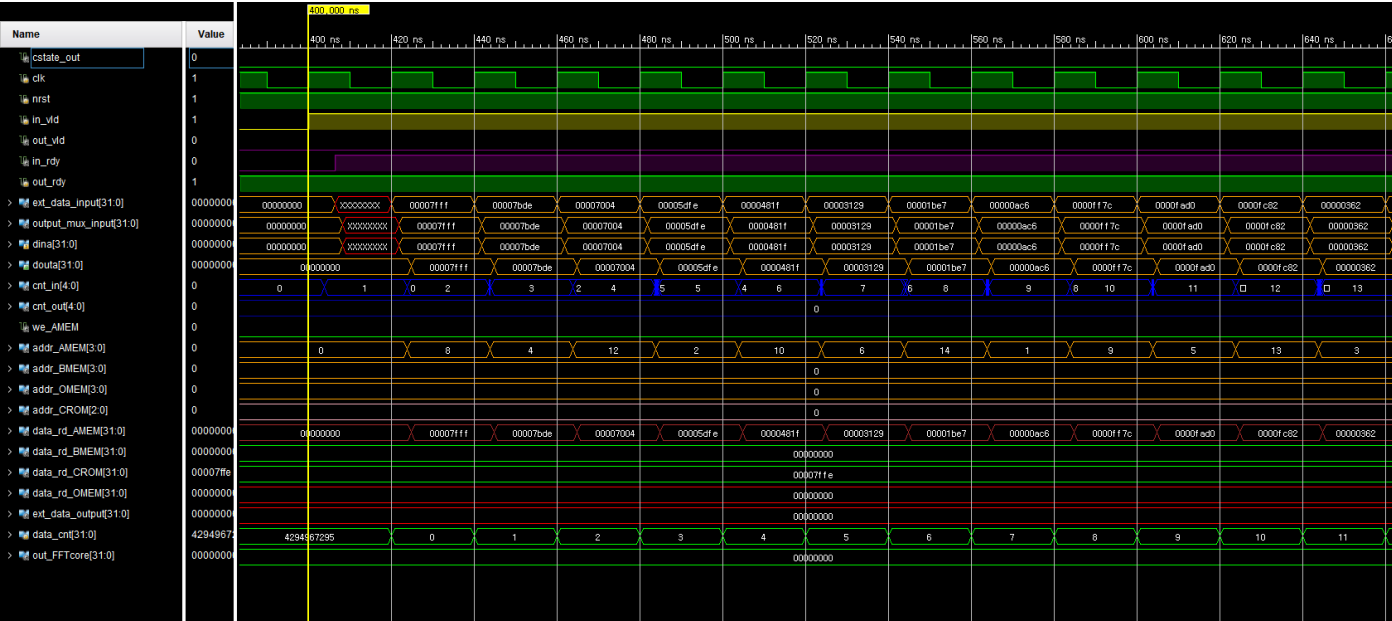
Divided : 0.000000000000000000

NSR : -inf

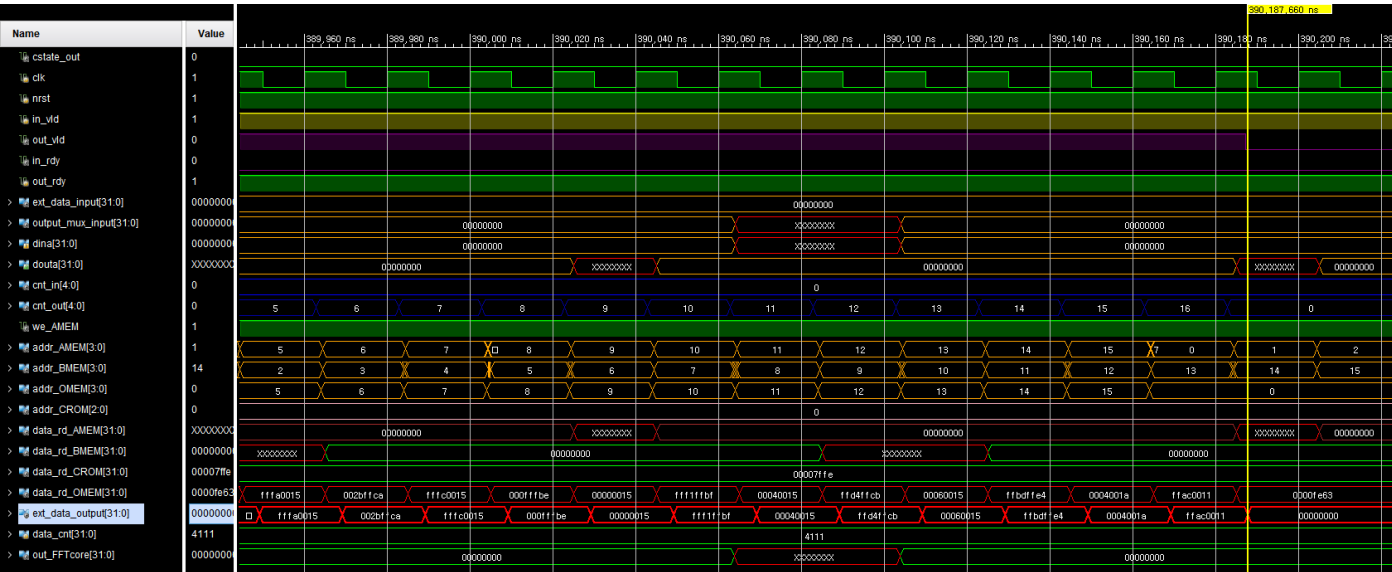
run: Time (s): cpu = 00:00:09 ; elapsed = 00:00:05 , Memory (MB): peak = 2343.930 ; gain = 0.000

### <Synthesis Timing simulation 결과>

▶ 처음 input 값이 들어가는 부분



▶ 마지막 output 값이 나오는 부분



4095 : signal : 00d83051a13c6 Temp : 00000ff581cb1

```
noise : 0000000000000000-dec : 0, signal : 00d83051a13c6-dec : 14856377471942
```

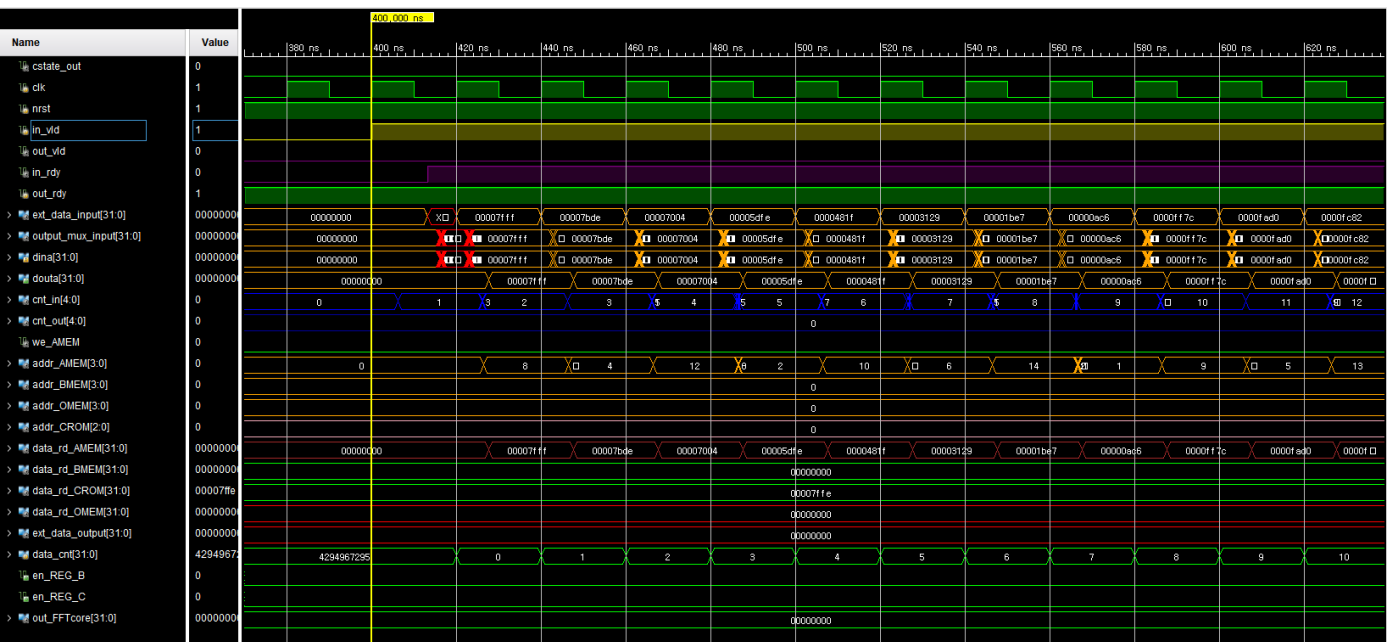
Divided : 0.000000000000000000000000

NSR : -inf

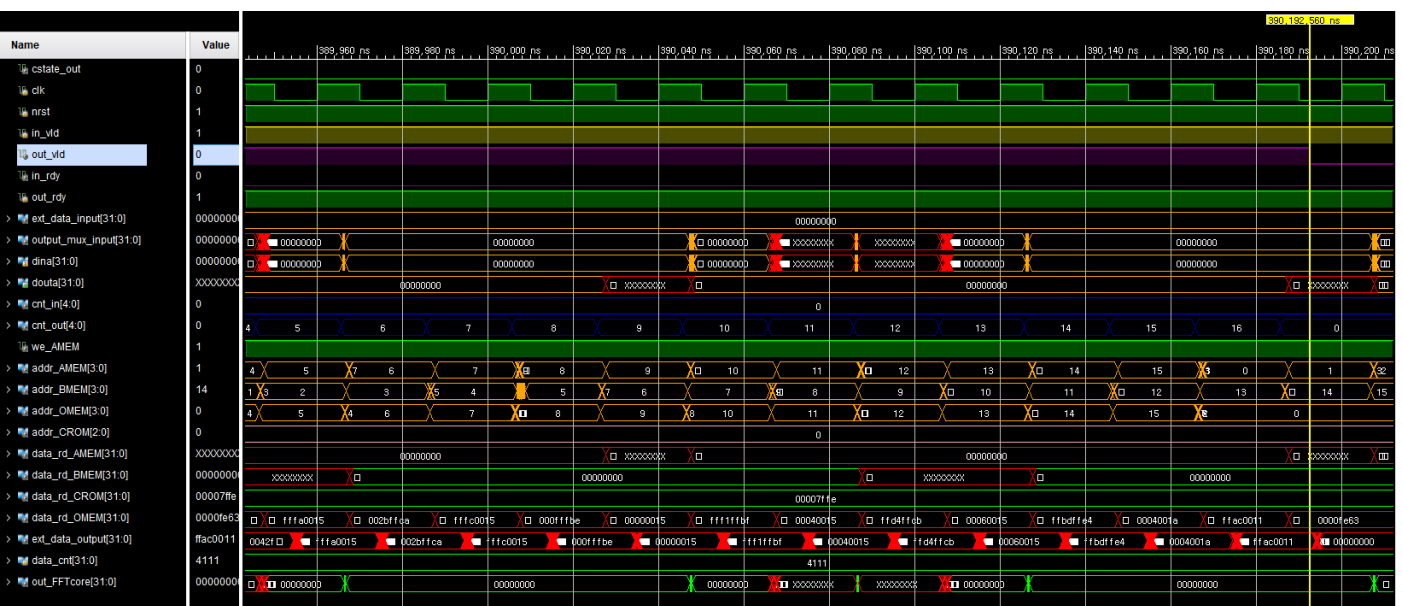
run: Time (s): cpu = 00:00:12 ; elapsed = 00:00:24 , Memory (MB): peak = 2165.402 ; gain = 0.000

### < Post - Implementation Simulation Timing >

▶ 처음 input 값이 들어가는 부분



▶ 마지막 output 값이 나오는 부분



signal : 00d8205c1f715

```
4095 : signal : 00d83051a13c6 Temp : 00000ff581cb1
```

```
noise : 0000000000000-dec :      0, signal : 00d83051a13c6-dec :    14856377471942
```

Divided : 0.00000000000000000000

NSR : -inf

```
run: Time (s): cpu = 00:00:17 ; elapsed = 00:00:49 , Memory (MB): peak = 1937.301 ; gain = 6.328
```

**< Timing Summary - Clock period : 20ns >**

## Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 8.323 ns	Worst Hold Slack (WHS): 0.022 ns	Worst Pulse Width Slack (WPWS): 8.750 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 25754	Total Number of Endpoints: 25754	Total Number of Endpoints: 13894

**All user specified timing constraints are met.**

- **Positive Slack**

Summary	
Name	↳ Path 1
Slack	<a href="#">8.323ns</a>
Source	design_1_i/axis_fft_test9_0/instTopFFT/BMEM/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[0].ram.r/prim_noinit.ram/Df
Destination	design_1_i/axis_fft_test9_0/instTopFFT/FFTcore/FFT/MULT/MULT_re/C[40] (rising edge-triggered cell DSP48E1 clocked by clk_fpga_0 {rise@0.000ns fall@10.000ns})
Path Group	clk_fpga_0
Path Type	Setup (Max at Slow Process Corner)
Requirement	20.000ns (clk_fpga_0 rise@20.000ns - clk_fpga_0 rise@0.000ns)
Data Path Delay	<a href="#">9.660ns</a> (logic 6.419ns (66.447%) route 3.241ns (33.553%))
Logic Levels	2 (DSP48E1=1 LUT4=1)
Clock Path Skew	<a href="#">-0.014ns</a>
Clock Uncertainty	<a href="#">0.302ns</a>

- Critical path delay : 9.660ns

**<DMA – Max burst size : 4>**

COM4 - Tera Term VT

메뉴(F) 수정(E) 설정(S) 제어(O) 창(W) 도움말(H)

```
<16-point Fourier Transform>
<Number of Iterations : 256>

Measured Accuracy (vs Floating Point): NSR(dB) = -64.206

Measured Accuracy (vs RTL): NSR(dB) = -inf

-----Benchmarking Start-----
Case 0: FFT Reference
Nr.      Max.      Min.      Average.  Fltr Avg.  Fltr Avg(us)
5,       256063,     255544,   255779,   255764,    767.292
Case 1: FFT Optimization
Nr.      Max.      Min.      Average.  Fltr Avg.  Fltr Avg(us)
5,       141713,     141513,   141567,   141536,    424.608
-----Benchmarking Complete-----

HW FFT is x1.81 faster than Reference
```

☐ Enable Asynchronous Clocks (Auto)  
☐ Enable Scatter Gather Engine  
☐ Enable Micro DMA  
☐ Enable Multi Channel Support  
☐ Enable Control / Status Stream  
Width of Buffer Length Register (8-23) 23 bits  
Address Width (32-64) 32 bits  
☒ Enable Read Channel      ☒ Enable Write Channel  
Number of Channels 1      Number of Channels 1  
Memory Map Data Width 32      Memory Map Data Width 32  
Stream Data Width 32      Stream Data Width (Auto) 32  
Max Burst Size 4      Max Burst Size 4  
☐ Allow Unaligned Transfers      ☐ Allow Unaligned Transfers  
☐ Use Rlength in Status Stream  
☐ Enable Single AXI4 Data Interface

## <Verilog Code >

### ▶ cnt\_addr, cnt\_cr

```
41 reg [4:0] cnt, cnt_out, cnt_in;
42 reg [3:0] cnt_addr, cnt_cr;
43 reg [2:0] cstate, nstate;
44 reg cstate_in, nstate_in, cstate_out, nstate_out;    // IDLE : 0, RUN : 1
45
46
47 localparam
48     IDLE    = 3'b000,
49     RUN     = 3'b001,
50     Stage1  = 3'b100,
51     Stage2  = 3'b101,
52     Stage3  = 3'b110,
53     Stage4  = 3'b111;
```

- **cnt\_addr** : Stage 2, 4의 addr\_AMEM와 Stage1, 3의 addr\_BMEM를 위한 카운터
- **cnt\_cr** : addr\_CROM을 위한 카운터

### ▶ cnt, cnt\_cr 선언

```
55 always@(posedge clk) begin
56     if(!nrst) begin
57         cnt <= 0;
58         cnt_cr <= 0;    // addr_CROM 값 만들기 위한 counter
59     end
60     else begin
61         if(in_vld == 1'b0 && out_rdy == 1'b0) begin    // 둘 다 0일때 종료
62             cnt <= 0;
63             cnt_cr <= 0;
64         end
65         else if(cnt == 5'd18) begin
66             cnt <= 0;
67             cnt_cr <= 0;
68         end
69         else if(cstate == IDLE) begin
70             cnt <= 0;
71             cnt_cr <= 0;
72         end
73         else begin
74             cnt <= cnt + 5'd1;
75             cnt_cr <= cnt;
76         end
77     end
78 end
79 end
```

- **cnt** : 0~18 카운터
- **cnt\_cr** : cnt의 1 clock delay된 카운터
- **in\_vld**와 **out\_rdy**가 모두 0일때만 카운터 종료

### ▶ cnt\_addr 선언

```
90 /* Stage 2, 4의 addr_AMEM와 Stage1, 3의 addr_BMEM를 위한 cnt의 2 clock delay된 카운터 */
91 always@(posedge clk) begin
92     if(!nrst) begin
93         cnt_addr <= -1;
94     end
95     else begin
96         if(in_vld == 1'b0 && out_rdy == 1'b0) begin    // 둘 다 0일때 FFT 종료
97             cnt_addr <= -1;
98         end
99         else if(cnt_addr == 4'd15) begin
100             cnt_addr <= 0;
101         end
102         else if(cstate == IDLE && in_rdy == 1) begin    // IDLE 일때
103             if(in_vld == 1) begin    // in_vld가 1일때만 카운터 작동
104                 cnt_addr <= cnt_addr + 4'd1;
105             end
106             else begin
107                 cnt_addr <= cnt_addr;    // in_vld가 0일때는 유지
108             end
109         end
110         else begin
111             if(cnt > 2) begin
112                 cnt_addr <= cnt - 2;
113             end
114             else
115                 cnt_addr <= 0;
116         end
117     end
118 end
```

- **IDLE & in\_rdy==1 & in\_vld==1** : 일반적인 카운터로 작동
- **나머지 state** : cnt의 2 clock period delay된 카운터로 동작
- **cstate == IDLE & in\_rdy == 1**
  - in\_vld == 1 : 카운터로 작동
  - in\_vld == 0 : 카운터 값 유지

### ▶ cnt\_in 선언

```
120 /* cnt_in */
121 always@(posedge clk) begin
122     if (!nrst) begin
123         cnt_in <= 0;
124     end
125     else if(in_vld == 1'b0 && out_rdy == 1'b0) begin // handshake 성립 안될 시, 종료
126         cnt_in <= 0;
127     end
128     else begin
129         case (cstate)
130             IDLE : begin // IDLE
131                 if(cnt_in == 5'd15) begin
132                     cnt_in <= 0;
133                 end
134                 else begin
135                     if(in_vld == 1'b1 && in_rdy == 1'b1) begin
136                         cnt_in <= cnt_in + 5'd1;
137                     end
138                     else begin
139                         cnt_in <= cnt_in;
140                     end
141                 end
142             end
143             Stage4 : begin // Stage4
144                 if(cnt_in == 5'd15) begin
145                     cnt_in <= 0;
146                 end
147                 else if(cnt < 3) begin
148                     cnt_in <= 0;
149                 end
150                 else begin
151                     cnt_in <= cnt_in + 5'd1;
152                 end
153             end
154             default begin
155                 cnt_in <= 0;
156             end
157         endcase
158     end
159 end
```

- hand shake 성립 안될 시 종료
- IDLE : hand shake 성립 시, 0~15 카운터로 작동
- Stage 4 : cnt=3일 때 0부터 시작

### ▶ cnt\_out 선언

```
156 /* cnt_out */
157 always@(posedge clk) begin
158     if(!nrst) begin
159         cnt_out <= 0;
160     end
161     else begin
162         if(in_vld == 1'b0 && out_rdy == 1'b0) begin // handshake 성립 안될 시, 종료
163             cnt_out <= 0;
164         end
165         else if(cnt_out == 5'd16) begin
166             cnt_out <= 0;
167         end
168         else if(cstate_out == RUN) begin
169             cnt_out <= cnt_out + 5'd1;
170         end
171         else begin
172             cnt_out <= 0;
173         end
174     end
175 end
```

- hand shake 성립 안될 시 종료
- cstate\_out == RUN 일때, 카운터 동작



## ▶ F/F 선언

```
178 always @(posedge clk)
179 begin
180     if(!nrst) begin
181         cstate <= IDLE;
182         cstate_in <= IDLE[0];
183         cstate_out <= IDLE;
184     end
185     else if ( in_vld == 1'b0 && out_rdy == 1'b0) begin // hand shake 성립 안될 시, 모든 state = IDLE, FFT 종료
186         cstate <= IDLE;
187         cstate_in <= IDLE[0];
188         cstate_out <= IDLE;
189     end
190     else begin
191         cstate <= nstate;
192         cstate_in <= nstate_in;
193         cstate_out <= nstate_out;
194     end
195 end
```

· hand shake 성립 안될 시, 모든 state를 IDLE 처리해 FFT module를 종료한다.

## ▶ Next State를 위한 Combination Logic 선언 – cstate

```
197 /* cstate, cstate_in, cstate_out */
198 always @(*) begin
199     case(cstate)
200         IDLE : begin
201             if(in_vld == 1'b1 && cnt_in == 5'd15) begin
202                 nstate <= Stage1;
203             end
204             else begin
205                 nstate <= IDLE;
206             end
207         end
208         Stage1 : begin
209             if(cnt == 5'd18) begin
210                 nstate <= Stage2;
211             end
212             else begin
213                 nstate <= Stage1;
214             end
215         end
216         Stage2 : begin
217             if(cnt == 5'd18) begin
218                 nstate <= Stage3;
219             end
220             else begin
221                 nstate <= Stage2;
222             end
223         end
224         Stage3 : begin
225             if(cnt == 5'd18) begin
226                 nstate <= Stage4;
227             end
228             else begin
229                 nstate <= Stage3;
230             end
231         end
232         Stage4 : begin
233             if(cnt == 5'd18) begin
234                 nstate <= Stage1;
235             end
236             else begin
237                 nstate <= Stage4;
238             end
239         end
240         default : nstate <= IDLE;
241     endcase
```

· default를 설정하므로써, 불필요한 Latch 제거

· IDLE일때는 맨 처음 입력값이 들어오는 단계  
이므로, 15가 될 때 state 변경

· 나머지 state에서는 cnt가 18이 될 때마다  
state 바뀌도록 설정

### ▶ Next State를 위한 Combination Logic 선언 – cstate\_in

```

IDLE[0] : begin
    if((cstate == IDLE && in_vld == 1) || (cstate == Stage4 && cnt == 2)) begin
        nstate_in <= RUN[0];
    end
    else begin
        nstate_in <= IDLE[0];
    end
end

RUN[0] : begin
    if((cstate == IDLE || cstate == Stage4) && cnt_in == 5'd15) begin // 15 달성시 IDLE로 복구
        nstate_in <= IDLE[0];
    end
    else begin
        nstate_in <= RUN[0];
    end
end

default : nstate_in <= RUN[0];

```

- IDLE : cnt이 2일때 RUN으로 변화
- RUN : cnt\_in이 15일 때, 다시 IDLE로 변화

### ▶ Next State를 위한 Combination Logic 선언 – cstate\_out

```

265 case(cstate_out)
266     IDLE[0] : begin
267         if(cstate == Stage4 && cnt == 5'd18) begin
268             nstate_out <= RUN[0];
269         end
270         else begin
271             nstate_out <= IDLE[0];
272         end
273     end
274
275     RUN[0] : begin
276         if(cnt_out == 16) begin
277             nstate_out <= IDLE[0];
278         end
279         else begin
280             nstate_out <= RUN[0];
281         end
282     end
283     default : nstate_out <= IDLE[0];
284 endcase
285 end

```

- IDLE : 현재 state가 Stage4이고, cnt가 18가 됐을 때, RUN으로 변화
- RUN : 16개의 출력을 위한, 카운트가 끝난 후, 다시 IDLE로 변화

\* 전부 하나의 always문 안에 있다.

### ▶ en\_REG\_A, en\_REG\_B, en\_REG\_C

```

289 assign en_REG_A = cnt[0];
290
291 always @(posedge clk)
292 begin
293     if(!nrst) begin
294         en_REG_B <= 0;
295         en_REG_C <= 0;
296     end
297     else if(cstate != IDLE) begin
298         en_REG_B <= en_REG_A; // delay
299         en_REG_C <= en_REG_B; // delay
300     end
301     else begin
302         en_REG_B <= 0;
303         en_REG_C <= 0;
304     end
305 end

```

- en\_REG\_A를 cnt의 LSB로 설정
- non-blocking 할당문을 통해 delay를 걸어주었다.

### ▶ addr\_CROM 할당

```
assign addr_CROM = (cnt > 0 && cnt < 17) ? (cstate == Stage1 ? 0 : (cstate == Stage2 ? cnt_cr[1] * 4 :  
                                (cstate == Stage3 ? 2*cnt_cr[1]+4*cnt_cr[2] : (cstate == Stage4 ? cnt_cr[1]+2*cnt_cr[2]+4*cnt_cr[3] : 0)))) : 0;
```

- 삼항 연산자를 통해 여러 조건에 따라 addr\_CROM 할당
- 기본 전제 :  $0 < \text{cnt} < 17$  일때만 참
- Stage 1 : 전부 0
- Stage 2 : cnt\_cr[1]를 활용해 주소 나타내었다
- Stage 3 : cnt\_cr[1]과 cnt\_cr[2] 활용해 주소를 나타내었다
- Stage 4 : cnt\_cr[1], cnt\_cr[2], cnt\_cr[3] 활용해 주소를 나타내었다

### ▶ out\_vld, in\_rdy 할당

```
315  
316 assign out_vld = cstate_out && cnt_out; // cnt_out >= 1 부터 참  
317 assign in_rdy = cstate_in; // cstate_in == RUN 일때 in_rdy = 1
```

- out\_vld : cstate\_out = RUN이고 cnt\_out가 1 이상일 때부터 참
- in\_rdy : cstate\_in = RUN일 때 참

### ▶ we\_AMEM, we\_BMEM, we\_OMEM 할당

```
318 assign we_AMEM = (cstate == Stage1 || cstate == Stage3) ? 1 : (cstate == IDLE ? 0 : (cnt < 3 ? 1 : 0));  
319 assign we_BMEM = (cstate == Stage2 || cstate == Stage4) ? 1 : (cstate == IDLE ? 1 : (cnt < 3 ? 1 : 0));  
320 assign we_OMEM = (cstate == Stage4 && cnt >= 3) ? 0 : 1;
```

- 제공된 Timing diagram을 참고해 삼항 연산자를 통해 신호 할당

### ▶ sel\_input, sel\_res, sel\_mem 할당

```
322 assign sel_input = in_rdy;  
323 assign sel_res = en_REG_C;  
324 assign sel_mem = (cstate == Stage2 || cstate == Stage4) ? 1 : 0;
```

- 제공된 Timing diagram을 참고해 신호를 알맞게 할당하였다.

### ▶ addr\_AMEM, addr\_BMEM, addr\_OMEM 할당

```
// stage1 : 그대로, stage2 : 3201, stage3 : 3021, stage4 : 0123  
assign addr_AMEM = we_AMEM ? (cstate == Stage1 ? {cnt[3], cnt[2], cnt[1], cnt[0]} : (cstate == Stage3 ? {cnt[3], cnt[0], cnt[2], cnt[1]} : 0)) // AMEM OUT  
                    : (cstate == Stage2 ? {cnt_addr[3], cnt_addr[2], cnt_addr[0], cnt_addr[1]} : {cnt_addr[0], cnt_addr[1], cnt_addr[2], cnt_addr[3]}); // AMEM IN  
  
// stage1 : 그대로, stage2 : 3201, stage3 : 3021, stage4 : 0321  
assign addr_BMEM = we_BMEM ? (cstate == Stage2 ? {cnt[3], cnt[2], cnt[0], cnt[1]} : (cstate == Stage4 ? {cnt[0], cnt[3], cnt[2], cnt[1]} : 0)) // BMEM OUT  
                    : (cstate == Stage1 ? {cnt_addr[3], cnt_addr[2], cnt_addr[1], cnt_addr[0]} : {cnt_addr[3], cnt_addr[0], cnt_addr[2], cnt_addr[1]}); // BMEM IN  
  
assign addr_OMEM = !(we_OMEM) ? {cnt_addr[0], cnt_addr[3], cnt_addr[2], cnt_addr[1]} : {cnt_out[3], cnt_out[2], cnt_out[1], cnt_out[0]};  
  
//////////Edit code above!//////////  
endmodule
```

- addr\_AMEM : Stage1, Stage3일때는 cnt를 활용해 규칙성을 찾아 신호 설정, Stage2, Stage4일때는 cnt\_addr를 활용해 신호 설정하였다.
- addr\_BMEM : addr\_BMEM과 마찬가지로, Stage1, Stage3에서는 cnt를 활용, Stage2, Stage4에서는 cnt\_addr를 활용해 신호를 설정하였다.
- addr\_OMEM : cnt\_addr와 cnt\_out를 활용해 신호를 설정하였다.