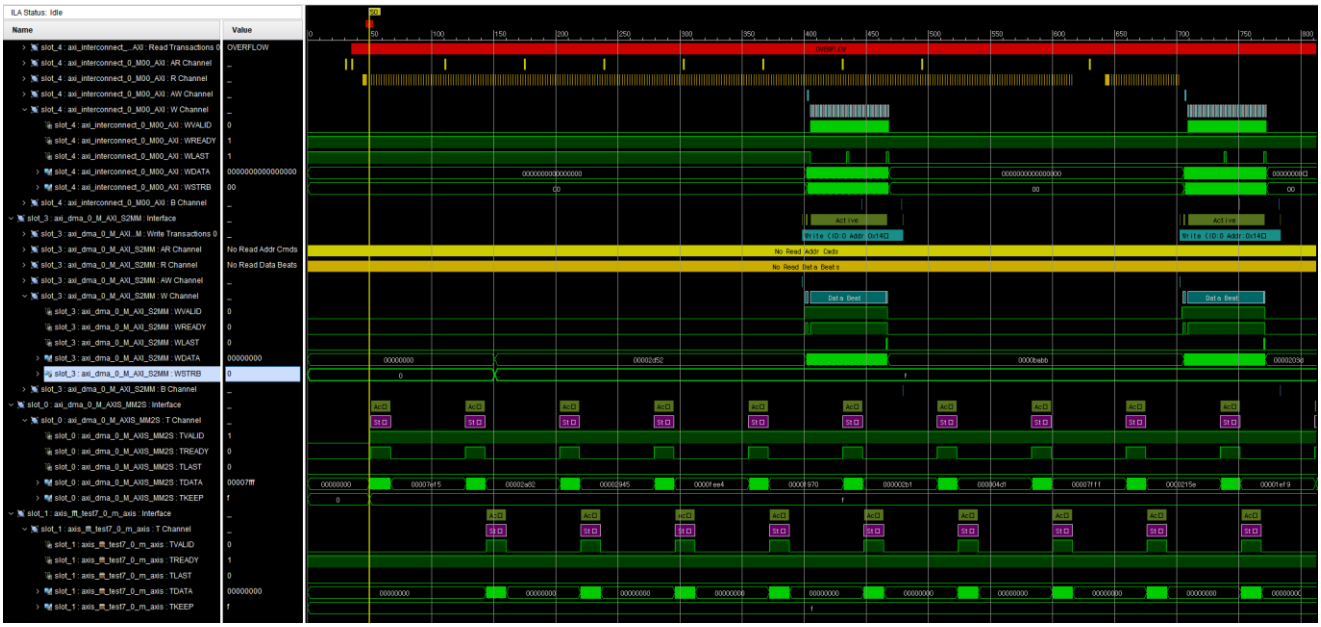
16-pt FFT
결과보고서

전기전자공학부
201810528 고려욱
201810845 박종혁

<SDK 출력 결과>

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

Measured Accuracy (vs Floating Point): NSR(dB) = -64.286

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
```

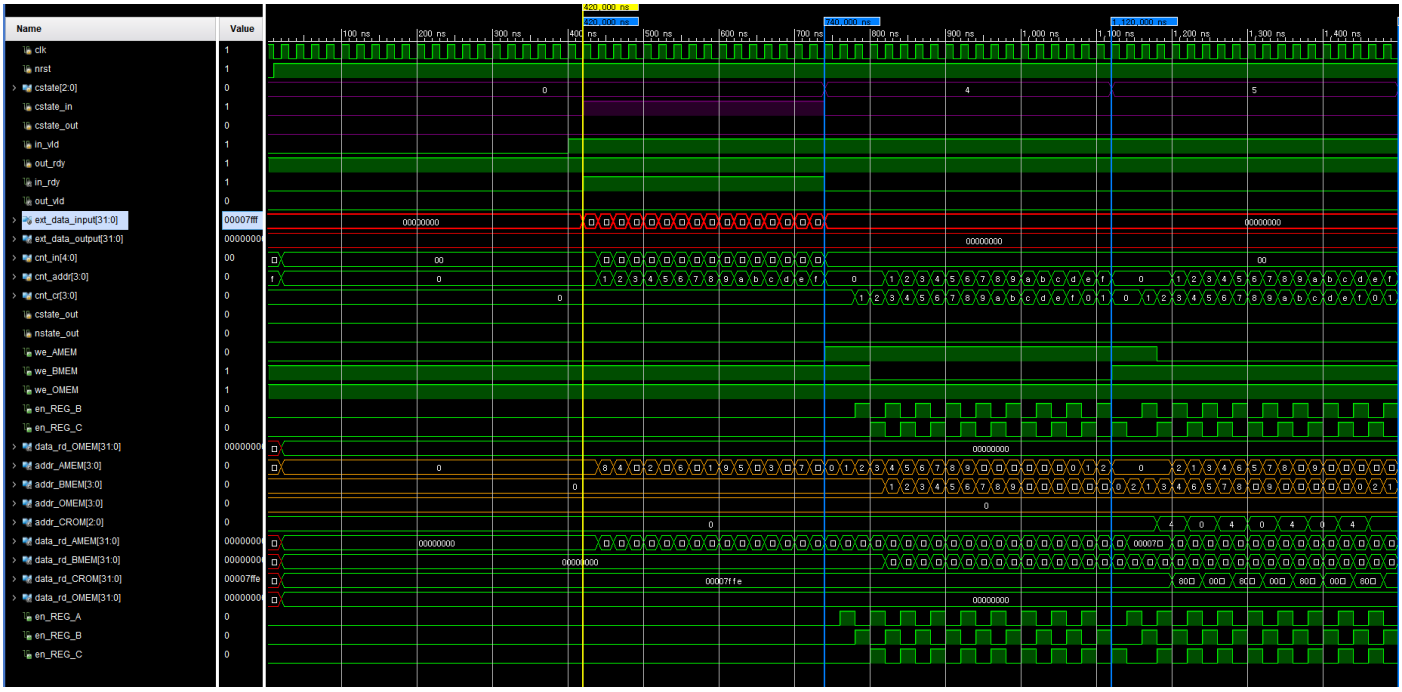
▶ ILA 파형 – 처음 Input data가 FFT module에 들어가는 부분



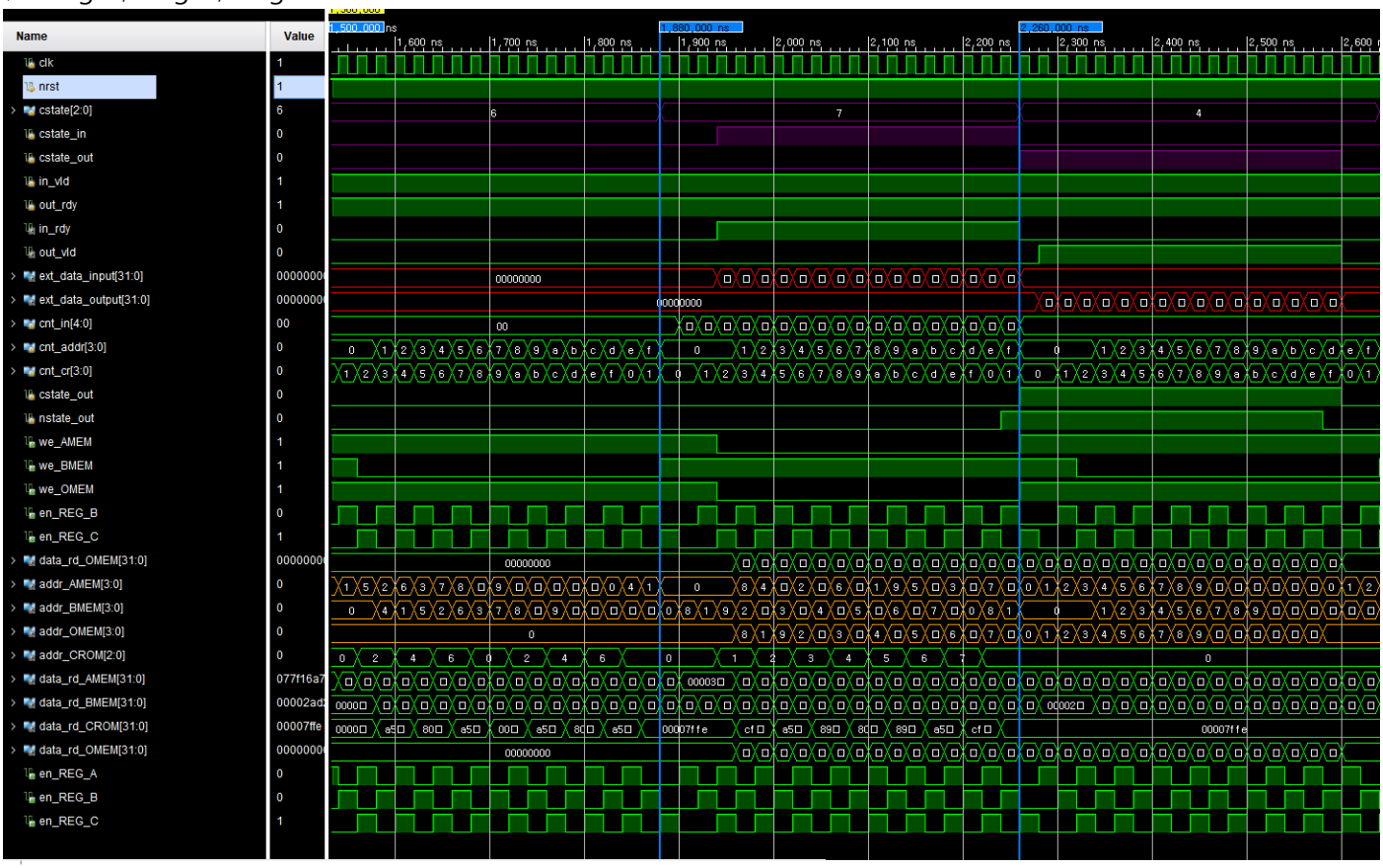
▶ ILA 파형 – FFT module의 마지막 출력 부분

<Behavioral Simulation>

▶ IDLE, Stage1, Stage2



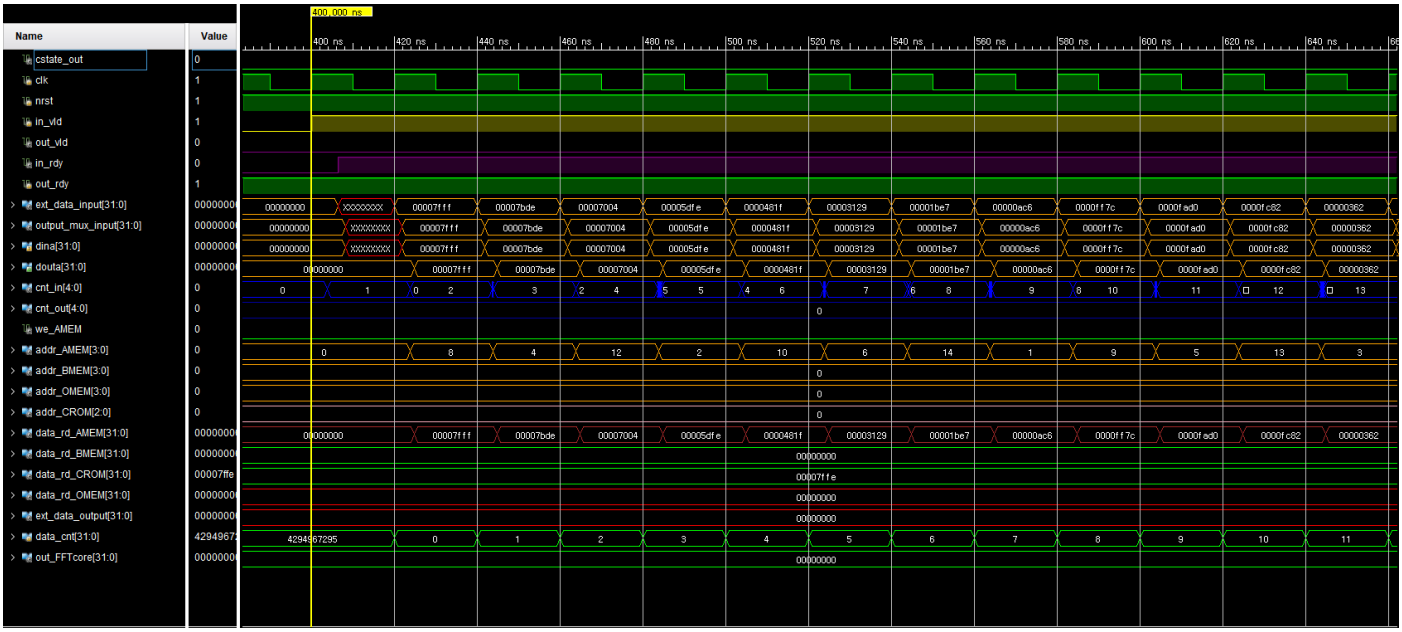
▶ Stage3, Stage4, Stage1



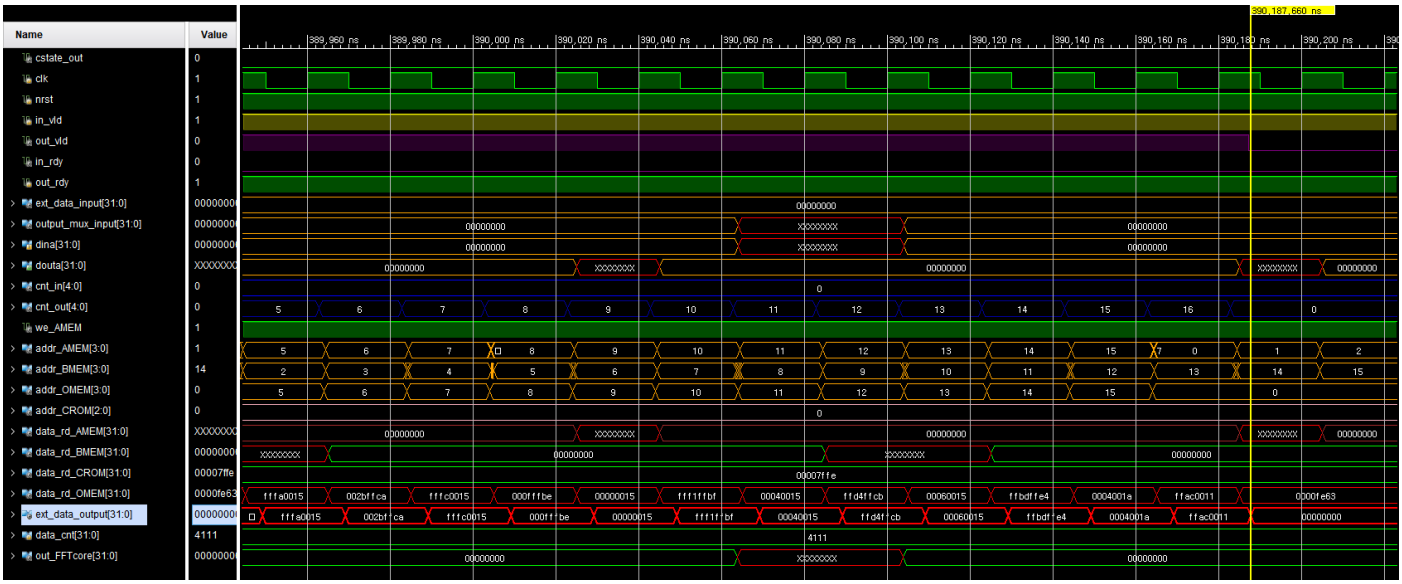
```
        4095 : signal : 00d83051a13c6 Temp : 00000ff581cb1

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

Divided : 0.000000000000000000000

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



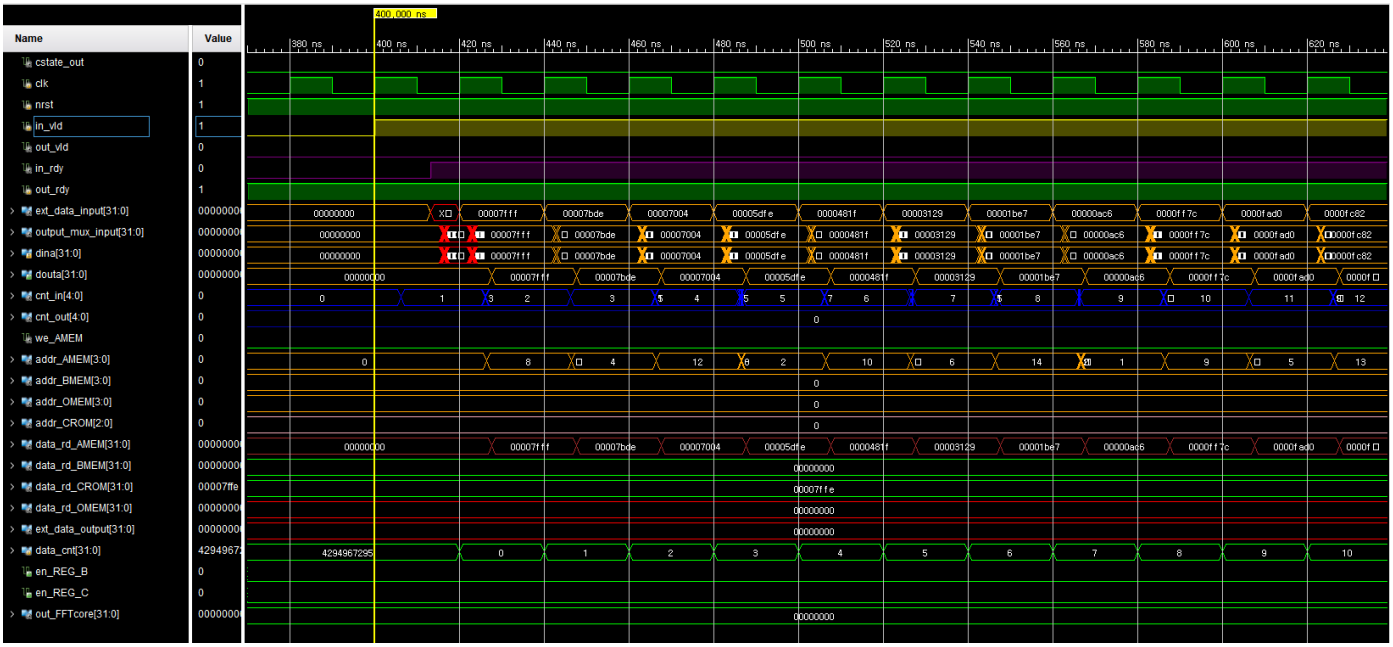
▶ 마지막 output 값이 나오는 부분



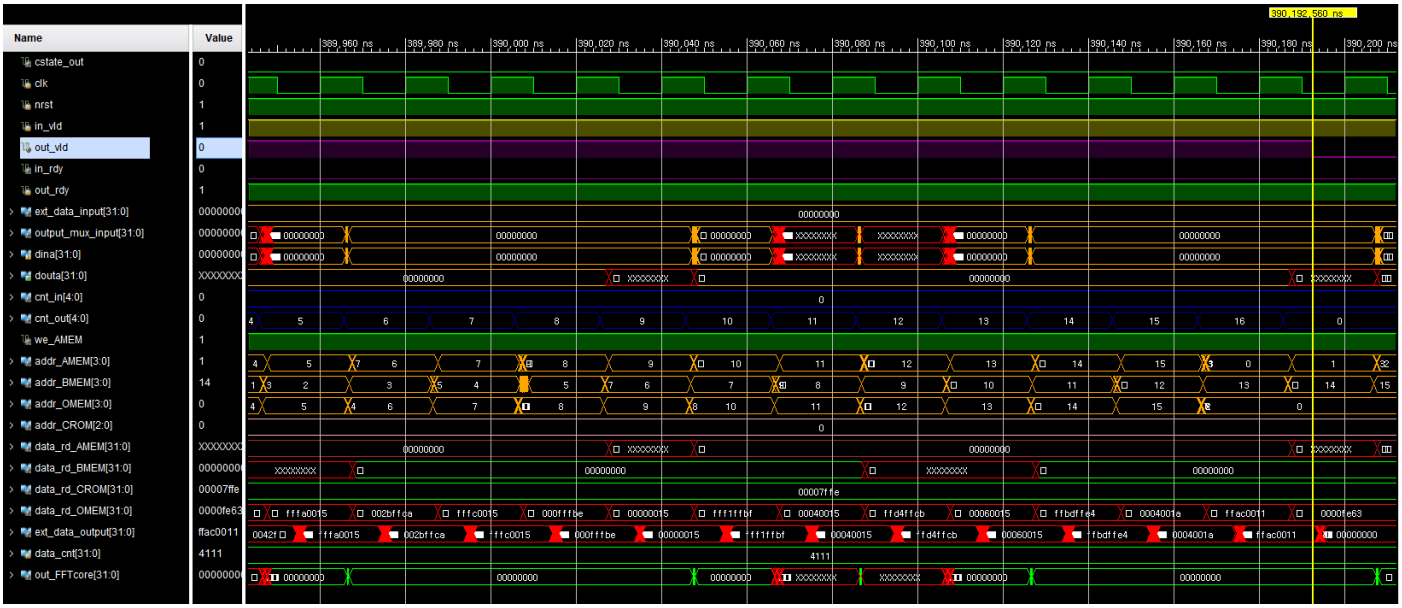

```
        4095 : signal : 00d83051a13c6 Temp : 00000ff581cb1

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

Divided : 0.00000000000000000000

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 : 0.0000000000000000000

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.

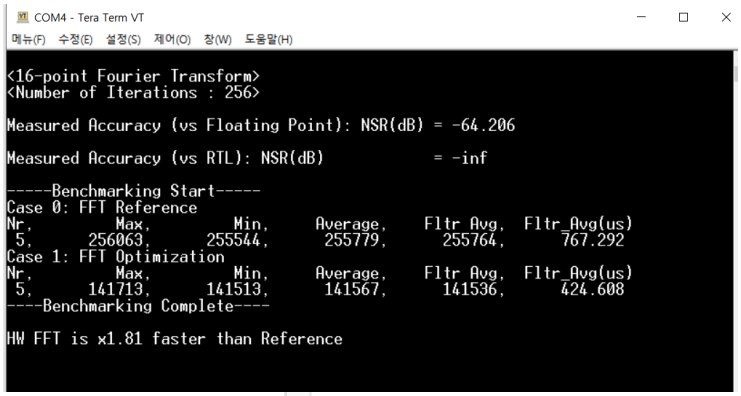
· **Positive Slack**

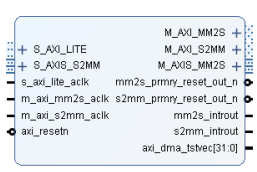
Summary

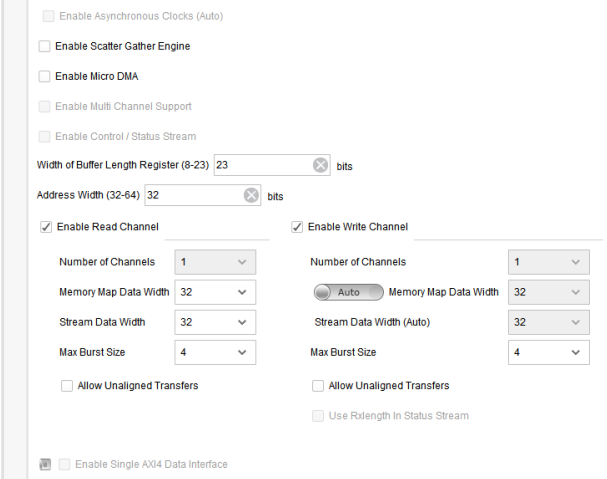
Name	Path 1
Slack	8.323ns
Source	design_1_i/axis_fft_test9_0/inst/TopFFT/BMEM/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[0].ram.r/prim_noinit.ram/DE
Destination	design_1_i/axis_fft_test9_0/inst/TopFFT/FFTcore/FFT/MULT/MULT_re/C[40] (rising edge-triggered cell DSP48E1 clocked by clk_fpga_0 {rise@0.000ns fall@1(
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	9.660ns (logic 6.419ns (66.447%) route 3.241ns (33.553%))
Logic Levels	2 (DSP48E1=1 LUT4=1)
Clock Path Skew	-0.014ns
Clock Un...rtainty	0.302ns

· Critical path delay : 9.660ns

<DMA – Max burst size : 4>







<Verilog Code>

▶ cnt_addr, cnt_cr

```
reg [4:0] cnt, cnt_out, cnt_in;
reg [3:0] cnt_addr, cnt_cr;
reg [2:0] cstate, nstate;
reg cstate_in, nstate_in, cstate_out, nstate_out;       // IDLE : 0, RUN : 1


localparam
    IDLE    = 3'b000,
    RUN     = 3'b001,
    Stage1  = 3'b100,
    Stage2  = 3'b101,
    Stage3  = 3'b110,
    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

```
always@(posedge clk) begin
    if(!nrst) begin
        cnt <= 0;
        cnt_cr <= 0;    // addr_CROM 값 만들기 위한 counter
    end
    else begin
        if(in_vld == 1'b0 && out_rdy == 1'b0) begin     // 둘 다 0일때 종료
            cnt <= 0;
            cnt_cr <= 0;
        end
        else if(cnt == 5'd18) begin
            cnt <= 0;
            cnt_cr <= 0;
        end
        else if(cstate == IDLE) begin
            cnt <= 0;
            cnt_cr <= 0;
        end
        else begin
            cnt <= cnt + 5'd1;
            cnt_cr <= cnt;
        end
    end
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

```
/* Stage 2, 4의 addr_AMEM와 Stage1, 3의 addr _BMEM를 위한 cnt의 2 clock delay된 카운터 */
always@(posedge clk) begin
    if(!nrst) begin
        cnt_addr <= -1;
    end
    else begin
        if(in_vld == 1'b0 && out_rdy == 1'b0) begin     // 둘 다 0일때 FFT 종료
            cnt_addr <= -1;
        end
        else if(cnt_addr == 4'd15) begin
            cnt_addr <= 0;
        end
        else if(cstate == IDLE && in_rdy == 1) begin    // IDLE 일때
            if(in_vld == 1) begin       // in_vld가 1일때만 카운터 작동
                cnt_addr <= cnt_addr + 4'd1;
            end
            else begin
                cnt_addr <= cnt_addr;   // in_vld가 0일때는 유지
            end
        end
        else begin
            if(cnt > 2) begin
                cnt_addr <= cnt - 2;
            end
            else
                cnt_addr <= 0;
        end
    end
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**

```
/* cnt_in */
always@(posedge clk) begin
    if (!nrst) begin
        cnt_in <= 0;
    end
    else if(in_vld == 1'b0 && out_rdy == 1'b0) begin    // handshake 성립 안될 시, 종료
            cnt_in <= 0;
    end
    else begin
        case (cstate)
            IDLE : begin    // IDLE
                if(cnt_in == 5'd15) begin
                    cnt_in <= 0;
                end
                else begin
                    if(in_vld == 1'b1 && in_rdy == 1'b1) begin
                        cnt_in <= cnt_in + 5'd1;
                    end
                    else begin
                        cnt_in <= cnt_in;
                    end
                end
            end
            Stage4 : begin  // Stage4
                if(cnt_in == 5'd15) begin
                    cnt_in <= 0;
                end
                else if(cnt < 3) begin
                    cnt_in <= 0;
                end
                else begin
                    cnt_in <= cnt_in + 5'd1;
                end
            end
            default begin
                cnt_in <= 0;
            end
        endcase
    end
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**

```
/* cnt_out */
always@(posedge clk) begin
    if(!nrst) begin
        cnt_out <= 0;
    end
    else begin
        if(in_vld == 1'b0 && out_rdy == 1'b0) begin     // handshake 성립 안될 시, 종료
            cnt_out <= 0;
        end
        else if(cnt_out == 5'd16) begin
            cnt_out <= 0;
        end
        else if(cstate_out == RUN) begin
            cnt_out <= cnt_out + 5'd1;
        end
        else begin
            cnt_out <= 0;
        end
    end
end
```

· hand shake 성립 안될 시 종료

· **cstate_out == RUN** 일때, 카운터 동작

▶ **F/F 선언**

```
always @(posedge clk)
begin
    if(!nrst) begin
       cstate <= IDLE;
       cstate_in <= IDLE[0];
       cstate_out <= IDLE;
    end
    else if ( in_vld == 1'b0 && out_rdy == 1'b0) begin     // hand shake 성립 안될 시, 모든 state = IDLE, FFT 종료
       cstate <= IDLE;
       cstate_in <= IDLE[0];
       cstate_out <= IDLE;
    end
    else begin
       cstate <= nstate;
       cstate_in <= nstate_in;
       cstate_out <= nstate_out;
    end
end
```

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

▶ **Next State를 위한 Combination Logic 선언 – cstate**

```
/* cstate, cstate_in, cstate_out */
always @(*) begin
    case(cstate)
            IDLE : begin
                if(in_vld == 1'b1 && cnt_in == 5'd15) begin
                   nstate <= Stage1;
                end
                else begin
                   nstate <= IDLE;
                end
            end
            Stage1 : begin
                if(cnt == 5'd18) begin
                    nstate <= Stage2;
                end
                else begin
                    nstate <= Stage1;
                end
            end
            Stage2 : begin
                if(cnt == 5'd18) begin
                    nstate <= Stage3;
                end
                else begin
                    nstate <= Stage2;
                end
            end
            Stage3 : begin
                if(cnt == 5'd18) begin
                    nstate <= Stage4;
                end
                else begin
                    nstate <= Stage3;
                end
            end
            Stage4 : begin
                if(cnt == 5'd18) begin
                    nstate <= Stage1;
                end
                else begin
                    nstate <= Stage4;
                end
            end
            default : nstate <= IDLE;
    endcase
```

· default를 설정하므로써, 불필요한 Latch 제거

· IDLE일때는 맨 처음 입력값이 들어오는 단계이므로, 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**

```
    case(cstate_out)
            IDLE[0] : begin
                if(cstate == Stage4 && cnt == 5'd18) begin
                   nstate_out <= RUN[0];
                end
                else begin
                   nstate_out <= IDLE[0];
                end
            end

            RUN[0] : begin
                if(cnt_out == 16) begin
                    nstate_out <= IDLE[0];
                end
                else begin
                    nstate_out <= RUN[0];
                end
            end
            default : nstate_out <= IDLE[0];
    endcase
end
```

** 전부 하나의 always문 안에 있다.*

· **IDLE** : 현재 state가 Stage4이고, cnt가 18가 됐을 때, RUN으로 변화

· **RUN** : 16개의 출력을 위한, 카운트가 끝난 후, 다시 IDLE로 변화

▶ **en_REG_A, en_REG_B, en_REG_C**

```
assign en_REG_A = cnt[0];

always @(posedge clk)
begin
    if(!nrst) begin
        en_REG_B <=0;
        en_REG_C <=0;
    end
    else if(cstate != IDLE) begin
        en_REG_B <= en_REG_A;       // delay
        en_REG_C <= en_REG_B;       // delay
    end
    else begin
        en_REG_B <= 0;
        en_REG_C <= 0;
    end
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 < 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

```
314
315   assign out_vld = cstate_out && cnt_out;     // cnt_out >= 1 부터 참
316   assign in_rdy = cstate_in;      // cstate_in == RUN 일때 in_rdy = 1
317
```

· **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

```
317
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;
321
```

· 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;
325
```

· 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 cnt을 활용해 규칙성을 찾아 신호 설정, Stage2, Stage4일때는 cnt_addr를 활용해 신호 설정하였다.

· **addr_BMEM :** addr_BMEM과 마찬가지로, Stage1, Stage3에서는 cnt을 활용, Stage2, Stage4에서는 cnt_addr를 활용해 신호를 설정하였다.

· **addr_OMEM :** cnt_addr과 cnt_out를 활용해 신호를 설정하였다.