64-pt FFT
결과보고서

전기전자공학부
201810528 고려욱
201810845 박종혁

<SDK 출력 결과>

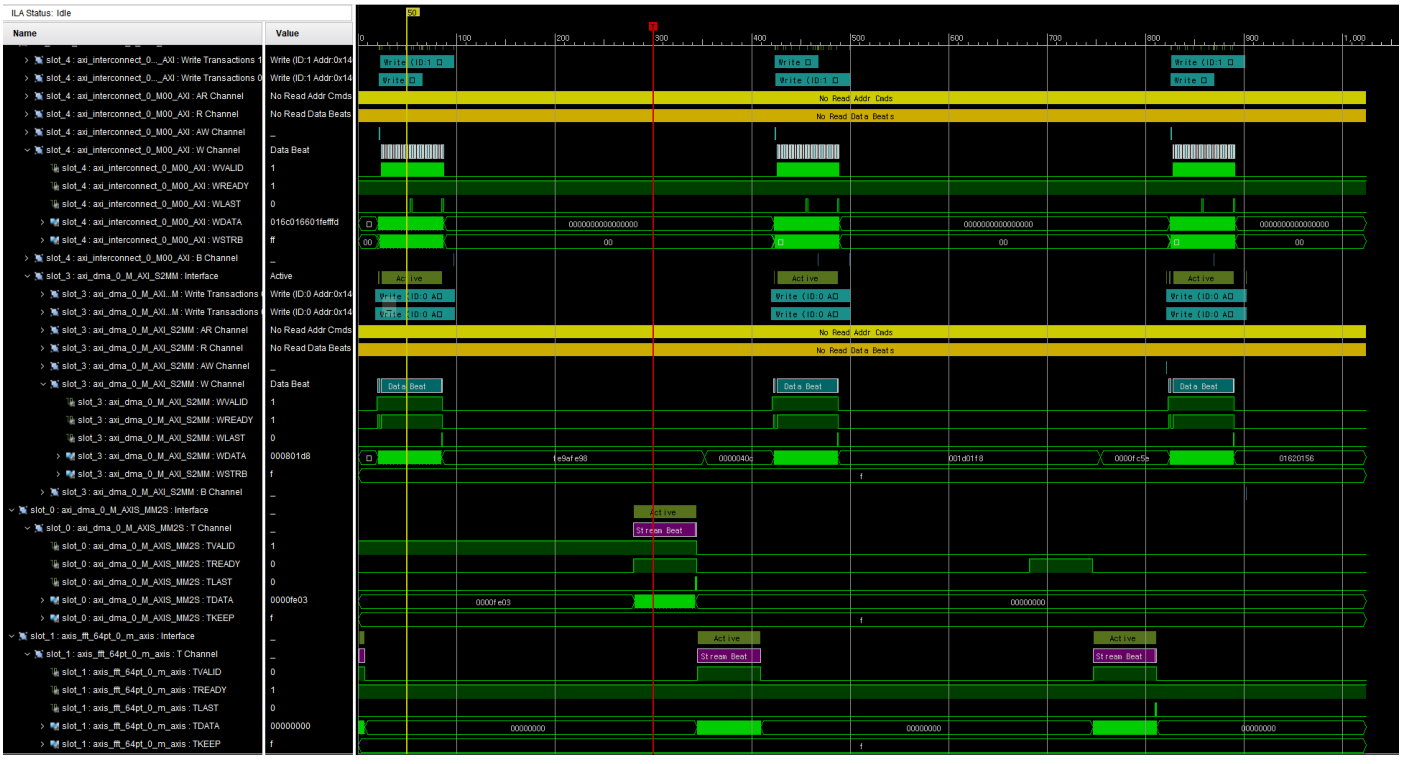
COM5 - Tera Term VT

메뉴(F) 수정(E) 설정(S) 제어(O) 창(W) 도움말(H)

```
<64-point Fourier Transform>
<Number of Iterations : 64>

Measured Accuracy (vs Floating Point): NSR(dB) = -57.459

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

-----Benchmarking Start-----
Case 0: FFT Reference
Nr,         Max,         Min,     Average,    Fltr Avg,  Fltr_Avg(us)
10,      340108,      339836,      339939,      339931,     1019.793
Case 1: FFT Optimization
Nr,         Max,         Min,     Average,    Fltr Avg,  Fltr_Avg(us)
10,      184336,      184250,      184265,      184258,      552.774
----Benchmarking Complete----

HW FFT is x1.85 faster than Reference
```

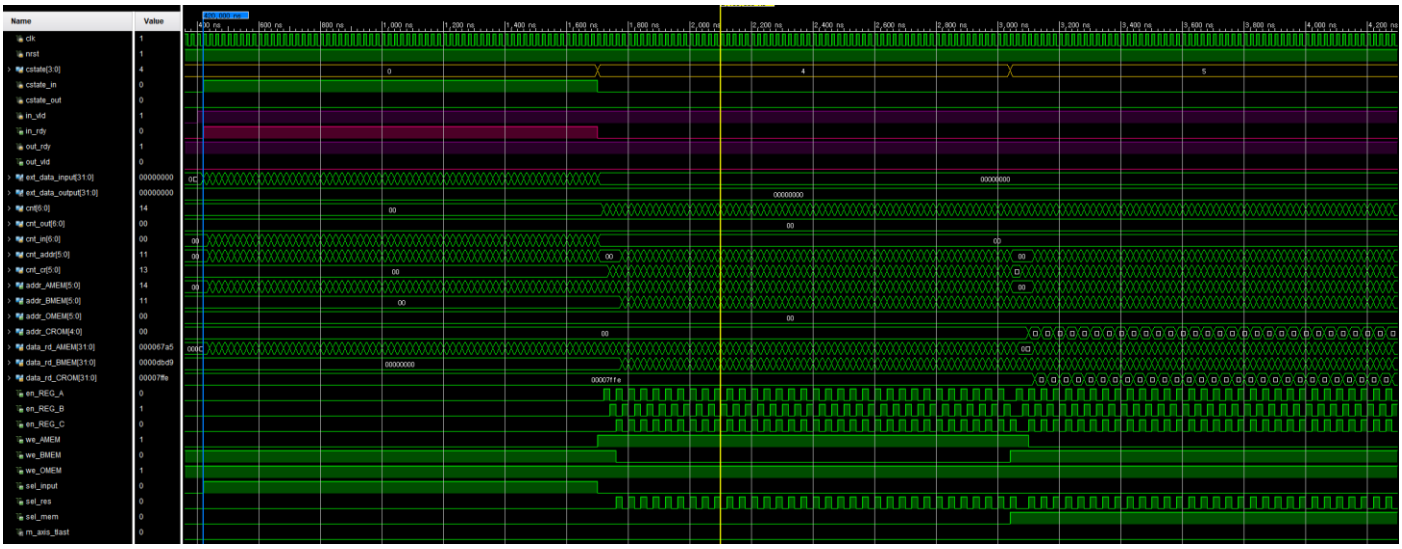
▶ ILA 파형 – 처음 Input data가 FFT module에 들어가는 부분

▶ ILA 파형 – FFT module의 마지막 출력 부분

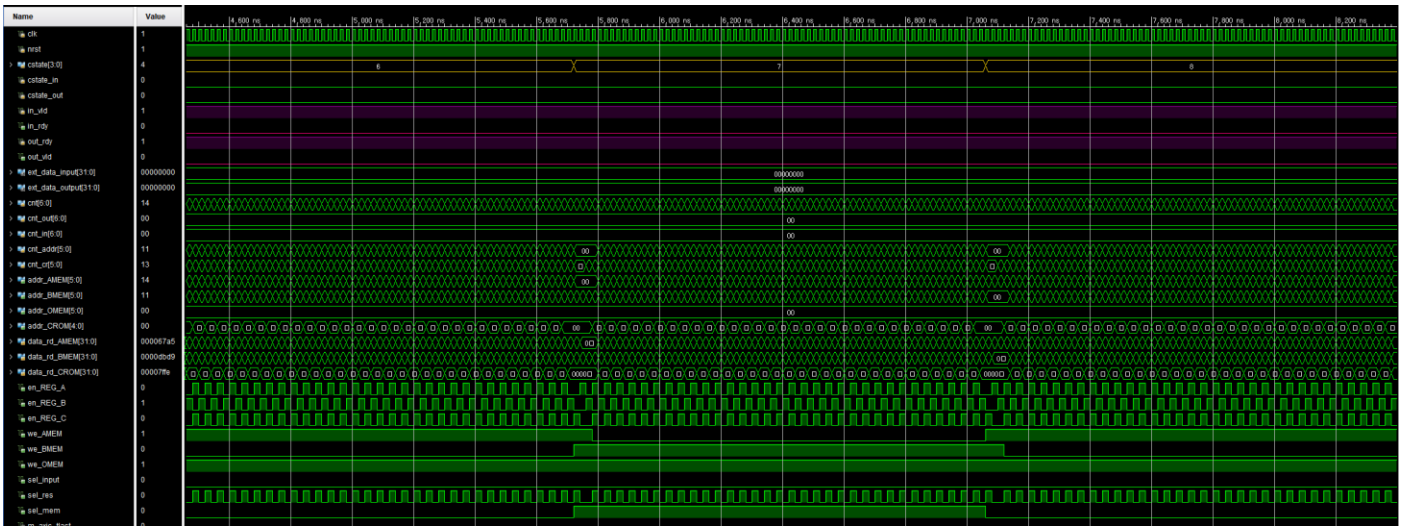


<Behavioral Simulation>

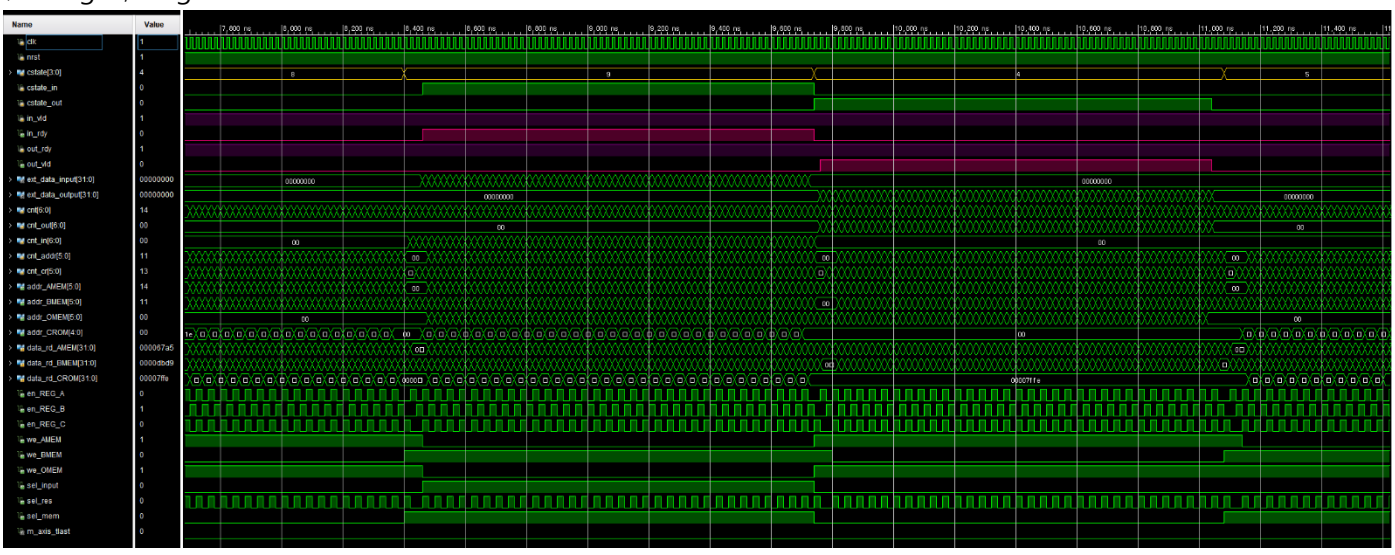
▶ IDLE, Stage1, Stage2



▶ Stage3, Stage4, Stage5

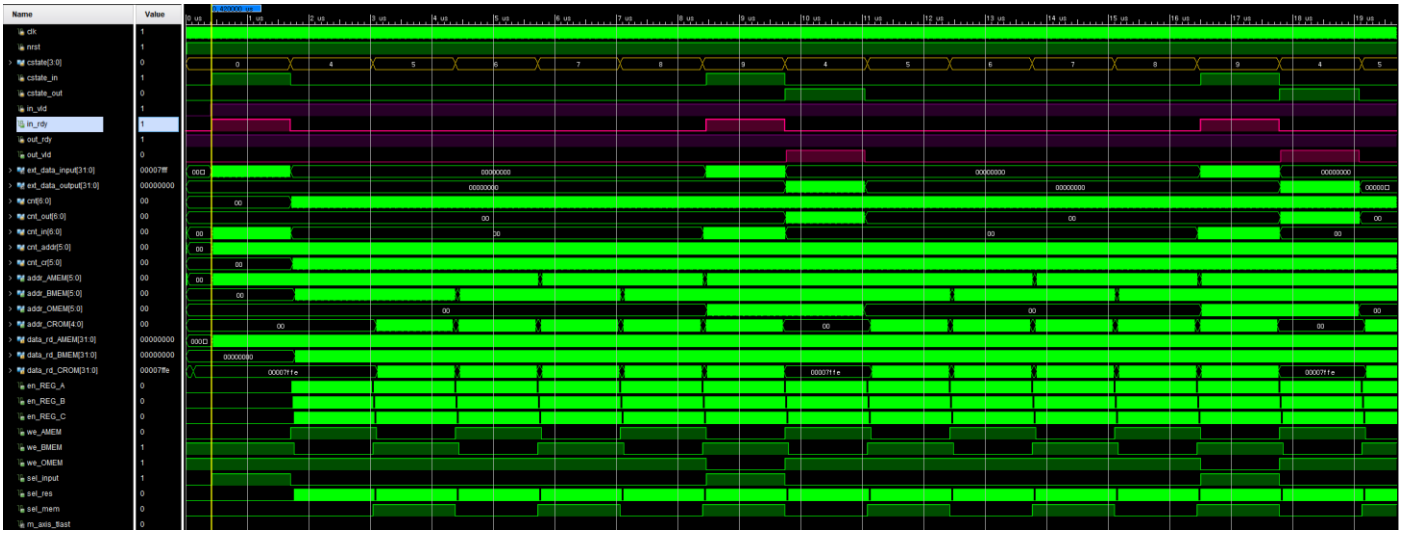


▶ Stage6, Stage1



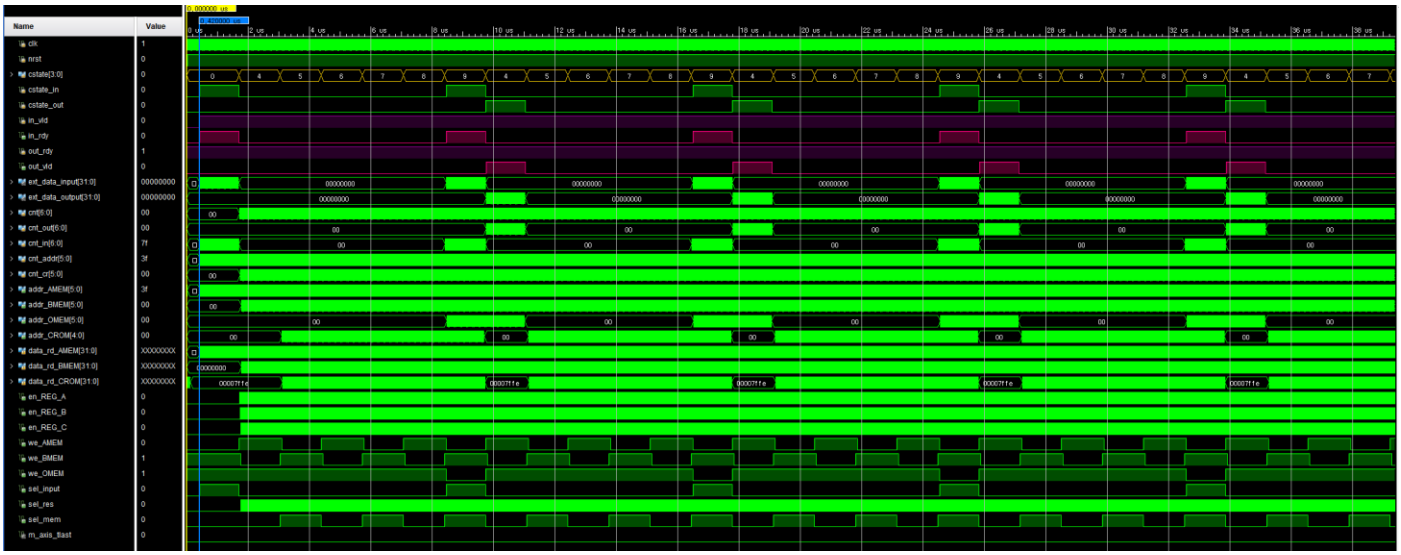
```
        4095 : signal : 00c3fddff6f7d Temp : 000000003b268

noise : 0000000000000-dec :                0, signal : 00c3fddff6f7d-dec :   13468446977917

Divided : 0.0000000000000000000

NSR : -inf
```

<Synthesis Timing simulation>



```
signal : 00c3fddfbbd15

        4095 : signal : 00c3fddff6f7d Temp : 000000003b268

noise : 0000000000000-dec :                0, signal : 00c3fddff6f7d-dec :   13468446977917

Divided : 0.00000000000000000000

NSR : -inf
```

< Post - Implementation Simulation Timing >



```
        4095 : signal : 00c3fddff6f7d Temp : 000000003b268

noise : 0000000000000-dec :                0, signal : 00c3fddff6f7d-dec :   13468446977917

Divided : 0.00000000000000000000

NSR : -inf
run: Time (s): cpu = 00:00:04 ; elapsed = 00:00:05 . Memory (MB): peak = 2386.477 ; gain = 0.000
```

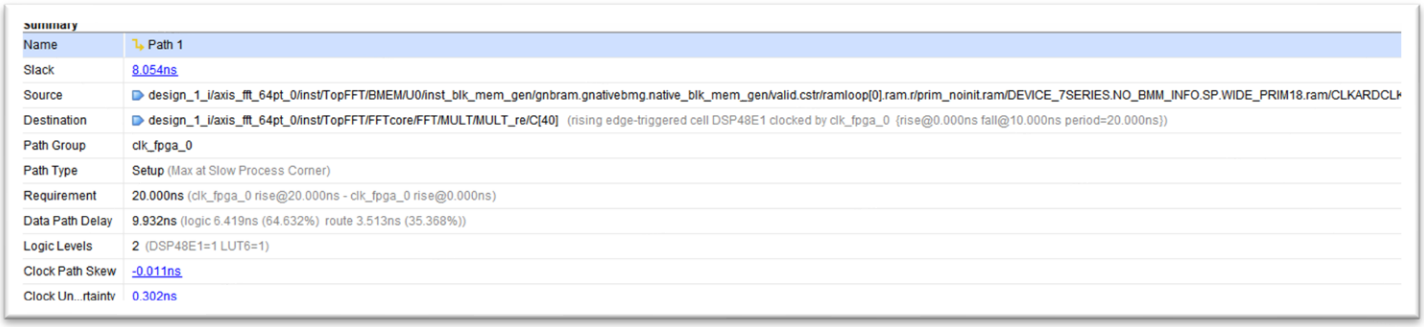
< Timing Summary - clock period : 20ns >

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.054 ns	Worst Hold Slack (WHS):	0.025 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:	25786	Total Number of Endpoints:	25786	Total Number of Endpoints:	13904

All user specified timing constraints are met.

· **Positive Slack**



Name	Path 1
Slack	8.054ns
Source	design_1_i/axis_fft_64pt_0/inst/TopFFT/BMEM/U0/inst_blk_mem_gen/gnbram.gnativebmg.native_blk_mem_gen/valid.cstr/ramloop[0].ram.r/prim_noinit.ram/DEVICE_7SERIES.NO_BMM_INFO.SP.WIDE_PRIM18.ram/CLKARDCLK
Destination	design_1_i/axis_fft_64pt_0/inst/TopFFT/FFTcore/FFT/MULT/MULT_re/C[40] (rising edge-triggered cell DSP48E1 clocked by clk_fpga_0 {rise@0.000ns fall@10.000ns period=20.000ns})
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	9.932ns (logic 6.419ns (64.632%) route 3.513ns (35.368%))
Logic Levels	2 (DSP48E1=1 LUT6=1)
Clock Path Skew	-0.011ns
Clock Un...rtainty	0.302ns

· Critical path delay : 9.932ns

<Verilog Code>

▶ 레지스터 변수 및 파라미터 선언

```
reg [6:0] cnt, cnt_out, cnt_in;
reg [5:0]  cnt_addr, cnt_cr;
reg [3:0] cstate, nstate;
reg cstate_in, nstate_in, cstate_out, nstate_out;       // IDLE : 0, RUN : 1


localparam
    IDLE    = 4'b0000,
    RUN     = 4'b0001,
    Stage1  = 4'b0100,
    Stage2  = 4'b0101,
    Stage3  = 4'b0110,
    Stage4  = 4'b0111,
    Stage5  = 4'b1000,
    Stage6  = 4'b1001;
```

· **cnt_addr**, **cnt_cr** 추가적으로 선언

▶ cnt, cnt_cr

```
always@(posedge clk) begin
    if(!nrst) begin
        cnt <= 0;
        cnt_cr <= 0;
    end
    else begin
        if(in_vld == 1'b0 && out_rdy == 1'b0) begin
            cnt <= 0;
            cnt_cr <= 0;
        end
        else if(cnt == 7'd66) 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 : 0~66 카운터

- in_vld와 out_rdy가 모두 0일때 카운터 초기화
- cstate == IDLE 일때 카운터 초기화

cnt_cr : addr_CROM을 위한 카운터

- cnt의 1 clock delay된 카운터

▶ cnt_addr

```
/* Stage 2, 4, 6의 addr_AMEM와 Stage1, 3, 5의 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 == 7'd63) begin
            cnt_addr <= 0;
        end
        else if(cstate == IDLE && in_rdy == 1&& in_vld == 1) begin  // IDLE 일때
            cnt_addr <= cnt_addr + 4'd1;
        end
        else begin      // Stage 1, 2, 3, 4, 5, 6
            if(cnt > 2) begin
                cnt_addr <= cnt - 2;
            end
            else
                cnt_addr <= 0;
        end
    end
end
```

cnt_addr : Stage 2, 4, 6의 addr_AMEM와 Stage1, 3, 5의 addr _BMEM를 위한 0~63 카운터

- IDLE && in_rdy ==1 & in_vld==1 : 일반적인 카운터로 작동
- 나머지 state에서는 cnt의 2 clock period delay된 카운터로 동작

▶ cnt_in 선언

```
/* cnt_in */
always@(posedge clk) begin
    if (!nrst) begin
        cnt_in <= -1;
    end
    else if(in_vld == 1'b0 && out_rdy == 1'b0) begin    // handshake 성립 안될 시, 종료
            cnt_in <= -1;
    end
    else begin
        case (cstate)
            IDLE : begin    // IDLE
                if(cnt_in == 7'd63) begin
                    cnt_in <= 0;
                end
                else begin
                    if(in_vld == 1'b1 && in_rdy == 1'b1) begin  // only when handshaking, cnt_in count
                        cnt_in <= cnt_in + 5'd1;
                    end
                    else begin
                        cnt_in <= 0;
                    end
                end
            end
            Stage6 : begin  // Stage6
                if(cnt_in == 7'd63) begin
                    cnt_in <= 0;
                end
                else if(cnt == 0) begin
                    cnt_in <= -2;   // -2로 시작하므로써, cnt_in의 시작 점을 나타낸다
                end
                else begin
                    cnt_in <= cnt_in + 5'd1;
                end
            end
            default begin
                cnt_in <= 0;
            end
        endcase
    end
end
```

cnt_in : input값을 입력하기 위한 카운터

- hand shake 성립 안될 시 종료
- IDLE : hand shake 성립 시, 0~15 카운터로 작동
- Stage4 : cnt=3일 때 0부터 시작이므로, cnt=0일 때 -2부터 시작하므로써 cnt_in 카운터 작동

▶ cnt_out 선언

```
158   /* cnt_out */
159   always@(posedge clk) begin
160       if(!nrst) begin
161           cnt_out <= 0;
162       end
163       else begin
164           if(in_vld == 1'b0 && out_rdy == 1'b0) begin
165               cnt_out <= 0;
166           end
167           else if(cnt_out == 7'd64) begin
168               cnt_out <= 0;
169           end
170           else if(cstate_out == RUN) begin
171               cnt_out <= cnt_out + 5'd1;
172           end
173           else begin
174               cnt_out <= 0;
175           end
176       end
177   end
```

cnt_out : output값을 출력하기 위한 카운터

- hand shake 성립 안될 시 종료
- cstate_out == RUN 일때, 카운터 동작

▶ F/F 선언

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

F/F : state를 바꿔주기 위한 Flip Flop

- 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 == 7'd63) begin
                   nstate <= Stage1;
                end
                else begin
                   nstate <= IDLE;
                end
            end
            Stage1 : begin
                if(cnt == 7'd66) begin
                    nstate <= Stage2;
                end
                else begin
                    nstate <= Stage1;
                end
            end
            Stage2 : begin
                if(cnt == 7'd66) begin
                    nstate <= Stage3;
                end
                else begin
                    nstate <= Stage2;
                end
            end
            Stage3 : begin
                if(cnt == 7'd66) begin
                    nstate <= Stage4;
                end
                else begin
                    nstate <= Stage3;
                end
            end
            Stage4 : begin
                if(cnt == 7'd66) begin
                    nstate <= Stage5;
                end
                else begin
                    nstate <= Stage4;
                end
            end
            Stage5 : begin
                if(cnt == 7'd66) begin
                    nstate <= Stage6;
                end
                else begin
                    nstate <= Stage5;
                end
            end
            Stage6 : begin
                if(cnt == 7'd66) begin
                    nstate <= Stage1;
                end
                else begin
                    nstate <= Stage6;
                end
            end
            default : nstate <= IDLE;
    endcase
```

· default를 설정하므로써, 불필요한 Latch 제거

· IDLE일때는 맨 처음 입력값이 들어오는 단계이므로, 63가 될 때 state 변경

· 나머지 state에서는 cnt가 66이 될 때마다 state 바뀌도록 설정

▶ Next State를 위한 Combination Logic 선언 – cstate_in

```
    // cnt_in 카운터만 사용해서 cstate_in의 case문 구성
    case(cstate_in)
            IDLE[0] : begin
                if((cstate == IDLE && in_vld == 1) || (cstate == Stage6 && cnt_in == 7'b1111111)) begin // cnt_in = -1 일때, RUN
                   nstate_in <= RUN[0];
                end
                else begin
                   nstate_in <= IDLE[0];
                end
            end

            RUN[0] : begin
                if((cstate == IDLE || cstate == Stage6) && cnt_in == 7'd63) begin   // 63 달성시 IDLE로 복구
                    nstate_in <= IDLE[0];
                end
                else begin
                    nstate_in <= RUN[0];
                end
            end
            default : nstate_in <= RUN[0];
    endcase
```

· cnt_in 카운터만 사용하여 구성해 복잡도를 줄임

· **IDLE** : cnt_in이 -1일 때 RUN으로 변화

· **RUN** : cnt_in이 63일 때, 다시 IDLE로 변화

▶ Next State를 위한 Combination Logic 선언 – cstate_out

```
    case(cstate_out)
            IDLE[0] : begin
                if(cstate == Stage6 && cnt == 7'd66) begin
                   nstate_out <= RUN[0];
                end
                else begin
                   nstate_out <= IDLE[0];
                end
            end

            RUN[0] : begin
                if(cnt_out == 64) begin
                    nstate_out <= IDLE[0];
                end
                else begin
                    nstate_out <= RUN[0];
                end
            end
            default : nstate_out <= IDLE[0];
    endcase
end
```

· **IDLE** : 현재 state가 Stage6이고, cnt가 66가 됐을 때, RUN으로 변화

· **RUN** : 64개의 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 < 65) ? (cstate == Stage1 ? 0 : (cstate == Stage2 ? cnt_cr[1] * 16 :
                   (cstate == Stage3 ? 8*cnt_cr[1]+16*cnt_cr[2] : (cstate == Stage4 ? 4*cnt_cr[1]+8*cnt_cr[2]+16*cnt_cr[3] :
                   (cstate == Stage5 ? 2*cnt_cr[1]+4*cnt_cr[2]+8*cnt_cr[3]+16*cnt_cr[4] :
                   (cstate == Stage6 ? 1*cnt_cr[1]+2*cnt_cr[2]+4*cnt_cr[3]+8*cnt_cr[4]+16*cnt_cr[5] : 0))))) : 0;
```

· 삼항 연산자를 통해 여러 조건에 따라 addr_CROM 할당

· **기본 전제** : 0 < cnt < 65 일때 만 참

· **Stage 1** : 전부 0

· **Stage 2** : cnt_cr[1]를 활용해 주소 나타내었다

· **Stage 3** : cnt_cr[1]과 cnt_cr[2] 활용

· **Stage 4** : cnt_cr[1], cnt_cr[2], cnt_cr[3] 활용

· **Stage 5** : cnt_cr[1], cnt_cr[2], cnt_cr[3], cnt_cr[4] 활용

· **Stage 6** : cnt_cr[1], cnt_cr[2], cnt_cr[3], cnt_cr[4], cnt_cr[5] 활용

▶ out_vld, in_rdy 할당

```
assign out_vld = cstate_out && cnt_out;     // cnt_out >= 1 부터 참
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

```
assign we_AMEM  = (cstate == Stage1 || cstate == Stage3 || cstate == Stage5) ? 1 : (cstate == IDLE ? 0 : (cnt < 3 ? 1 : 0));
assign we_BMEM  = (cstate == Stage2 || cstate == Stage4 || cstate == Stage6) ? 1 : (cstate == IDLE ? 1 : (cnt < 3 ? 1 : 0));
assign we_OMEM  = (cstate == Stage6 && cnt >= 3) ? 0 : 1;
```

· 제공된 Timing diagram을 참고해 삼항 연산자를 통해 신호 할당

▶ sel_input, sel_res, sel_mem 할당

```
assign sel_input = in_rdy;
assign sel_res = en_REG_C;
assign sel_mem  = (cstate == Stage2 || cstate == Stage4 || cstate == Stage6) ? 1 : 0;
```

· 제공된 Timing diagram을 참고해 신호를 알맞게 할당하였다.

▶ addr_AMEM, addr_BMEM, addr_OMEM 할당

```
// stage1 : 그대로, stage2 : 543201, stage3 : 543021, stage4 : 540321, stage5 : 504321, stage6 : 012345
assign addr_AMEM =  we_AMEM ? (cstate == Stage1 ? {cnt[5], cnt[4], cnt[3], cnt[2], cnt[1], cnt[0]}
                            : (cstate == Stage3 ? {cnt[5], cnt[4], cnt[3], cnt[0], cnt[2], cnt[1]}
                            : (cstate == Stage5 ?  {cnt[5], cnt[0], cnt[4], cnt[3], cnt[2], cnt[1]}   :  0  )))     // AMEM OUT
                            : (cstate == Stage2 ? {cnt_addr[5], cnt_addr[4], cnt_addr[3], cnt_addr[2], cnt_addr[0], cnt_addr[1]}
                            : (cstate == Stage4 ? {cnt[5], cnt[4], cnt[0], cnt[3], cnt[2], cnt[1]}
                            : {cnt_in[0], cnt_in[1], cnt_in[2], cnt_in[3], cnt_in[4], cnt_in[5]}));   // AMEM IN

// stage1 : 그대로, stage2 : 543201, stage3 : 543021, stage4 : 540321, stage5 : 504321, stage6 : 054321
assign addr_BMEM = we_BMEM ? (cstate == Stage2 ? {cnt[5], cnt[4], cnt[3], cnt[2], cnt[0], cnt[1]}
                            : (cstate == Stage4 ? {cnt[5], cnt[4], cnt[0], cnt[3], cnt[2], cnt[1]}
                            : (cstate == Stage6 ? {cnt[0], cnt[5], cnt[4], cnt[3], cnt[2], cnt[1]} : 0 )))      // BMEM OUT
                            : (cstate == Stage1 ? {cnt_addr[5], cnt_addr[4], cnt_addr[3], cnt_addr[2], cnt_addr[1], cnt_addr[0]}
                            : (cstate == Stage3 ? {cnt_addr[5], cnt_addr[4], cnt_addr[3], cnt_addr[0], cnt_addr[2], cnt_addr[1]}
                            : {cnt_addr[5], cnt_addr[0], cnt_addr[4], cnt_addr[3], cnt_addr[2], cnt_addr[1]}));     // BMEM IN (1, 3, 5)

assign addr_OMEM = !(we_OMEM) ? {cnt_addr[0], cnt_addr[5], cnt_addr[4], cnt_addr[3], cnt_addr[2], cnt_addr[1]}
                            : {cnt_out[5], cnt_out[4], cnt_out[3], cnt_out[2], cnt_out[1], cnt_out[0]};

//////////Edit code above!!//////////
////////////////////////////////////

endmodule
```

· **addr_AMEM** : Stage1, Stage3, Stage5일때는 cnt을 활용해 규칙성을 찾아 신호 설정, Stage2, Stage4, Stage6일때는 cnt_addr를 활용해 신호 설정하였다.

· **addr_BMEM** : addr_BMEM과 마찬가지로, Stage1, Stage3, Stage5에서는 cnt을 활용, Stage2, Stage4, Stage6에서는 cnt_addr를 활용해 신호를 설정하였다.

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