AI디지털집적회로

Assignment #3

전기전자공학부

202110410

조민우

서론

과제 2에서는 과제 1에서 합성한 RCA 기반 8-bit multiplier를 하위 모듈(submodule)로 포함하는 MAC을 대상으로 hierarchical synthesis를 진행하고 Static Timing Analysis(STA)를 수행하였었다.

과제 3에서는 과제 2에서 만든 mac8 unit을 하위 모듈로 사용해 systolic array(sa4x4) 모듈을 hierarchical sysnthesis하고, STA(Static Timing Analysis)를 수행해 SDF(Standard Delay Format) 파일을 생성한다. 여기서 더 나아가, 이 SDF 파일을 QuestaSim 시뮬레이터에 back-annotation하여, 합성된 회로의 delay가 반영된 동적 타이밍 시뮬레이션을 진행하려한다.

앞서 과제들과 마찬가지로 다양한 clk 주파수에서의 timing violation을 검증하여 실제 동작 가능한 주파수를 찾을 예정이다. 또한, PPA간의 trade-off를 확인해보려 한다.

본격적으로 분석에 들어가기 앞서, STA와 post-synthesis timing simulation 두 가지 분석의 차이점에 대해 알아보고, 왜 두 가지 분석이 모두 필요한지 고찰해보고자 한다.

구분	Primetime	Questasim
방식	Static Timing Analysis(STA)	Dynamic Gate level Simulation
입력 Data	Netlist, Timing Library	Netlist, SDF, Testbench
타이밍 모델	이론적 추정(RC, Path Delay 계산)	실제 SDF delay
고려 요소	Setup/Hold, Slew, OCV, Skew 등	실제 신호 전파, Clock Skew, Glitch, Race Condition
결과 형태	Slack, Path Delay, Timing Report	Waveform, functional simulation
장점	Worst-case Timing Analysis 가능	Real Timing Analysis
단점	Dynamic Analysis, glitch 검출 불가	느리고 입력 벡터 의존적
목적	Slack 확보 및 동작 frequency 확인	실제 동작 시 오류 여부 확인
활용 시점	synthesis 또는 layout 후 STA 단계	Post-synthesis/Post-layout 타이밍 검증 단계

Fig 1.1 Comparison between PrimeTime & QuestaSim

* 입력 벡터(Test Vector), 회로의 동작을 stimulate하기 위해 넷리스트에 적용되는 입력 신호를 나타내는 테스트 벡터.

* glitch: combinational logic gate에서 delay로 인해 발생하는 오류 pulse. Gate-level Simulation은 입력 벡터로 실제 신호 전파를 시뮬레이션하므로 glitch를 확인하고 오류 발생 여부를 사전에 검증할 수 있다.

Fig 1.1은 각 툴(Tool)에 따른 타이밍 분석 방법의 목적과 방식을 나타낸다. 가장 큰 차이점은 STA(Static Timing Analysis)는 정적 환경에서 이론적으로 timing violation을 검증

하는 반면, Post-Synthesis Timing Simulation in QuestaSim은 실제 신호 전파와 지연을 포함한 동적 시뮬레이션 기반 검증을 수행한다는 점이다.

구체적으로 STA는 회로의 모든 타이밍 경로를 수학적으로 계산하여 worst-case 조건에서 delay를 예측하고 Timing violation 여부를 판단한다.
이 과정에서는 입력 벡터 없이 전체 경로를 분석하므로 모든 가능한 조합을 빠르게 평가할 수 있지만, glitch나 race condition 등 실제 동작 중 발생할 수 있는 동적 오류는 검출하지 못한다.

반면 QuestaSim의 Post-Synthesis Simulation은 합성된 gate netlist와 SDF 파일을 사용하여, 셀의 실제 지연이 반영된 상태에서 신호가 시간에 따라 어떻게 전파되는지를 시뮬레이션한다. 따라서 STA에서는 문제가 없더라도, 실제 시뮬레이션에서는 clock skew, setup margin 부족, signal collision 등으로 인해 오류가 발생할 수 있다. 이는 QuestaSim 기반 시뮬레이션이 실제 칩 동작 환경에 더 가까운 realistic timing behavior를 반영하기 때문이다.

결과적으로, STA는 이론적 타이밍 여유 분석에, Post-Synthesis Simulation은 실제 동작 타이밍 검증에 각각 특화되어 있으며, 최종적인 타이밍 검증은 두 결과를 상호 보완적으로 해석하는 것이 좋다.

본론

1. RTL code Preview

-se8

```
13 module se8 (
14         input                           clk,
15         input                           rstn,
16         input                           en,
17         input                           req_out,
18         input   [7:0]           data_from_left,
19         input   [15:0]          data_from_top,
20         output  reg     [7:0]   data_to_right,
21         output  reg     [15:0]  data_to_bottom
22 );
23
24         wire    [15:0]  mac_out;
25         mac8    U_mac8  (       .clk(clk), .rstn(rstn), .en(en), .A(data_from_left), .B(data_from_top[7:0]), .M(mac_out) );
26
27         reg             [1:0]   req_out_cycle;
28         always @ (posedge clk) begin
29                 if (~rstn)
30                         req_out_cycle <= 0;
31                 else if(req_out)
32                         req_out_cycle <= 1;
33                 else if (req_out_cycle != 0)
34                         req_out_cycle <= req_out_cycle == 3 ? 0 : req_out_cycle + 1;
35                 else
36                         req_out_cycle <= req_out_cycle;
37         end
38
39         always @ (posedge clk) begin
40                 if (~rstn) begin
41                         data_to_right <= 0;
42                         data_to_bottom <= 0;
43                 end else begin
44                         data_to_right <= en ? data_from_left : data_to_right;
45                         data_to_bottom <= req_out ? mac_out : (req_out_cycle != 0) ? data_from_top : en ? data_from_top : data_to_bottom;
46                 end
47         end
```

<Fig 2.1.1 se8>

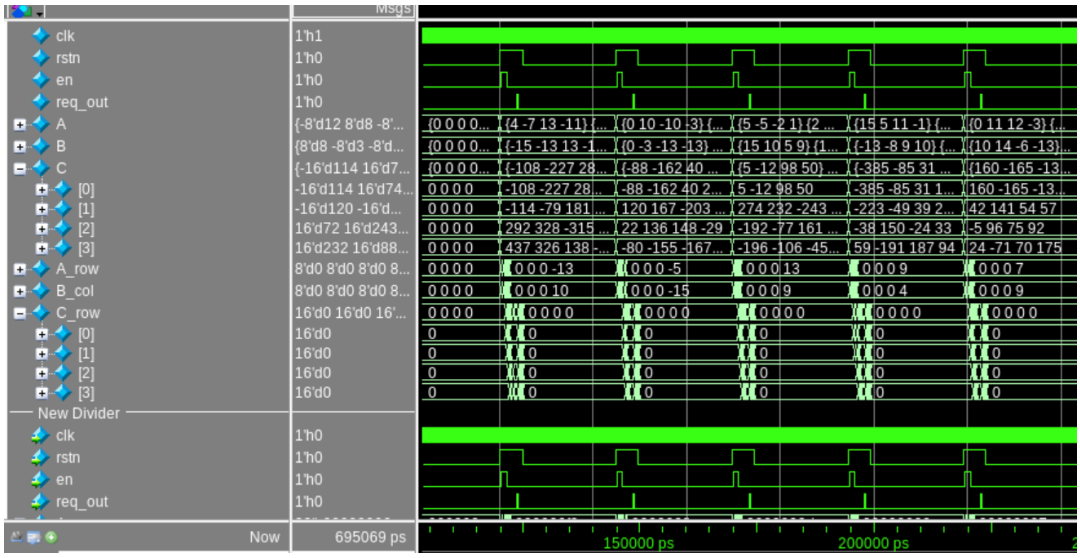
se8 모듈은 4x4 systolic array의 PE(Processing Element) 역할을 하는 모듈이다. PE의 역할은 mac8 unit을 instance하여 이웃해 있는 PE로부터 데이터를 전달받아 mac 연산을 진행하고, req_out 신호에 따라 output reg에 값을 할당한다. 그리고, cycle 신호에 따라 FSM 형태의 logic을 구현하여 출력을 제어한다.

-sa4x4

```
module sa4x4(
        input                           clk,
        input                           rstn,
        input                           en,
        input                           req_out,
        input   [31:0]          A,
        input   [31:0]          B,
        output  reg     [63:0]  C
);

        genvar i, j;

        reg             [7:0]   en_;
        always @ (posedge clk) begin
                en_ <= {en_[6:0], en};
        end

        wire    [7:0]   se_data_from_left   [0:3][0:3];
    wire    [15:0]  se_data_from_top    [0:3][0:3];
    wire    [7:0]   se_data_to_right    [0:3][0:3];
    wire    [15:0]  se_data_to_bottom   [0:3][0:3];
    wire            se_en               [0:3][0:3];
        for (i = 0; i < 4; i = i + 1) begin
        for (j = 0; j < 4; j = j + 1) begin
                assign se_en[i][j] = (i + j == 0) ? en : en_[i + j - 1];
            assign se_data_from_left[i][j] = (j == 0) ? A[31 - 8*i -: 8] : se_data_to_right[i][j-1];
            assign se_data_from_top[i][j] =  (i == 0) ? {{8{B[31 - 8*j]}}, B[31 - 8*j -: 8]} : se_data_to_bottom[i-1][j];
        end
    end
        generate
                for (i = 0; i < 4; i = i + 1) begin: se_row
                        for (j = 0; j < 4; j = j + 1) begin: se_col
                                se8             U_se8   (       .clk(clk), .rstn(rstn), .en(se_en[i][j]), .req_out(req_out), .data_from_left(se_data_from_left[i][j]), .data_from_top(se_data_from
_top[i][j]), .data_to_right(se_data_to_right[i][j]), .data_to_bottom(se_data_to_bottom[i][j])  );
                        end
                end
        endgenerate

        for (i = 0; i < 4; i = i + 1) begin
                always @ (*) begin
                        C[63 - 16*i -: 16] = se_data_to_bottom[3][i];
                end
        end

endmodule
```

<Fig 2.1.2 sa4x4>

sa4x4 모듈은 앞서 살펴본 se8 모듈을 instance화하여 se8 unit으로부터 나온 연산 결과를 연결해주는 controller의 역할을 한다. 16개의 PE를 4x4구조로 배치하며, 각 행의 첫 번째 PE는 data A를 받고, 각 열의 첫 번째 PE는 data B를 받는다. PE들은 서로 연결되어 데이터와 partial sum이 전달되며, 마지막 행의 PE들에서 최종 sum이 출력된다.



<Fig 2.1.3 waveform for sa module>

2. Synthesis(Design Compiler)

```
# Base Working Directory
set MODULE_BASE_DIR "/home/home25/shmoon/KU_lecture/Fall_2025/3421_AI_Digital_IC/assn3"

# Current Design Target Module
set TARGET_MODULE "sa4x4"

set RTL_SOURCE_FILES [list \
        "${MODULE_BASE_DIR}/rtl/sa.v" \
]

set SUBMODULES [list \
        "mult8" \
        "mac8" \
```

<Fig 2.2.1 module_setup.tcl >

모듈들의 설정을 관리하는 file에 접근하여 Target module을 sa로 설정해주자. 여기서 주목할 부분은 SUBMODULE이다. 과제 2와 비슷하게 sa4x4의 하위 모듈인 mult8, mac8 모듈을 RTL이 아닌, 이미 합성된 gate level netlist로 다룬다고 설정하였음을 확인할 수 있다.

```
## Read lower hierarchy design when doing hierarchical design
if {[llength $SUBMODULES] > 0} {
    foreach name $SUBMODULES {
        if {$name eq "mult8"} {
            read_ddc "../../assn1/dc/mult8_4.0GHz_LVT/results/mult8.mapped.ddc"
        } elseif {$name eq "mac8"} {
            read_ddc "../../assn2/dc/mac8_4.0GHz_LVT/results/mac8.mapped.ddc"
        }
    }
}

```

<Fig 2.2.2 synth.tcl >

이미 합성된 모듈(mult8, mac8)을 이용하여 target module을 합성을 해야하기에 Fig 2.2.2와 같이 직접 읽어올 경로를 지정해주어야한다.

```
################################################
# Save the design database
################################################
## Remove submodule design to write output
if {[llength $SUBMODULES] > 0} {
    foreach name $SUBMODULES {
        remove_design -hierarchy "${name}"
    }
}
```

<Fig 2.2.3 synth.tcl.>

sa4x4 모듈을 합성하면 submodule로 지정된 mult8, mac8의 정의까지 합성 결과에 포함되게 된다. 하지만, 우리가 사용할 mult8, mac8모듈은 이미 합성이 완료된 모듈이므로 DC 내부에서 mult8의 정의를 제거하고 다시 write하도록 설정해주도록 한다.

이후에는 과제1,2에서 진행한 것과 같이 timing.tcl에서 clk period를 조정하고, 이에 맞는 하위 모듈 mult8과 mac8 netlist를 설정하고 합성하면 sa4x4 모듈 netlist를 구현할 수 있다.

3. Primetime STA

```
31 # Read verilog files
32 set svr_enable_vpp true
33
34 read_ddc "../../dc/${TARGET_MODULE}_4.0GHz_LVT/results/${TARGET_MODULE}.mapped.ddc"
35
36 ## Read lower hierarchy design when doing hierarchical design
37 if {[llength $SUBMODULES] > 0} {
38     foreach name $SUBMODULES {
39         if {$name eq "mult8"} {
40             read_ddc "../../../assn1/dc/mult8_4.0GHz_LVT/results/mult8.mapped.ddc"
41         } elseif {$name eq "mac8"} {
42             read_ddc "../../../assn2/dc/mac8_4.0GHz_LVT/results/mac8.mapped.ddc"
43         }
44   }
```

<Fig 2.3.1 pt.tcl>

sa4x4 유닛의 합성결과로 STA를 진행하고자 한다. 앞서 합성한 mac 유닛과 동일한 주파수로 분석하기 위해 pt.tcl파일에서 사용할 mac유닛을 수정하여 분석한다.

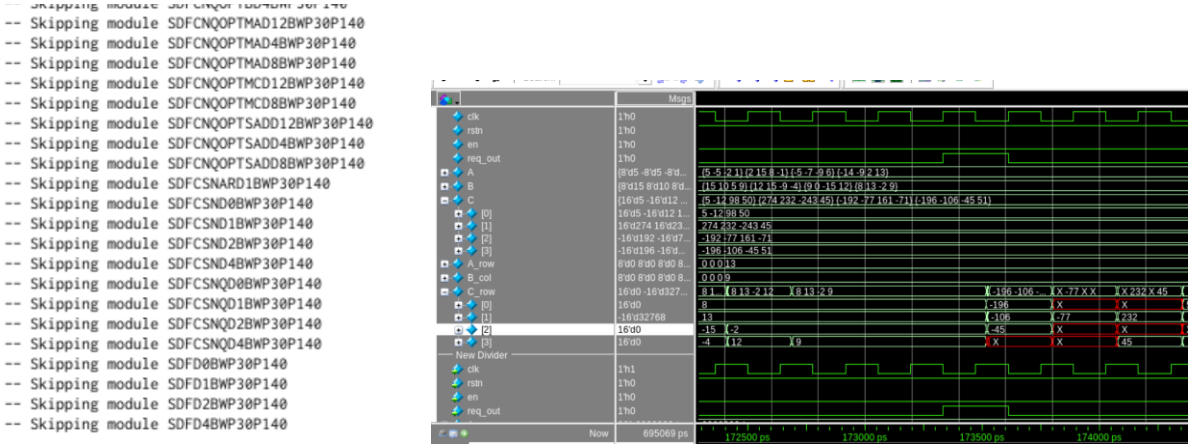
4. post-synthesis simulation in Questa

```
 7 vlib work
 8 vmap work work
 9
10 # Include std cell verilog
11 vlog +acc -incr /tech/TSMC_HPC28/Standard_Cell/core_cell_library/gate_length_30nm/9-track/tcbn28hpcplusbwp30p140_190a/TSMCHOME/digital/Front_End/verilog/tcbn28hpcplusbwp30p140_110a/tcbn28hpcplusbwp30p140.v
12
13 # Include Netlist - List all of your verilog files that will be used
14 vlog +acc -incr ../../../assn1/dc/mult8_4.0GHz_LVT/results/mult8.mapped.v
15 vlog +acc -incr ../../../assn2/dc/mac8_4.0GHz_LVT/results/mac8.mapped.v
16 vlog +acc -incr ../../dc/sa4x4_4.0GHz_LVT/results/sa4x4.mapped.v
17
18 # Include Testbench - List one testbench file that will be used
19 vlog +acc -incr ../tb/tb_sa.v
20
21 # Run Simulator - Write the testbench module name
22 vsim +acc -t ps -lib work -sdfmax /tb_sa/U_sa4x4=../../pt/fe/sa4x4_4.0GHz_LVT/results/sa4x4.sdf tb_sa
23
24 # Load Waveform Configuration - If exist, remove the comment to simply load it
25 do wave.do
26
27 # Run Simluation
28 run -all
29
```

<Fig 2.4.1 runsim.do>

Primetime에서 STA를 수행하고 나면 ~qsim/fe 디렉토리에 있는 runsim.do 파일을 이용하여 post-synthesis simulation을 진행할 수 있다. Fig 2.4.1을 보면 어떤 모듈들을 이용하여 simulation하는지 확인할 수 있는데, 합성된 모듈 netlist mult, mac, sa와 SDF, tb, standard cell 등을 포함한다. 또한, waveform 분석을 위해 여러 signal 추가 등의 waveform 설정을 변경하고 저장해둘 수 있는데, 해당하는 file을 불러와 유지할 수 있다.

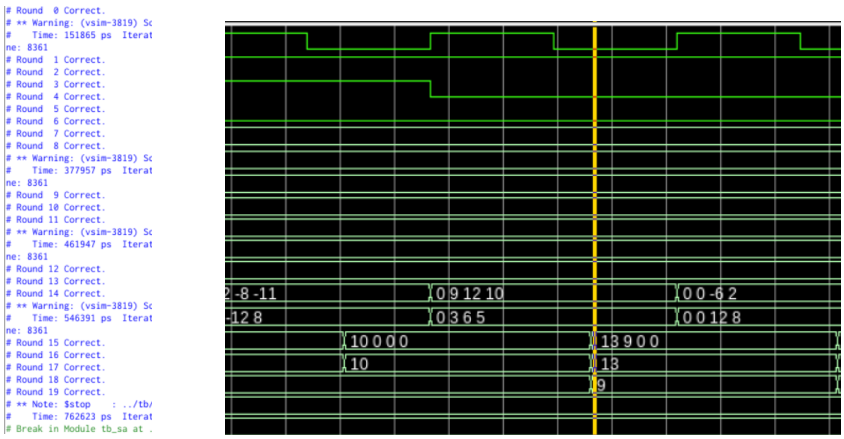
또한, 현재 runsim.do의 22번째 줄을 보면 sdfmax를 이용해 maximum delay를 고려하겠다고 명시해주고 있음을 볼 수 있다. 이를 이용해 delay 정보를 반영해줄 수 있다.



<Fig 2.4.2 post-synthesis simulation timing verification in 4.0GHz LVT>

vsim & 명령어를 이용해 simulation을 해보면, Fig 2.4.2와 같이 Skipping module... 이 console에 뜨는 것을 알 수 있는데 이는 include한 standard cell file들에서 사용하지않는 module들을 제거하는 과정이다.

Fig2.4.2의 waveform을 보면 timing이 맞지 않아 일부 값들에서 floating 현상이 발생한 것을 볼 수 있다. 이러한 방식으로, timing met 여부를 검증할 수 있다.

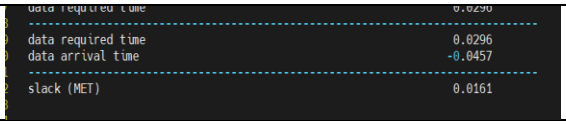


<Fig 2.4.3 post-synthesis simulation timing verification in 2.2GHz RVT>

Fig 2.4.3는 2.2GHz로 tb 동작 주파수를 설정하고 2.2GHz로 합성된 mult, mac, sa를 post-synthesis simulation을 진행한 결과이다. Clk과 data transfer가 behavior simulation 처럼 이상적이지 않지만, 모든 verification round에서 통과하였음을 확인할 수 있다. 이러한 방식으로 testbench, waveform을 이용하여 SDF를 이용해 post-synthesis simulation을 진행할 수 있다.

*** Addition**

Assn2에서 우리 조는 mac8을 합성 시, Submodule이었던 mult8을 합성할 때 사용했던 주파수와 동일 주파수로 mac8을 합성하였다. 하지만, 이번에는 더 나은 성능을 위해 최적화된 높은 주파수에서 합성된 mult8 netlist를 이용해 mac8 module을 재합성하였다. 그리고, 이와 동일하게 다양한 주파수에서 합성된 mult8, mac8 모듈을 조합하여 sa4x4 모듈을 합성하였다.



sa4x4_2.4GHz	sa4x4_2.4GHz_mac8_2.6GHz_mult8_2.4GHz

<Fig 2.4.4 STA Compare Variation of SUBMODULE>

예를 들어, Fig2.4.4는 동일한 주파수 2.4GHz로 mult8, mac8, sa4x4로 합성한 결과의 STA slack 결과와 다양한 주파수로 나누어 합성한 STA 결과이다. sa4x4는 동일하게 2.4GHz에서 합성되지만, SUBMODULE들의 합성 주파수를 바꾸어 timing violation을 해결한 모습이다.

주목할 만한 점은 mult8_2.4GHz를 이용하여 mac8 unit을 2.6GHz로 합성하였을 때는 timing violation이 발생하였으나, mac8_2.6GHz와 mult8_2.4GHz를 사용하여 sa4x4를 2.4GHz로 합성한 결과에서는 timing violation이 발생하지 않았다는 것이다.

Hierarchical Synthesis에서의 Timing Context 변화

이러한 현상은 hierarchical synthesis의 특성에 의해 나타난다. Submodule을 독립적으로 합성할 때와 top-level module에 통합되었을 때의 timing context가 다르기 때문이다.

mac8 모듈을 2.6GHz에서 합성할 경우, synthesis tool은 해당 모듈의 모든 path가 2.6GHz constraint를 만족하도록 최적화를 시도한다. 이 과정에서 내부 critical path가 target freqeuncy를 만족하지 못하면 timing violation이 발생한다.

하지만 sa4x4 모듈에 통합되었을 때, 실제 동작 주파수는 2.4GHz이므로 mac8의 internal path 역시 2.4GHz만 만족하면 충분하다. 또한, top-level synthesis 과정에서 전체 design의 critical path가 변하며, mac8 내부 path가 더 이상 전체 design의 critical path가 아닐 수 있다.

Submodule 합성 주파수와 실제 동작 주파수의 차이

앞서 언급한 내용을 비슷하지만 다른 시각으로 바라보면, submodule의 "합성 주파수"와 "실제 동작 주파수"를 구분하는 것이다. mac8을 2.6GHz로 합성했다는 것은 해당 주파수에서 동작 가능하도록 optimization을 시도했다는 의미이지, 반드시 그 주파수로 동작

시켜야 한다는 의미는 아니다.

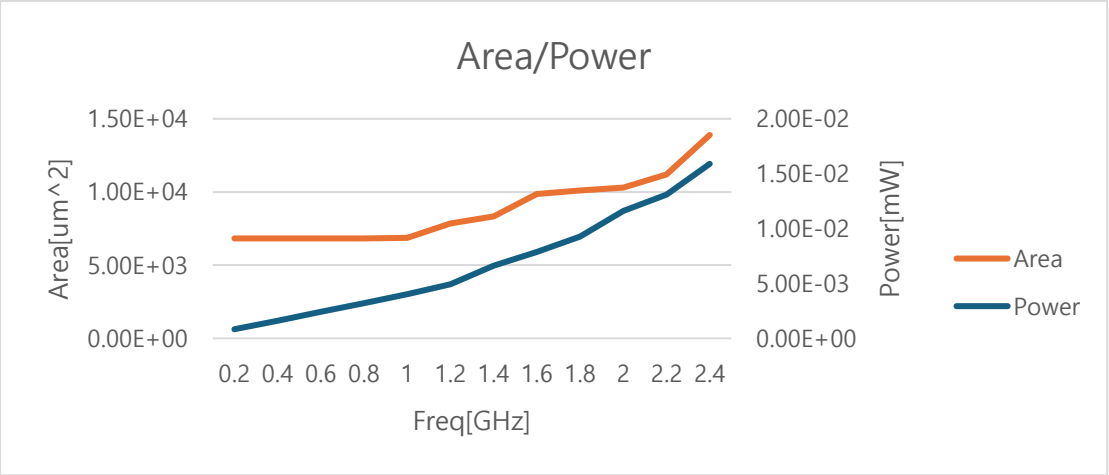
최종 설계의 타이밍 검증은 top-level인 sa4x4에서 2.4GHz 기준으로 수행되며, 이 STA가 pass하면 전체 design은 2.4GHz에서 안전하게 동작한다고 판단할 수 있다. 이러한 hierarchical synthesis 기법은 각 submodule에 대해 다양한 최적화 수준을 적용하여 전체 design의 성능과 timing을 조절할 수 있다.

5.1 Results

Syn Freq(GHz)	Max Sim Freq(GHz)	Slack	Timing Met	Power	Area
0.2	1	2.7081	O	8.34E-04	6.83E+03
0.4	1	0.8417	O	1.61E-03	6.83E+03
0.6	1	0.4659	O	2.41E-03	6.83E+03
0.8	1	0.1598	O	3.21E-03	6.83E+03
1	1	0.0024	O	4.02E-03	6.86E+03
1.2	1.2	0.0002	O	4.94E-03	7.85E+03
1.4	1.4	0.0001	O	6.62E-03	8.34E+03
1.6	1.6	0.0001	O	7.86E-03	9.86E+03
1.8	1.8	0.0005	O	9.28E-03	1.01E+04
2	2	0	O	1.16E-02	1.03E+04
2.2	2.2	0.0006	O	1.31E-02	1.12E+04
2.4	2.4	0.0001	O	1.63E-02	1.44E+04
2.6	2.4	-0.0235	X	1.96E-02	1.57E+04
2.8	2.4	-0.0363	X	2.12E-02	1.46E+04
3	2.4	-0.0552	X	2.15E-02	1.60E+04

<Fig 2.5.1 Frequency Variation Analysis in QuestaSim>

* STA에서 violation이 발생하였더라도, testbench의 clk frequency를 조절하여 functionality를 검증할 수 있었다.



<Fig 2.5.2 Valid Operating Frequency Graph(freq vs (area & power)

Area & Power	
Hierarchy / Int Power / Switch Power / Leak Power / Total Power / % / Area sa4x4 1.44e-02 1.38e-03 8.67e-05 1.59e-02 100.0 1.39e+04 se_row_3__se_col_0__U_se8 (se8_3) 9.02e-04 8.03e-05 5.43e-06 9.88e-04 6.2 874.441 U_mac8 (mac8) 6.93e-04 5.35e-05 5.10e-06 7.51e-04 4.7 798.588 U_mult (mult8) 5.74e-04 3.82e-05 4.63e-06 6.17e-04 3.9 706.356 genblk2_3__U_rca (rca_N16_3) 5.22e-05 1.73e-06 6.86e-07 5.46e-05 0.3 94.122 genblk1_10__U_FA (full_adder_50) 3.61e-07 1.77e-07 7.35e-08 6.11e-07 0.0 10.080	
Area : 1.39e4[um^2]	Power: 1.59e-2[mW]

<Fig 2.5.3 Max Frequency[2.4GHz] Area/Power Report>

5.2 Results Analysis

Fig 5.1.1를 보면 timing violation이 발생하지 않는 **maximum frequency**는 **2.4GHz**임을 확인할 수 있다.

```
13 # Include Netlist - List all of your verilog files that will be used
14 vlog +acc -incr ../../../assn1/dc/mult8_2.4GHz/results/mult8.mapped.v
15 vlog +acc -incr ../../../assn2/dc/mac8_2.6GHz_mult8_2.4GHz/results/mac8.mapped.v
16 vlog +acc -incr ../../dc/sa4x4_2.4GHz_mac8_2.6GHz_mult8_2.4GHz/results/sa4x4.mapped.v
17
18 # Include Testbench - List one testbench file that will be used
19 vlog +acc -incr ../tb/tb_sa.v
20
21 # Run Simulator - Write the testbench module name
22 vsim +acc -t ps -lib work -sdfmax /tb_sa/U_sa4x4=../../pt/fe/sa4x4_2.4GHz_mac8_2.6GHz_mult8_2.4GHz/results/sa4x4.sdf tb_sa
```

<Fig 2.5.4 runsim.do setting>

*Fig 2.5.4를 보면 알 수 있듯이, sa4x4 합성 **max freq 2.4GHz**는 SUBMODULE mult8_2.4GHz와 mac8_2.6GHz를 조합하여 얻어졌다.

Fig 2.5.2의 graph를 보면 valid한 frequency 영역에서의 area, power의 변화는 다음과 같다.

1. Power는 대체로 주파수에 비례하여 증가하는 모습을 보였다.

2. Area는 0.2GHz~1GHz까지는 증가하지 않다가 1GHz~2.4GHz까지 증가하였다.

Optimal frequency는 power, performance, area 중 어느 환경이냐에 따라 상이하게 optimal frequency가 존재하겠지만, 여기서는 ppa 모두의 성능 고르게 반영하는 frequency를 찾아보려고한다.

			Scores			
Syn Freq(GHz)	Area	Power	**F/(A+P)**	**F/A + F/P**	**F/(A*P)**	**F^2(A*P)**
0.2	1.00E+00	1.00E+00	1.00E-01	4.00E-01	2.00E-01	4.00E-02
0.4	1.00E+00	1.05E+00	1.95E-01	7.80E-01	3.80E-01	1.52E-01
0.6	1.00E+00	1.10E+00	2.85E-01	1.14E+00	5.43E-01	3.26E-01
0.8	1.00E+00	1.16E+00	3.71E-01	1.49E+00	6.91E-01	5.53E-01
1	1.00E+00	1.21E+00	4.51E-01	1.82E+00	8.22E-01	8.22E-01
1.2	1.14E+00	1.27E+00	4.97E-01	1.99E+00	8.24E-01	9.89E-01
1.4	1.21E+00	1.38E+00	5.39E-01	2.17E+00	**8.34E-01**	1.17E+00
1.6	1.43E+00	1.47E+00	5.53E-01	2.21E+00	7.64E-01	1.22E+00
1.8	1.46E+00	1.56E+00	5.95E-01	2.38E+00	7.89E-01	1.42E+00
2	1.49E+00	1.71E+00	6.24E-01	2.51E+00	7.82E-01	1.56E+00
2.2	1.62E+00	1.81E+00	**6.41E-01**	**2.57E+00**	7.49E-01	**1.65E+00**
2.4	2.00E+00	2.00E+00	6.00E-01	2.40E+00	6.00E-01	1.44E+00

* Max score는 빨간색으로 표기

<Fig 2.5.5 Optimal Frequency Assessment(Data are normalized)>

Power, Performance, Area 모든 특성의 변화를 동일하게 반영하기위해 valid한 영역(0.2GHz~2.4GHz)내에서 min-max normalization을 진행하고 계산을 위하여 +1의 offset을 하여 data-preprocessing을 진행했다. 정규화된 값들을 바탕으로 총 4번의 서로 다른 efficiency 계산을 진행하였다. 그리고, 이 4번의 평가 방식에서 3번의 우수 성적을 받은 **2.2GHz**와 1번의 우수 성적을 받은 **1.4GHz**를 Optimal Frequency로 결정할 수 있겠다. 만약 Performance를 우선으로 고려한다면, 2.2GHz를 선택하고 Area와 Power를 우선으로 생각하면 1.4GHz를 선택한다. 여기서는 3번의 우수 성적을 받은 **2.2GHz를 Optimal Frequency**로 선택하였다.

Min-max normalization을 선택한 이유는 power/performance/area가 서로 비교할 수 없는 단위들을 가지고 trade-off 관계를 형성하기에 동일한 scale로 가공할 필요가 있다고 생각했기 때문이다.

결론

1. Result Summary

Maximum frequency without violation : 2.4GHz (mac8_2.6GHz, mult8_2.4GHz)

Maximum frequency in Practical Simulation: 2.4GHz

Optimal Frequency : 2.2GHz

* STA Violation이 있어도 SDF를 이용한 simulation에서는 functionality를 검증하기위해 testbench의 clk frequency를 조절하여 확인한 결과, simulation에서도 2.4GHz에서 max로 동작하였다.

2. Additional discussion for Maximum Frequency

Maximum frequency를 높여 성능을 올림과 동시에 회로가 정상적으로 동작하려면 timing violation이 있어서는 안된다. 우리의 STA 분석에서는 2.6GHz부터 Slack이 음수로 나타나며 timing constraint를 위반한다. 따라서 2.6GHz 이상에서 정상 동작을 위해서는 critical path를 다음과 같은 방법들로 다시 설계할 필요가 있다.

1) RVT -> LVT synthesis

현재 설계는 RVT 셀 기반으로 합성되어 있어 속도 측면에서 한계가 존재한다. 이를 LVT(Low-Vth) 셀로 변경할 경우 Vth가 낮아져 cell delay가 감소하므로 critical path의 propagation delay를 효과적으로 줄일 수 있다. 다만, Ileakage 증가에 따른 power consumption이 동반된다.

LVT					
Syn Freq(GHz)	Max Sim Freq(GHz)	Slack	Timing Met	Power	Area
3	3	-0.0014	X	2.06E-02	1.28E+04
3.2	3.2	-0.0077	X	2.34E-02	1.42E+04
3.4	3.2	-0.0212	X	2.64E-02	1.53E+04
3.6	3.2	-0.0322	X	2.91E-02	1.47E+04
3.8	3.4	-0.0419	X	3.20E-02	1.46E+04
4	3.4	-0.0484	X	3.06E-02	1.47E+04

<Fig 3.1.1 LVT Analysis>

*실제로 LVT library를 include하여 합성하고 simulate 해본 결과, Fig 3.1.1과 같이 max sim freq를 향상 시킬 수 있었다.

2) Submodule의 합성 주파수 향상

Timing violation이 발생하는 경로가 특정 submodule 내부에 집중되어 있을 경우, 해당 모듈의 합성 주파수를 높여 synthesis tool이 더 강한 optimization을 적용하도록 유도할 수 있다. 이를 통해 tool이 gate sizing, logic restructuring 등을 통해 delay를 줄이는 데 효과적이다.

3) Critical path에 buffer 삽입

STA 결과를 토대로 critical path를 세분화하여 분석하면, 특정 구간에 buffer를 삽입하거나 기존 buffer를 재배치하여 load를 분산시키고 경로의 전체 delay를 줄일 수 있다.

4) Critical path 구간에 LVT 적용

전체 회로를 LVT 기반으로 합성할 경우 전력과 면적 증가가 문제로 이어질 수 있다. 그 대신, timing에 민감한 critical path 구간만 선택적으로 LVT 셀을 적용할 수 있다.

3. Additional discussion for Optimal Frequency

이번 과제 역시 optimal Frequency를 찾는데에는 min-max normalization을 이용하여 attribution의 변화를 고려했다. 실제 설계에서는 설계 목적에 맞게 각 attribution의 변화에 weight를 추가해서 조정해가며 Efficiency를 평가하는 것이 좋을 것이다.

Assessments

한민준 – STA Synthesis을 진행해주었음.

주형훈 – STA 결과 및 post-synthesis simulation 결과를 정리하여 체계적으로 공유해주었음.