AI디지털집적회로

Assignment #2

전기전자공학부

202110410

조민우

서론

이번 과제에서는 앞선 과제에서 합성한 RCA 기반 8-bit multiplier를 하위 모듈(submodule)로 포함하는 MAC을 대상으로 hierarchical synthesis를 진행하고 Static Timing Analysis(STA)를 수행하고자 한다.

과제1과 마찬가지로 다양한 clk 주파수에서의 timing violation을 검증하여 실제 동작 가능한 주파수를 찾을 예정이다. 또한, power, power, area간의 trade-off를 확인해볼 예정이다.

본격적으로 분석에 들어가기 앞서, Design Compiler의 report와 PrimeTime의 report, 두 가지 분석 결과의 차이점에 대해 알아보고, 왜 두 가지 분석이 모두 필요한지 고찰해보고자 한다.

구분	Design Compiler	PrimeTime
목적	합성(Synthesis)	Static Timing Analysis(STA)
Delay 연산	Wire load 기반	실제 RC 기반
Clock 처리	Ideal Clock 가정	실제 skew까지 고려
Corner 고려	Typical	Slow/fast/OCV 고려

Fig 1.1 Comparison between DC & PT

Fig 1.1은 각 툴(Tool)에 따른 분석 방법의 목적 및 방식을 나타낸다. 가장 큰 차이점은 Design Compiler(DC)는 합성(Synthesis) 과정 중에 타이밍을 추정하기 때문에 부정확한 반면, PrimeTime(PT)은 합성이 완료된 넷리스트(netlist)를 기반으로 정밀한 정적 타이밍 분석(Static Timing Analysis, STA)을 수행한다는 점이다.

구체적으로 DC는 Wire Load Model을 사용하여 지연(delay)을 추정하고, Ideal Clock을 가정하며, Typical corner만을 고려한다. 반면 PT는 실제 RC 기반의 정확한 지연을 계산하고, clock skew와 On-Chip Variation(OCV)을 포함한 slow/fast/OCV corner를 모두 고려한다.

따라서 DC에서 timing이 met 됐더라도 PT 분석에서는 violation이 발생할 수 있다. 이는 PT가 실제 칩 동작 환경에 더 가까운 worst-case(Corner 고려) 조건을 반영하기 때문이다. 그러므로 최종적인 타이밍 검증은 PT 결과를 기준으로 수행하는 것이 좋다.

본론

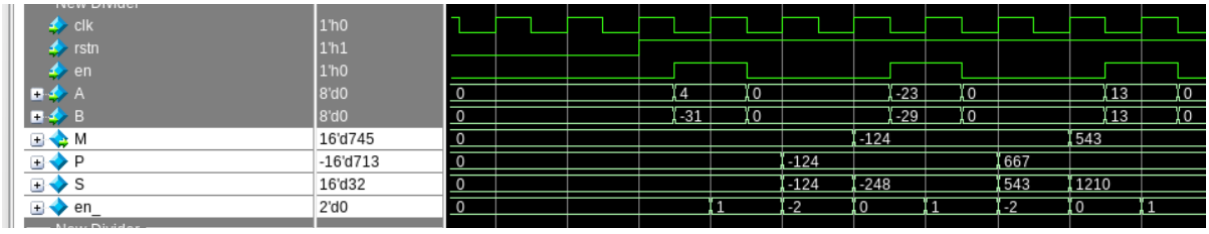
1. Behavior simulation

```
10
11 `timescale 1ns/1ps
12
13 module mac8(
14         input                           clk,
15         input                           rstn,
16         input                           en,
17         input   [7:0]           A,
18         input   [7:0]           B,
19         output  reg     [15:0]  M
20 );
21
22         wire    [15:0]  P;
23         mult8   U_mult  ( .clk(clk), .rstn(rstn), .en(en), .A(A), .B(B), .P(P) );
24
25         wire    [15:0]  S;
26         rca     #(16)   U_rca   ( .A(P), .B(M), .S(S), .Cout( ) );
27
28         reg             [1:0]   en_;
29         always @ (posedge clk) begin
30                 en_ <= {en_[0], en};
31         end
32         always @ (posedge clk) begin
33                 if (~rstn) begin
34                         M <= 0;
35                 end else if (en_[1]) begin
36                         M <= S;
37                 end else begin
38                         M <= M;
39                 end
40         end
41 endmodule
mac.v" 41L, 983C
```

<Fig 2.1.1 mac.v>

과제에서 target하고 있는 모듈인 mac 유닛이다. Multiply 연산을 위해 과제 1에서 합성한 mult모듈을 instance화 하고, add연산을 위해 adder 모듈을 instance화 하였다.

en_[1] 신호가 1인 경우 adder의 out port로 나온 값인 S(Sum)를 M reg에 할당한다. en_[1] 신호가 1이 아닌 경우 mult의 out port로 나온 값인 M을 M reg에 할당한다. 이러한 구조를 통해, MAC 유닛은 하위 모듈들의 연산 결과 dataflow를 결정하는 역할을 하게 된다.



<Fig 2.1.2 waveform for mac.v>

2. Synthesis(Design Compiler)

```
4 # Current Design Target Module
5 set TARGET_MODULE "mac8"
6
7 set RTL_SOURCE_FILES [list \
8         "${MODULE_BASE_DIR}/rtl/add.v" \
9         "${MODULE_BASE_DIR}/rtl/mac.v" \
10 ]
11
12 set SUBMODULES [list \
13         "mult8" \
14 ]
```

<Fig 2.2.1 module_setup.tcl >

모듈들의 설정을 관리하는 file에 접근하여 확인해보자. 여기서 주목할 부분은 SUBMODULE이다. 과제 1에서와 달리 mac8의 하위 모듈인 mult8 모듈을 RTL이 아닌, 이미 합성된 gate level netlist로 다룬다고 설정하였다.

```
41
42 ## Read lower hierarchy design when doing hierarchical design
43 if {[llength $SUBMODULES] > 0} {
44     foreach name $SUBMODULES {
45                 if {$name ne ""} {
46                         read_ddc "../../assn1/dc/${name}_3.0GHz/results/${name}.mapped.ddc"
47         }
48         }
49 }
50
```

<Fig 2.2.2 synth.tcl >

이미 합성된 모듈을 이용하여 target module을 합성을 해야하기에 Fig 2.2.2와 같이 직접 읽어올 경로를 지정해주어야한다.

```
190 ################################################
191 # Save the design database
192 ################################################
193 ## Remove submodule design to write output
194 if {[llength $SUBMODULES] > 0} {
195     foreach name $SUBMODULES {
196         remove_design -hierarchy "${name}"
197     }
198 }
```

<Fig 2.2.3 synth.tcl.>

Mac 모듈을 합성하면 submodule로 지정된 mult8의 정의까지 합성 결과에 포함되게 된다. 하지만, 우리가 사용할 mult8모듈은 이미 합성이 완료된 모듈이므로 DC 내부에서 mult8의 정의를 제거하고 다시 write한다.

이후에는 과제1에서 진행한 것과 같이 timing.tcl에서 clk period를 조정하고, 이에 맞는 mult를 설정하면 mult8를 submodule로 이용한 mac 모듈 netlist를 구현할 수 있다.

3. PrimeTime

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

```
36 ## Read lower hierarchy design when doing hierarchical design
37 if {[llength $SUBMODULES] > 0} {
38     foreach name $SUBMODULES {
39                 if {$name ne ""} {
40                         read_ddc "../../../assn1/dc/${name}_3.0GHz/results/${name}.mapped.ddc"
41         }
42         }
43 }
```

<Fig 2.3.1 pt.tcl>

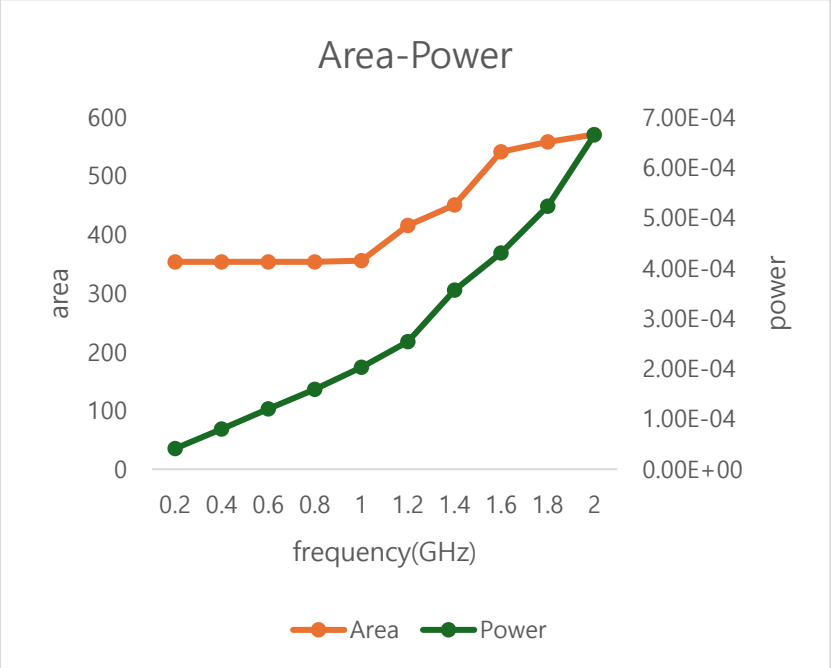
3.1 Results

Frequency [GHz]	Slack[ns]	Timing Met	Area[um^2]	Power[mW]
0.2	2.9124	O	353.682	4.11E-05
0.4	1.0403	O	353.682	8.01E-05
0.6	0.5535	O	353.682	1.20E-04
0.8	0.1598	O	353.682	1.59E-04
1	0.0547	O	355.446	2.03E-04
1.2	0.0002	O	415.926	2.54E-04
1.4	0?	O	450.576	3.56E-04
1.6	0.0001	O	541.296	4.30E-04
1.8	0.0005	O	558.054	5.23E-04
2	0	O	570.654	6.65E-04
2.2	-0.0021	X	627.102	7.66E-04
2.4	-0.0028	X	823.662	1.02E-03
2.6	-0.0228	X	910.981	1.23E-03
2.8	-0.0358	X	106.848	2.91E-04
3	-0.0546	X	930.763	1.36E-03

<Fig 3.1.1 Frequency Variation Analysis in PrimeTime>

frequency[GHz]	Slack[ns]	Timing Met	Area[um^2]	Power[mW]
0.2	2.91	O	354.69	3.85E-02
0.4	1.04	O	354.69	7.50E-02
0.6	0.66	O	354.69	0.112
0.8	0.28	O	354.69	0.149
1	0.07	O	354.69	0.189
1.2	0	O	415.17	0.292
1.4	0	O	449.82	0.393
1.6	0	O	492.03	0.542
1.8	0	O	492.66	0.595
2	0	O	536.13	0.651
2.2	0	O	575.064	0.754
2.4	0	O	581.364	0.935
2.6	0	O	580.356	1.036
2.8	0	O	670.068	1.181
3	-0.06	X	931.518	1.356

<Fig 3.1.2 Frequency Variation Analysis in Synthesis(design compiler)>



<Fig 3.1.3 Valid Operating Frequency Graph(freq vs (area & power)

Fig 3.1.1과 Fig 3.1.2를 비교해보면 서론에서 언급한바와 같이 dc분석에서 timing이 met 됐더라도 PT 분석에서는 2.2GHz부터 violation이 발생하였음을 알 수 있다.

3.2 Results Analysis

Fig 3.1.1를 보면 timing violation이 발생하지 않는 **maximum frequency**는 **2.0GHz**임을 확인할 수 있다.

Fig 3.1.3의 valid한 frequency 영역에서의 area, power의 변화는 다음과 같다.

1. Power는 대체로 주파수에 비례하여 증가하는 모습을 보였다.

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

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

			Scores			
frequency(GHz)	Area	Power	**F/(A+P)**	**F/A + F/P**	**F/(A*P)**	**F^2/(A*P)**
0.2	1	1	0.1	0.4	0.2	0.04
0.4	1	1.0625	0.193938	0.776467	0.376467	0.1505868
0.6	1	1.1265	0.282159	1.1326409	0.532641	0.3195845
0.8	1	1.1890	0.365468	1.4728498	0.67285	0.5382799
1	1.00813	1.2595	0.44099	1.7859034	**0.787565**	0.787565
1.2	1.28688	1.3412	0.456601	1.827185	0.695245	0.8342941
1.4	1.44657	1.5047	0.474367	1.8982046	0.643175	0.9004454
1.6	1.86469	1.6233	0.458712	1.8436745	0.528572	0.8457151
1.8	1.94193	1.7724	0.48461	1.9424863	0.522971	0.9413482
2	2	2	**0.5**	**2**	0.5	**1**

* Max score는 빨간색으로 표기

<Fig 3.2.1 Optimal Frequency Assessment>

Power, Performance, Area 모든 특성의 변화를 동일하게 반영하기위해 valid한 영역(0.2GHz~2.0GHz)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 **2GHz**와 1번의 우수 성적을 받은 **1GHz**를 Optimal Frequency로 결정할 수 있겠다. 만약

Performance를 우선으로 생각한다면, 2GHz를 선택하고 Area와 Power를 우선으로 생각하면 1GHz를 선택한다. 여기서는 3번의 우수 성적을 받은 **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.

Area & Power

```
****************************************
Report : Averaged Power
         -hierarchy
         -nosplit
         -area
Design : mac8
Version: W-2024.09-SP4
Date   : Tue Oct 14 18:00:07 2025
****************************************



                                    Int      Switch   Leak     Total
Hierarchy                           Power    Power    Power    Power    %          Area
--------------------------------------------------------------------------------------
mac8                                5.79e-04 8.29e-05 3.31e-06 6.65e-04 100.0     570.654
  clk_gate_M_reg (SNPS_CLOCK_GATE_HIGH_mac8) 1.02e-05 5.71e-06 8.80e-09 1.59e-05   2.4    1.890
  clk_gate_M_reg_0 (SNPS_CLOCK_GATE_HIGH_mac8_0) 1.02e-05 5.09e-06 8.80e-09 1.53e-05   2.3    1.890
  U_rca (rca_N16)                   2.10e-05 4.50e-06 2.65e-07 2.58e-05   3.9      48.384
    genblk1_10__U_FA (full_adder_110) 1.07e-06 3.81e-07 1.90e-08 1.47e-06   0.2       3.402
    genblk1_8__U_FA (full_adder_112)  1.07e-06 2.99e-07 1.46e-08 1.39e-06   0.2       2.772
    genblk1_4__U_FA (full_adder_116)  1.04e-06 2.72e-07 1.45e-08 1.33e-06   0.2       2.772
    genblk1_7__U_FA (full_adder_113)  1.13e-06 3.92e-07 1.93e-08 1.54e-06   0.2       3.402
    genblk1_3__U_FA (full_adder_117)  1.07e-06 3.51e-07 1.94e-08 1.45e-06   0.2       3.402
    genblk1_6__U_FA (full_adder_114)  1.08e-06 2.89e-07 1.46e-08 1.38e-06   0.2       2.772
```

Area : 570.654[um^2]	Power: 0.665[mW]

<Fig 3.2.3 Optimal Frequency[2GHz] Area/Power Report>

결론

1. Result Summary

Maximum frequency without violation : 2.0GHz

Optimal Frequency : 2.0GHz

2. Additional discussion for Maximum Frequency

Maximum frequency를 높여 성능을 올림과 동시에 회로가 정상적으로 동작하려면 timing violation이 있어서는 안된다. 우리의 설계를 Primetime으로 분석해보면 2.0GHz부터 Slack이 음수로 나타나며 timing constraint를 위반한다. 따라서 2.0GHz 이상에서 정상 동작을 위해서는 critical path를 다음과 같은 방법들로 다시 설계할 필요가 있다.

1) Cell Upsizing

Critical path 상의 standard cell들을 더 큰 cell로 교체한다. 라이브러리에 있는 더욱 큰 크기의 cell로 교체를 하게 되면, 더욱 큰 transistor가 output load를 빠르게 구동하여 propagation delay를 줄일 수 있다.

2) Buffer Insertion

Critical path가 긴 경우, 중간에 buffer를 삽입하여 delay를 분산시킬 수 있다.

3) Logic Restructuring

동일한 논리 기능을 유지하면서 logic depth를 줄이는 방법이다. 순차적으로 연결된 논리 구조를 병렬로 재배치하여 critical path를 단축할 수 있다

3. Additional discussion for Optimal Frequency

이번 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 – dc synthesis rpt 결과를 표로 정리하여 공유하였음

주형훈 – pt rpt 결과를 표로 정리해 체계적으로 공유해주었음.