ETRI 0.5um CMOS Std-Cell DK:

회로도와 레이아웃에서 추출한 회로의 비교(LVS)

연구과제명	반도체 기술 개발 지원 고경력 전문인력 활용 사업(25JB1710)
연구기간	2025 년 6 월~2026 년 12 월
연구책임자	고상춘
기록자	국일호
확인자	
작성일자	2025 년 7 월 18 일

ETRI 한국전자통신연구원
www.etri.re.kr

회로도와 레이아웃에서 추출한 회로 비교(LVS)

목차

[부록] Netgen 실습: 네트리스트 비교(LVS) 및 시뮬레이션

목차

---

1. 학습 개요

회로도(circuit diagram)는 인간이 읽고 쓰기 수월하도록 전자회로 요소들을 직관성이 좋은 그래픽 심볼들로 표현하고 상호 연결한 도면이다. 반도체로 제작하려면 레이아웃 도면이 필요하다. 회로도에서 레이아웃을 완벽하게 그려주는 자동화 도구는 없다. 도면을 손수 그려야 하는데 인간이 개입되면서 오류가 끼어들 여지가 매우 크다. 이런 위험을 방지하려고 다양한 방법들을 동원한다. 그중 회로도(Schematics)에서 추출한 네트리스트와 레이아웃(Layout)에서 추출한 네트리스트를 비교하는 LVS(Layout-versus-Schematic)는 반도체 설계 과정에서 필수 확인 절차다.

Netgen 은 LVS를 수행 하는 오픈-소스 도구다. 앞서 학습한 XSchem의 회로도[연구노트5]나 Magic의 레이아웃[연구노트6]은 저마다 고유의 파일 형식을 가지고 있다. 두 도구 모두 LVS를 위해 공통의 SPICE 네트리스트를 생성할 수 있다. 추상화 수준이 다른 두 도구에서 추출되는 네트리스트의 내용이 달라질 수 있다는 점에 유의하면서 회로도와 레이아웃의 네트리스트 비교 도구 netgen의 사용법을 알아본다.

2. 추상화 수준과 설계 생산성

회로도는 인간이 보기에 읽고 쓰기 수월하다. 이에 비하여 레이아웃에서 회로를 파악하기 위해서 좀 더 많은 추리를 필요로 한다. 회로도는 회로 요소들을 직관적인 상징(심볼)으로 표현하고 구체적 특성 인자들을 변경하기 수월하며 그 사이의 배선이 자유로운 반면 레이아웃은 제한된 평면 위에서 일일이 부품을 그려 주어야 했다. 회로도는 인간에게 가깝고 레이아웃은 제조 공장에 가깝다. 회로도는 레이아웃보다 추상성이 높다. 레이아웃은 회로도보다 구체적으로 실물에 가깝다.

회로도로부터 제조용 생산도면을 작성하려면 레이아웃 그리기의 과정을 거쳐야 한다. 처음부터 레이아웃을 그리지 않고 번거롭게 회로도 작성에서 시작하는 이유를 찾아보자. 가장 큰 이유는 생산성이다. 설계는 고도의 추상적인 지적 활동으로 이를 구체화 하려면 회로 그리기와 검증에 이은 수정의 무한 반복이다. 레이아웃에서 설계를 한다면 처음 도면 그리기도 수월치 않을 뿐더러 시뮬레이션 후 수정은 더 많은 시간을 소요한다. 게다가 설계자의 피로도 상승으로 인한 오류의 가능성이 높아지고 수정 회피의 유혹에 빠지기 쉽다. 이에 비하여 회로도로 설계를 한다면 그나마 매우 수월하다. 부품을 배치하고 배선해야 할 평면의 제약이 거의 없다. 충분히 검증한 회로도를 바탕으로 레이아웃을 단번에 그린다면 그 생산성은 이루 말할 수 없이 높아질 것은 자명하다.

회로도와 레이아웃의 회로 표현 방식은 다르다. 따라서 표현 방식이 다른 두 회로가 물리적으로 일치하는지 확인하는 절차가 필요하다. 다행히 회로도와 레이아웃 모두 네트리스트를 추출하는 기능을 가지고 있다. SPICE 라는 네트리스트는 시뮬레이션 용도 뿐만 아니라 두 회로를 비교하는 용도로도 쓰인다.

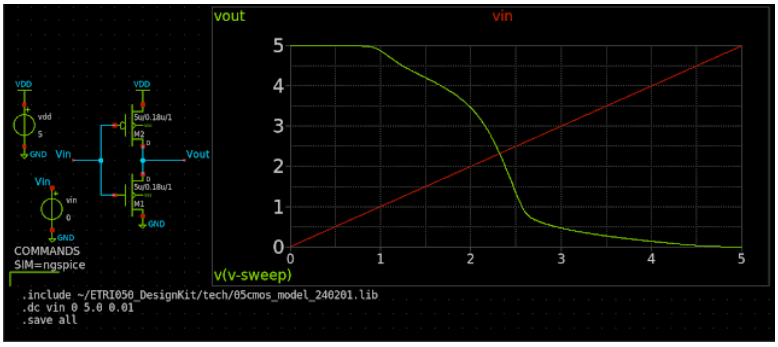
3. 회로도와 레이아웃의 네트리스트 비교

검증은 추상화의 수준이 달라지면 반드시 거쳐야 하는 절차로서 두 설계물이 일치하는지 확인한다. 회로도의 네트리스트에는 배선으로 인하여 발생하는 물리적 인자들이 없는 반면 실물에 가까운 레이아웃은 사정이 다르다. 부품의

배선으로 인한 저항(resistivity)과 인덕턴스(inductance), 배선 사이에서 기인하는 기생 커페시턴스(parasite capacitance)가 끼어든다.

추상화 수준이 서로 다른 두 설계물을 검증할 내용은 상위 수준에서 묘사된 회로가 하위 수준에 동일하게 반영되었는지 확인한다. 일치의 여부는 회로의 연결도와 사용된 부품의 특성(트랜지스터 채널의 길이와 폭)을 포함한다. 배선으로 인한 기생 성분은 레이아웃을 완성 한 후에 비로서 알려지는 것으로 회로도에 표현 하는 설계인자(design parameter)가 되지 못한다. 회로도와 레이아웃은 모두 반도체 제조공장에서 제공한 PDK의 반도체 물질의 물리적 특성을 공유하지만 트랜지스터를 형성하는 채널의 길이와 폭은 설계인자(design parameter)다. 회로도에서 트랜지스터 채널의 폭과 길이는 임의 숫자로 지정할 수 있으나 레이아웃은 그리기 규칙을 따라야 한다.

NSPL 0.5um CMOS 노드(내칩 제작 서비스의 공정)의 레이아웃의 그리기 규칙에 따르면 트랜지스터 채널의 최소 길이는 0.5um, 폭은 1.6um 다. 회로도를 작성할 때 이 규칙이 적용되어야 한다. 레이아웃에서 그리기 규칙의 위반(DRC violation)을 즉시 알려 주지만 한 단계 높은 추상화 수준의 회로도에서는 자유롭다. 지난번 학습에서 이를 무시하고 채널 길이를 0.18um 로 잡은 트랜지스터의 시뮬레이션[연구노트5]에서 기괴한(?) 파형이 나왔었다. 설령 회로도의 시뮬레이션에서 원하는 파형이 나왔다 하더라도 제조 불가능하면 소용없는 일이다.



서로 다른 추상화 수준의 설계물 표현법 사이에 일치를 보장하기 위한 절차가 LVS다. Layout-versus-Schematic의 약자다. LVS는 상위 수준의 회로도에서 기술된 내용이 레이아웃에 반영되었는지 확인하는 것이지 실물 회로의 동작을 보장하지 않는다. 배선으로 인한 지연 같은 물리적 특성은 LVS 후에 레이아웃으로부터 회로를 추출하여 시뮬레이션을 실시해야 하지만 물리 시뮬레이션은 워낙 긴 시간을 든다. 소자의 갯수가 적고 입력에 따른 출력의 변화 모습(shape)을 따지는 아날로그 회로에서는 기생 RLC를 포함한 회로 시뮬레이션(SPICE)을 실시한다. 디지털 회로는 0과 1로 상태가 변하는 시간 간격에 관심을 두고 시뮬레이션 대신 배선 지연을 분석(analysis)한다. 소자의 지연과 배선의 길이를 따져 총 지연을 산출한다. 이런 방식을 정적 시간 분석 STA(Static Timing Analysis)라고 한다. 회로의 규모(트랜지스터의 갯수)가 방대해지면 시뮬레이션에 엄청난 시간이 소요된다. 추상화 수준이 트랜지스터 단계에 이르면 시뮬레이션 검증보다 네트리스트 비교 확인을 실시한다.

4. Netgen, 오픈-소스 LVS 도구

Netgen은 SPICE 네트리스트를 비교해주는 오픈소스 도구다. 회로도에서 추출한 네트리스트와 레이아웃에서 추출한 네트리스트를 비교하여 등가성을 확인해 준다. Netgen을 사용하여 두 네트리스트를 비교해보자. 먼저 XSchem에서 추출한 네트리스트는 아래와 같다. 인버터 회로에 사용된 두 트랜지스터를 제외한 시뮬레이션 명령들은 주석처리 했다.

```
** sch_path: /home/goodkook/ETRI050_DesignKit/Tutorials/1-
1_Inverter_XSchem/inverter1.sch
**.subckt inverter1
M1 out in GND GND nfet w=5u l=0.18u m=1
M2 out in VDD VDD pfet w=5u l=0.18u m=1
*vdd VDD GND 5
*vin Vin GND 0
**** begin user architecture code
*.include ~/ETRI050_DesignKit/tech/05cmos_model_240201.lib
*.dc vin 0 5.0 0.01
*.save all
***** end user architecture code
***.ends
*.GLOBAL VDD
*.GLOBAL GND
*.end
```

Magic에서 추출한 네트리스트는 아래와 같다. 기생 커페시턴스 부분은 주석처리 되었다.

```
* NGSPICE file created from inverter.ext - technology: scmos
.option scale=0.15u
*.subckt inverter in out vdd gnd
M1000 out in vdd vdd pfet w=12 l=4
    +  ad=0.192n pd=56u as=0.18n ps=54u
M1001 out in gnd gnd nfet w=12 l=4
    +  ad=0.168n pd=52u as=0.168n ps=52u
*C0 out gnd 3.0088f
*C1 in gnd 2.36385f
*C2 vdd gnd 5.93253f
*.ends
```

두 네트리스트의 비교를 위해 Netgen을 실행한다.

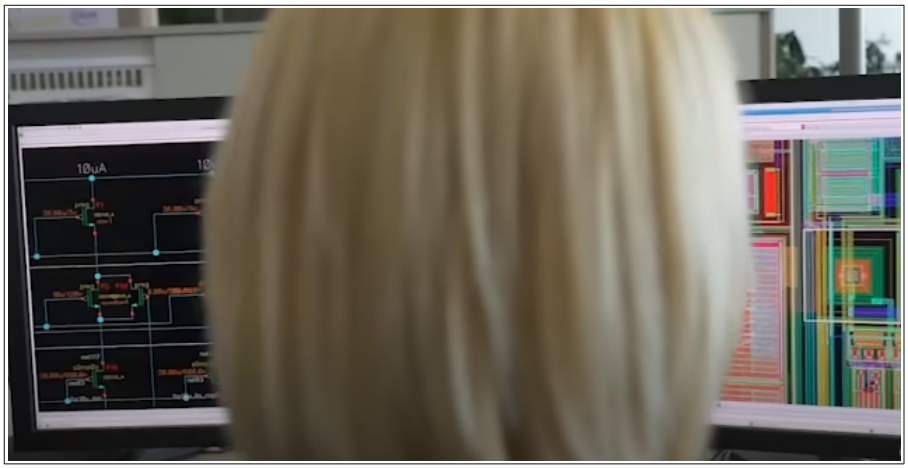
```
$ netgen -batch lvs ../1-2_Inverter_Magic/inverter.spice \
            ../1-1_Inverter_XSchem/simulation/inverter1.spice  \
            ./etri050_setup.tcl comp.out
```

LVS의 결과,

```
Subcircuit summary:
Circuit 1: ../1-2_Inverter_Magic/inverter. |Circuit 2: ../1-
1_Inverter_XSchem/simulation
```

```
-----------------------------------------|---------------------
pfet (1)                                 |pfet (1)
nfet (1)                                 |nfet (1)
Number of devices: 2                     |Number of devices: 2
Number of nets: 4                        |Number of nets: 4
-------------------------------------------------------------------
Netlists match uniquely with property errors.
nfet:1001 vs. nfet:1:
      L circuit1: 4   circuit2: 1.8e-07   (delta=200%, cutoff=1%)
      W circuit1: 12   circuit2: 5e-06   (delta=200%, cutoff=1%)
pfet:1000 vs. pfet:2:
      L circuit1: 4   circuit2: 1.8e-07   (delta=200%, cutoff=1%)
      W circuit1: 12   circuit2: 5e-06   (delta=200%, cutoff=1%)
Cells have no pins;  pin matching not needed.
Final result: Circuits match uniquely.
Property errors were found.
```

회로는 일치하지만 두 트랜지스터의 속성(property)이 다르다는 보고다. 회로도를 작성할 때 트랜지스터의 채널 길이와 폭이 레이아웃 그리기 규칙을 위배하고 있으므로 이를 수정해 주어야 한다.



단 두개의 트랜지스터가 사용된 회로에서도 불일치의 가능성이 있다. 인간의 실수가 끼어들 여지가 많기 때문이다. 설계 규모가 커지고 레이아웃 평면도가 기발한 아날로그 회로의 경우 자동화된 확인 절차는 매우 중요하다. 가끔 두 화면을 심각하게 바라보는 장면을 본다. 뭔가 중요한 일을 하는 양 하지만 연출된 장면이 확실하다. 눈으로 봐서 확인할 수 있는 일이 아니다.

5. 결론

설계와 검증은 사양을 만족하기까지 무한 반복이다. 생산성을 위해 상위 추상화 수준에서 이뤄진 설계는 제조를 위해 하위 수준으로 전환 된다. 이 과정에서 표현된 회로의 등가성이 보장되어야 한다. 각 수준에서 회로 시뮬레이션의 수행이 가장 확실한 방법이지만 회로 규모의 증가에 따른 시간적 부담이 너무 크다.

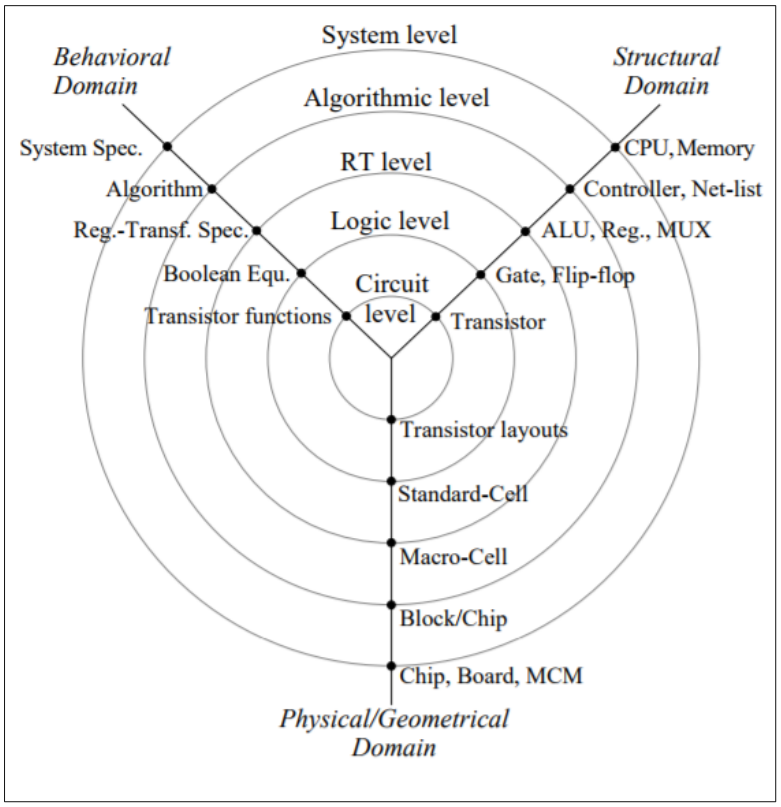
설계과정에서 추상화 수준의 전환에 따른 상이한 도구들이 동원된다. 저마다 고유의 파일 형식을 사용하며 검증을 위해 공용 형식으로 변환이 이뤄진다. 추상화 수준이 상이한 만큼 생산된 네트리스트에 검증의 요소가 아닌 부분이 포함되는 경우가 있다. 그때마다 손수 편집을 한다면 매우 위험하다. 설계 자동화는 인간이 실수가 끼어들 여지를 최대한 줄이기 위한 방안이다. 설계 자동화 도구는 용도에 맞는 네트리스트를 생성하는 장치를 갖추고 있다. 오픈-소스 도구 XSchem와 Magic 도 예외는 아니다.

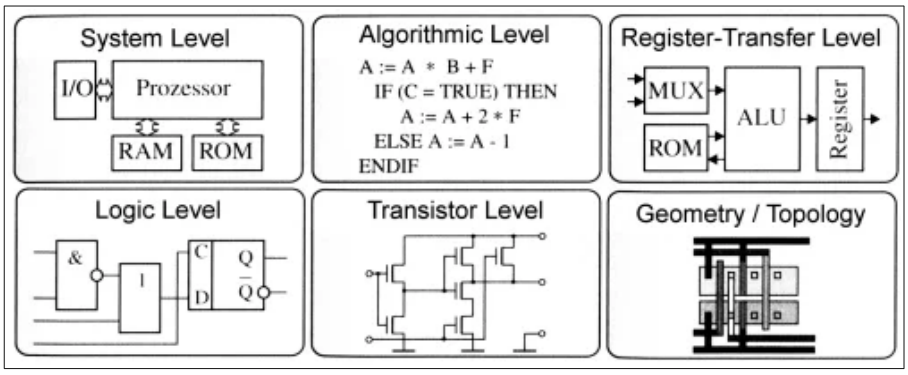
[부록] Netgen 실습: 네트리스트 비교(LVS) 및 시뮬레이션

목차

I. 개요

이번 실습편은 상이한 추상화 수준 또는 방법론이 다른 두 설계물을 등가성을 검증한다. 동일한 기능을 상이한 방법으로 설계하는 가장 큰 이유는 생산성 때문이다. 설계는 목표한 사양에 이르기까지 입력과 검증 그리고 수정의 반복이다. 검증을 위한 시뮬레이션은 자동화 되어 있지만 입력과 수정은 인간의 지적 능력에 달렸다. 인간이 다루기 수월한 형식을 취하기 위해 상위 추상화 수준에서 설계하고 검증을 마친 후 하위 수준으로 이전함으로서 생산성을 높인다.





Y 차트는 설계 방법론과 추상화 수준을 잘 설명하고 있다. 그림에서 설계 방법을 3가지 영역(domain)으로 나누고 있으며 추상화 수준을 낮춰(refinement) 최종 목표인 설계 도면을 얻는 과정을 표현한다. Y 차트의 중심으로 향하는

과정에서 자동화 도구(EDA Tools)의 조력을 받는다. 영역 사이를 오가면서 사용하는 자동화 도구는 설계 생산성 향상에 크게 기여 한다. 트랜지스터 회로도(schematic)와 레이아웃(layout) 설계를 Y-차트의 추상화 수준에서 보면 비슷한 수준 "Circuit Level" 이나 방법론은 매우 다르다. 비슷한 추상화 수준이라도 방법론(설계의 표현방법)을 달리하는 이유는 역시 생산성 향상을 얻기 위함이다. 추상화 수준은 물론 설계 방법의 전환이 있게 되면 반드시 검증의 절차를 따라야 한다. 회로를 표현하는 방법이 달라도 기능적으로 동일성이 확인 되어야 한다.

LVS(Layout versus Schematic)은 트랜지스터 수준에서 두 회로의 등가성을 확인한다. 다른 방식으로 표현된 회로를 각각 공통의 SPICE 네트리스트로 추출 하여 비교한다. 비교될 항목은 보다 높은 추상화 수준의 회로를 따른다. 회로도 보다 실제에 가까운 레이아웃에서 추출된 기생 C 값은 비교 항목에서 제외 된다. 기능적으로 동일한 두 회로의 물리적(시간적) 차이와 특성 파악은 시뮬레이션을 통해 이뤄진다.

II. LVS, 네트리스트 비교

오픈-소스 도구 Netgen 은 SPICE 네트리스트 비교한다. 회로의 연결 관계 뿐만 아니라 회로 요소의 세부 파라메터 까지 비교해 준다. 트랜지스터의 경우 기하학적 모습(채널의 폭과 길이)를 포함한다.

II-1. 작업 폴더

작업 디렉토리로 이동,

```
$ cd ~/ETRI050_DesignKit/Tutorials/1-3_Inverter_Netgen
```

Magic 의 환경 설정 파일 복사,

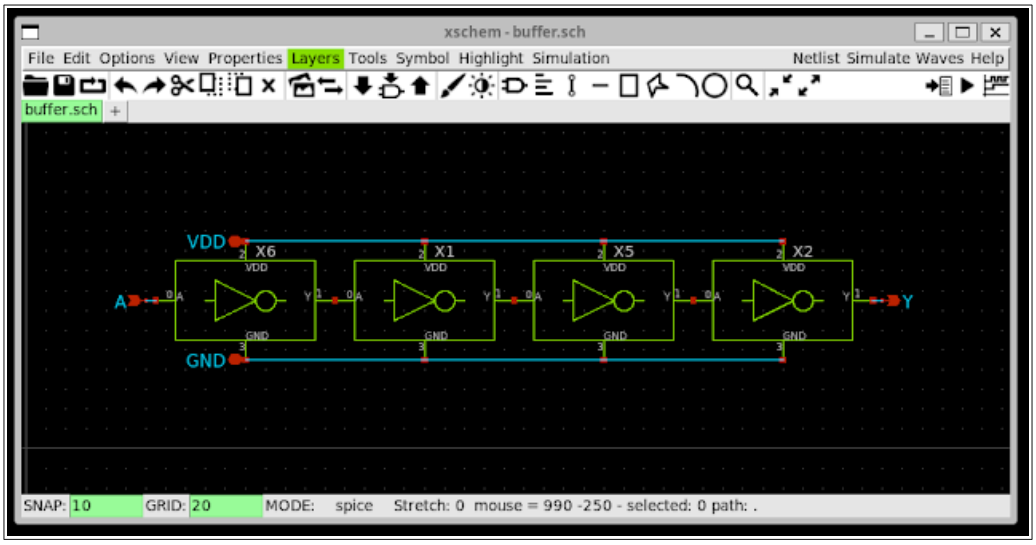
```
$ cp ~/ETRI050_DesignKit/tech/etri050.magicrc .magicrc
```

Netgen의 환경설정 파일 복사,

```
$ cp ~/ETRI050_DesignKit/tech/etri050_setup.tcl netgen_setup.tcl
```

II-2. 계층적 회로도

XSchem은 계층적 회로도를 작성할 수 있다. 앞선 실습에서 만들어둔 인버터를 하위 회로로 4개를 잇달아 연결한 버퍼 회로를 작성한다. 생성된 네트리스트를 보면 하위회로 inverter 4개가 연속으로 이어진 회로로 기술된 것을 알수 있다. SPICE 의 .subckt 는 하위회로를 정의하는 명령이다.




```
** sch_path: ~/ETRI050_DesignKit/Tutorials/1-3_Inverter_Netgen/buffer.sch
**.subckt buffer A Y VDD GND
*.ipin A
*.opin Y
*.iopin VDD
*.iopin GND
X6 A net1 VDD GND inverter
X1 net1 net2 VDD GND inverter
X5 net2 net3 VDD GND inverter
X2 net3 Y VDD GND inverter
**.ends
* expanding   symbol:  ~/Tutorials/1-1_Inverter_XSchem/inverter.sym
** sym_path: ~/Tutorials/1-1_Inverter_XSchem/inverter.sym
** sch_path: ~/Tutorials/1-1_Inverter_XSchem/inverter.sch
.subckt inverter A Y VDD GND
*.ipin A
*.opin Y
*.iopin VDD
*.iopin GND
M2 Y A VDD VDD pfet w=2.0u l=0.6u m=1
M1 Y A GND GND nfet w=2.0u l=0.6u m=1
.ends
.end
```

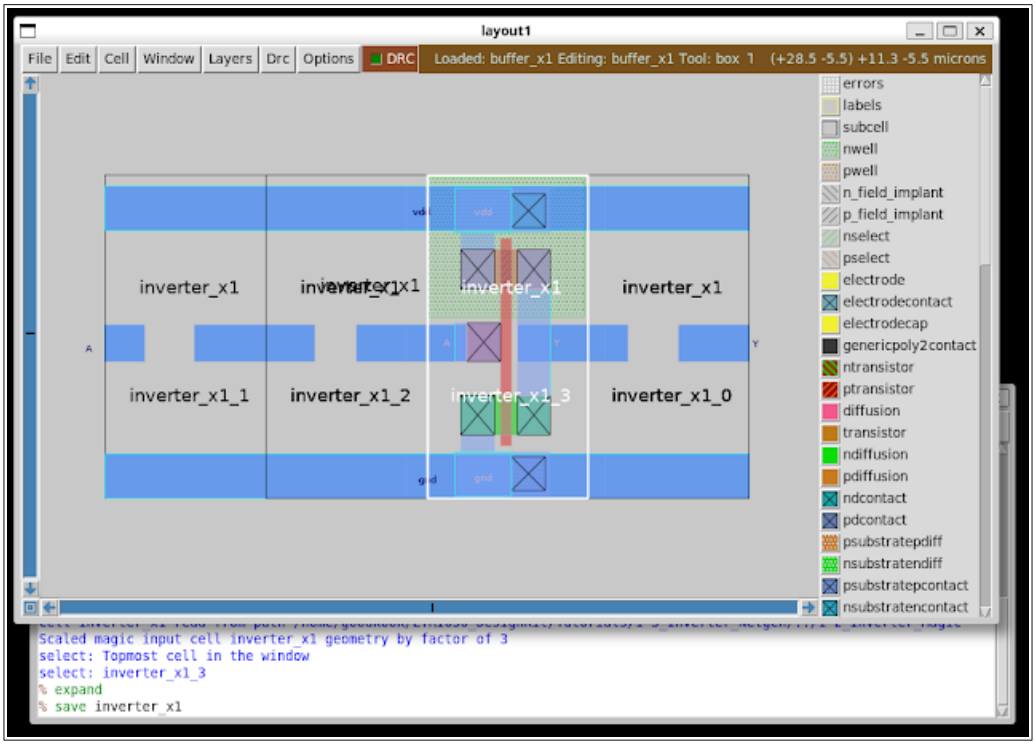
상위 회로에 .subckt 를 붙이지 않고 있는 점에 유의하자. 정의한 하위회로를 실제로 존재하게 해야(사례화, instantiate) 비로서 회로가 꾸며진다.

II-3. 계층적 레이아웃

Magic은 계층적 레이아웃 설계를 지원한다. 앞서 제작해둔 인버터 레이아웃을 하위 셀(sub-cell)로 불러올 수 있다. 현재 위치의 환경 파일 .magicrc에 하위 셀이 저장된 경로를 추가해 준다. 상대 경로도 가능하다.

```
addpath ../1-2_Inverter_Magic
```

Magic의 명령창에서 앞서 제작해둔 인버터 inverter_x1 을 불러온 후 이를 복사하여 버퍼를 만들고 buffer_x1 으로 저장한다.



하위 셀들을 불러오는 명령은 getcell 이다.

```
% getcell inverter_x1
```

불러온 하위 셀을 복사(단축키 'c')하여 배치한 후 metal1 레이어를 이용하여 배선한다. 배선이 완료되면 상위 셀에서 입출력 포트를 붙여 주도록 한다. 상위 설계에서 하위 셀의 내부를 보여주는 명령은 expand (단축키 x) 다. 반대는 unexpand (단축키 'X')다.

II-4. 네트리스트 추출(쉘 스크립트 활용예)

레이아웃에서 네트리스트의 생성은 용도에 따라 차이가 있다. 앞의 Magic 레이아웃 실습[기술노트6]에서는 회로 시뮬레이션을 위한 용도로 배선으로 인한 기생 커페시턴스까지 추출 하였다. 이를 기생 커페시턴스는 없는 상위 수준의 XSchem 회로도[기술노트5]에서 추출한 네트리스트와 비교할 경우 회로 불일치 오류를 발생한다.

한 레이아웃에서 용도에 따라 다른 네트리스트를 생성하기 위해 편집(수정과 삭제)기능이 있는 도구를 매번 실행시키면 자칫 실수가 끼어들 수 있다. 특히 단축키들의 사용이 광범위한 그래픽 기반의 편집 도구들은 실수로 인한 설계물의 변경 위험이 너무 크다. 이런 위험을 배제하기 위해 스크립트들이 적극 활용되고 있다. Magic 역시 그래픽

도구를 띄우지 않고 명령줄 실행이 가능하다. Magic을 실행 할 때 명령줄에 디스플레이 옵션을 null로 주면 Tcl 명령 프롬프트 % 가 뜬다. 레이아웃 파일을 불러들여 네트리스트를 추출하는 명령을 줄 수 있다.

```
$ magic -dnull -noconsole
% load inverter_x1
% extract all
% ext2spice default
% ext2spice format ngspice
% ext2spice scale off
% ext2spice subcircuit on
% ext2spice hierarchy on
% ext2spice
```

이미 그려놓은 레이아웃에서 옵션만 변경하여 LVS용 네트리스트를 추출하려면 다소 복잡한 절차가 필요하다. 이 네트리스트가 필요할 때마다 이 절차를 tcl 명령줄에 입력하기도 피곤하다. 리눅스의 쉘 스크립트를 활용하자. 리눅스의 쉘 스크립트는 간이형 프로그래밍 언어로서 손색이 없다. 파일의 형식변환, 네트리스트 생성과 같은 일련의 작업이 필요한 경우 쉘 스크립트로 작성해 두면 매우 유용하다. Magic 레이아웃에서 시뮬레이션용 네트리스트를 생성하는 배쉬-쉘 스크립트를 아래와 같이 작성해 놓으면 매우 유용하게 쓸 수 있다.

```
#!/usr/bin/bash
### filename: extract_spice_sim.sh
if [ -f ".magicrc" ] ; then
    echo "Magic rc found"
else
    echo "Magic rc NOT found"
    exit 2
fi
if [[ $# -ne 1 ]]; then
    echo "usage: extract <Magic Layout>"
    echo "       Extract spice netlist for simulation"
    exit 2
fi
magic -dnull -noconsole << EOF
    drc off
    box 0 0 0 0
    load $1 -force
    drc off
    extract all
    ext2spice default
    ext2spice format ngspice
    ext2spice scale off
    ext2spice subcircuit on
    ext2spice hierarchy on
    ext2spice
    quit -force
    EOF
```

쉘 스크립트는 양식없는 문서(plain text) 파일 이므로 배치 파일로서 실행 될 수 있도록 파일 속설을 바꿔 주어야 명령줄에서 실행 시킬 수 있다.

```
$ chmod +x extract_spice_sim.sh
$ ./extract_spice_sim.sh
```

II-5. LVS 실행 (파이썬 활용 예)

Netgen으로 XSchem 회로도에서 생성한 네트리스트와 Magic 레이아웃에서 생성한 네트리스트를 비교하는 명령은 다음과 같다.

```
$ netgen -batch lvs \
"./simulation/buffer_top.spice buffer" \
"buffer_x1.spice buffer_x1"\
./netgen_setup.tcl comp.out
```

두 SPICE 네트리스트 내에 비교할 하위 회로를 지정하고 있다. 실행은 매우 단순하지만 입력해야 할 명령줄이 너무 길다. LVS 를 수행하는 스크립트를 파이썬으로 작성하면 아래와 같다.

```
#!/usr/bin/env python3
### filename: run_lvs.py
import os, sys
if len(sys.argv) != 3:
    print('lvs: Specify two netlist folder and '
                    'filename to compare.')
    sys.exit(1);
os.system('netgen -batch lvs \
                  "{}.spice {}"  \
                  "./simulation/{}_top.spice {}" \
                   ./netgen_setup.tcl comp.out' \
                   .format( sys.argv[1], sys.argv[1],  \
                   sys.argv[2], sys.argv[2]))
sys.exit(0)
```

명령줄의 첫번째 인수로 Magic에서 생성한 네트리스트, 두번째 인수는 XSchem에서 생성한 네트리스트다. 각 네트리스트가 생성된 경로에 유의한다. 위의 파이썬 스크립트를 실행하여 LVS를 실시한다.

```
$ ./run_lvs.py buffer_x1 buffer
```

두 네트리스트의 비교 결과 불일치의 보고를 받을 경우 결과 보고 파일 comp.out 을 살펴보자.

```
Flattening unmatched subcell inverter in circuit
./simulation/buffer.spice (0)(4 instances)
Subcircuit summary:
Circuit 1: ./simulation/buffer.spice |Circuit 2: buffer_x1.spice
-------------------------------------|-----------------------------
```

```
pfet (4)                                |pfet (4)
nfet (4)                                |nfet (4)
(no matching element)                   |c (5)
Number of devices: 8 **Mismatch**       |Number of devices: 13
Number of nets: 7                       |Number of nets: 7
---------------------------------------------------------------------
NET mismatches: Class fragments follow (with fanout counts):
```

불일치의 원인은 XSchem 회로도에서 추출한 네트리스트에 없던 커페시터(c) 5개가 Magic 레이아웃 네트리스트에 포함되었기 때문이다. 레이아웃 네트리스트를 추출할 때 기생 C 가 없는 LVS 용으로 추출하는 스크립트는 아래와 같이 작성한다.

```
#!/usr/bin/bash
### filename: extract_spice_lvs.sh
if [ -f ".magicrc" ] ; then
    echo "Magic rc found"
else
    echo "Magic rc NOT found"
    exit 2
fi
if [[ $# -ne 1 ]]; then
    echo "usage: extract <Magic Layout>"
    echo "       Extract spice netlist for simulation"
    exit 2
fi
magic -dnull -noconsole << EOF
    drc off
    box 0 0 0 0
    load $1 -force
    drc off
    extract all
    ext2spice default
    ext2spice format ngspice
    ext2spice cthresh infinite
    ext2spice rthresh infinite
    ext2spice scale off
    ext2spice subcircuit on
    ext2spice hierarchy on
    ext2spice
    quit -force
    EOF
```

시뮬레이션용 네트리스트 추출 스크립트와 다른 점은 기생 C 와 R 의 추출 한계치 cthresh와 rthresh 를 무한대 infinite 로 놓고 있다. LVS용 스크립트를 이용하여 생성한 네트리스트를 가지고 비교를 수행한다.

```
$ ./extract_spice_lvs.sh buffer_x1
$ ./run_lvs.py buffer_x1 buffer
Contents of circuit 1:  Circuit: './simulation/buffer.spice'
Circuit ./simulation/buffer.spice contains 8 device instances.
```

```
Class: pfet                  instances:   4
Class: nfet                  instances:   4
Circuit contains 7 nets.
Contents of circuit 2:  Circuit: 'buffer_x1.spice'
Circuit buffer_x1.spice contains 8 device instances.
  Class: pfet                  instances:   4
  Class: nfet                  instances:   4
Circuit contains 7 nets.
Circuit 1 contains 8 devices, Circuit 2 contains 8 devices.
Circuit 1 contains 7 nets,    Circuit 2 contains 7 nets.
Final result:
Circuits match uniquely.
Logging to file "comp.out" disabled
LVS Done.
```

두 회로가 일치한다는 보고를 얻는다.

III. 시뮬레이션

방법론은 물론 추상화 수준이 전환되면 두 설계물의 등가성 확인은 필수 사항이다. 이와 더블어 전환 전후의 회로의 물리적인 변화는 시뮬레이션을 통해 확인 되어야 한다. 전환 전의 시뮬레이션을 Pre-Simulation, 전환된 후의 시뮬레이션을 Post-Simulation 이라 한다. 실물에 가까울 수록 시간 지연을 예상 할 수 있다.

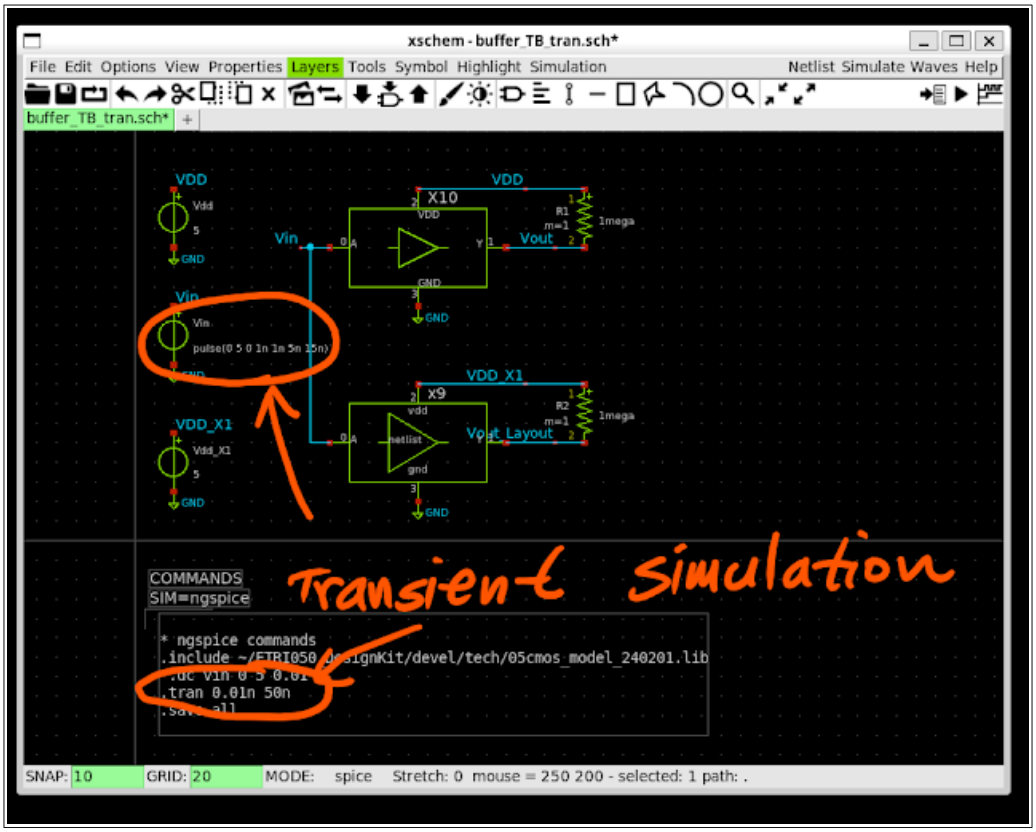
III-1. 두종류 네트리스트: 시뮬레이션용 vs. LVS 용

레이아웃은 실물에 가깝다. 트랜지스터의 채널은 물론 드레인과 소스 기하학적 모양과 배선으로 인한 기생 성분이 추가된다. 미리 작성해둔 스크립트를 이용하여 레이아웃으로부터 시뮬레이션용 네트리스트를 추출한다.

```
$ ./extract_spice_sim.sh buffer_x1
```

III-2. 테스트 벤치

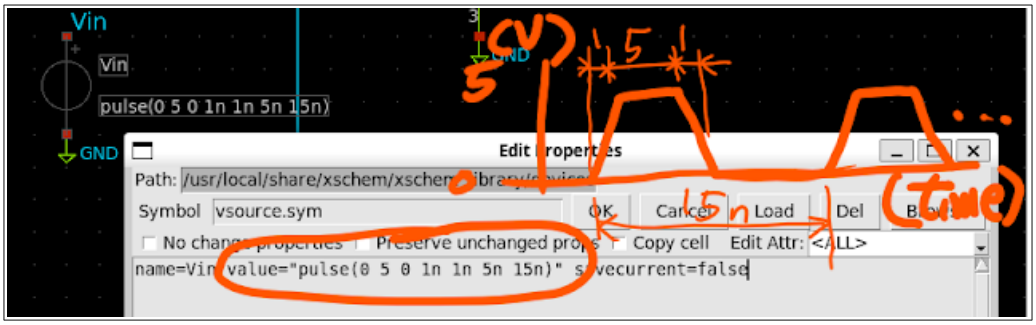
XSchem으로 SPICE 시뮬레이션용 아래와 같은 테스트 벤치를 작성한다. 테스트벤치 두개의 동일한 버퍼 심볼이 사례화 되어 있는 점에 주목하자. 하나는 회로도의 네트리스트이며 다른 하나는 레이아웃에서 추출한 네트리스트이다. 두 네트리스트에 동일한 테스트 입력을 주었다.



위의 테스트벤치 회로도에서 추출한 시뮬레이션용 SPICE 네트리스트 buffer_TB_tran.spice 의 일부를 보면 다음과 같다. X9는 Magic 레이아웃에서 추출한 buffer_x1 이며 X10은 XSchem으로 작성한 회로도에서 추출한 것임을 알 수 있다.

```
Vdd VDD GND 5
R1 VDD Vout 1mega m=1
R2 VDD_X1 Vout_Layout 1mega m=1
Vin Vin GND pulse(0 5 0 1n 1n 5n 15n)
Vdd_X1 VDD_X1 GND 5
X9 Vin Vout_Layout VDD_X1 GND buffer_x1
X10 Vin Vout VDD GND buffer
```

두 회로는 기능적으로 동일 하므로 입력에 대한 전압 변화의 관찰(voltage sweep)은 의미없다. 레이아웃에서 추출한 네트리스트에는 트랜지스터의 기하학적 모양과 배선으로 인한 기생 성분을 반영한 것이므로 지연을 예상 할 수 있다. 시간축상의 출력 변화를 살펴보기로 한다. 이를 통과시간(transient) 시뮬레이션이라고 한다. 테스트벤치는 아래와 같다. 입력으로 시간상 변화하는 전압을 pulse()로 표현하였다. SPICE는 pulse외에 수학 함수를 포함하여 다양한 방식으로 전압원을 생성할 수 있다[1].



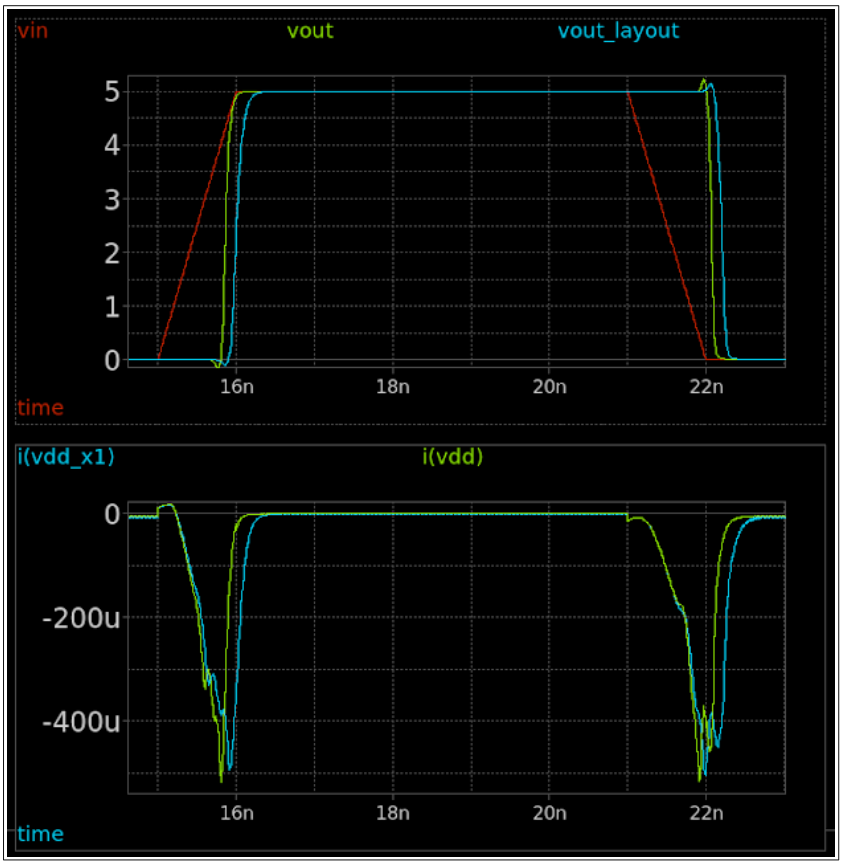
시뮬레이션 명령은 .tran 이다.

```
.tran 0.01n 50n
```

50 나노초까지 시뮬레이션을 진행하며 관찰 간격은 0.01나노초다.

III-3. 시뮬레이션 결과

테스트벤치에 두 설계물을 각각 사례화 한 네트리스트를 회로 시뮬레이션 하였다. 시뮬레이션 결과는 아래와 같다. 레이아웃에서 추출한 네트리스트에 포함된 기생 RC의 영향으로 약간의 지연을 보여준다.



간단한 인버터 회로를 대상으로 오픈-소스 설계 자동화 도구 XSchem(회로도 입력)과 Magic(레이아웃 그리기) 그리고 Netgen(네트리스트 비교 LVS)의 사용법을 살펴봤다. 오픈-소스 반도체 설계 자동화 도구와 상용 도구를 굳이 비교할 필요는 없다. 도구 사용법은 사용자 편의성을 고려했다 뿐이지 다 거기서 거기다. 어떤 도구를 가졌는가 보다 어떤 내용을 담을지 생각하자. 다음 편에서는 좀더 복잡한 회로를 다뤄보기로 한다.

[참조]

[1] Netgen version 1.5 netlist comparison (LVS) and format manipulation, http://opencircuitdesign.com/netgen/
[2] NSPL 0.5um Si-CMOS 2P3M Design Rule
[3] An introduction to the MAGIC VLSI design Layout System, https://terpconnect.umd.edu/~newcomb/vlsi/magic_tut/Magic_x3.pdf