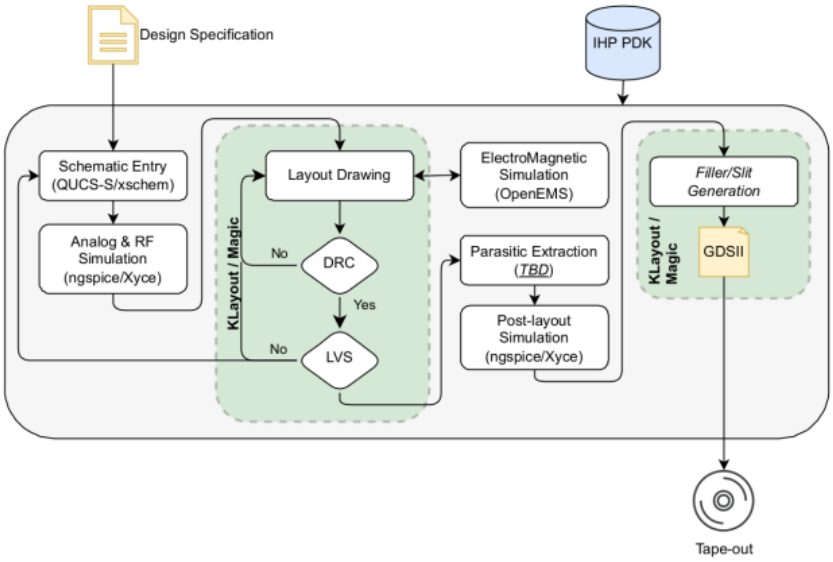
Instrukcja instalacji wymaganego oprogramowania i plików technologicznych oraz wykonania prostego projektu w technologii IHP SG13G2/cmos5l

Autor: Bogdan Pankiewicz, lipiec-sierpień 2026

Spis treści

Wstęp

Niniejszy dokument opisuje sposób wykorzystania oprogramowania CAD Free / Open Source wraz z Open PDK (ang. Process Design Kit czyli zestaw plików konfiguracyjnych dostosowujących środowisko do konkretnej technologii produkcji) firmy IHP dla technologii z rodziny SG13 [1]. Dokumentacja PDK jest dostępna na stronach firmy IHP [2]. Wykorzystany zostanie tzw. flow typu FULL-CUSTOM, czasami nazywany ANALOG/RF. Dla tego rodzaju podejścia do projektowania należy wykonać szereg czynności jak to przedstawiono na rys. 1.



Rys. 1. Poszczeglne etapy przebiegu projektowania typu Analog/RF wraz z wyszczególnieniem możliwych programów CAD typu Free/Open Source realizujących poszczególne etapy [1].

Instalacja środowiska projektowego oraz PDK

Jest kilka możliwych sposobów wykonania instalacji środowiska projektowego oraz zestawu plików technologicznych (PDK – ang. Process Design Kit). Poniżej przedstawione zostaną 3 opcje.

1) Oficjalne źródła firmy IHP

Najpewniejszą oraz najbardziej wymagającą możliwością jest zainstalowanie PDK oraz oprogramowania projektowego dokładnie wg zaleceń firmy IHP. Instrukcja jest opisana na stronie producenta [3]. Proces instalacji można w skrócie podzielić na następujące etapy:

a) instalacja systemu operacyjnego Linux Ubuntu 24.04 LTS
b) klonowanie PDK z githuba: [1]
c) instalacja zalecanych paczek
d) instalacja kompilatora i kompilacja modeli Verilog-A
e) instalacja oprogramowania projektowego: XSCHEM, NGSPICE, Xyce, Pygmid, Magic i innych niezbędnych do poprawnej pracy.

2) Dystrybucja IIC-OSIC-TOOLS

Najłatwiejszą metodą szybkiego dostępu do oprogramowania jest skorzystanie z dystrybucji narzędzi wykonanej przez **Department for Integrated Circuits (ICD), Johannes Kepler University** i dostępnych pod nazwą IIC-OSIC-TOOLS na githubie [4]. Narzędzia te instalują się bardzo szybko, wymagają wcześniejszej instalacji wirtualizacji Docker Desktop i otrzymujemy przygotowane od razu środowisko wraz z zestawem Open PDK od kilku dostawców, w tym z firmy IHP jak również bardzo

szeroki zakres oprogramowania Open CAD, znacznie wykraczający poza potrzeby projektów Full-Custom. Instalację oprogramowania w systemie Windows wykonuje się następująco:

a) należy zainstalować oprogramowanie wirtualizacji Docker Desktop (https://docs.docker.com/desktop/setup/install/windows-install/) oraz GIT (https://git-scm.com/install/windows),

b) należy uruchomić terminal ***Windows PowerShell*** i w katalogu głównym użytkownika wydać polecenie:

```
git clone https://github.com/iic-jku/IIC-OSIC-TOOLS.git
```

c) należy wejść do katalogu ściągniętego repozytorium wydając polecenie:

```
cd IIC-OSIC-TOOLS
```

d) uruchamiamy proces instalacji poleceniem: **`start_x.bat`**

e) nastąpi ściągnięcie obrazu i jego uruchomienie, otworzy się terminal co oznacza gotowość systemu do pracy

f) kolejne uruchomienia (bez instalacji obrazu) wykonuje się poprzez uruchomienie oprogramowania ***Docker Desktop*** a następnie wybranie opcji ***Containers*** i na obrazie ***iic-osic-tools*** wybieramy przycisk ***Start***

Dla przypadku użycia ww. instalacji, już po instalacji i uruchomieniu obrazu należy:

a) w uruchomionym terminalu przejść do katalogu roboczego (utworzyć i wejść przy pierwszym uruchomieniu)
b) katalog roboczy widoczny jest w katalogu domowym systemu hosta w podkatalogu **`eda`**
c) uruchomić skrypt konfiguracyjny technologii ***IHPSG13CMOS5L***:
`sak-pdk ihp-sg13cmos5l`

Od tej chwili dostępne będą odpowiednio skonfigurowane zmienne środowiskowe oraz oprogramowanie projektowe Open CAD.

3) Dystrybucja dostępna w zasobach KSME

Na serwerze ***intel5*** w ramach zasobów Katedry Systemów Mikroelektronicznych, WETI PG zainstalowana została dystrybucja PDK IHP SG13G2 wraz z oprogramowaniem projektowym z kategorii Full Custom Flow w postaci: XSCHEM, NGSPICE, Pygmid i Magic. Po zalogowaniu do komputera należy:

a) uruchomić terminal i upewnić się, że używamy powłoki BASH (polecenie ***ps***) a w razie innej powłoki wydać polecenie ***bash***
b) przejść do katalogu roboczego (utworzyć i wejść przy pierwszym uruchomieniu)
c) uruchomić skrypt konfiguracyjny:
`source /techfiles_ldap/openpdk/ihp-sg13cmos5l.csh`

Od tej chwili dostępne będą odpowiednio skonfigurowane zmienne środowiskowe oraz oprogramowanie projektowe Open CAD. Do połączenia z komputerem **`intel5`** można użyć 2 następujących metod:

1. użycie oprogramowania RDP wbudowanego w system Windows lub Linux, w tym celu dla Windows wpisujemy w wyszukiwarkę ciąg ***mstsc*** i powinna pojawić się ikona programu podłączenia pulpitu zdalnego, po jego uruchomieniu w miejsce komputer wpisujemy ***intel5*** i kontynuujemy połączenie podają swoje dane logowania,

2. użycie polecenia **`ssh -X login_użytkownika@intel5`** z terminala komputera w systemie Linux lub też z terminala Linux uruchomionego w WSL systemu Windows.

W zależności od konfiguracji stacji roboczej może się okazać, że jedna z tych metod daje wyraźnie lepsze odczucia użytkownika przy projektowaniu topografii w programie Klayout. W przypadku logowania się spoza sieci komputerowej katedry należy przetunelować porty za pośrednictwem komputera **`staff.ue.eti.pg.gda.pl`** (np. poprzez PUTTY). Dla połączenia SSH jest używany port 22 a dla RDP jest to port 3389.

Procedura wykonania projektu prostego podbloku układu scalonego

W celu przejścia pełnej ścieżki projektowej typu Full – Custom zaprojektowany zostanie najprostszy układ cyfrowy – bramka inwertera CMOS. Kolejno wykonane zostaną następujące etapy (w nawiasach podano wykorzystane oprogramowanie):

a) uruchomienie i konfiguracja środowiska, utworzenie katalogu roboczego (terminal z powłoką BASH),
b) narysowanie schematu elektrycznego inwertera (XSCHEM),
c) stworzenie symbolu inwertera (XSCHEM),
d) umieszczenie symbolu inwertera w układzie testowym (testbench) oraz wykonanie symulacji elektrycznych (XSCHEM, gnspice),
e) narysowanie topografii inwertera (KLAYOUT),
f) wykonanie operacji DRC (ang. – Design Rules Check) (KLAYOUT),
g) wykonanie operacji LVS (ang. – Layout Versus Schematic) (KLAYOUT),
h) wykonanie operacji PEX (ang. Parasitics Extraction) (Magic w tle z użyciem skryptu),
i) wykonanie symulacji po ekstrakcji topografii (XSCHEM, ngspice).

Po wykonaniu powyższych kroków uzyskamy projekt topografii bramki inwertera CMOS wraz z jej symulowanymi parametrami elektrycznymi. Nie ma znaczenia, wg. której ścieżki zainstalowane zostały narzędzia projektowe jak i PDK, zasady i kroki projektowe pozostają niezmienne.

XSCHEM - Przydatne, często używane skróty klawiszowe i myszki

Tab. 1. Skróty klawiszowe/myszki XSCHEM.

Klawisz skrótu	Funkcja
Insert	Wstawienie komponentu z biblioteki lub wstawienie własnego symbolu
F	Wyskalowanie schematu aby mieścił się w całości na ekranie
Lewy klawisz myszki	Zaznaczanie elementu, z przyciśniętym klawiszem Shift wielu elementów, można też zaznaczyć przez przeciągnięcie prostokąta nad zaznaczany obszar
Środkowy klawisz myszki	Przeciąganie powoduje przesunięcie schematu
Prawy klawisz myszki	Otwarcie menu kontekstowego
Rolka myszki	Zoom
Rolka + Shift	Przesuwanie lewo – prawo
Rolka + Ctlr	Przesuwanie góra – dół
A	Utworzenie symbolu z bieżącej zawartości okna schematu
E	Wejście do hierarchii zaznaczonego obiektu (jeśli posiada)

Ctrl + E	Wyjście z hierarchii bieżącego obiektu (jeśli istnieje hierarchia powyżej)
Shift + R, Alt + R	Obrót w prawo wokół kursora lub w miejscu
Shift + F, Alt + F	Odbicie wokół osi pionowej
Shift + V, Alt + V	Odbicie wokół osi pionowej
Ctrl + lewy klawisz myszki	Uruchomienie wskazanego myszką Launchera (zielona strzałka na schemacie) co skutkuje akcją w postaci np. uruchomienia symulacji, aktualizacji wykresów, i.t.d.

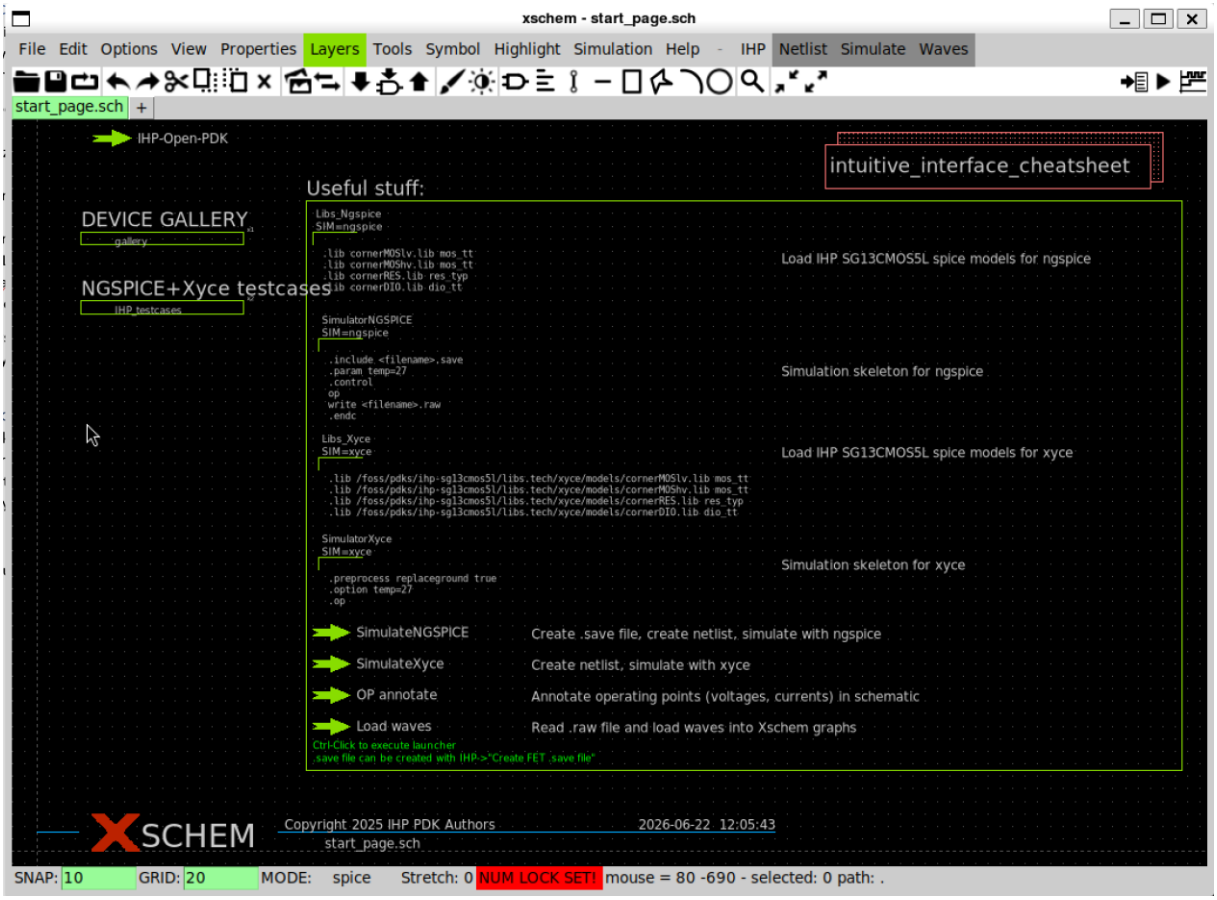
Narysowanie schematu elektrycznego inwertera w oprogramowaniu XSCHEM

Wprowadzenie schematu wykonane zostanie za pomocą programu XSCHEM. Poniżej w teście niniejszego dokumentu przedstawiono użycie tylko podstawowych funkcji. Szczegółowa instrukcja oraz poradnik dostępne są na stronie programu [5].

Założono, że użytkownik znajduje się w katalogu roboczym o nazwie **`test1`** i ma prawidłowo skonfigurowane środowisko oraz zainstalowane oprogramowanie jak to przedstawiono w poprzednim rozdziale. W celu uruchomienia edytora schematów, w terminalu wpisujemy polecenie (uwaga: Linux jest czuły na wielkość liter więc polecenia powinny być wprowadzone bez żadnych zmian):

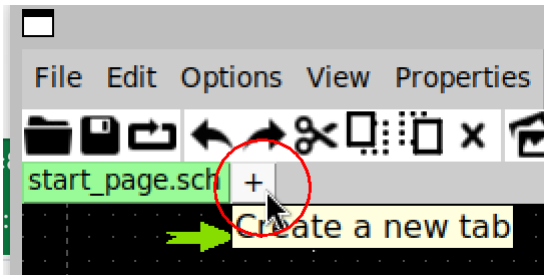
```
xschem &
```

Powinno pojawić się okno jak na rys. 2. Automatycznie wczytywany jest schemat z pliku „start_page.sch”, na którym umieszczone są podpowiedzi edycyjne dla Open PDK IHP. Edytor schematów jest hierarchiczny, co oznacza, że każdy z bloków może mieć rozwinięte wnętrze, w którym umieszczona jest jego zawartość. Hierarchia schematu może być wielopoziomowa, to znaczy w danym bloku może być podblok, który też jest hierarchiczny. Wejście do wnętrza bloku hierarchicznego wykonuje się poprzez zaznaczenie bloku lewym klawiszem myszki i wciśnięcie klawisza **E**, wyjście z podbloku do hierarchii wyżej poprzez kombinację klawiszy **Ctlr-E**. Aby utworzyć nowy schemat (ten domyślny proponuję zostawić bo z niego będzie można skopiować ustawienia symulatora do wykonania symulacji dla PDK IHP) należy nacisnąć pole ze znakiem **+** jak to przedstawiono na rys. 3.

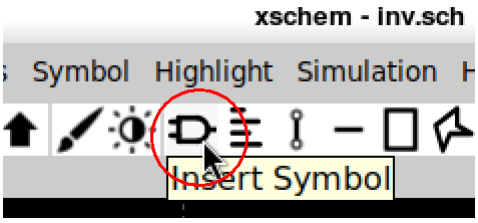


Rys. 2. Okno programu XSCHEM po pierwszym uruchomieniu ze skonfigurowanym PDK IHP SG13CMOS5L.

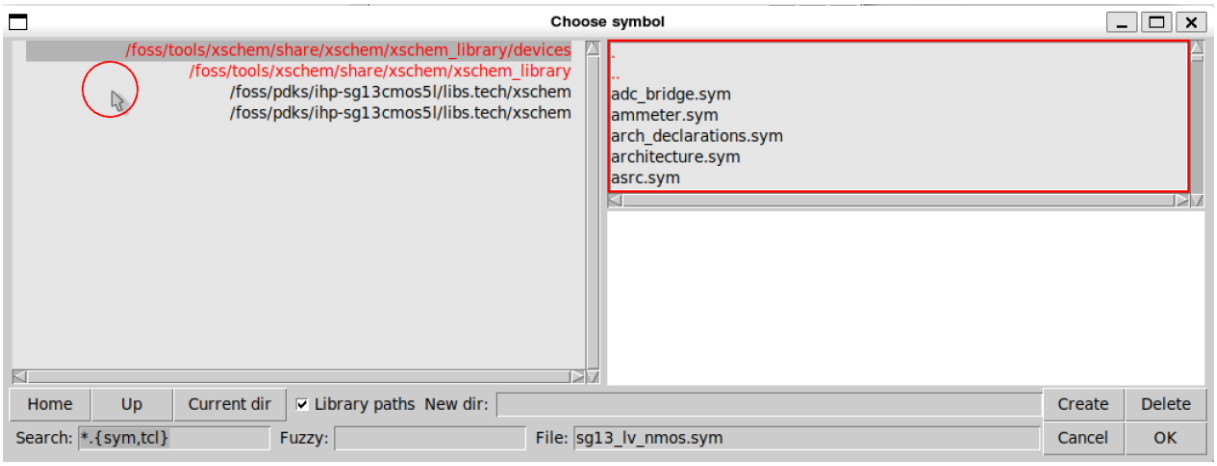
Następnie ten schemat warto zapisać od razu pod docelową nazwą (**Ctlr+Shift+S**). Umieszczanie elementów na schemacie polega na pobieraniu ich z bibliotek. W tym celu naciskamy klawisz **Insert** lub wybieramy menu jak na rys. 4. Pojawi się kolejne okno jak na rys. 5, gdzie można wybrać odpowiedni element z zestawów pogrupowanych w katalogach. Są 2 zestawy elementów: ogólne (katalogi na czerwono) i związane z technologią docelową (na czarno). Tworząc schemat bloku, który ma być fizycznie zaprojektowany do wykonania w postaci układu scalonego powinniśmy (zazwyczaj, bo niektóre połączeniowe jak np. PINy z ogólnej) wybierać elementy zgrupowane na czarno. W przypadku schematów służących do testowania bloków wybieramy elementy zgrupowane w katalogach opisanych czerwoną czcionką. W naszym konkretnym przypadku do połączeń użyjemy PINów z biblioteki ogólnej oraz tranzystorów z biblioteki technologicznej. Po umieszczeniu elementu możemy zmienić jego parametry zaznaczając go myszką i wciskając klawisz **Q** lub przez dwuklik. Poniżej w tabeli 2 przedstawiono listę elementów potrzebnych do narysowania schematu, ich funkcję oraz parametry do ustawienia. Po narysowaniu schematu pełen układ inwertera powinien wyglądać jak na rys. 6. Niektóre elementy (np. PINy) mają ukrytą widoczność nazwy wstawienia ale zamiennie widoczna jest etykieta, która definiuje późniejszą nazwę sieci do której jest wyprowadzenie połączone elektrycznie. Kolejność i nazwy elementów na schemacie nie mają istotnego znaczenia dla działania układu, muszą one być unikalne aby jednoznacznie zidentyfikować dany element. Unikalną dla technologii IHP SG13* jest konieczność umieszczania bloków ptap1 i ntap1 służących do podłączeń podłoży tranzystorów MOS. W topografii są to fizyczne kontakty do podłoża i wyspy. Brak umieszczenia tych elementów na schemacie spowoduje błędny proces LVS wykonywany w późniejszym etapie projektu.



Rys. 3. Utworzenie nowej zakładki (nowego schematu). Warto zaraz po utworzeniu nowego schematu zapisać go pod nową, docelową nazwą (Ctrl+Shift+S).



Rys. 4. Wstawienie nowego komponentu z biblioteki / wstawienie własnego, uprzednio przygotowanego symbolu.

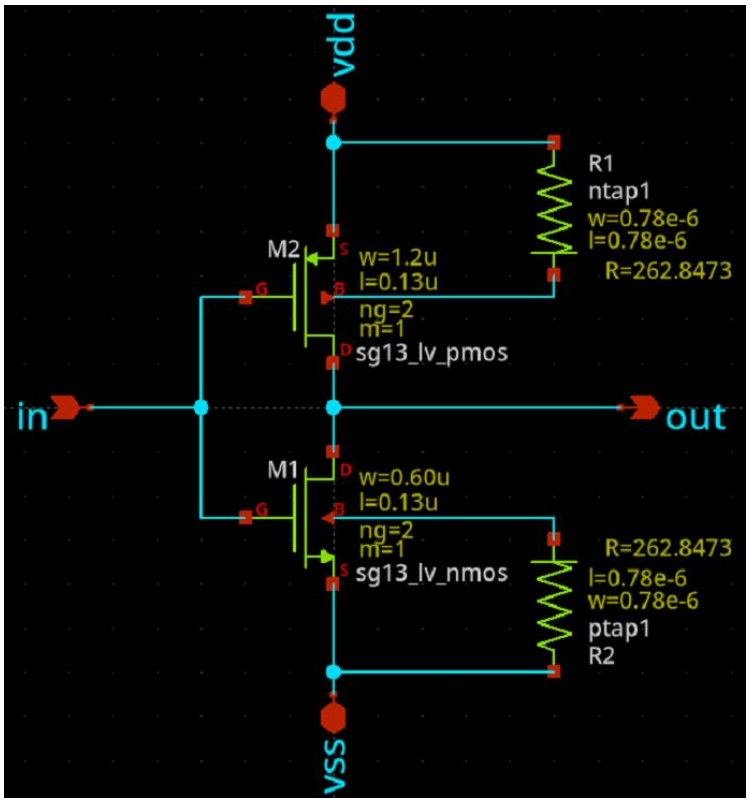


Rys. 5. Okno wyboru symbolu do wstawienia na schemacie. Symbole są umieszczone w katalogach wyszczególnionych po lewej stronie. Kolorem czerwonym zanznaczone są katalogi w elementami sytemowymi natomiast czarnym z elementami w technologii docelowej.

Tab. 2. Lista elementów umieszczonych na schemacie inwertera CMOS (plik inv.sch).

L.P.	Nazwa	Pełniona funkcja	Nazwa elementu w bibliotece	Biblioteka	Zmodyfikowane parametry
1	p1	wejście	ipin.sym	.../xschem_library/devices	lab=in
2	p2	wyjście	opin.sym	.../xschem_library/devices	lab=out
3	p3	zasilanie	iopin.sym	.../xschem_library/devices	lab=vdd
4	p4	masa	iopin.sym	.../xschem_library/devices	lab=vss
5	M1	tranzystor NMOS	sg13_lv_nmos.sym	…/xschem/sg13cmos5l_pr	w=0.60u ng=2

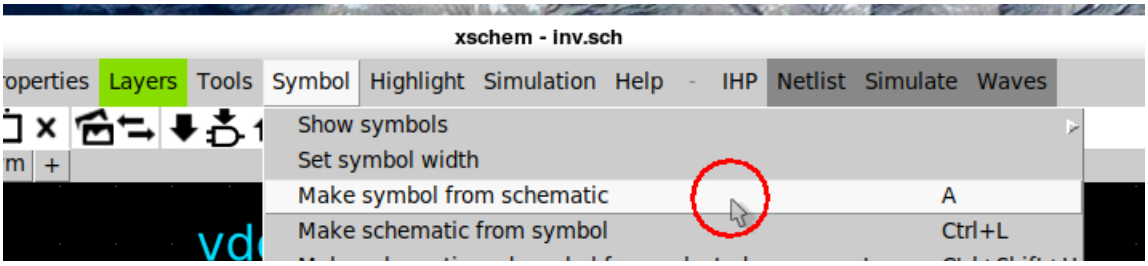
6	M2	tranzystor PMOS	sg13_lv_pmos.sym	…/xschem/sg13cmos5l_pr	w=1.2u ng=2
7	R1	polaryzacja wyspy do „vdd" (zasilania)	ntap1.sym	…/xschem/sg13cmos5l_pr	
8	R2	polaryzacja podłoża do „vss" (masy)	ptap1.sym	…/xschem/sg13cmos5l_pr	
9	p5	nazwa sieci	lab_wire.sym	.../xschem_library/devices	lab=subp
10	p6	nazwa sieci	lab_wire.sym	.../xschem_library/devices	lab=subn



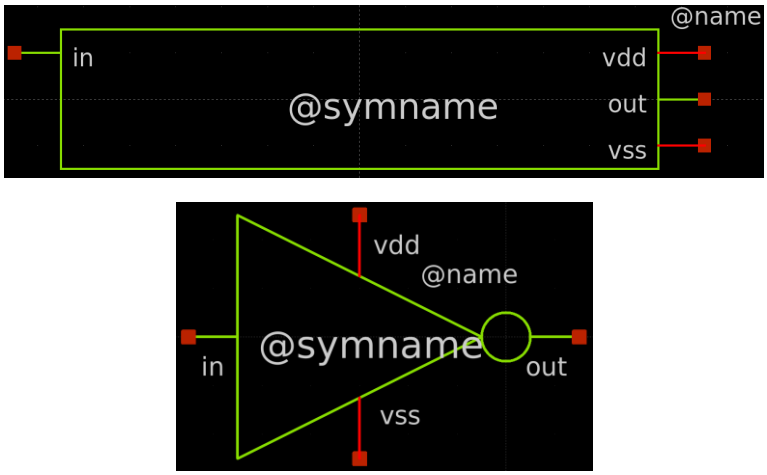
Rys. 6. Schemat inwertera CMOS. Parametry zostały ustawione jak w tabeli 2.

Stworzenie symbolu inwertera

Symbol podbloku jest niezbędny aby można było go osadzić jako podukład w większym projekcie lub w środowisku testowym, które potocznie nazywa się testbenchem. W niniejszych materiałach bloki testbenchy będą nazywana jak badany blok z dodanym zakończeniem nazwy w postaci „**_tb**", testbench do bloku inwertera przybierze więc nazwę „**inv_tb.sch**". Aby stworzyć symbol podbloku najłatwiej jest wykorzystać wbudowany generator symbolu ogólnego a następnie zmodyfikować wygląd. Otwieramy schemat bloku, któremu chcemy utworzyć symbol (np. jak na rys. 6), następnie wybieramy menu ***Symbol/Make symbol from schematic*** jak to przedstawiono na rys. 7. W katalogu użytkownika zostanie utworzony symbol, który powinniśmy otworzyć i zmodyfikować. W tym celu dodajemy zakładkę jak wcześniej to już robiliśmy na rys. 3 a następnie otwieramy utworzony symbol np. menu ***File/Open*** i wybieramy nazwę **inv.sym**. Teraz za pomocą możliwych narzędzi do rysowania usuwamy prostokąt i rysujemy kształt typowego inwertera z wyprowadzeniami zasilającymi jak na rys. 8.



Rys. 7. Utworzenie symbolu z bieżącego schematu. Piny ze schematu zostaną automatycznie umieszczone na symbolu. Użytkownik powinien jedynie zmodyfikować kształt symbolu na odpowiadający ogólnej funkcji podbloku.



Rys. 8. Modyfikacja wyglądu symbolu inwertera z inicjalnej wersji automatycznie utworzonej (góra) do zmodyfikowanego ręcznie kształtu (dół).

TESTBENCH i symulacje

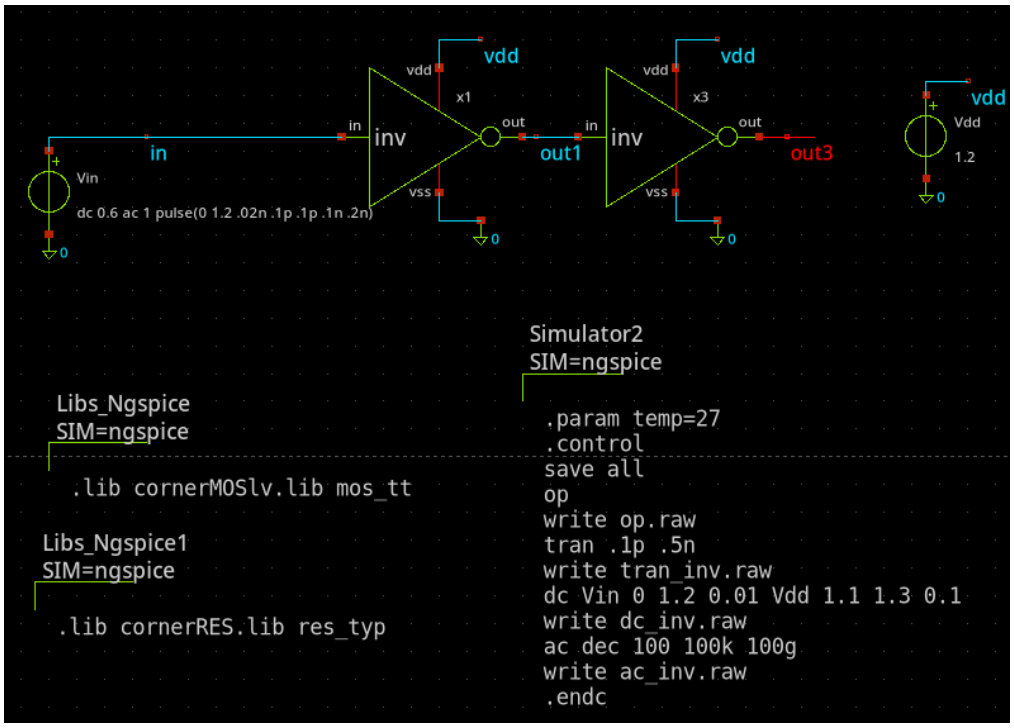
W celu wykonania symulacji umieszcza się badany układ (jego symbol) na schemacie zawierającym pozostałe niezbędne komponenty takie jak źródła sygnału, napięcie zasilające, obciążenie, linki do plików bibliotecznych modeli elementów, itd. W tym celu dodajemy zakładkę jak wcześniej to już robiliśmy na rys. 3 a następnie zapisujemy jako (**`Ctrl + Shift + S`**) plik o nazwie **`inv_tb.sch`**. Następnie nanosimy elementy do uzyskania schematu przedstawionego na rys. 9. Lista komponentów wraz z głównymi parametrami przedstawiona jest w tabeli 3. Warto również, nie tylko w testbenchu ale także na każdym schemacie, umieszczać jego tytuł i autora. Na schemacie poniżej wstawiona jest kaskada 2 inwerterów, parametry badane są na pierwszym z nich, drugi stanowi wyłącznie obciążenie. Fragmenty kodu oraz każdy inny element można kopiować pomiędzy schematami z wykorzystaniem schowka. W tym celu wybieramy dowolny schemat i zaznaczamy fragment, który chcemy skopiować. Następnie wciskamy klawisz C, który umieszcza zaznaczoną zawartość w schowku. Kolejno wybieramy schemat docelowy i lewym klikiem myszki umieszczamy skopiowane elementy. Kopiowanie nie zadziała jeśli w docelowym schemacie jest cokolwiek zaznaczone, więc przed rozpoczęciem procesu warto kliknąć lewym klawiszem myszki w puste miejsce na schemacie.

Program XSCHEM sam w sobie nie wykonuje symulacji elektrycznych. Jednak ze schematu tworzy listę połączeniową, która następnie poddawana jest symulacji w programie zewnętrznym. Do listy połączeniowej dołączane są kody umieszczone na schemacie. W naszym przypadku będziemy używać symulatora **`ngspice`** , możliwe są jednak inne opcje takie jak **`xyce`**, **`verilator`**, **`ghdl`** i inne. Wiąże się z tym konieczność zapewnienia aby kod był zgodny z docelowym programem. Symulator **`ngspice`**

jest oprogramowaniem Open Source, jego składnia jest w dużej części zgodna z ze składnią typowego SPICEA, jednak są istotne różnice w sterowaniu symulacją i analizach parametrycznych. Szczegółowa instrukcja jest dostępna na stronie programu [6].

Tab. 3. Lista najważniejszych komponentów umieszczonych na schemacie testbench-a (inv_tb.sch).

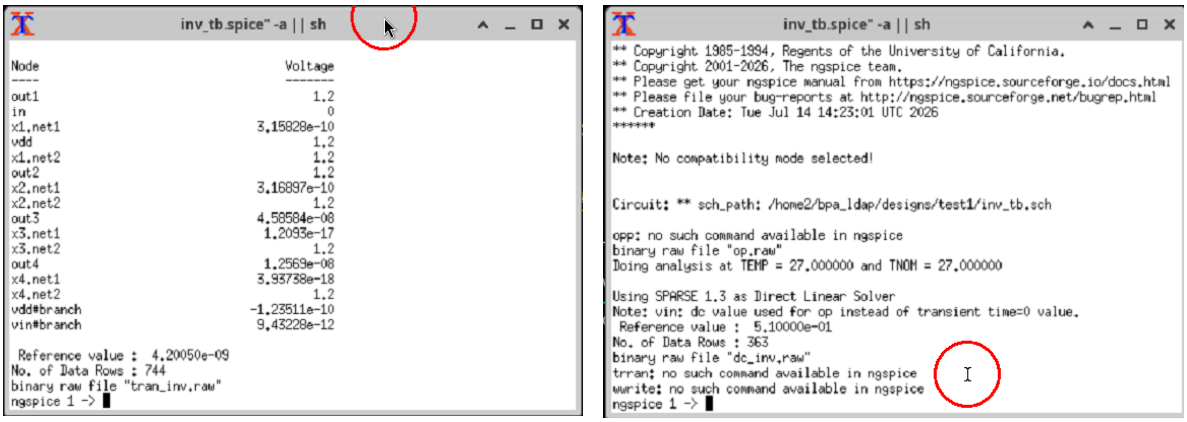
L.P.	Nazwa	Pełniona funkcja	Nazwa elementu w bibliotece	Biblioteka	Zmodyfikowane parametry (dostępne po zaznaczeniu komponentu i naciśnięciu klawisza Q) lub inny komentarz
1	x1	Wstawienie inwertera	inv.sym	.../test1	Brak
2	x2	Wstawienie inwertera	inv.sym	.../test1	brak
3	Vin	Wstawienie źródła sygnału	vsource.sym	.../xschem_library/devices	value="dc 0.6 ac 1 pulse(0 1.2 .02n .1p .1p .1n .2n)"
4	Vdd	Wstawienie źródła zasilania	vsource.sym	.../xschem_library/devices	value=1.2
5	Liczne nazwy węzłów	Nadanie nazw węzłom połączeniowym układu	lab_wire.sym	.../xschem_library/devices	lab=in lab=vdd lab=out1 lab=out2
6		Liczne wstawienia masy układu (węzeł 0)	gnd.sym	.../xschem_library/devices	Brak
7	Libs_Ngspice	Wstawienie biblioteki modeli MOS	simulator_commands_shown.sym	.../xschem_library/devices	name=Libs_Ngspice simulator=ngspice only_toplevel=false value=" .lib cornerMOSlv.lib mos_tt " Najprościej jest skopiować ze schematu start_page/IHP-testcases/dc_lv_nmos.sch
8	Libs_Ngspice1	Wstawienie biblioteki modeli ntap1 i ptap1	simulator_commands_shown.sym	.../xschem_library/devices	name=Libs_Ngspice1 simulator=ngspice only_toplevel=false value=" .lib cornerRES.lib res_typ " Najprościej jest skopiować ze schematu start_page/IHP-testcases/dc_ntap1.sch
9	Simulator2	Polecenia symulacyjne przekazywane do symulatora gnspice	simulator_commands_shown.sym	.../xschem_library/devices	name=Simulator2 simulator=ngspice only_toplevel=false value=" .param temp=27 .control save all op write op.raw dc Vin 0 1.2 0.01 Vdd 1.1 1.3 0.1 write dc_inv.raw tran .1p .5n write tran_inv.raw .endc " Najprościej jest skopiować ze schematu start_page/IHP-testcases/dc_lv_nmos.sch lub innego i zmodyfikować stosownie do potrzeb
10	l5	Tytuł schematu	title.sym	.../xschem_library/devices	Imię i nazwisko autora lub inne dane wg własnych potrzeb



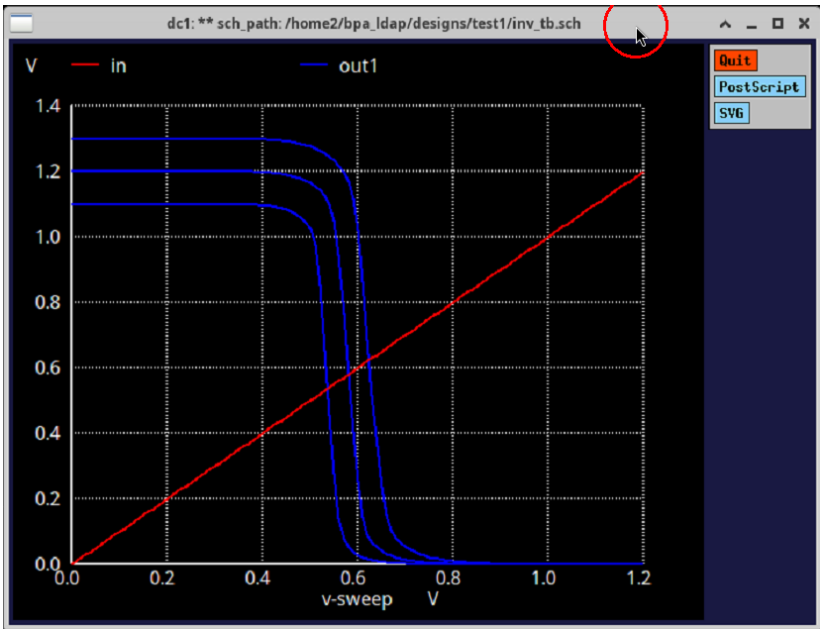
Rys. 9. Wstawione 2 inwertery CMOS wraz ze źródłem zasilania i sygnałem wejściowym oraz niezbędnymi komponentami do wykonania symulacji elektrycznych.

Aby wykonać symulację należy kolejno kliknąć w przycisk Netlist a następnie w przycisk Simulate. To wywoła przeprowadzenie symulacji (domyślnie wybrany jest progam **ngspice** w tybie interaktywnym). W przypadku braku błędów pojawi się okno jak na rys. 10 po stronie lewej. W takim przypadku należy zweryfikować kod poleceń jak również schemat. Bardzo pomocne może być przejrzenie utworzonej listy połączeniowej (ang. netlist) poprzez wydanie polecenia: ***Simulation/Edit netlist***. Należy pamiętać, że za każdym razem symulowana jest wcześniej utworzona netlista więc po jakichkolwiek zmianach w schemacie lub umieszczonym kodzie niezbędne jest najpierw wygenerowanie nowej wersji netlisty!

W przypadku pomyślnego wykonania symulacji można wyświetlić wyniki na wykresie poleceniem wydanym w konsoli symulatora: **plot in out1**. To powinno wykreślić przebiegi wymienione w poleceniu na wykresie w nowym oknie dla ostatnio wykonanej analizy. Na rys. 11 przedstawiono wyniki dla analizy DC. Polecenie powyższe wykorzystuje wewnętrzny poler programu **ngspice**. Można wykorzystać program niezależny, domyślnie **gaw**, poprzez przycisk: **Waves/External Viewer**.



Rys. 10. Okno symulatora po poprawnej (na leweo) i błędnej symulacji (na prawo).



Rys. 11. Wyniki symulacji inwertera CMOS wyświetlane w wewnętrznym procesorze graficznym programu ngspice.

Lepszą metodą jest umieszczenie wyników symulacji bezpośrednio na wykresie umieszczonym na schemacie. W tym celu należy wykonać kolejno (ilustracja na rys. 13):

- menu: ***Simulation/Graphs/Add waveform graph*** a następnie należy wskazać myszką na schemacie miejsce umieszczenia wykresu

- przycisk: ***Waves/Load first analysys found*** a następnie wybieramy plik z zapisanymi wynikami symulacyjnymi w celu załadowania ich do pamięci, w przykładzie poniżej wykorzystano analizę DC, plik o nazwie **dc_inc.raw**

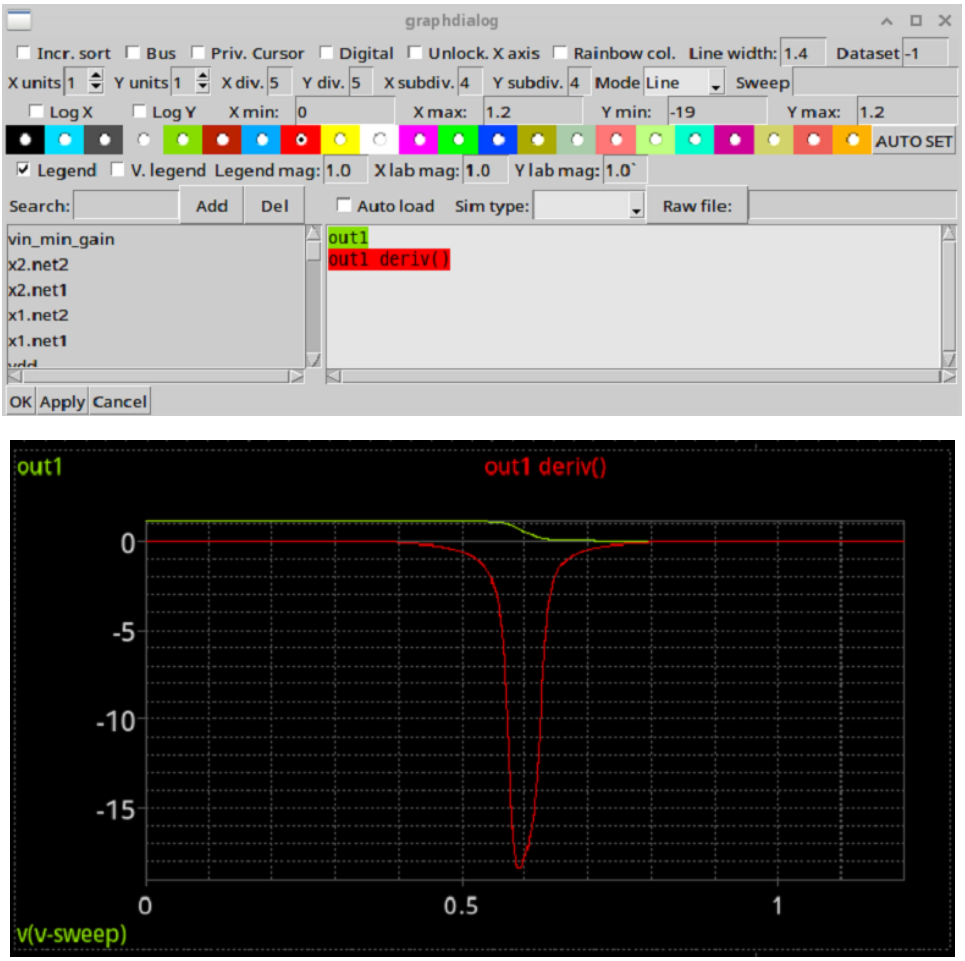
- najeżdżamy na środek wykresu i podwójnie klikamy lewym klawiszem myszki, pojawi się menu wyboru przebiegów do wyświetlania, kolejno wybieramy z lewego panelu sygnał do wyświetlenia następnie wybieramy kolor i przycisk ***Add*** , po dodaniu wszystkich sygnałów pojawią się one na wykresie, niestety domyślne skale osi X i Y wykresu są zazwyczaj złe i przebiegi nie są w ogóle widoczne lub widoczne tylko częściowo

- zmiana skali odbywa się poprze najechanie na oś i wciśniecie klawisza ***F***, powinniśmy to zrobić na obu osiach wówczas pojawią się wykresy w pełnych zakresach zmienności sygnałów

- zmiany skali i przesunięcia: ***Rolka*** myszki oraz ***Shift+Rolka*** myszki, zmiana występuje na osi wskazanej kursorem

- można umieścić na osiach do 2 kursorów, klawisze A oraz B (oś X) oraz Shift-A i Shift-B (oś Y), wartości sygnałów dla kursora A są prezentowane bezpośrednio pod nazwami sygnałów.

Bardzo ciekawą funkcją jest możliwość umieszczania na wykresie nie tylko samych wartości wybranych w przeglądarce sygnałów a także wartości matematycznych na tych sygnałach. Używany tu zapis jest w składni RPN (ang. Reverse Polish Notation), w skrócie najpierw zmienna/e a następnie operacja/funkcje. W tym celu modyfikujemy tekst reprezentujący sygnał do umieszczenia na wykresie poprzez zmianę nazwy sygnału na sygnał i operację, jak np. przedstawiono to na rysunku poniżej.



Rys. 12. Dodanie do wyświetlania, oprócz samego napięcia w węźle out1, również jego pochodnej względem osi X.

Warto też umieścić na schemacie tzw. **Lunchery**, które po ich wskazaniu myszką z równocześnie wciśniętym klawiszem ***Ctrl*** powodują wykonanie akcji w nich zdefiniowanej. Może to być np. generacja netlisty i wykonanie symulacji czy wczytanie wyników na wykres. Wstawienie następuje przez klawisz Insert a następnie element …/xschem_library/devices/launcher.sym. Następnie wskazujemy naniesiony **Launcher** i klawisz ***Q*** i uzupełniamy kod do wykonania w czasie aktywacji przez ***Ctrl-Lewy_klik***. Poniżej zamieszczony jest kod dla wywołania automatycznej generacji netlisty i symulacji ngspice:

name=h4

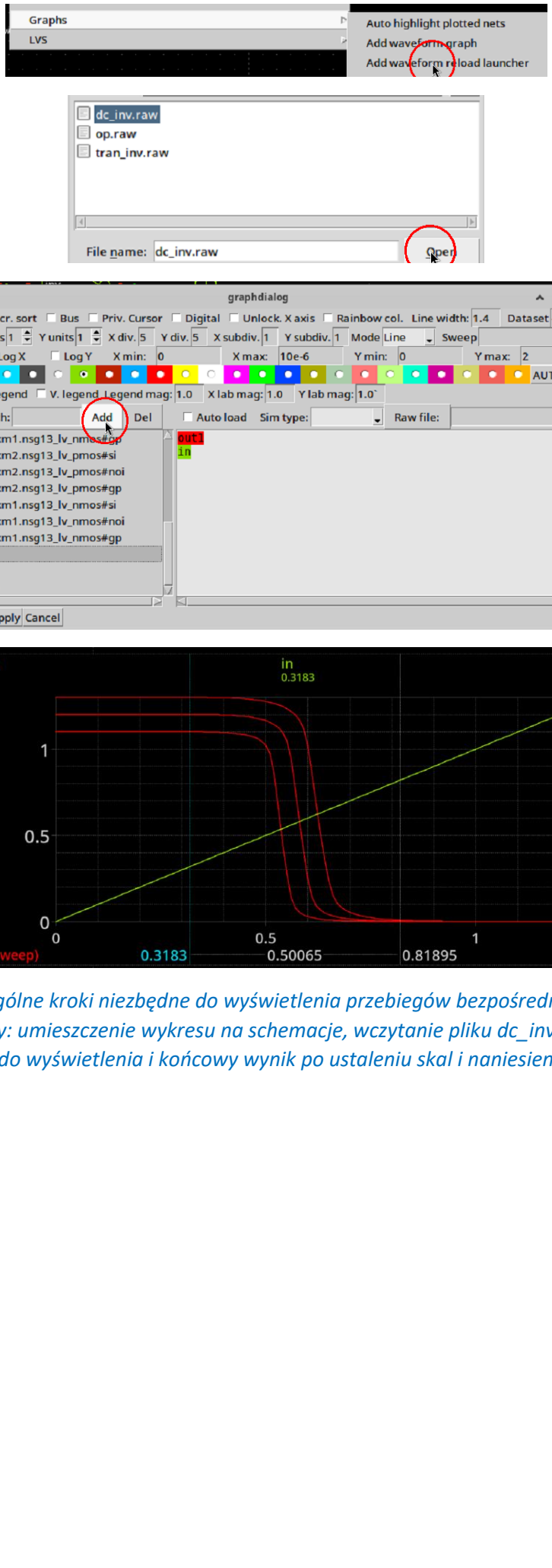
descr=SimulateNGSPICE

```
tclcommand="
# Setup the default simulation commands if not already set up
# for example by already launched simulations.
set_sim_defaults
puts $sim(spice,1,cmd)
# Change the Xyce command. In the spice category there are currently
# 5 commands (0, 1, 2, 3, 4). Command 3 is the Xyce batch
# you can get the number by querying $sim(spice,n)
set sim(spice,1,cmd) {ngspice  \"$N\" -a}
# change the simulator to be used (Xyce)
set sim(spice,default) 0
# run netlist and simulation
xschem netlist
simulate
"
```

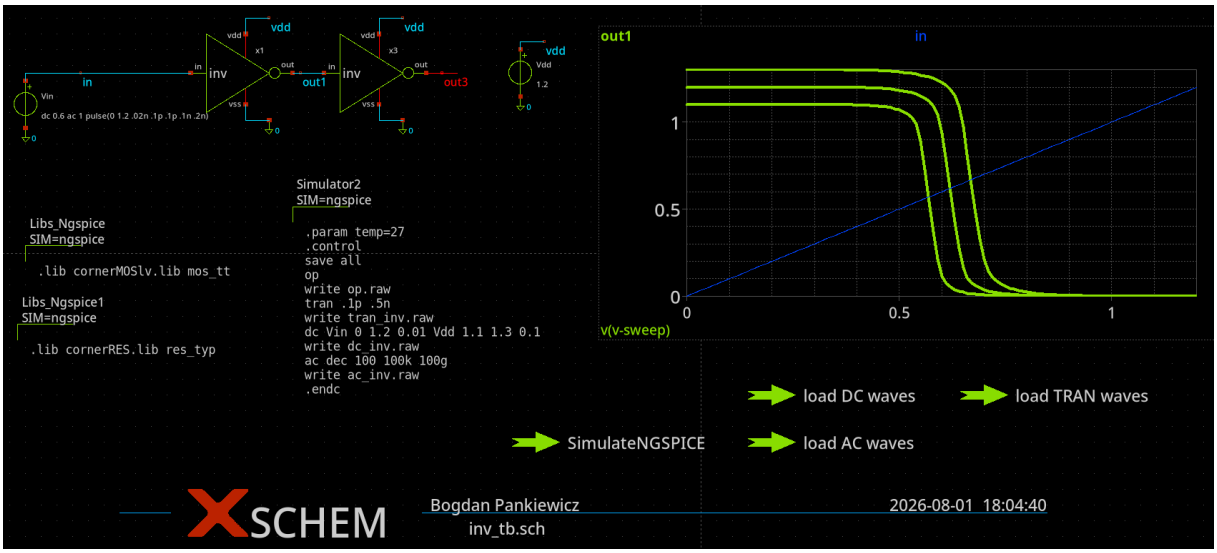
Kod dla wczytania wyników symulacji DC na wykres, w przypadku zmiany rodzaju analizy i nazwy pliku z wynikami należy go odpowiednio zmodyfikować:

```
name=h5
descr="load DC waves"
tclcommand="xschem raw_read $netlist_dir/dc_inv.raw dc"
```

Pełny schemat testbecha zawierający: schemat środowiska testowego wraz z jego zasilaniem oraz sygnałem wejściowym, polecenia symulacyjne, wykres z wynikami, podpis schematu oraz ***Lunchery*** przedstawiony jest na rys. 14.



Rys. 13. Poszczególne kroki niezbędne do wyświetlenia przebiegów bezpośrednio na schemacie układu. Od góry: umieszczenie wykresu na schemacje, wczytanie pliku dc_inv.raw, wybranie przebiegów do wyświetlenia i końcowy wynik po ustaleniu skal i naniesieniu kursorów.



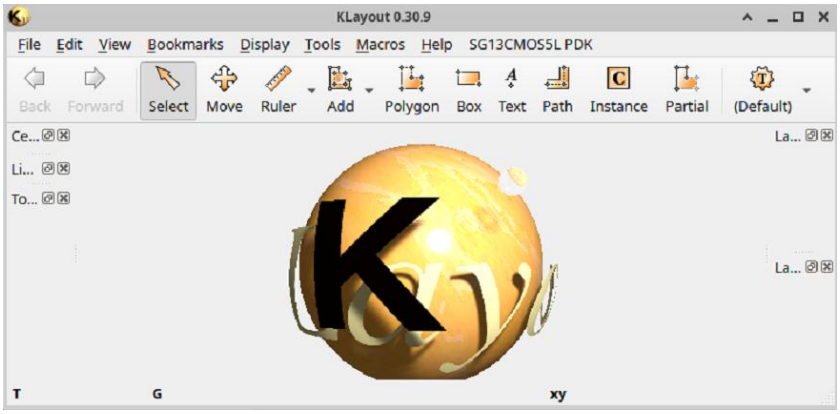
Rys. 14. Zawartość pełnego schematu testbencha inwertera.

Wykonanie projektu topografii w programie KLAYOUT

Do zaprojektowania topografii zostanie wykorzystany program KLAYOUT [7]. Jego uruchomienie (należy to wykonać w katalogu roboczym projektu lub w podkatalogu przeznaczonym na topografię) wykonuje się poprzez poniższe polecenie, parametr -e jest niezbędny aby program pracował w trybie edycji topografii:

```
klayout -e &
```

Po uruchomieniu pojawi się okno jak na rys. 15.



Rys. 15. Okno programu KLAYOUT bezpośrednio po uruchomieniu w trybie edycji.

Pierwszą czynnością jest wczytanie topografii lub utworzenie nowej. Utworzymy nową topografię poprzez menu: ***File/New Layout*** a następnie wybieramy technologię ***sg13cmos5l***, nazwę ***Top Cell*** zmieniamy na nasz inwerter czyli na ***inv*** a pozostałe parametry pozostawiamy niezmienione. Powinno pojawić się okno jak na rys. 16. Pierwszą czynnością powinien być wybór technologii poprzez strzałkę obok przycisku zaznaczonego na rysunku czerwonym okręgiem i wybranie ***sg13cmos5l***. Okno programu podzielone jest na kilka stref, które zostały wyszczególnione na rys. 16 a ich znaczenie jest następujące:

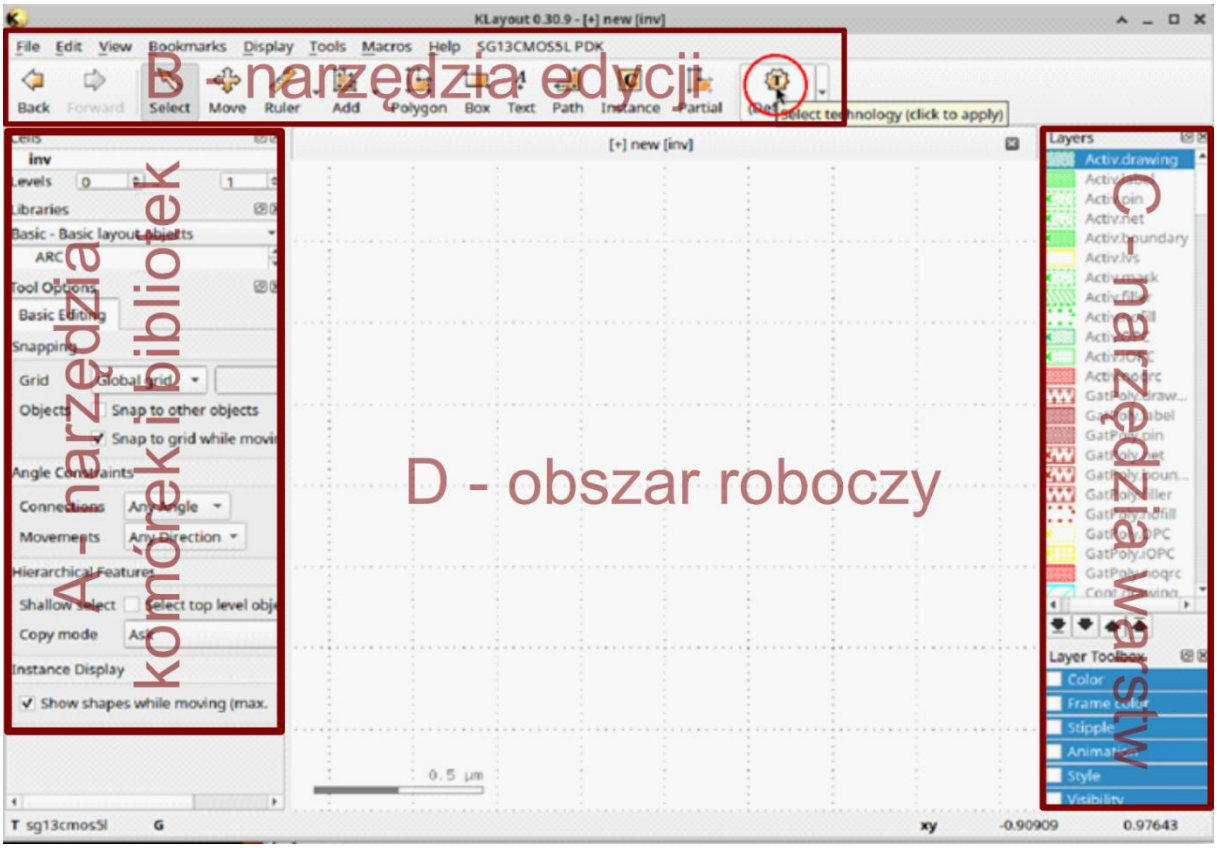
A) narzędzia komórek i bibliotek: tu umieszczone są edytowane/tworzone komórki oraz możliwy jest wybór tzw. PCELL (komórek parametryzowanych), komórka parametryzowana jest to standardowy layout wybranego rodzaju komponentu np. tranzystora NMOS czy też kontaktu

lub przelotki, dzięki takim komórkom rysowanie topografii jest znacząco szybsze, dzięki parametryzacji można podać docelowe wymiary tranzystora czy liczbę przelotek między warstwami,

B) narzędzia edycji: tu wybierany jest rodzaj operacji do wykonania na projektowanej topografii,

C) narzędzia warstw: lista warstw tu umieszczona zależna jest od wybranej technologii, warstwy tu umieszczone można wybierać, filtrować ich widzialność, używać do umieszczania w obszarze roboczym,

D) obszar roboczy: miejsce na umieszczenie projektu topografii.



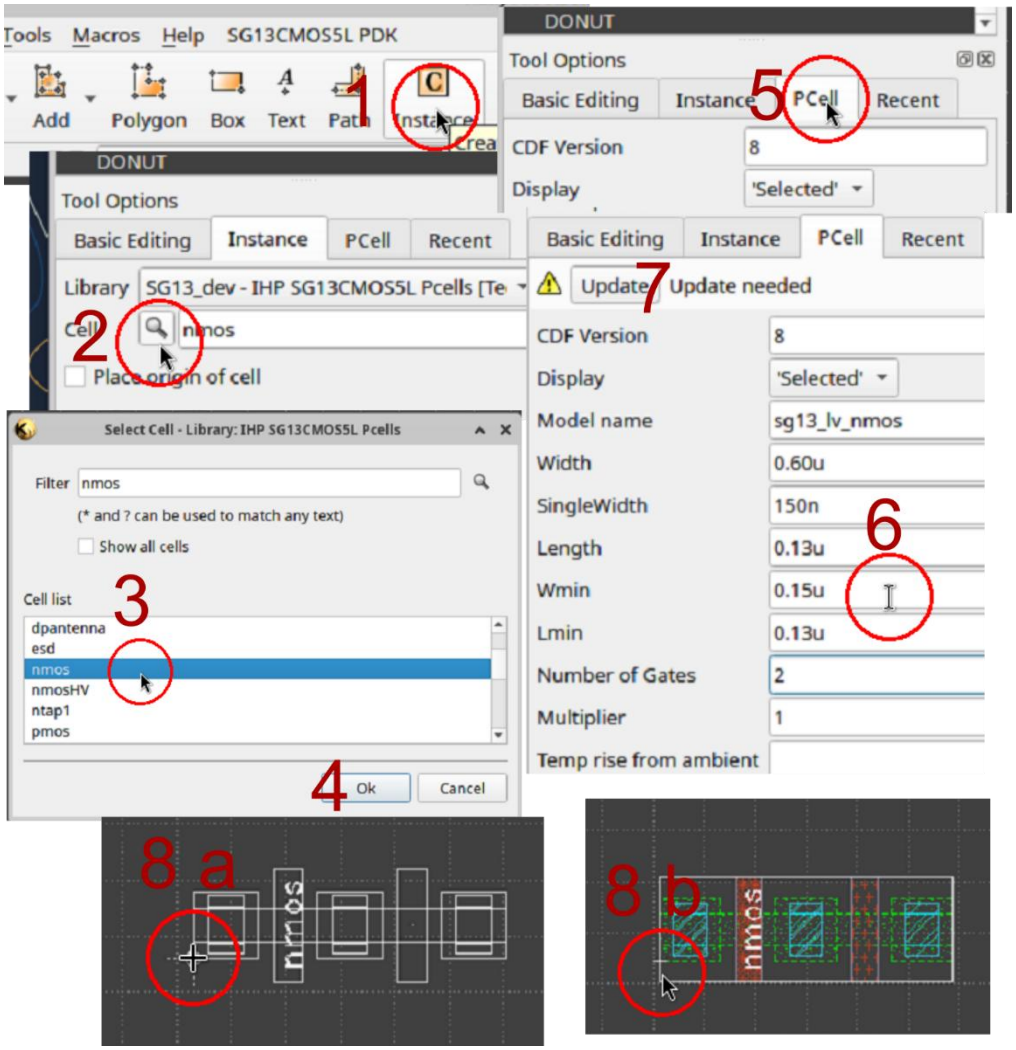
Rys. 16. Wygląd okna edytora topografii KLAYOUT bezpośrednio po powołaniu nowego pustego projektu wraz z zaznaczeniem stref narzędziowych.

Zasadniczo narysowanie topografii polega na wybraniu i zaznaczeniu warstwy, która ma być użyta (obszar C) a następnie na wybraniu narzędzia (obszar B) i użyciu go w głównym polu edytora (obszar D) w celu utworzenia odpowiedniego kształtu warstwy lub opisu. Każda z warstw fizycznych posiada kilka kategorii, które są wymieniane po kropce, np. Metal1.drawing, Metal1.pin, Metal1.label itd. Fizycznie jest to ta sama warstwa ale użycie odpowiedniej kategorii jest konieczne w celu prawidłowej interpretacji w czasie procesu projektowego, np. podczas operacji LVS lub PEX. Dla przyspieszenia tworzenia typowych struktur można alternatywnie najpierw wybrać narzędzie **`Instance`** (obszar B), następnie wybrać interesującą nas komórkę PCELL z biblioteki (obszar A), ustawić jej parametry (obszar A) i umieścić w odpowiednim miejscu w obszarze roboczym (miejsce D). Możliwe jest również wstawienie innej wcześniej utworzonej komórki w bieżącej. W taki sposób powstaje projekt w formie hierarchicznej. Komórki wstawione w ten sposób reprezentowane są poprzez ich prostokąt obejmujący co przyspiesza projektowanie dużych układów. Komórki PCELL też są komórkami hierarchicznymi. Domyślnie edytor topografii nie wyświetla zawartości elementów wstawionych hierarchicznie, aby

zobaczyć ich zawartość, należy w obszarze A w prawym polu **Levels** ustawić wartość poziomu hierarchii do którego chcemy widzieć zawartość, wyższe poziomy będą reprezentowane przez prostokąty.

Poniżej, w kolejnych punktach wyszczególniono wszystkie niezbędne czynności do wykonania topografii układu inwertera, którego symulację wykonano wcześniej. Poszczególne kroki zostały opisane skrótowo, dodano dla nich także ilustracje w postaci rysunków, na których naniesiono numery zgodne z kolejnością wykonywanych czynności.

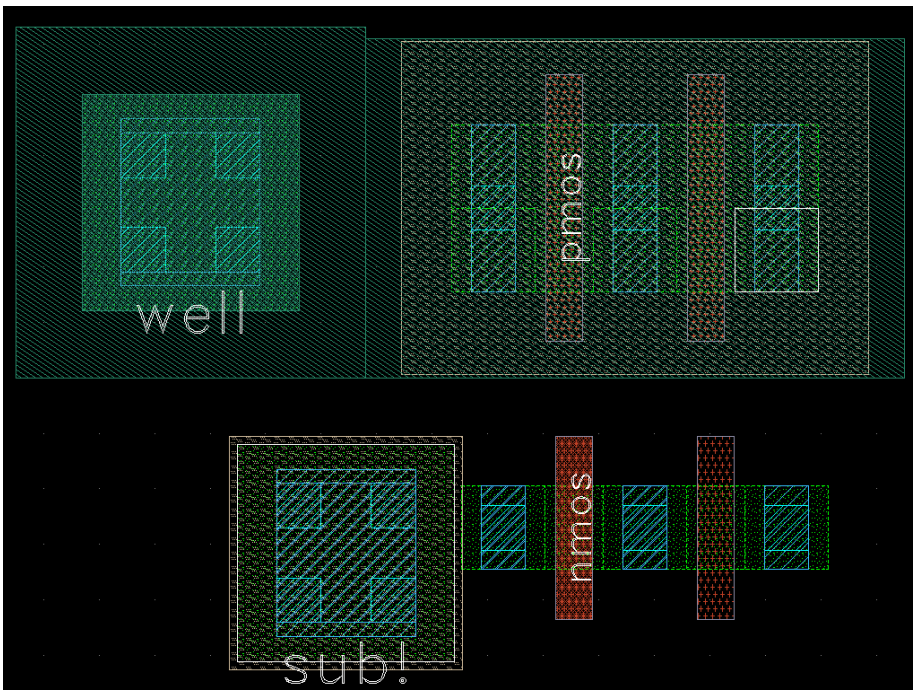
1. Wstawienie tranzystora NMOS poprzez wykorzystanie komórki parametryzowanej (PCELL). 1 – wybór narzędzia **Create a cell instance**, 2,3,4 – wyszukanie i wybranie PCELL nmos, 6,7 – wpisanie pożądanych parametrów komórki (jak na schemacie) i aktualizacja, 8 – wskazanie myszką miejsca i obszarze roboczym i wstawienie komórki, 8 a widok bez ustawienia rozszerzenia widocznej hierarchii, 8 b widok z rozszerzeniem widoczności. Na zakończenie wciskamy klawisz ESC co powoduje do powrotu do ustawienia narzędzia **Select**.



Rys. 17. Niezbędne kroki do wstawienia PCELL tranzystora NMOS.

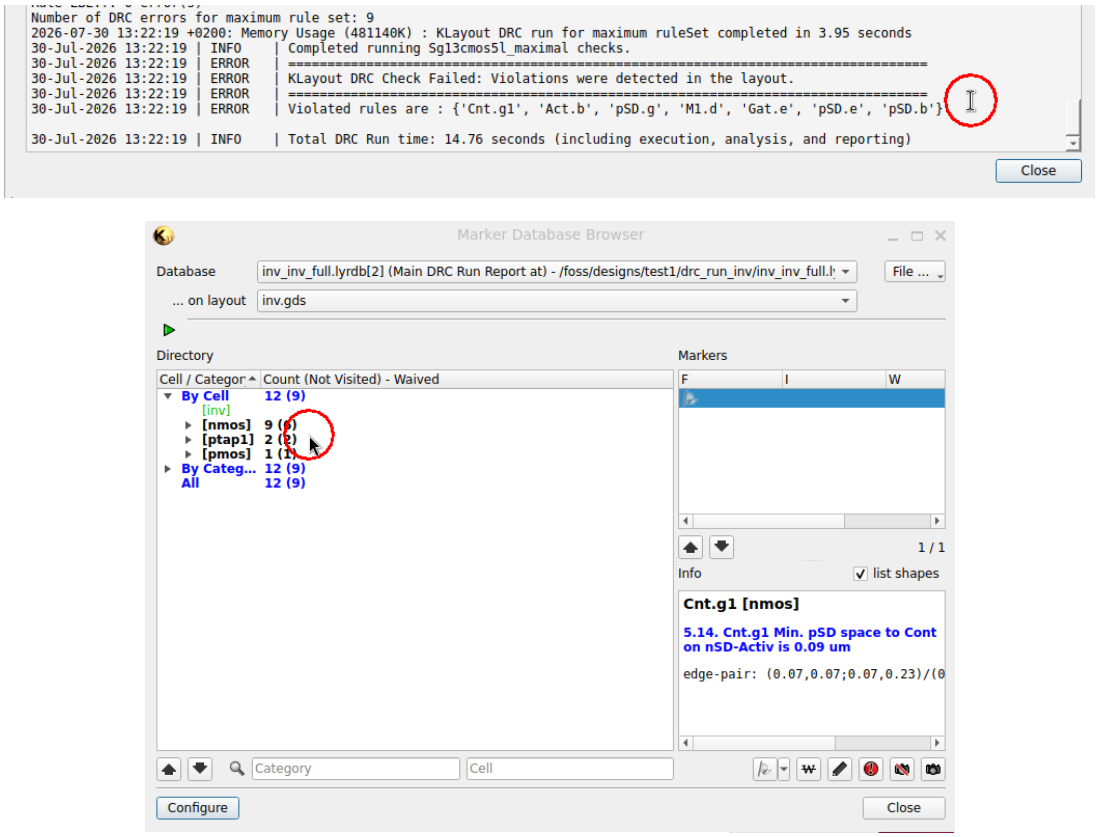
2. Wstępne wstawienie pozostałych komórek PCELL występujących na schemacie inwertera, tj.: tranzystora PMOS, kontaktów do wyspy i podłoża NTAP1 i PTAP1. Po tym kroku obszar roboczy może wyglądać jak poniżej. Parametry poszczególnych komórek można zmieniać na bieżąco poprzez podwójne kliknięcie w PCELL już wstawioną w obszarze roboczym i poprawienie parametrów w

formularzu. W tym miejscu warto wykonać test DRC w celu znalezienia ewentualnych zbyt bliskich umieszczeń elementów NMOS i PMOS pomiędzy sobą. Usunięcie takich błędów na tym etapie może przyspieszyć poprawki DRC na zakończonym projekcie. Aby to wykonać wybieramy menu: ***SG13CMOS5L PDK/Run KLayout DRC*** i czekamy na zakończenie procesu. Pojawią się 2 okna, jak to przedstawiono na rys. 19. Okno logu można zamknąć, natomiast okno przeglądarki błędów warto umieścić obok głównego okna programu.



Rys. 18. Widok obszaru roboczgo po wstępnym umieszczeniu wszystkich komórek PCELL.

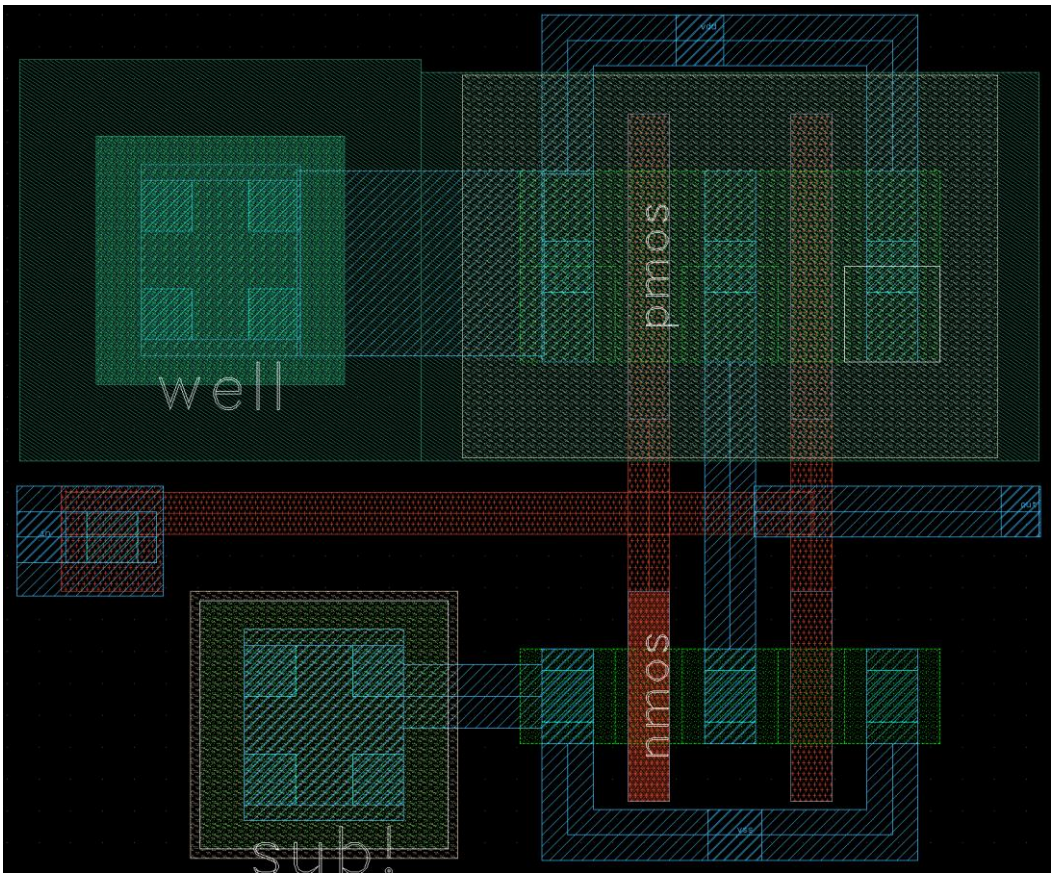
W oknie przeglądarki wyników, linie ze strzałą można rozwinąć a wewnątrz widoczne są pogrupowane kategorie reguł projektowych. Linie w kolorze czarnym oznaczają złamane reguły, ich opis jest widoczny po prawej stronie u dołu okna. Kliknięcie na regułę podświetla w obszarze roboczym miejsca gdzie niespełniona reguła występuje oraz jakie liczbowe parametry są niezbędne aby reguła była zachowana. Na tym etapie warto zwrócić uwagę wyłącznie na reguły dotyczące zbyt małych odległości i w przypadku ich występowania porozsuwać elementy je powodujące. Pełne sprawdzenie reguł będzie wykonane w późniejszym etapie projektu.



Rys. 19. Wstępne wyniki DRC w celu odnalezienia ewentualnych zbyt bliskich odległości pomiędzy poszczególnymi komórkami PCELL. Okno logów (u góry) oraz okno przeglądarki błędów (na dole). Linie w kolorze czarnym zawierają błędy DRC. Po rozwinięciu można po rawej stronie otrzymać opis błędu i dokonać zmian w topografii w celu jego usunięcia.

3. Dokładne spozycjonowanie komórek i wykonanie połączeń elektrycznych. Następnym krokiem jest takie spozycjonowanie komórek między sobą aby wygodnie można było dokonać połączeń. Np. patrząc na rys. 18 warto przesunąć tranzystory tak aby ich bramki (kolor czerwony, warstwa ***GatPoly.drawing***) były dokładnie nad sobą, bez żadnych przesunięć. W przypadku gdy kursor skacze ze zbyt dużym krokiem można zmienić grid poleceniem menu ***View/Grid***. Do połączeń należy wykorzystywać warstwy z nazwą kończącą się ***.drawing***, np. ***Metal1.drawing*** (połączenia na warstwie Metal1), ***GatPoly.drawing*** (ścieżki polisilikonowe i bramki tranzystorów), ***Cont.drawing*** (kontakty do polisilikonu i dyfuzji), itd. Opisy poszczególnych warstw dla technologii IHP SG13G2CMOS5L można znaleźć w dokumentacji [1,2,3]. Aby umieścić dany kształt najpierw wybieramy warstwę (np. ***Metal1.drawing***), następnie narzędzie (***Polygon***, ***Box*** lub ***Path***) a na koniec w obszarze roboczym przy użyciu kursora umieszczamy warstwę w pożądanym miejscu i kształcie. Do połączeń między metalami można użyć PCELL o nazwie ***via_stack*** co znacząco przyspiesza ich rysowanie.

4. Oznaczenie miejsc traktowanych jako połączenia (PINy) oraz nadanie nazw węzłem wewnętrznym (opcjonalne). W topografii nadaje się oznaczenia miejscom, które w późniejszym etapie będą traktowane jako wyprowadzenia danej komórki. W tym celu wykorzystuje się warstwę z rozszerzeniem ***.pin***, np. ***Metal1.pin***. Następnie, tym razem na warstwie ***.text*** należy nanieść opis tekstowy (przycisk ***Text*** z górnego menu). Naniesienie tekstu na warstwie ***.drawing*** (bez ***.text***) nadaje nazwę wewnętrzną sieci elektrycznej w danej komórce. Topografia po tym etapie może wyglądać jak na rys. 20.



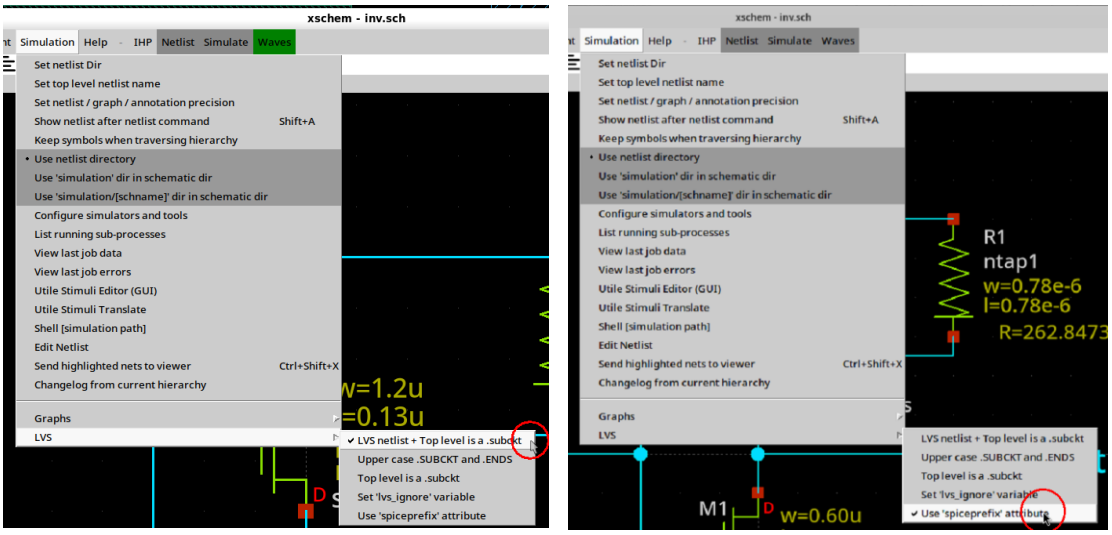
Rys. 20. Topografia inwertera z naniesionymi PINami.

Wykonanie sprawdzenia poprawności reguł projektowych (DRC).

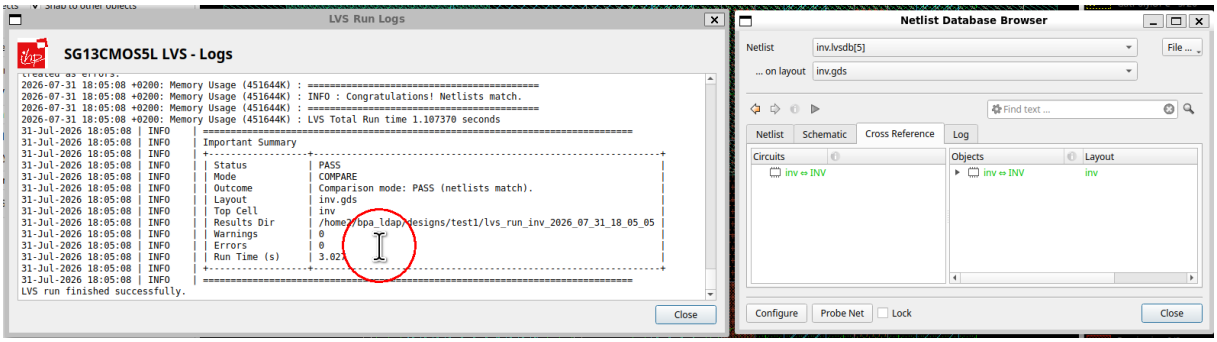
W tym momencie powinniśmy sprawdzić poprawność reguł technologicznych i w przypadku wystąpienia błędów usunąć wszystkie poprzez przeorganizowanie topografii (zazwyczaj zwiększenie odległości i szerokości obszarów). W tym celu, podobnie ja w kroku 2, wykonujemy menu ***SG13CMOS5L PDK/Run KLayout DRC*** i tym razem musimy wyczyścić wszystkie błędy.

Wykonanie porównania schematu elektrycznego z topografią.

W tym celu najpierw musimy przygotować netlistę zgodną z wymaganiami programu KLayout. Niestety jest ona niezgodna z symulatorem ngspice, stąd musimy wygenerować odpowiednią wersję. W tym celu w programie XSCHEM otwieramy schemat inwertera (inv.sch), następnie wybieramy menu ***Simulation/LVS*** i tak ustawiamy opcje aby była aktywna wyłącznie pierwsza linia jak na poniższym rysunku po stronie lewej. Następnie wybieramy przycisk ***Netlist*** na górnym pasku programu i w katalogu **simulations** powinien powstać plik netlisty o nazwie **inv.spice**. Elementy w tym pliku rozpoczynają się literami M, R a nie X co jest wyznacznikiem poprawnej zmiany opcji nietlisty. W tej chwili porównana będzie netlista z programu XSCHEM z topografią w KLayout. W tym celu wybieramy menu (w programie KLayout) ***SG13CMOS5L PDK/SG13CMOS5L LVS Options*** i w polu ***Netlist Path*** wskazujemy wygenerowaną wcześniej netlistę w programie XSCHEM. Następnie wybieramy menu ***SG13CMOS5L PDK/Run KLayout LVS*** i rozpoczyna się proces porównania. Po jego zakończeniu otrzymujemy raport oraz otwiera się przeglądarka wyników, identyczna jak poprzednio po procesie DRC, rys. 22.



Rys. 21. Ustawienie opcji netlisty dla procesu LVS (po lewej) i dla symulacj ngspice (po pawej).



Rys. 22. Log procesu LVS oraz przeglądarka ewentualnych błędów.

Wykonanie ekstrakcji topografii (ang. PEX)

Ekstrakcja topografii wykonywana jest poprzez uruchomienie w tle programu „magic" przy użyciu skryptu autorstwa Haralda Pretla z Johannes Kepler University, Department for Integrated Circuits. To samo można wykonać ręcznie uruchamiając program magic, następnie wczytując topografię i wydając odpowiednie polecenia. Podobnie, konkurencyjny pakiet **kpex** wywołuje w tle program magic a następnie wykonuje ekstrakcję schematu z topografii. Tu przedstawione zostanie wykorzystanie skryptu **sak – pex.sh**. W tym celu należy wydać polecenie w terminalu Linux:

```
sak-pex.sh -m 2 inv.gds
```

Po ekstrakcji topografii otrzymujemy plik z końcówką ***.pex.spice**, poniżej przykładowa jego zawartość:

```
* PEX produced on Fri Jul 31 06:06:35 PM CEST 2026 using
/techfiles_ldap/openpdk/tools/sak-pex.sh with m=2 and s=1

* NGSPICE file created from inv.ext - technology: ihp-sg13cmos5l

.subckt inv out in vdd vss

X0 vss in out vss sg13_lv_nmos ad=0.102p pd=1.28u as=57f ps=0.68u w=0.3u l=0.13u

X1 out in vss vss sg13_lv_nmos ad=57f pd=0.68u as=0.102p ps=1.28u w=0.3u l=0.13u

X2 vdd in out vdd sg13_lv_pmos ad=0.204p pd=1.88u as=0.114p ps=0.98u w=0.6u l=0.13u

X3 out in vdd vdd sg13_lv_pmos ad=0.114p pd=0.98u as=0.204p ps=1.88u w=0.6u l=0.13u
```

```
C0 vdd in 0.18434f
C1 out vdd 0.23575f
C2 out in 0.05761f
C3 out vss 0.43801f
C4 vdd vss 0.22646f
.ends
```

Symulacje układu inwertera po ekstrakcji topografii

Jak widać w wyżej przywołanej zawartości pliku **inv.pex.spice** powstałego po ekstrakcji topografii, zawarta w nim jest definicja podukładu o nazwie **inv** i 4 wyprowadzeniach. Wyprowadzenia umieszczone są w przypadkowej kolejności, innej niż w definicji schematu **inv.sch**. Aby wykonać symulację z użyciem pliku PEX, należy w definicji symbolu dodać informację aby był użyty opis netlisty zamiast schematu (przy symulacji ze schematu jest on też wcześniej zamieniany na netlistę). W tym celu w programie XSCHEM modyfikujemy właściwości wstawianego bloku (zaznaczenie bloku i klawisz Q). Nowa zawartość powinna w być jak poniżej (# na początku linii oznacza komentarz):

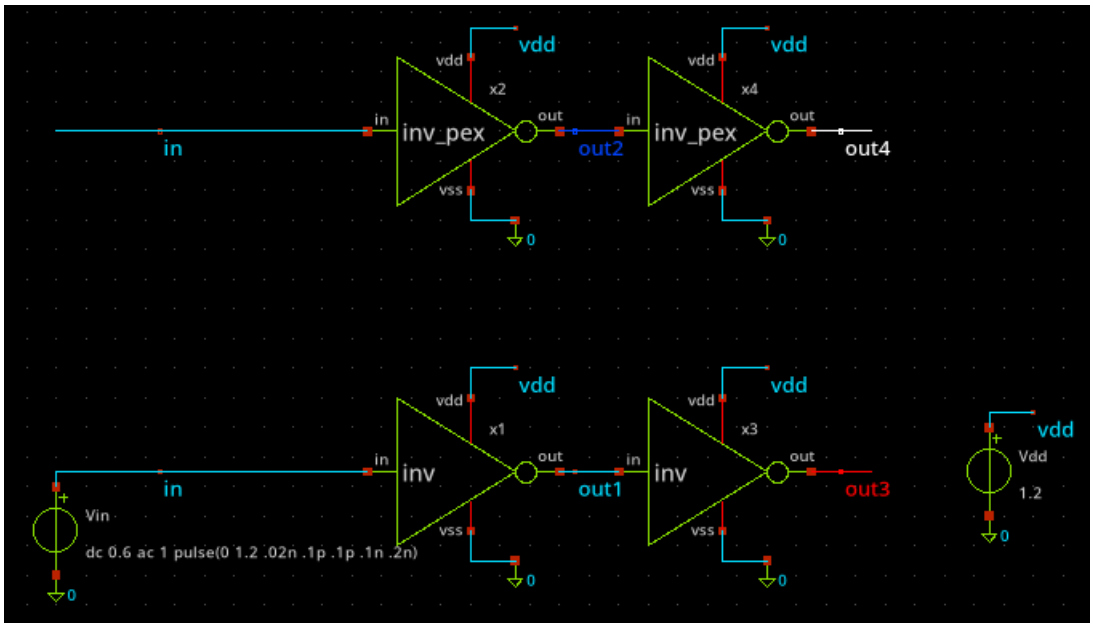
```
name=x2
# poniższa linia zamienia nazwę używanego podukładu z
# domyślnej "inv" na "inv_pex"
schematic=inv_pex
# poniższe polecenie zamienia domyślne poszukiwanie schematu
# na wczytanie gotowej netlisty z pliku, trzeba oczywiście
# zamienić /katalog_projektu/ na własną ścieżkę, która jest
# indywidualna dla każdego użytkownika
spice_sym_def=".include /katalog_projektu/inv.pex.spice"
# poniższe polecenie umożliwia wczytanie pliku w XSCHEM
# przez wybranie symbolu i wciśnięcie Ctrl-H
tclcommand="textwindow /katalog_projektu/inv.pex.spice"/inv.pex.spice"
```

Jak widać powyżej, zmieniono nazwę podukładu z **inv** na **inv_pex**. Zrobiono to celowo aby umożliwić równoczesną symulację inwertera ze schematu i inwertera po ekstrakcji w celu porównania wartości sygnałów w nich występujących. Dodatkowo takie wstawienie bloku jak wyżej przedstawiono, powoduje automatyczną poprawę kolejności wyprowadzeń w podukładach. Schemat testbencha po dodaniu 2 inwerterów wygląda jak poniżej na rys. 23. Aby symulacja powiodła się należy jeszcze w pliku **inv.pex.spice** zmienić nazwę podbloku na identyczną jak nadana powyżej, tj. zamienić linię:

```
.subckt inv out in vdd vss
```

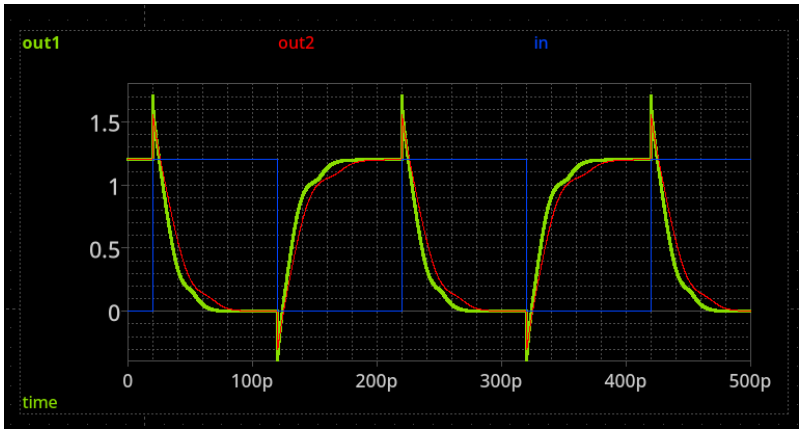
na następującą:

```
.subckt inv_pex out in vdd vss
```



Rys. 23. Wstawione 2 inwertery (powielone ze schematu niżej). Należy zwrócić uwagę na nazwy sieci gdyż sieci o tych samych nazwach traktowane są jako zwarte ze sobą. Inwertey x2 oraz x4 mają wpisane właściwości jak w kodzie przedstawionym powyżej. Symbol wstawienia (x1,x2,x3,…) musi być dla każdego inwertera inny.

Po wykonaniu symulacji i dodaniu sygnałów na wykresie, symulacja TRAN da wyniki jak przedstawiono na rys. 24. Przebiegi na wyjściu układu po ekstrakcji są niemal takie same jak przed ekstrakcją z tą różnicą, że ze względu na dodane pojemności pasożytnicze nastąpiło lekkie opóźnienie odpowiedzi inwertera. Dla symulacji DC nie powinno być żadnych różnic (bo brak ekstrakcji rezystancji) a dla AC powinniśmy uzyskać nieco niższe pasmo 3dB. Niniejszy punkt kończy opis podstawowych czynności niezbędnych i wystarczających do wykonania projektu w technice FULL-CUSTOM. W kolejnych rozdziałach opisano szereg udogodnień, które można i należy wykorzystywać w trakcie projektowania układów scalonych gdyż bardzo usprawniają proces projektowy.



Rys. 24. Symulacja TRAN kaskady inwerterów przed (out1) i po ekstrakcji topografii (out2).

Podukłady i symulacje parametryzowane, dziedziczenie połączeń

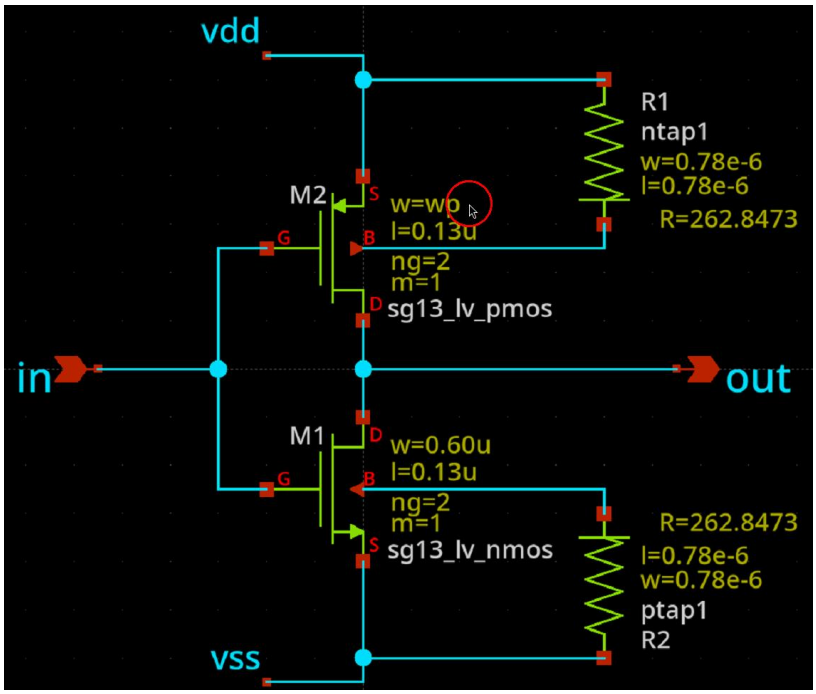
W przypadku powtarzających się wyprowadzeń (np. zasilanie i masa) często okazuje się bardzo korzystne wykorzystanie połączeń dziedziczonych, które nie są widoczne na schemacie i upraszczają projektowanie dla wielu powtarzających się elementów, np. dla wielokrotnie wstawianych bramek logicznych. Dodatkowo w czasie projektowania, gdy dobierane są wymiary tranzystorów w celu

uzyskania zakładanych wyników, łatwiej jest wykorzystać sparametryzowanie i przemiatanie danego wymiaru (czy też innej właściwości elementu) i wykonanie szeregu symulacji w zautomatyzowanej serii. Poniżej przedstawione zostanie przerobienie inwertera na wersję z dziedziczonymi połączeniami zasilania oraz z sparametryzowanym wymiarem szerokości tranzystora PMOS. W tym celu w programie XSCHEM wczytujemy schemat inwertera (**inv.sch**) a następnie zapisujemy go pod nazwą **inv_par_inh.sch**. Na schemacie zmieniamy następujące elementy:

a) tranzystor PMOS: zmieniamy oryginalny wymiar na zmienną **wp**

b) usuwamy PINy zasilające a w ich miejsce nadajemy nazwy **vdd** oraz **vss** węzłom zasilającym

Schemat po modyfikacji przedstawiony jest na rys. 25.



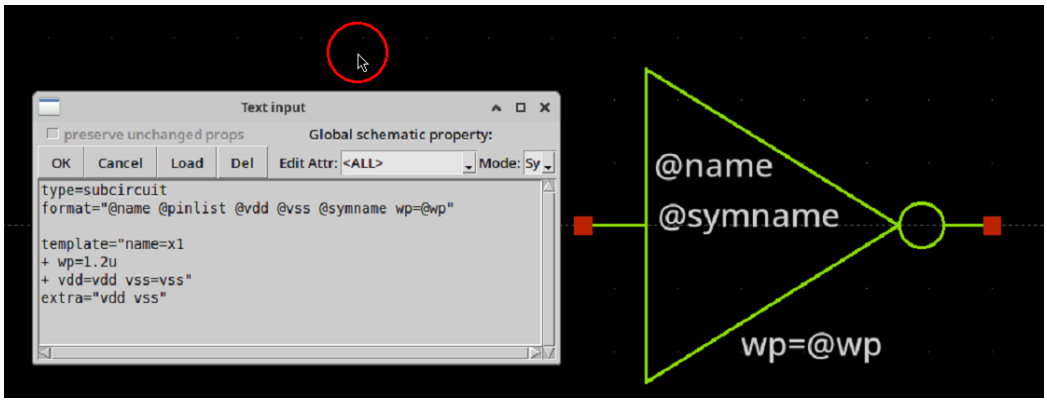
Rys. 25. Schemat inwertera po usunięciu PINów zasilających i nadaniu parametrowi szerokości tranzystora PMOS wartości zmiennej „wp”.

Kolejną czynnością jest dostosowanie symbolu inwertera do nowej, hierarchicznej i sparametryzowanej formy. W tym celu wczytujemy uprzednio utworzony symbol inwertera **inv.sym** i zapisujemy go jako **inv_par_inh.sym**. Następnie w tym symbolu zmieniamy następujące składniki:

a) usuwamy PINy **vdd** i **vss** oraz linie je łączące
b) edytujemy wysokość napisu **@name** (oznacza nazwę wstawienia) i **@symnane** (oznacza nazwę wstawianego podukładu) na 0.2 oraz przesuwamy je do środka symbolu
c) dodajemy tekst (klawisz T) i wprowadzamy zawartość: wp=@wp, to spowoduje, że na schemacie z powołanym symbolem pojawi się napis z faktycznie stosowaną wartością wskazanej zmiennej
d) usuwamy napisy **IN** oraz **OUT** (są one wyłącznie dekoracją)
e) modyfikujemy właściwości ogólne symbolu, klikamy myszką w puste miejsce i wciskamy klawisz Q, otworzy się okno właściwości, uzupełniamy zawartością jak poniżej podana, tu w kodzie został dopisany opis poszczególnych części wstawianego kodu, nie jest to konieczne w opisie podukładu.

```
type=subcircuit
# ta część informuje jaka będzie kolejność wstawienia przy
# generacji netlisty
format="@name @pinlist @vdd @vss @symname wp=@wp"
# ta część informuje jak domyślnie będzie wstawiany dany symbol
# w schemacie
template="name=x1
+ wp=1.2u
+ vdd=vdd vss=vss"
# poniższa linie informuje które węzły są dodane jako opcjonalne
# dodatkowe względem PINów na schemacie
extra="vdd vss"
```

Schemat po modyfikacji wraz o otwartym oknem dialogowym modyfikacji właściwości przedstawione są na rys. 26.



Rys. 26. Modyfikacja symbolu inwertera oraz jego właściwości (klawisz Q na wolnym polu).

Mając zmodyfikowany i schemat i symbol naszego inwertera wstawiamy go do schematu testbencha. Dla przyspieszenia pracy zapisujemy poprzednio przygotowany plik testbencha **`inv_tb.sch`** jako **`inv_par_nh_tb.sch`** i modyfikujemy go poprzez:

a) usuwamy wszystkie inwertery ze schematu, pozostawiamy źródło wejściowe Vin i zasilające Vdd

b) dodajemy sekcję kodu poprzez wstawienie (klawisz Insert) z biblioteki **`xschem/library_devices`** elementu **`simulator_commands_shown.sym`** oraz zmodyfikowanie jego zawartości do postaci:

```
name=Parameters

simulator=ngspice

only_toplevel=false

value="

.param param_wp1=1.2u

+ param_wp2=0.6u

"
```

c) powyższy kod definiuje 2 parametry: **param_wp1** oraz **param_wp2**, parametry te zostaną użyte do zdefiniowania właściwości wstawianych podbloków

d) wstawiamy 2 inwertery: **inv_par_inh.s**, modyfikujemy ich właściwości (wybieramy dany inwerter i wciskamy klawisz Q) , parametry modyfikujemy jak poniżej:

inwerter x1

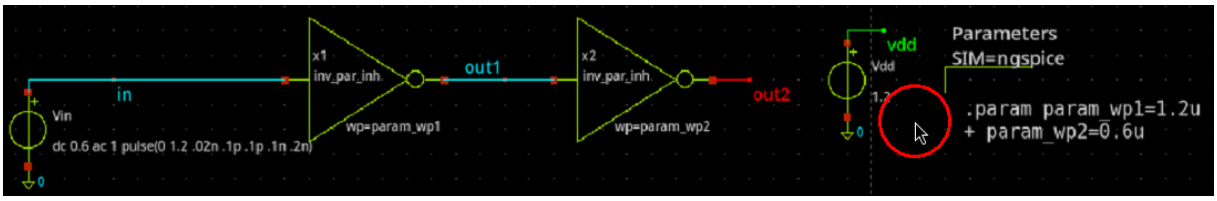
```
name=x1
+ wp=param_wp1
+ vdd=vdd vss=0
```

inwerter x2

```
name=x2
+ wp=param_wp2
+ vdd=vdd vss=0
```

e) teraz możemy wykonać generację netlisty i symulację, należy zauważyć że w netliście ujemne zasilanie **vss** zostało podłączone do **0** a parametry **wp** dla inwerterów zostały ustawienie odpowiednio na **param_wp1** oraz **param_wp2**

Schemat fragmentu zmodyfikowanego testbencha wraz definicją parametrów przedstawione są na rys. 27.



Rys. 27. Schemat testbencha ze wstawionymi inwerterami z zasilaniem połączonym przez hierarchię dziedziczenia oraz zmodyfikowanymi parametrami szerokości tranzystorów PMOS.

Wykorzystanie dziedziczonego połączenia zasilania upraszcza schemat, należy jednak pamiętać o ich prawidłowym zdefiniowaniu we właściwościach każdego wstawianego komponentu. Użycie parametryzowanych właściwości tranzystora umożliwia wstawienie komponentów z różnymi ich wartościami. Sparametryzowanie daje jednak największą korzyść poprzez umożliwienie łatwego doboru parametrów w celu uzyskania założonych właściwości projektowanego układu. Poniżej zamieszczony jest zmodyfikowany kod symulacji DC ze zmianą parametru **wp**. Z całego zestawu przemiatanych wartości interesuje nas taki dobór szerokości aby uzyskać V(out1)=0.6V dla Vin=0.6V. Parametr **wp** zmieniany jest od 0.8u do 1.6u z krokiem 0.2u. Niestety symulator ngspice nie ma wbudowanego typowego dla składni SPICE polecenia **.step**, można jednak za pomocą sekcji **.control** uzyskać podobny rodzaj symulacji. Na rys. 28 umieszczono listing kodu, który symuluje polecenie **.step**. Jak widać z symulacji najbliżej założonego napięcia wyjściowego jest dla wp=0.8u. Na kolejnym rysunku umieszczono listing dla pętli z wykorzystaniem zmiennych i wektorów. Polecenie **reset** niestety usuwa wszystkie wcześniejsze deklaracje wektorów (deklarowanych poprzez **let**) i stąd niezbędne jest wykorzystanie zmiennej (deklaracja poprzez polecenie **set**). Niestety do zmiennych nie można wykonywać przypisywać bezpośrednio wyników operacji matematycznych stąd pętla jest kontrolowana pośrednio przez wektor **var1** i jego zamianę na zmienną również o nazwie **var1**. **var1** jest w części kodu zmienną a w części kodu wektorem!

```
name=Simulator2
simulator=ngspice
only_toplevel=false
value="
.param temp=27
.param param_wp = 1u
.control
save all
op
write op.raw
tran .1p .5n
meas tran prop_L TRIG v(in) val=0.6
+ rise=2 targ v(out1) val=0.6 fall=2
meas tran prop_H TRIG v(in) val=0.6
+ fall=2 targ v(out1) val=0.6 rise=2
write tran_inv.raw
ac dec 100 100k 100g
write ac_inv.raw
foreach wp_tmp 0.8u 1u 1.2u 1.4u 1.6u
   echo wp_tmp = $wp_tmp
   alterparam param_wp = $wp_tmp
   reset
   dc Vin 0 1.2 0.01 ;Vdd 1.1 1.3 0.1
   write dc_inv.raw
   set appendwrite
end
plot dc1.v(out1) dc2.v(out1) dc3.v(out1) dc4.v(out1) dc5.v(out1)
.endc
"
```

Rys. 28. Modyfikacja kodu sterującego symulacją DC skutkująca przemiatniem parametru wp w zakresie od 0.8u do 1.6u.

```
name=Simulator2
simulator=ngspice
only_toplevel=false
value="
.param temp=27
.param param_wp = 1u
.control
save all
op
write op.raw
tran .1p .5n
meas tran prop_L TRIG v(in) val=0.6
+ rise=2 targ v(out1) val=0.6 fall=2
meas tran prop_H TRIG v(in) val=0.6
+ fall=2 targ v(out1) val=0.6 rise=2
write tran_inv.raw
ac dec 100 100k 100g
write ac_inv.raw
set var1 = 0.8u
echo var1 =$var1
while $var1 < 2u
   alterparam param_wp = $var1
   reset
   dc Vin 0 1.2 0.01 ;Vdd 1.1 1.3 0.1
   write dc_inv.raw
   set appendwrite
```

```
    let var1 = $var1 + 0.2u
    set var1 = "$&var1"
    echo var1(in loop) = $var1
end
echo var1 =$var1
.endc
"
```

Rys. 29. Manipulacje na zmiennych. Do operacji matematycznych potrzebny jest wektor (deklaracja poleceniem let) natomiast aby nie został on wyzerowany poleceniem reset nezbędna jest zmienna (deklaracja poleceniem set ale nie można na niej robić operacji matematycznych). W powyższym kodzie operacje są na wektorach i są przypisywane później do zmiennych.

Adnotacja punktu pracy na schemacie, operacje na wynikach obliczeń oraz polecenie measure symulatora ngspice

Symulator **ngspice** jest w ok 90% zgodny ze składnią SPICE znaną z symulatorów komercyjnych. Zgodne jest tworzenie listy połączeniowej oraz tworzenie i umieszczanie podukładów. W poleceniach zlecania i sterowania przebiegiem symulacji są dość istotne różnice. Poniżej wymieniono kilka najbardziej znaczących:

1) Polecenia rozpoczynają się od znaku kropki (jak w SPICE) jednak jeśli są umieszczone w bloku **.control** wówczas kropka nie występuje. Blok sterujący rozpoczynamy linią zawiewającą kod **.control** i kończymy linią **.endc**. Każda linia pomiędzy powyższymi zawiera jedno polecenie i nie potrzebne są kropki na początku linii.

2) Aby polecenia poza blokiem **control** się wykonały musi być polecenie **run** lub symulator musi być uruchomiony z opcją **-a** lub też pracując w trybie interaktywnym należy manualnie wprowadzić polecenie **run**.

3) W ngspice nie ma polecenia **.step**, które umożliwia analizę parametryczną. Są natomiast metody obejścia tego problemu. Odpowiedni przykład został przedstawiony we wcześniejszym rozdziale.

4) Są różne polecenia do zmiany różnych składników kodu sterującego. I tak aby zmienić wartość parametru (element deklarowany poleceniem **.param**) używamy polecenie **alterparam** jednak nowa wartość jest używana dopiero po poleceniu **reset**. Polecenie **let** deklaruje i nadaje wartość wektorowi. Zdeklarowany wektor jest widoczny od razu. Polecenie **set** deklaruje i ustawia wartość zmiennej. Zmienne ani wektory nie są usuwane po poleceniu **reset**, polecenie to ładuje nową netlistę ze zaktualizowanymi parametrami. Aby uzyskać wartość wektora lub zmiennej należy stosować znaki wyłuskania wartości **$** lub **$&** odpowiednio dla wektora i zmiennej. Dobry przykład jest przedstawiony w poprzednim rozdziale. Do debugowania kodu NGSPICE warto używać polecenia **print nazwa_zmieninnej** lub **print nazwa_wektora**, ewentualnie zamiennika **echo $nazwa_zmieninnej** lub **echo $&nazwa_wektora**. Po wykonaniu symulacji może się wydawać, że wektory zniknęły – nie jest to jednak prawda gdyż one są, tyle że wykonanie symulacji automatycznie powołuje i uaktywnia nowy **setplot** a wektory mogą należeć do poprzedniego. Wektory zdeklarowane przed wykonaniem jakiejkolwiek symulacji są widoczne zawsze, wektory zdeklarowane po symulacji należą tylko do **setplot**u tej symulacji!

5) Standardowo do zapisu wyników używamy polecenia **write nazwa_pliku**, co automatycznie uruchamia go w trybie tworzenia nowego pliku co w konsekwencji kasuje wyniki już tam umieszczone jeśli taki plik istniał. Aby dopisywać wyniki symulacji do pliku już istniejącego należy wydać polecenie **set appendwrite**, w przeciwnym wypadku poprzednie dane zostaną usunięte. Usuniecie dopisywania (czyli włączenie nadpisywania) poprzez polecenie **unset appendwrite** .

W przypadku gdy podamy polecenie obliczenia punktu pracy **op** oraz zapiszemy wyniki do pliku, np.: **write op.raw** możemy to wykorzystać do adnotacji napięć na schemacie w programie XSCHEM. W tym celu wybieramy przycisk ***Waves*** i następnie ***OP Annotate*** a następnie wybieramy zapisany wcześniej plik zawierający punkt pracy. Na schemacie pojawią się wartości napięć węzłowych. Napięcia te będą też widoczne w podukładach. Można też użyć opcji z menu ***Simulation/Graphs/…*** aby adnotacje napięć pojawiały się automatycznie po każdym nowym przebiegu symulacji i wyświetleniu kursora B na wykresie.

Symulator ngspice umożliwia realizację szeregu operacji arytmetycznych na wynikach przeprowadzonych obliczeń. Obliczenia te można umieścić na wykresach lub wyprowadzić w postaci liczbowej. Szczegóły można znaleźć w instrukcji programu [6], tutaj na kilku przykładach przedstawione zostanie wykorzystanie kilku często używanych odczytów. Polecenie **.meas** umieszcza wyniki w pliku tekstowym konsoli symulatora natomiast operacje matematyczne na przebiegach mogą być wykreślone na wykresach. Przykłady:

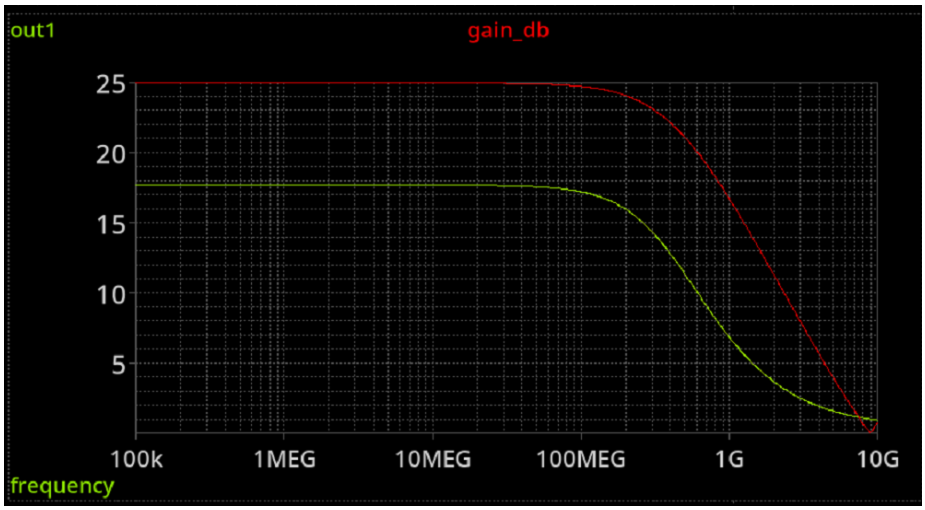
1) Symulacja AC, znalezienie maksymalnego wzmocnienia i pasma 3dB:

```
*** AC simulation
reset
ac dec 100 100k 10g
let Gain = mag(v(out1))
let Gain_dB = db(Gain)
let Max_Gain_dB = vecmax(Gain_dB)
meas ac f_3dB_low when Gain_dB = (Max_Gain_dB-3)  fall=1
print Max_Gain_dB f_3dB_low
write ac_inv.raw
```

Po wykonaniu symulacji otrzymujemy w terminalu programu ngspice wygenerowane wyniki np. jak poniżej:

```
max_gain_db = 2.495e+01
f_3db_low = 4.172e+08
```

Dodatkową zaletą kodu powyżej jest również to że wektor Gain_dB będzie zapisany w pliku wyników i można go umieścić na wykresie jak pokazano poniżej na rys. 30.



Rys. 30. Wzmocnienie inwertera wykreślone w dB po analizie wg kodu z przykładu 1).

2) Symulacja TRAN, znalezienie czasów propagacji, narastania i opadania:
Kod:

```
*** Transient simulation
reset
tran .1p .5n
meas tran prop_L TRIG v(in) val=0.6
+ rise=2 targ v(out1) val=0.6 fall=2
meas tran prop_H TRIG v(in) val=0.6
+ fall=2 targ v(out1) val=0.6 rise=2
meas tran rise_time TRIG v(out1)
+ VAL=0.12 RISE=2 TARG v(out1) VAL=1.08 RISE=2
meas tran fall_time TRIG v(out1)
+ VAL=1.08 FALL=2 TARG v(out1) VAL=0.12 FALL=2
write tran_inv.raw
```

Po wykonaniu symulacji otrzymujemy w terminalu programu ngspice wygenerowane wyniki np. jak poniżej:

```
No. of Data Rows : 5038
prop_l           =  1.15238e-11 targ=  2.31574e-10 trig=  2.20050e-10
prop_h           =  1.51775e-11 targ=  3.35328e-10 trig=  3.20150e-10
rise_time        =  3.30117e-11 targ=  3.60063e-10 trig=  3.27051e-10
fall_time        =  2.05061e-11 targ=  2.46068e-10 trig=  2.25562e-10
binary raw file "tran_inv.raw"
```

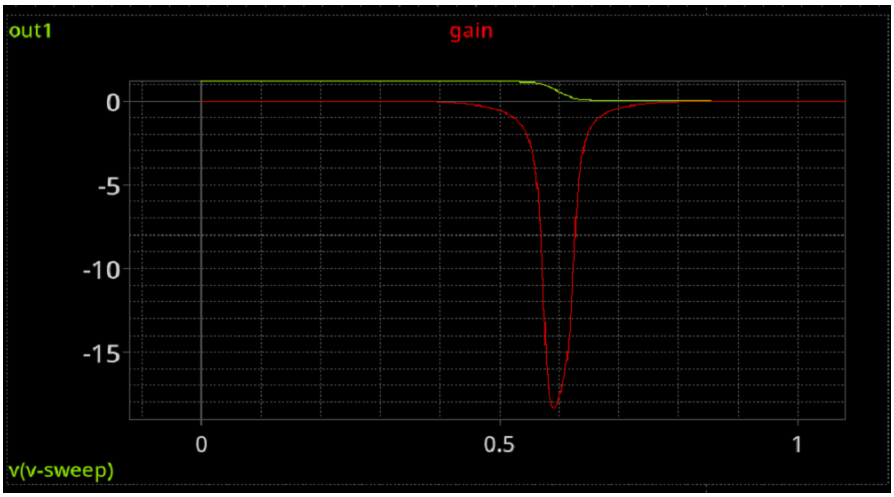
3) Symulacja DC, znalezienie wzmocnienia jako pochodnej sygnału wyjściowego:
Kod:

```
reset
dc Vin 0 1.2 0.001
let Gain = deriv(v(out1))
let Min_Gain = vecmin(Gain)
print Min_Gain
meas dc vin_min_gain min_at Gain
print vin_min_gain
write dc_inv.raw
```

Wynik obliczenia, w rzeczywistości jest to ujemne wzmocnienie gdyż inwerter zmienia fazę sygnału

```
min_gain = -1.83592e+01
vin_min_gain        =  5.91000e-01 with=  -1.83592e+01
vin_min_gain = 5.910000e-01
```

Dodatkową zaletą kodu powyżej jest również to że wektor Gain będzie zapisany w pliku wyników i można go umieścić na wykresie jak pokazano poniżej na rys. 31.

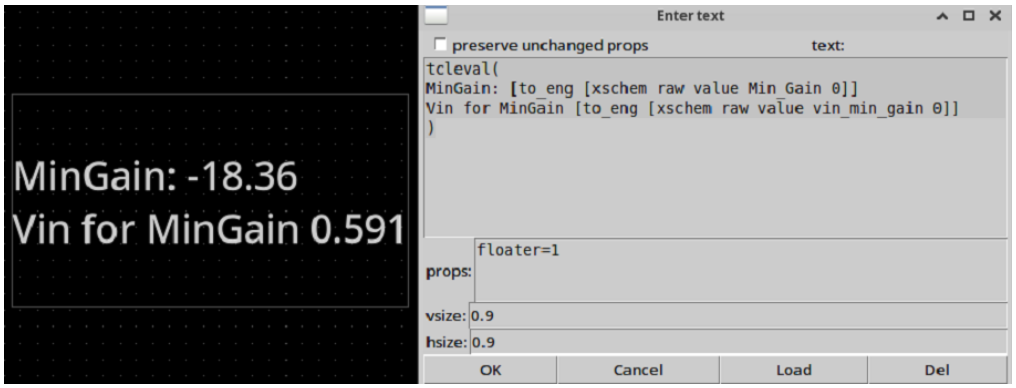


Rys. 31. Wzmocnienie inwertera jako pochodna napięcia wyjściowego out1 po analizie wg kodu z przykładu 3).

Wartości wyliczonych parametrów zapisane w plikach wyników .raw można wczytywać (podobnie jak dane na wykresy) w postaci tekstu uwidacznianego bezpośrednio na schemacie. Aby to zrobić należy umieścić odpowiednio sformatowany tekst na schemacie (umieszczenie tekstu menu: ***Tools/Insert text*** lub klawisz ***T***). Poniższy kod odpowiada symulacji 3) powyżej, wczytanie danych umieści zarówno wyniki na wykresie jak i w tekście. Kod tekstu:

```
tcleval(
MinGain: [to_eng [xschem raw value Min_Gain 0]]
Vin for MinGain [to_eng [xschem raw value vin_min_gain 0]]
)
```

We właściwościach (pole ***props***) należy dopisać ciąg: **floater=1**. Taki tekst będzie aktualizowany po każdym wczytaniu pliku wyników, w przykładzie 3) powyżej jest to plik **dc_inv.raw**.



Rys. 32. Tekst i jego pole props (na lewo) oraz tekst generowany na schemacie po wczytaniu danych zawierających wartości: Min_Gain i vin_min_gain.

Symulacja zniekształceń nieliniowych

Aby znaleźć wartość zniekształceń nieliniowych należy najpierw wykonać analizę czasową (**tran**) a następnie wynik tej analizy poddać analizie Fouriera (**fourier**). Czas końca symulacji powinien zapewnić stan ustalony na wyjściu i równocześni czas dłuższy niż okres odpowiadający częstotliwości

rozwinięcia Fouriera. Poniżej przedstawiono przykładowy fragment kodu sterującego realizujący taką operację, przyjęto 10-cio kronie dłuższy czas symulacji niż okres harmonicznego przebiegu wejściowego. Założono że sygnałem wejściowym jest źródło napięciowe o nazwie **Vin** a napięcie wyjściowe jest badane w węźle o nazwie **out** oraz częstotliwość sygnału wejściowego przyjęto jako równą 10kHz. Amplituda przyłożonego sygnału określona jest w definicji źródła Vin i w poniższym przykładzie wynosi 1V. Wyniki symulacji są drukowane w terminalu NGSPICE.

```
Vin in 0 dc 0 ac 1 sin(0 1 10k)
...
.control
tran 1u 1m
fourier 10k v(out)
.endc
```

Sporządzenie wykresu THD = f(Vin) wymagałoby wielu powtórzeń symulacji ze zmienianą wartością amplitudy sygnału wejściowego oraz ręcznego sporządzenia wykresu poza symulatorem NGSPICE. Program NGSPICE, dzięki części kontrolnej, umożliwia wykonanie w pętli powtórzeń symulacji i zebranie wyników wewnątrz NGSPICE. Poniższy kod umożliwia wykonanie tej czynności automatycznie. Dane o wartościach THD są zgromadzone w wektorze **thd_vec** a odpowiadające im amplitudy pobudzeń harmonicznych w wektorze **amp_vec**. Użytkownik powinien jedynie wypełnić fragment początkowy definiując zakres amplitud, częstotliwość pobudzenia oraz nazwy sygnału wejściowego i badanego napięcia na wyjściu.

```
.control
*** THD simulations and plot THD vs Amplitude
*** (C) Bogdan Pankiewicz
*** User definitions of amplitude range, frequency of sine wave, and Input and Output
let amp_start = 10m              ; starting amplitude of input signal
let amp_step  = 10m              ; amplitude increment
let steps = 10                   ; no of steps, final amplitude = amp_start + amp_step*steps
let frequency = 10k              ; put frequency of sine wave
set Input_source = Vin           ; put here name of used input voltage source, for ex. Vin
set Output_voltage = v(out)      ; put here name of observed output voltage, for ex. v(out)
*** End of user definitions

echo Input_source = $Input_source
echo Output_voltage = $Output_voltage
let amp_current = amp_start
let thd_vec = vector(steps)
let amp_vec = vector(steps)
let index = 0
let tran_end = 10/frequency
let tran_step = 0.001/frequency
let loop_no = 1

*** THD vs amplitude plot
unset appendwrite
echo THD analysis START
print tran_end
print tran_step
print steps
print frequency
repeat $&steps
   alter @{$Input_source}[sin] [ 0.4 $&amp_current $&frequency ]
   tran {$&tran_step} {$&tran_end}
   fourier {$&frequency} {$Output_voltage}
   write thd_test_tran.raw
   set appendwrite
   set THD = thd{$&loop_no}1
   let curr_thd = $&{$THD}
   echo Amplitude = $&amp_current
   echo THD = $&curr_thd
   let amp_vec[index] = amp_current
```

```
   let thd_vec[index] = curr_thd
   let amp_current = amp_current + amp_step
   let loop_no = loop_no + 1
   let index = index + 1
end
plot thd_vec vs amp_vec xlabel 'Input Sine Wave Amplitude' ylabel 'THD at Output [%]'
.endc
```

Symulacje Monte Carlo

Symulacje statystyczne rozrzutu parametrów elementów wynikające z dopasowania w procesie produkcji w komercyjnych symulatorach wykonuje się używając polecenia symulacji typu Monte Carlo (MC). W symulatorze ngspice nie ma odpowiednika tego polecenia a takie symulacje wykonuje się w nieco okrężny sposób. Najpierw definiuje się parametry, których wartości będą losowane zgodnie z góry ustaloną dystrybucją, następnie wykonuje się ustaloną ilość losowań i symulacji dla tych losowań. Dla technologii SG13CMOS5L możliwe jest losowanie parametrów tranzystorów. Przebieg konfiguracji symulacji dla tego przypadku jest następujący:

1) Używamy biblioteki i sekcji w bibliotece, która zawiera modele statystyczne:

   `.lib cornerMOSlv.lib mos_tt_mismatch`

   lub biblioteki

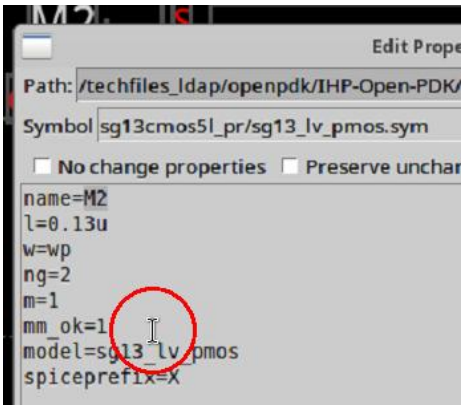
   `.lib cornerMOSlv.lib mos_tt_stat`

   Pierwsza z nich zawiera modele statystyczne rozrzutu względnego dla tranzystorów typowych i nie zawiera rozrzutów procesu (odpowiednik MISMATCH dla Virtuoso) a druga zawiera rozrzut procesowy bez rozrzutu względnego (odpowiednik opcji PROCESS dla Virtuoso).

2) Ustalamy które elementy mają podlegać losowaniom parametrów. Wykonuje się to przez ustawienie parametru **`mm_ok=1`** tych elementów (klawisz Q na danym elemencie).

3) W sekcji kontrolnej używamy pętli która wielokrotnie powtarza tą samą symulację. Wyniki dla każdej symulacji będą nieco inne gdyż za każdym razem będą losowane parametry elementów zgodnie z definicjami dystrybucji statystycznej umieszczonej w bibliotece elementów.

Przykład, który realizuje 10 krotne losowanie i symulację DC przedstawiony jest poniżej. Na początku upewniamy się, że tranzystory mają odpowiednio ustawiony parametr **`mm_ok`** (rys. 33).



Rys. 33. Ustawienie parameteru mm_ok=1. Najeżdżamy myszką na dany tranzystor i wciskamy klawisz Q.

Następnie należy napisać odpowiedni kod sterujący. Przykładowy, pełny kod poniżej (nie potrzebne są oddzielne fragmenty z indywidualnymi deklaracjami bibliotek):

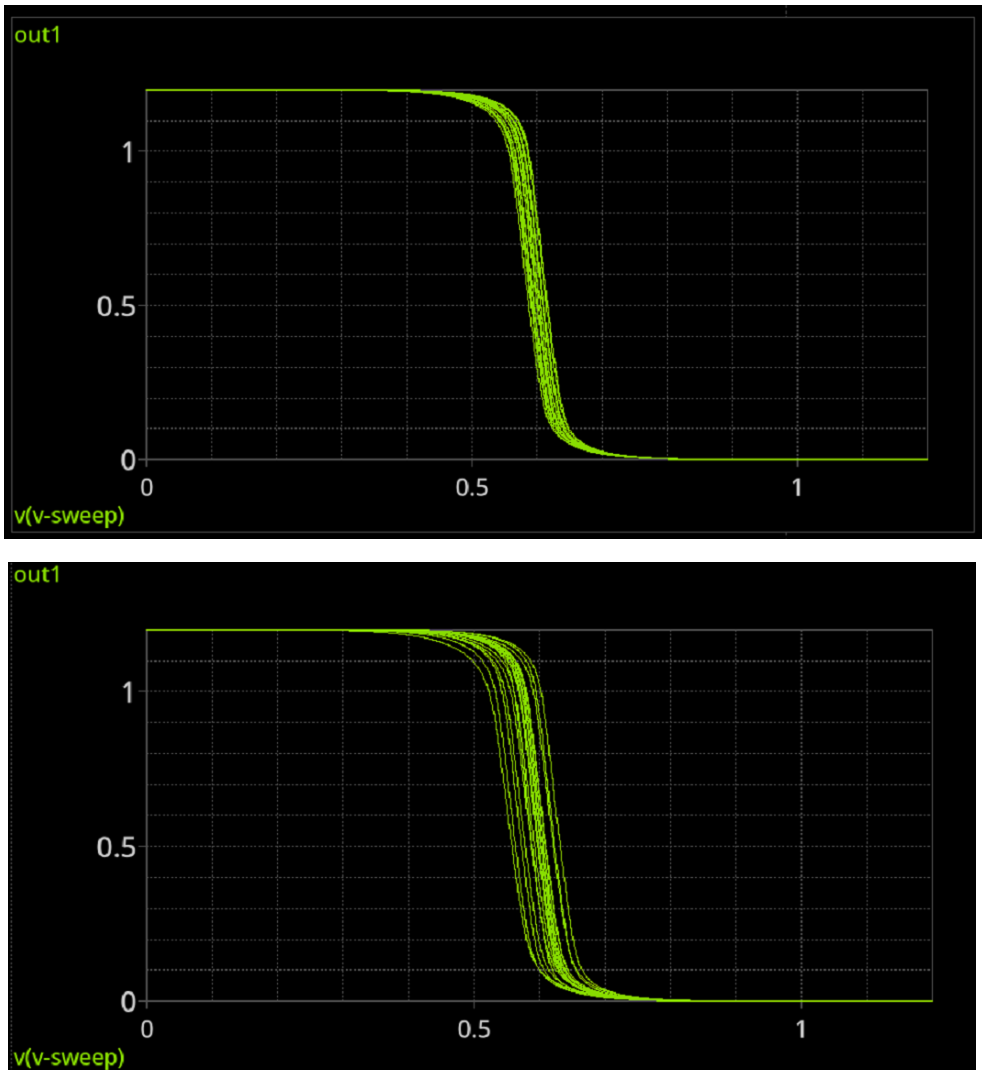
```
name=Sim_Code
simulator=ngspice
only_toplevel=false
value="
*** MOS core models
*.lib cornerMOSlv.lib mos_tt
.lib cornerMOSlv.lib mos_tt_mismatch
*.lib cornerMOSlv.lib mos_tt_stat

*** resistor models
.lib cornerRES.lib res_typ

.param temp=27
.param param_wp = 0.8u
.control
save all
shell rm dc_MC.raw
set appendwrite
let mc_runs = 10
let run = 0
dowhile run < mc_runs
   reset
   set run = $&run
   let run = run + 1
   dc Vin 0 1.2 0.001
   let Gain = deriv(v(out1))
   let Min_Gain = vecmin(Gain)
   meas dc vin_min_gain min_at Gain
   write dc_MC.raw
end
.endc
"
```

Wykonujemy symulację i zamiast pojedynczego przebiegu otrzymujemy 10 wykresów jak na rys. 34. Należy zauważyć, że dla symulacji z modelami PROCESS zmiany są znacznie większe. Zmiany PROCESS generują duże różnice w parametrach tranzystorów w stosunku do modelu typowego jednak dla wszystkich tranzystorów tego samego typu zmiana jest identyczna, tj. wszystkie tranzystory NMOS są

identyczne ale mają dużą zmianę w stosunku do typowego NMOS. Zmiany typu MISMATCH generują mniejszy zakres losowań ale występuje on w obrębie samych typów tranzystorów.



Rys. 34. Wyniki 10-cio krotnej symulacji DC z losowanymi parametrami MISMATCH (góra) i PROCESS (dół) tranzystorów MOS.

Referencje:

[1] https://github.com/IHP-GmbH/IHP-Open-PDK, repozytorium zawierające Open PDK firmy IHP dla technologii SG13G2.

[2] https://www.ihp-microelectronics.com/services/research-and-prototyping-service/fast-design-enablement/open-source-pdk, dokumentacja PDK IHP SG13G2.

[3] https://ihp-open-pdk-docs.readthedocs.io/en/latest/install.html, instrukcja instalacji Open PDK IHP SG13G2.

[4] https://github.com/iic-jku/IIC-OSIC-TOOLS , Open source tools and PDKs prepared in the form of Docker Image, prepared at Department for Integrated Circuits (ICD), Johannes Kepler University.

[5] https://xschem.sourceforge.io/stefan/xschem_man/xschem_man.html , Instrukcja oraz poradniki wykorzystania programu XSCHEM.

[6] https://ngspice.sourceforge.io/ strona domowa oprogramowania **ngspice**, kod, instrukcje, poradniki.

[7] https://www.klayout.de/, strona domowa oprogramowania ***KLAYOUT***.

[8] https://cace.readthedocs.io/en/latest/getting_started/index.html, dokumentacja systemu CACE (ang. Circuit Automatic Characterization Engine).