Appendix A

Clock mesh Simulation in NGSPICE

A.1 NGSPICE

NGSPICE is a mixed-level/mixed-signal, open source circuit simulation program for nonlinear and linear analyses. It is derived from Spice3f5, the last Berkeley's release of Spice3 simulator family. It is an amalgamation of three programs: Spice3f5, Xspice and Cider1b1. As already stated in chapter 3 and 4, NGSPICE was extensively used in this research for clock mesh simulation. In this appendix, we briefly describe the NGSPICE code to simulate our clock mesh.

A.1.1 Uniform distributed RC model(URC)

In Chapter 3, the π-model of the mesh was presented. We have used the URC facility of NGSPICE to form the π-model of the clock mesh. The model is accomplished by a subcircuit type expansion of the URC line into a network of lumped RC segments with internally generated nodes. The general structure of the URC model is UXXXX n1 n2 n3 mname len n1 and n2 are the two element nodes the RC line connects, while n3 is the node to which the capacitances are connected. mname is the model name, len is the length of the RC line in meters. Depending on the length of the wire, this model internally decides the number of lumped segments needed to accurately model the wire. For example

```
.MODEL URCMOD URC(RPERL=300K CPERL=0.16nF)
Umeshwire 1 2 0 URCMOD L=500n
```

represents a uniform distributed model of a wire between nodes 1 and 2 of length 500 nm. RPERL represents the resistance per unit length of the wire and CPERL represents the capacitance per unit length of the wire. RPERL=300K translates to a resistance of 0.3 Ohm/μm and CPERL=0.16nF translates to a capacitance 0.16 fF/μm, the values in 45 nm as specified by ISPD [59].

A.1.2 NGSPICE file for a sample mesh

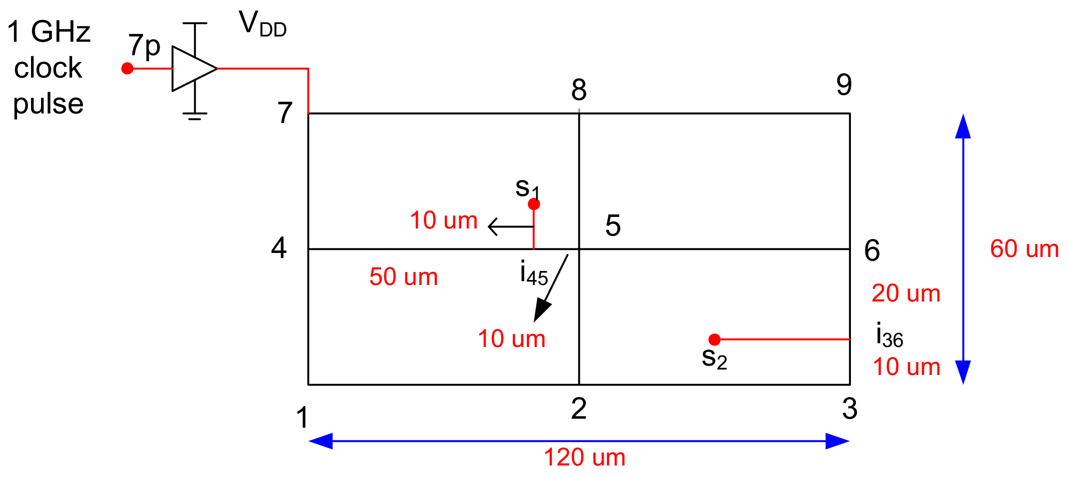

Figure A.1: A simple mesh to illustrate the NGSPICE file formation

Below is the ngspice description of the above mesh. The entire mesh is driven by a single buffer at node 7. s_1 and s_2 are the 2 clock sinks having a capacitance of 30fF each. i_{36} represents an intermediate node of the mesh edge between nodes 3 and 6.

```
*Simulation of test mesh in NGSPICE
*45nm PTM model file is used to simulate buffers
.include tuned_45nm_HP.pm
U1 1 2 0 URCMOD L=60u
U2 2 3 0 URCMOD L=60u
U3 1 4 0 URCMOD L=30u
U4 2 5 0 URCMOD L=30u
U5 3 i36 0 URCMOD L=10u
U6 i36 6 0 URCMOD L=20u
U15 i36 s2 0 URCMOD L=30u
C2 s2 0 30f
.probe v(s2)
U7 4 i45 0 URCMOD L=50u
U8 i45 5 0 URCMOD L=10u
U16 i45 s1 0 URCMOD L=10u
C1 s1 0 30f
```

```
.probe v(s1)
U9 5 6 0 URCMOD L=60u
U10 4 7 0 URCMOD L=30u
U11 5 8 0 URCMOD L=30u
U12 6 9 0 URCMOD L=30u
U13 7 8 0 URCMOD L=60u
U14 8 9 0 URCMOD L=60u
.MODEL URCMOD URC(RPERL=300K CPERL=0.16nF)
* modelling the clock input at the buffer
VIN 7p 0 DC 0 PULSE(0 1 0.1n .05n .05n 0.4n 1n)
VDD 10 0 DC 1
* MOS buffer
MN1 0 7p ib 0 TN L=45N w= 1560nm
MP1 10 7p ib 10 TP L=45N w= 2200nm
MN2 0 ib 7 0 TN L=45N w= 1560nm
MP2 10 ib 7 10 TP L=45N w= 2200nm
.probe v(7p)
.probe i(VDD)
.tran 0.001n 1.2n
.measure tran pwr INTEG i(VDD) from=0 to=1.2n
.end
```

'.probe' command is used to save the voltage at a node. we save the volatages at nodes at which we want to find the clock latency - at sinks s1 and s2. A clock is fed at node '7p' by the PULSE command. The arguments of PULSE command are as follows
PULSE(V1 V2 TD TR TF PW PER)

1. V1 - initial voltage valuse
2. V2 - pulsed voltage value
3. TD - Delay time
4. TR - Rise time
5. TF - Fall time
6. PW - Pulse width
7. PER -Pulse period

The buffer is given a DC power supply of 1 volt. We save the current drawn from V_{DD} using the '.probe' command. This is needed to calculate the total power which is calculated by integrating the current drawn from the power supply in 1 cycle. '.tran' command does a transient simulation for the time specified(1.2 ns).

A.1.3 Executing the .cir file

There are two ways to simulate the .cir file in NGSPICE - batch mode and interactive mode. In the batch mode, we run the program 'window1'.cir as follows

ngspice -b -r window1.raw -o window1.log window1.cir

ngspice will start, simulate according to the .tran command and store the output data in a rawfile window1.raw. Comments, warnings and other informations will go to log file window1.log created in the folder where window1.cir is present.

In interactive mode, we run the program 'window1.cir' as follows:

ngspice

NGSPICE will respond as

ngspice 1 ->

we can then source the .cir file as follows

ngspice 1 -> **source window1.cir**

ngspice 2 ->**run**

ngspice 3 ->**plot allv**

allv command plots all voltages which are probed in the .cir file. If the NGSPICE file has control statements(as will be the case when we do monte carlo simulations), batch mode should not be used and we need to invoke NGSPICE directly as follows

ngspice window1.cir

A.1.4 Analysing the NGSPICE output file(.raw)

Typically, the clock distribution network will have a few thousand sinks and its not possible to manually analyse the clock signal at the sinks. Also, the skew will be in the order of ps which cannot be ascertained accurately by visual inspection. The .raw file contains the value of voltages(which were probed in the .cir file) at sinks during the entire duration of the transient simulation. This is stored as a vector. The HSPICE toolbox for MATLAB ([65]) is a collection of programs to import this information in the raw file into MATLAB. The main program is a mex program called **loadsig** that reads binary output files of transient, DC, or AC sweep data generated by HSPICE or NGSPICE into Matlab or Octave. The following steps can be followed to create an extraction environment in MATLAB.

1. Download "Hspice Toolbox for Matlab® and Octave (also for use with Ngspice)" from `http://www.cppsim.com/download_hspice_tools.html`
2. Place these two downloaded folders in the ToolBox folder of MATLAB installation in the system.
3. Open MATLAB. In File choose **Set Path** $\cdots$ **Add Folder** option, select the two new folders added in Toolbox folder. **SAVE**
4. In MATLAB, make HSPICE toolbox as present folder and give command
   mex loadsig.c

The extraction setup is ready. Supposing the output file of testmesh.cir(Figure A.1) is testmesh.raw,
> z = loadsig('testmesh.raw');
will load the simulation data of testmesh.cir into a structure z
>lssig(z)
lists the signals in z structure. They are 7p, s1, s2 and i(V_{DD}).
> plotsig(z, '7p ,s1 ,s2')
plots the input clock and clock at the sinks s_1 and s_2. As can be seen in Figure.A.2, the latency is visible, but the skew. i.e the difference in clock arrival time between s_1 and s_2 is not very evident since it is in pico seconds.

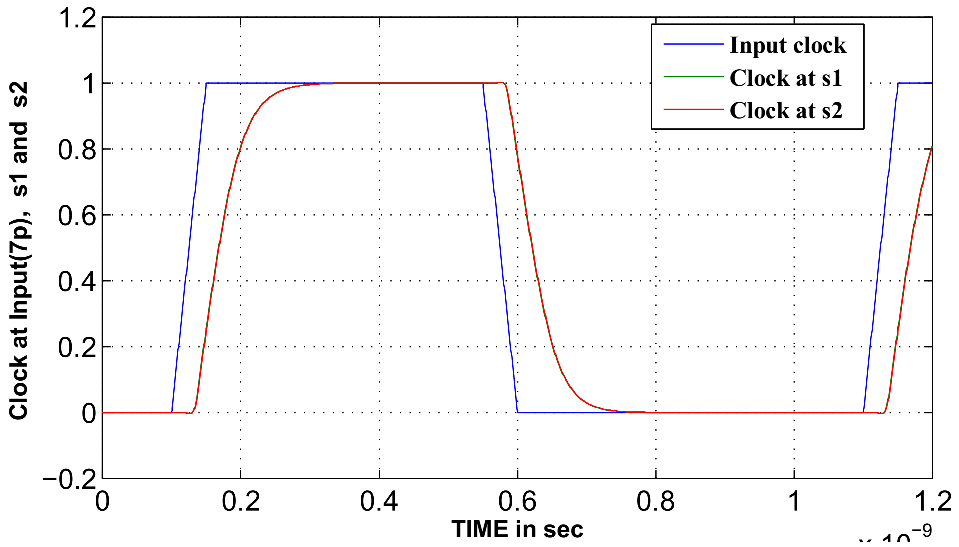

Figure A.2: Simulated clock waveform at input(7p), s1 and s2

If we zoom a certain section near the fall of the first clock pulse, we can observe the skew as shown in Figure.A.3. But since we have to analyse hundreds of sinks, the time (t_1, t_2, $\cdots$,t_n) the clock takes to reach 50% of its maximum value at sinks s_1, s_2, $\cdots$,s_n can be calculated from the z structure by writing a MATLAB program.

Clock latency at $s_i = t_i - t_{\text{input clock}}$

Then, Clock skew is {maximum latency - minimum latency} of all sinks.

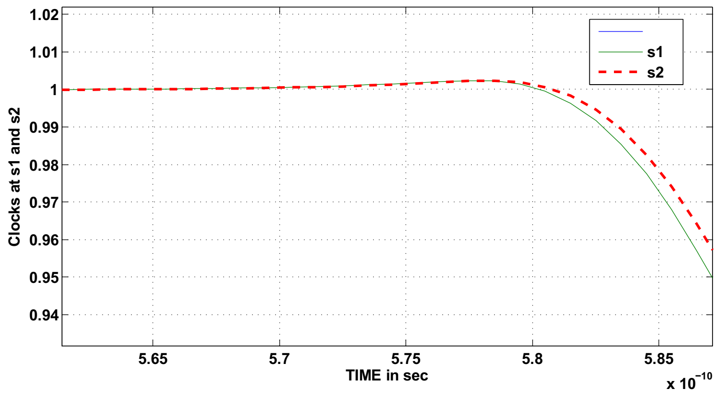

Figure A.3: A certain section zoomed to depict skew between s1 and s2

A.1.5 Power dissipation

As already stated,

.measure tran pwr INTEG i(VDD) from=0 to=1.2n

will display the integrated value of current drawn from 0 to 1.2 nano seconds.

pwr= 4.1234 e-11

This is divided by the time period of 1 clock cycle to get the total power dissipated in the clock mesh in 1 clock cycle.

Appendix B

Monte Carlo Simulation in NGSPICE: Incorporating Process, Voltage and system variations

B.1 Monte Carlo(MC) simulation

Monte Carlo methods refer to a broad class of experiments that rely on repeated random sampling to predict the behavior of an entity whose behavior cannot be predicted in a deterministic manner. In VLSI, Monte carlo methods are often used to study the effect of voltage, temperature and within-die process variations on the performance and characteristics of a circuit. This is done by simulating the circuit over a wide range of randomly chosen values for the parameters which are subject to variation. For example, if supply voltage V_{DD} is subject to variations, we simulate the circuit over many possible values of V_{DD}. We know that the V_{DD} directly affects the propagation delay of a circuit. Monte Carlo simulation attempts to predict the variation in delay of a circuit in response to variation in V_{DD} by studying the distribution of the delay across variations in V_{DD}. In deep Sub micron technology, there may be many variations like process variations, temperature , V_{DD} affecting the circuit at the same time. In these cases, Monte Carlo simulations vary all the parameters simultaneously and perform numerous simulations and measure the circuit characteristics like delay, power etc. The mean and variance of the measured circuit characteristic obtained by plotting the simulation results can give an estimate of the performance of the circuit in the presence of variations. Although computationally intensive, MC simulation are extensively used to study the fabrication readiness of integrated circuits in semi-conductor industry. The increase in

Table B.1: Variations considered in MC simulations

	Parameter	Distribution	Variation
Process	Channel length	Gaussian	3σ with mean 45 nm
	V_{th}	Gaussian	3σ with mean 0.404 V for nmos and -0.384 V for pmos
Voltage	V_{DD}	Gaussian	3σ with mean 1V
System	Inter-buffer skew	Uniform	0-50 ps

computation power of computers due to parallel processors in recent years make MC simulation worth and will be so in the near future. The NGSPICE scripting language may be used to run MC simulations with statistically varying device or model parameters. NGSPICE scripting language supports different statistical distributions for the parameters to be varied

1. uniform distribution
2. gaussian distribution
3. poisson distribution
4. exponential distribution

As already stated in Chapter 4, Monte Carlo simulations with the variations considered in Table B.1 were performed on the clock mesh. In this appendix, we introduce NGSPICE scripting language and describe how it can be used to vary the parameters during circuit simulation in an automated manner. Thus MC simulation is achieved with minimal manual effort.

B.1.1 NGSPICE scripting language

Expressions, functions, commands, variables, vectors and other pseudo-code like structures(example if...else, while...do loop) may be assembled into scripts within a .control $\cdots$ .endc section of the original .cir file. The script allows to automate complex ngspice behavior: simulations are performed, output data are the analyzed, simulations are repeated with modified parameters and output data are assembled. Detailed description of the scripting language, its capabilities and executing method in the simulation flow can be accessed from [72].

Inside the .control statement, we define our two distributions as follows:
define gauss(nom, var, sig) (nom + (nom*var)/sig * sgauss(0))
define unif(nom, var) (nom + (nom*var) * sunif(0))
A call to sgauss(0) will return a gaussian distributed random number as a vector of length 1. similarly, sunif(0) returns a uniformly distributed random number as a unit length vector. The function gauss(nom,var,sig) returns nominal value plus variation drawn from Gaussian distribution with mean 0 and standard deviation var (relative to nominal) divided by sigma.

B.1.2 Incorporating Process variation

We vary channel length and threshold voltage of the MOSFET buffers to incorporate the effect of process variations in our clock mesh. Supposing TN and TP are the NMOS and PMOS transistor models, the threshold voltage V_{th} can be varied using the **altermod** command as follows:

```
set n1vth0=@TN[vth0]
set p1vth0=@TP[vth0]
altermod @TN[vth0]=gauss($n1vth0, 0.05, 3)
altermod @TP[vth0]=gauss($p1vth0, 0.05, 3)
```

The above statement varies the nominal threshold voltage available in the model file as a Gaussian distribution with 3σ variation where σ is 5% of nominal value. **altermod** operates on models and can be used to change any transistor model parameter. In the simulation loop the **altermod** command changes the model parameters before a call to **tran**.

In NGSPICE scripting language, **alter** command changes the value for a device or a specified parameter of a device or model. Using it, we can vary the channel length as follows:

```
alter MN1 L= gauss(45n, 0.05, 3)
alter MP1 L= gauss(45n, 0.05, 3)
```

MN1 and MP1 are instantiation of NMOS and PMOS and they are varied as a Gaussian distribution with 3σ variation where σ is 5% of nominal value.

B.1.3 Incorporating voltage variation

We would have already specified V_{DD} as

```
VDD n1 0 DC 1 *represents 1 V Dc value between node n1 and ground
```

we can vary it as

```
alter VDD dc = gauss(1, 0.05, 3)
```

which means that the nominal supply voltage of 1 V is varied as a Gaussian distribution with 3σ variation where σ is 5% of nominal value.

B.1.4 Incorporating system variation

Incorporating the system variation i.e the inter-buffer skew is a rather strenuous task in NGSPICE. As already stated in Chapter 4, the inter-buffer skew is induced by variations in the top-level tree driving the mesh. Hence,the clock will not reach all the mesh buffer inputs at the same time. To incorporate this uncertainty in clock arrival time, we have to introduce uncertainty in the PULSE command. As already stated, the PULSE command has a TD parameter which specifies the delay in the PULSE signal. We have to incorporate a 0-50 ps difference in the clock arrival time at mesh buffers. We choose 125 ps as the mean arrival time. By uniformly varying 125 ps by 20%, we will get arrival times between 100 ps

(125-25 ps) and 150 ps (125+25 ps). We form a variable called tdi to denote the delay of the pulse at i^{th} buffer.

```
let td1 = unif(0.125n, 0.2)
let td2 = unif(0.125n, 0.2)
. . .
let tdn = unif(0.125n, 0.2) *for nth mesh buffers
```

and then feed the delayed clock to the mesh buffer inputs as

```
alter @VIN1[PULSE] = [ 0 1 $&td1 .05n .05n 0.4n 1n ]
alter @VIN2[PULSE] = [ 0 1 $&td2 .05n .05n 0.4n 1n ]
. . .
alter @VINN[PULSE] = [ 0 1 $&tdn .05n .05n 0.4n 1n ]
```

B.1.5 Structure of the whole NGSPICE file for a particular benchmark

```
.include tuned_45nm_HP.pm
C45 45 0 6.435000e+000f *these are sinks inside the window
.probe v(45)
C46 46 0 6.435000e+000f
.probe v(46)
. . .
C977 977 0 1.071000e+001f
.probe v(977)
C978 978 0 15f
.probe v(978)
*No of sinks inside this window is 292
*steiner tree
URC4 819 815com 0 URCMOD l=7.737836e+003n
URC5 1s 815com 0 URCMOD l=5.702164e+003n
URC6 815 1s 0 URCMOD l=1920n
URC7 815com 815stub 0 URCMOD l=1.772324e+004n
URC8 220 220stub 0 URCMOD l=6000n
*steiner tree
URC9 205 205com 0 URCMOD l=26400n
URC10 201 205com 0 URCMOD l=26400n
URC11 205com 205stub 0 URCMOD l=52800n
```

```
··· other steiner trees
URC244 810 810stub 0 URCMOD l=1.767500e+003n rooms with only one sink connected
to mesh by stub
URC245 917 917stub 0 URCMOD l=750n
URC246 799 799stub 0 URCMOD l=3000n
URC247 804 804stub 0 URCMOD l=12000n
URC248 88 88stub 0 URCMOD l=960n
URC249 93 93stub 0 URCMOD l=960n
...
URC373 1n 3n 0 URCMOD l=117900n *mesh edges,1n and 3n represents mesh nodes
URC374 1n 2n 0 URCMOD l=48720n
URC375 2n 4n 0 URCMOD l=117900n
...
C193n 193n 0 2.464125e+001f *sinsk capacitances outside the window are lumped at the
nearest meshnode
C194n 194n 0 5.142750e+001f
C195n 195n 0 1.071375e+001f
...
VIN1 1inp 0 DC 0 PULSE(0 1 0NS .05NS .05NS .40NS 1nS) *clock at mesh buffers
VIN2 2inp 0 DC 0 PULSE(0 1 0NS .05NS .05NS .40NS 1nS)
...
.model URCMOD URC(RPERL=300K CPERL=0.16nF)
VDD 1v 0 DC 1.00
MN1 0 1inp 2i 0 TN L=45N w=82n *mesh buffer at mesh node
MP1 1v 1inp 2i 1v TP L=45N w=114n
MN2 0 2i 2n 0 TN L=45N w=1718n
MP2 1v 2i 2n 1v TP L=45N w=2451n
...
.control

define gauss(nom, var, sig) (nom + (nom*var)/sig * sgauss(0))
define unif(nom, var) (nom + (nom*var) * sunif(0))

let mc_runs=200
let count=1

dowhile count ≤ mc_runs

set n1vth0=@TN[vth0]
set p1vth0=@TP[vth0]
altermod @TN[vth0]=gauss($n1vth0, 0.05, 3)
altermod @TP[vth0]=gauss($p1vth0, 0.05, 3)
alter VDD dc = gauss(1, 0.05, 3)

let td1 = unif(0.125n, 0.2)
let td2 = unif(0.125n, 0.2)
. . .
let td207 = unif(0.125n, 0.2)
let td208 = unif(0.125n, 0.2)
alter @VIN1[PULSE] = [ 0 1 $&td1 .05n .05n 0.4n 1n ]
alter @VIN2[PULSE] = [ 0 1 $&td2 .05n .05n 0.4n 1n ]
. . .
alter MN1 L= gauss(45n, 0.05, 3)
alter MP1 L= gauss(45n, 0.05, 3)
. . .
tran 0.001n 1.4n
meas tran pwr INTEG i(VDD) from=0 to=1.4n
save i(VDD)
let power=-pwr/1.4n
save power
if count = 1
write LL1.raw
end
. . .
if count = 200
write LL200.raw
end

let count= count +1
reset
end
.endc
.end
```