Abstract

LIPA, STEVEN. Frequency Stability in Distributed Latch Travelling Wave Oscillators. (Under the direction of Dr. Paul Franzon and Dr. Michael Steer) This work presents analysis and modelling of phase noise and jitter in travelling wave oscillators based on distributed CMOS latches (DLTWOs). These oscillators are capable of generating GHz rate square waves and distributing them over large integrated circuits with low skew.

The results show that the frequency stability of DLTWOs is comparable to that of well-designed ring oscillators. A time-domain perturbation analysis is presented in which the DLTWO is broken up into a number of stages like a ring oscillator. The time-domain perturbation analysis shows that excess phase is mainly introduced when voltage noise is amplified in the transitioning latches. The time-domain perturbation analysis is verified by impulse sensitivity function (ISF) simulations. Simulation techniques are described and techniques for drastically reducing the number of simulations required to implement the ISF are introduced. It is shown that very accurate results are possible with as few as 41 simulations and that fairly accurate results are possible with as few as three simulations. Simulation results are compared to measured results for the first integrated DLTWO circuits ever produced. Design implications are explored giving designers insight into designing DLTWOs with reduced frequency instability.

FREQUENCY STABILITY IN DISTRIBUTED LATCH TRAVELLING WAVE OSCILLATORS

BY

Steven Lipa

A DISSERTATION SUBMITTED TO THE GRADUATE FACULTY OF
NORTH CAROLINA STATE UNIVERSITY
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

ELECTRICAL ENGINEERING

RALEIGH
December 2004

APPROVED BY:

CO-CHAIR OF ADVISORY COMMITTEE	CO-CHAIR OF ADVISORY COMMITTEE

Biography

Steven Lipa received the BS in Electrical Engineering from the University of Virginia in 1980 and the MS in Electrical Engineering from North Carolina State University in 1993. His Masters research was in the area of transmission line modelling. Over the years he has worked for several companies as an integrated circuit designer, laboratory manager, and consultant.

Acknowledgements

First I must thank Dr. Paul Franzon, and Dr. Michael Steer, who provided generous support and encouragement throughout my career as a graduate student. Dr. Harvey Charlton provided support and great courses in applied mathematics. Dr. Griff Bilbro kindly joined my committee when I needed him and was a source of interesting ideas.

Next I must thank John Wood, the innovator and independent thinker who invented the DLTWO and therefore made this work possible, and his company Multi-GiG, which provided working DLTWOs for me to measure.

Michele Joyner provided tremendous support for our entire research group by making sure that everything ran smoothly.

Finally, I must thank my family and many friends for their support and ideas.

Table of Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

$\alpha(\omega_0\tau)$	-	noise modulating function
B'_t	-	Brownian motion process
$\beta(\omega_0 t)$	-	capacitance modulating function
C	-	generalized capacitance
c_0	-	speed of light in vacuum
C'	-	capacitance per unit length
C_{ll}	-	line-to-line capacitance
C_{ls}	-	line-to-substrate capacitance
C_L	-	capacitive loading per stage
CMF	-	capacitance modulating function
CMOS	-	complementary metal oxide semiconductor
D	-	phase diffusion constant
dB	-	decibels
dBc	-	decibels below the carrier
dBm	-	decibels above a milliwatt
DLTWO	-	distributed latch travelling wave oscillator
DSO	-	digital sampling oscilloscope
DUT	-	device under test
Δq	-	change in charge
Δf	-	frequency band
$\Delta\phi$	-	excess phase
$\Delta\Theta$	-	phase shift
$\Delta\tau$	-	timing error

$\Delta\omega$	-	frequency offset
$\Delta\omega_{f^{-3}}$	-	third order corner frequency
ϵ_0	-	permittivity of free space
f	-	frequency (Hz)
f_c	-	carrier frequency
g_{d0}	-	transistor conductance at zero bias
g_m	-	transconductance
g_M	-	differential transconductance
GS	-	Ground-Signal
GSG	-	Ground-Signal-Ground
γ	-	channel-length proportionality constant
$\Gamma(\omega_0\tau)$	-	impulse sensitivity function
Γ_{CMF}	-	modified ISF (capacitance)
Γ_{NMF}	-	modified ISF (noise)
Γ_{rms}	-	rms value of impulse sensitivity function
Γ_0	-	DC component of impulse sensitivity function
$h_\phi(t,\tau)$	-	normalized phase response
$H(j\omega)$	-	open-loop transfer function
i	-	generalized AC current
I	-	generalized DC current
I_{dc}	-	DC current
I_n	-	current noise
I_{EE}	-	nominal tail current
I_{SS}	-	nominal tail current
ISF	-	impulse sensitivity function $\Gamma(\omega_0\tau)$
κ	-	figure of merit for oscillator stability
κ_R	-	component of κ due to thermal noise
κ_{samp}	-	component of κ due to sampling
κ_{tail}	-	component of κ due to tail current
L	-	generalized inductance

L'	-	inductance per unit length
$\mathcal{L}(\Delta\omega)$	-	single-sideband noise spectral density
LTI	-	linear time-invariant
LTV	-	linear time-varying
MOSFET	-	metal oxide semiconductor field effect transistor
μ_0	-	permeability of free space
NMF	-	noise modulating function
NMOS	-	N-type MOSFET
PLL	-	phase-locked loop
Q	-	resonator quality factor
$q_{\max}$	-	resonator quality factor
Q_{total}	-	total charge on node
PMOS	-	P-type MOSFET
R	-	generalized resistance
RSS	-	root sum of squares
SG	-	Signal-Ground
$S_{\Delta\phi}(\omega)$	-	spectral density of phase uncertainty
$S_{\phi}(\omega)$	-	power spectral density of phase
$S_{RF}(\omega)$	-	two-sided radio frequency spectrum
$S_y(\omega)$	-	power spectral density of fractional frequency deviation
σ_{ctc}	-	cycle-to-cycle jitter
σ_{edge1}	-	RMS jitter at first edge after trigger
σ_{min}	-	RMS jitter of trigger system
σ_t	-	time variance
σ_v	-	voltage variance
σ_τ	-	jitter over interval τ
$\sigma_{\tau,eff}$	-	effective jitter over interval τ
$\sigma_y^2(\tau)$	-	Allan variance
σ_ϕ	-	phase jitter
T	-	temperature
t_d	-	delay per stage

V_0 - signal amplitude
$V_c(t)$ - instantaneous signal voltage
V_{dd} - supply voltage
V_{in} - input voltage
V_{out} - output voltage
V_n - voltage noise
v_p - velocity of propagation
V_T - thermal voltage kT/q_e
ω_c - carrier frequency
X - generalized oscillator input
Y - generalized oscillator output
$y(t)$ - normalized instantaneous frequency fluctuation
ω - frequency (radians)
$\omega_{f^{-1}}$ - flicker knee frequency

Chapter 1

Introduction

1.1 Distributed Latch Travelling Wave Oscillators (DLTWOs)

Travelling wave oscillators based on distributed latches were developed primarily for high-speed clock distribution in large digital integrated circuits such as microprocessors.

The idea was first introduced in 2001.[1, 2] It consists of a differential transmission line in the shape of a ring, buffered by cross-coupled latches. The differential transmission line is cross-coupled at an odd number of sites along the length of the ring. Figure 1.1 shows block diagrams of two simple examples of this architecture. In Figure 1.1 (a) there is a single crossing of the differential transmission line. In Figure 1.1 (b) the same pattern of latches is used, but now there are seven crossings of the differential transmission line. Increasing the number of crossovers helps to confine the electric field between the conductors reducing electromagnetic compatibility (EMC) problems such as coupling to adjacent conductors. The configuration of the ring results in a single clock edge that traverses the ring at a speed that depends to a first order on the effective shunt capacitance per unit length of the differential line as well as its effective shunt inductance per unit length. In order to understand the genesis of the wavefront it is instructive to refer to Figure 1.2. In this figure we see a semi-infinite differential transmission line with cross-coupled inverters on the right, driven by a low-impedance pair of switches on the left. Initially, the latches in the semi-infinite transmission line are set so that the top half of the differential

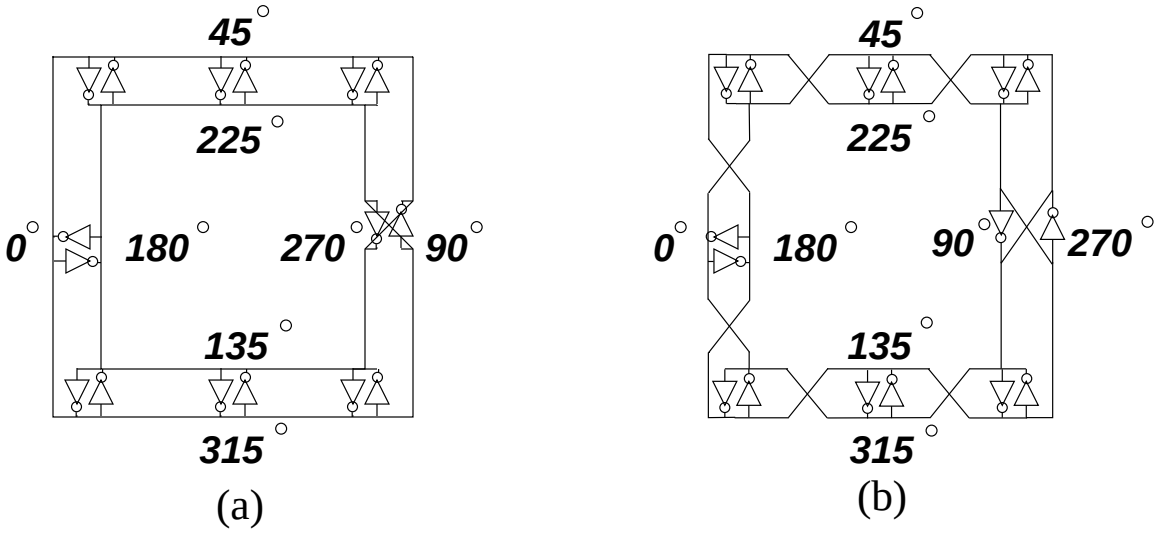

Figure 1.1: (a) A simple DLTWO with one crossover. (b) a similar DLTWO with seven crossovers.

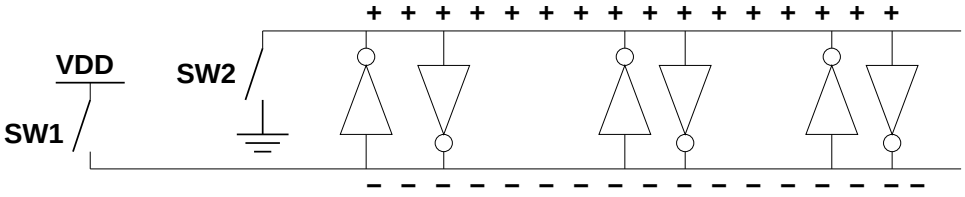

Figure 1.2: The genesis of the DLTWO wavefront.

transmission line is high, and the bottom half is at ground potential. If both of the switches on the left side of the diagram are switched simultaneously, charge from the positive rail will flow in to the bottom half of the differential transmission line and charge from the top half of the differential transmission line will flow to ground. The effect of the switches is essentially the same as the output of a latch that has just changed state. If the impedance of the switches is low enough, the charge will flow quickly enough to change the state of the first latch. Now that the first latch has been toggled, more charge is drained from the top half of the differential transmission line towards ground and the inverter in the first latch is helping switch one pull up the bottom half of the differential transmission line. As long as the characteristic impedance of the transmission line is low enough, the wavefront will continue from left to right indefinitely, causing a positive-going edge on the lower half of the transmission line, and a negative-going edge on the upper half. In the case of the DLTWO we simply cross-couple the differential transmission line like a Möbius strip, and the process continues indefinitely. Note that for the left-to-right moving wavefront in our example, current in the top line is flowing toward the left, while in the bottom line it is flowing toward the right.

Obviously, the speed of the wavefront is of particular interest, because as can be seen by referring back to Figure 1.1, the time that it takes the wavefront to traverse the ring twice determines the operating frequency of the ring. This is because each time the wavefront passes any given point on the ring the phase at that point changes by 180°, or π radians, so a full cycle requires two passes and the fundamental frequency of the ring f_c is given as:

$$f_c = \frac{v_p}{2l_{ring}} \tag{1.1}$$

where v_p is the velocity of propagation and l_{ring} is the electrical length of the ring. Ideally, v_p would be constant, but of course if this were the case f_c would be constant and the frequency stability of the ring would be perfect. A major goal of this work is to determine and model the mechanisms that give rise to short- and longer-term changes in v_p which give rise to phase noise.

If we view the DLTWO ring as a differential transmission line which is loaded by

the capacitance of the latches that are spread around it, it is possible to estimate v_p and therefore f_c by noting that

$$v_p = \frac{1}{\sqrt{L'C'}} \tag{1.2}$$

where L' and C' represent the approximate effective inductance and capacitance of the loaded line per unit length respectively. Of course this is only an approximation, but it turns out that simple approaches to estimating L' and C' combined with the combination of Equations 1.1 and 1.2 predict v_p with surprising accuracy. The simplest approach is to break the ring into small sections and estimate L' using an analytic formula and to estimate C' as a combination of the line-to-line capacitance and the loading capacitance presented by the inverters. For example, approximate L' as

$$L' = \frac{\mu_0}{\pi} \ln\left(\frac{\pi s}{w + t_c} + 1\right) \tag{1.3}$$

where μ_0 is the relative permeability of free space, s is the separation between the conductors, w is the width of the conductors and t_c their thickness. [3] Then approximate C' using the simple model of the latch shown in Figure 1.3 (a) and (b). In practice, it is found that the parasitic capacitances of the latch circuitry account for the majority of C'. Once v_p is known it is possible to control f_c by varying the length of the ring. It is possible to dynamically control v_p by varying L' and C' thus changing the fundamental frequency of the ring. Probably the most practical approach is to vary C' by introducing varactors or switching in more or less parasitic capacitance along the line. Once some degree of frequency control is available it may be possible to phase lock the ring to another oscillator.

1.1.1 Arrays of DLTWOs

The single DLTWO is good because it can generate arbitrarily many phases of a high-speed clock with very fast slew rate and drive a heavy load. [4] The most interesting aspect of the DLTWO, however, is that DLTWOs can be arrayed into arbitrarily large clocking systems. The key is that the DLTWOs must be interlinked in such a way that the shared nodes are always at the same phase. For example,

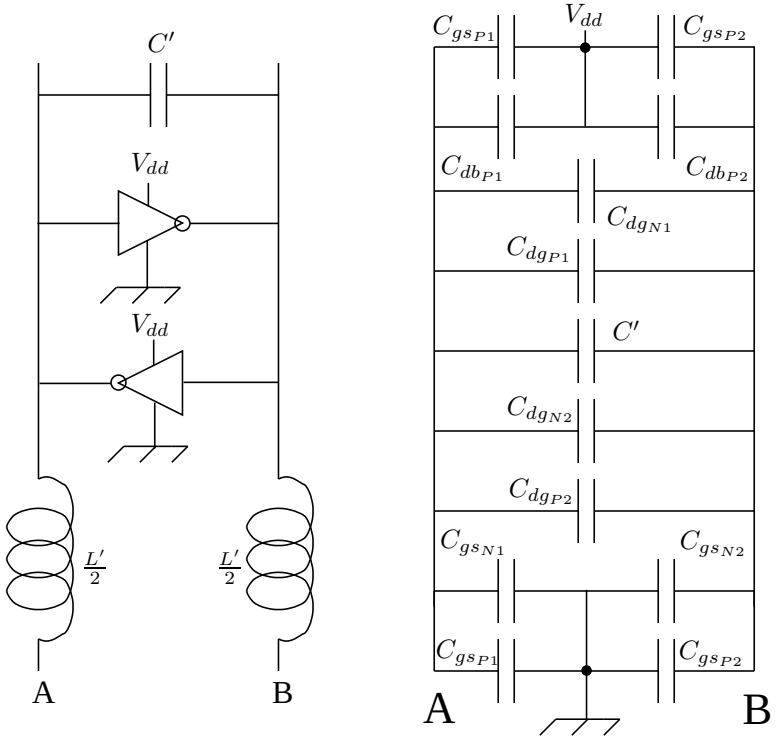

Figure 1.3: (a) simple model of DLTWO section. (b) latch parasitic capacitances.

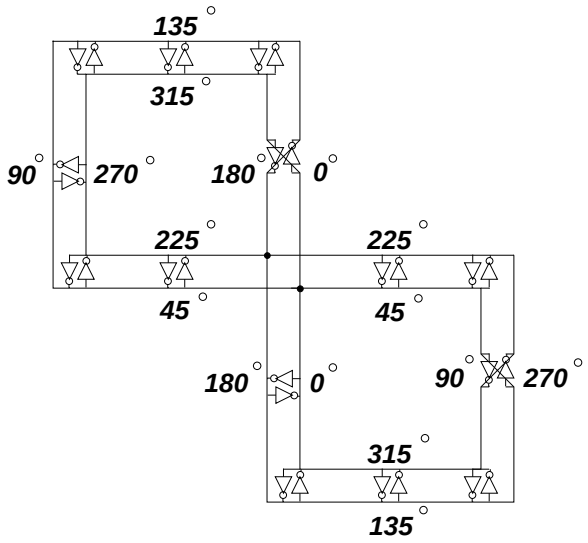

Figure 1.4: A simple DLTWO array with two rings

see Figure 1.4. In this system, each ring supports a wavefront. In order for both rings to simultaneously support individual wavefronts, the wavefronts must reach the common point at precisely the same time. The quality of the impedance match at the common point is important to maintain the sharp edge of the wavefront.

Building on this theme, it is possible to link together larger arrays of DLTWOs as illustrated in Figures 1.5 and 1.6. The points labeled = indicate points of equivalent

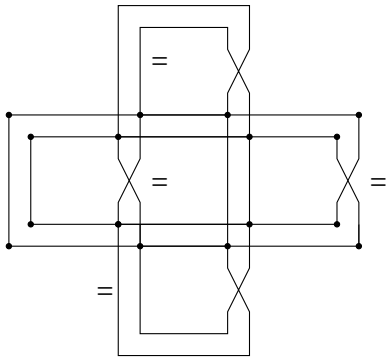

Figure 1.5: Five interlinked rings.

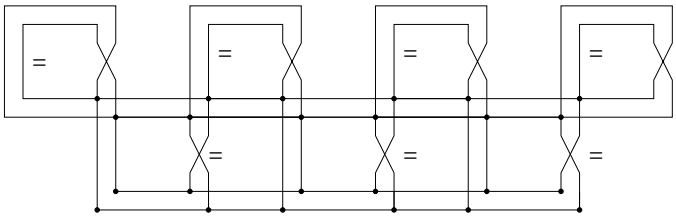

Figure 1.6: Nine interlinked rings.

relative phase. That is, the waves traversing the individual rings hit these points simultaneously. Of course these points are just examples; for any given point on any of the rings it is possible to determine the point of equal relative phase on all the other rings. Rings that are as complicated as these require proper attention to matching and it is important to note that the direction of the wavefront is arbitrary unless the ring is biased one way or the other, which can be important for multiphase clock distribution.

By building up an array that covers an entire integrated circuit it is possible to distribute points of equal phase (and therefore theoretically zero clock skew) over the entire chip as shown in Figure 1.7. For a single-phase clocking system, it is then simply necessary to tap off of the nearest ring at the nearest point of equal phase to drive a small local clock buffer and/or distribution system such as an H-tree. For two-phase systems, the inverse phase is immediately at hand due to the differential nature of the transmission line. Theoretically any arbitrary number of phases can be accommodated by tapping off at the appropriate place(s).

1.2 Frequency Stability

While the DLTWO was originally invented with clock distribution in large digital systems in mind, the purpose of this work is to consider the DLTWO as a multi-purpose oscillator, and to study and model all aspects of its frequency and phase stability. Integrated circuit oscillators are much cheaper than oscillators that rely on

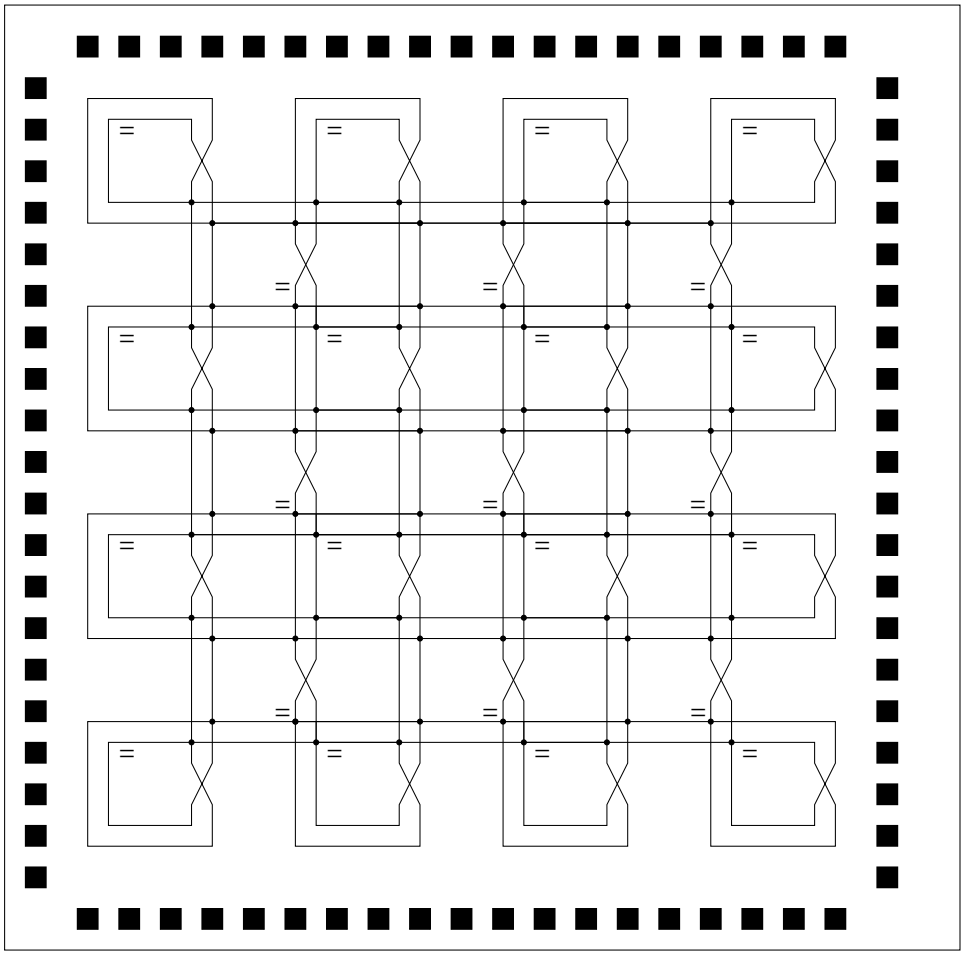

Figure 1.7: An DLTWO array covering a full chip.

external resonators such as quartz crystals, but generally have inferior frequency stability characteristics. It may be that properly designed DLTWO circuits can provide performance closer to that of more expensive implementations.

There are many reasons for studying the frequency stability characteristics of oscillators. For oscillators used as timing signals in digital systems, knowledge of the frequency stability characteristics makes it possible to determine appropriate setup and hold time rules and predict how the oscillator will perform in phase-lock. In digital communications systems, phase noise contributes to bit-error-rate (BER).

In an analog system such as the the local oscillator of a multichannel receiver, knowledge of the frequency stability characteristics of the local oscillator allows for the prediction of adjacent channel interference. In doppler radar systems, phase noise reduces sensitivity and spatial resolution. In test equipment such as spectrum analyzers and network analyzers, phase noise of the local oscillator is the limiting factor in frequency resolution. For example, in spectrum analyzers resolution of a weak signal directly adjacent to a strong signal is limited by phase noise exactly as it is in a multichannel receiver.

Each type of practical oscillator implementation is unique in some way, but all oscillators include components which introduce well known noise processes such as Johnson noise, flicker and burst noise, to name a few. The frequency stability characteristics of each type of oscillator are unique because the combination of noise sources is to some degree unique and the different types of oscillators combine these noise processes in unique ways. Many types of oscillators have been thoroughly studied in the past, but so far no one has studied the DLTWO. The purpose of this work is to determine how the unique architecture of the DLTWO causes noise processes to combine to create phase noise and jitter.

The goal of the following chapters is to:

1. Provide a review of the various approaches previously applied to the study of frequency stability in similar oscillators.

2. Identify the unique properties of the DLTWO that determine its frequency stability characteristics

3. Develop techniques for efficiently and accurately modelling these characteristics.
4. Provide design guidelines for optimizing the frequency stability of DLTWOs.
5. Present measurements on the first DLTWO integrated circuits ever made.

Chapter 2 provides a review of publications relevant to the study of DLTWOs. Chapter 3 provides a time-domain perturbation analysis of the DLTWO which pinpoints the mechanisms that give rise to frequency instability. This analysis is verified in Chapter 4 through simulation and measurement of actual DLTWO integrated circuits. Chapter 4 also introduces techniques for reducing the number of simulations required to accurately estimate the frequency stability of the DLTWO. Chapter 5 combines the lessons of the previous chapters, providing insight for the design of DLTWOs with improved frequency stability. Chapter 6 summarizes the conclusions and contributions of this work.

1.3 Original contributions

- A detailed description of the mechanisms that lead to excess phase in the DLTWO – Sections 3.1-3.3.
- Identification of the noise sources unique to the DLTWO architecture – Section 3.2
- Identification of the characteristics of the DLTWO architecture that can be exploited to reduce the number simulations required to perform ISF analysis of the DLTWO. – Section 4.2.3
- Analysis of several design implications resulting from the insights gained in Chapters 3 and 4 as applied to the following design parameters –
  1. Number of rings – Section 4.3.5
  2. Number of DLTWO stages per ring – Section 5.2
  3. Power supply voltage – Section 5.3
  4. Waveform symmetry – Section 5.4

Chapter 2

Literature Review

2.1 The Study of Frequency Stability

Frequency stability has been a subject of intense study since before World War II [5]. Since that time, the explosion of technology that has resulted in global communications networks, global computer networks, ubiquitous doppler radars, atomic clocks, and thousands of miles of fiber optics has necessitated advancements in the theory of frequency stability that have resulted in hundreds, if not thousands of publications. This chapter will attempt to distill some of the most important and relevant results out of this vast body of knowledge so that they are available to the reader as background material for subsequent chapters.

2.2 Fundamentals

The study of frequency stability naturally divides into several distinct, though related areas. Fundamentally, we are interested in the phenomena that give rise to fluctuations in the frequency or phase of oscillators. A block diagram of a typical oscillator is shown in Figure 2.1. While this is not the most general representation of an oscillator, it is representative of all of the oscillators with which we will be concerned. For the oscillator in Figure 2.1 to oscillate with no external input, the phase shift around the loop must be exactly zero, and the gain around the loop must be at least 1. A

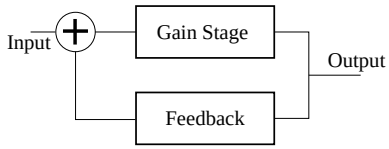

Figure 2.1: Block diagram of typical oscillator.

perfect oscillator might be described in the time domain by the equation

$$V_c(t) = V_0 \cos \omega_c t \tag{2.1}$$

where $V_c(t)$ represents the instantaneous signal voltage, V_0 represents the amplitude of the signal and ω_c denotes the angular frequency in radians per unit time. Obviously this equation provides no facility for describing frequency or phase errors. To describe a less perfect, and more practical oscillator we might resort to an equation of the form

$$V_c(t) = V_0 \cos(\omega_c t + \phi(t)) \tag{2.2}$$

where $\phi(t)$ represents a function of time that affects the *phase* of the signal. In general it is assumed that $\phi(t)$ is low in frequency relative to ω_c. The frequency stability of $V_c(t)$ can be characterized in a number of ways, although they are all mathematically related.

2.2.1 Frequency domain

One popular characterization is the spectral density of the instantaneous fractional frequency fluctuations. In other words, given that $\phi(t)$ is a time-domain representation of the instantaneous phase error of the oscillator, how does $\phi(t)$ effect the instantaneous frequency of the oscillator? Barnes *et al.* [7] proposed defining

$$y(t) \stackrel{\text{def}}{=} \frac{\dot{\phi}(t)}{\omega_c} \tag{2.3}$$

which represents the fractional (normalized) instantaneous frequency fluctuation of the oscillator. The spectral density of this function, denoted $S_y(\omega)$ is equivalent to the frequency spectrum seen at the output of an ideal phase detector with $y(t)$ as its input.

A more popular characterization seems to be the spectral density of $\phi(t)$ itself, denoted $S_\phi(\omega)$. This is because it is relatively easy to derive from measuring the radio frequency spectrum of the oscillator. [6] We have

$$S_{RF}(\omega) \approx \frac{P}{2}\left(S_\phi(\omega+\omega_c) + S_\phi(\omega-\omega_c)\right) \tag{2.4}$$

where S_{RF} is the two-sided radio frequency spectrum of the oscillator, and P is the power in the carrier. Thus the $S_\phi(\omega)$ information is easily extracted from the sidebands. Another benefit of knowing $S_\phi(\omega)$ is that it is possible to derive $S_y(\omega)$ from it. This is because [6]

$$S_{\dot{\phi}}(\omega) = \omega^2 S_\phi(\omega) \tag{2.5}$$

where $S_{\dot{\phi}}(\omega)$ is equivalent to the frequency spectrum seen at the output of an ideal *frequency modulation* detector with $\phi(t)$ as its input. This is useful because

$$S_y(\omega) = \omega_c^{-2} S_{\dot{\phi}}(\omega) \tag{2.6}$$

so substituting Equation (2.5) into Equation (2.6) yields $S_y(\omega)$.

The most common approach is to report one sideband of the spectral density $S_\phi(\omega)$ relative to the carrier per hertz

$$\mathcal{L}(\Delta\omega) = 10\log\left[\frac{S_\phi(\Delta\omega)}{2P_{carrier}}\right] \tag{2.7}$$

In other words, at each frequency offset $\Delta\omega$ from the carrier, the power in one of the sidebands in a one hertz bandwidth around that frequency is divided by the total power in the carrier, and the logarithm is taken. Thus the units of the result are dBc/Hz.

2.2.2 Time domain

Time domain characterization of an oscillator's frequency stability is also useful. The foundation for virtually all modern time domain characterization of oscillators is the work of Allan [8] in the study of atomic clocks. The accuracy of a clock obviously depends on the stability of some sort of frequency generator. If two measurements

are made of a clock's accuracy, the fractional frequency departure is defined as

$$y(t) \stackrel{\text{def}}{=} \frac{\text{change in clock's time error}}{\text{time between measurements}} \tag{2.8}$$

If $y(t)$ is calculated for two adjacent intervals, with y_n the result for the first interval and y_{n+1} the result for the second interval, the difference between the two is a measure of the frequency stability of the oscillator. The Allan variance

$$\sigma^2(\tau) = \frac{\langle (y_{n+1} - y_n)^2 \rangle}{2} \tag{2.9}$$

where the angle brackets represent the infinite time average

$$\langle f(t) \rangle = \lim_{T \to \infty} \int_{-T/2}^{T/2} f(t) dt \tag{2.10}$$

and τ is the time between measurements yields a simple single dimensionless number which provides a good indication of the frequency stability of an oscillator. Some typical values of Allan variance are presented in Table 2.1.

Table 2.1: Some typical values of Allan variance ($\tau = 1s$)

Ring oscillator	$5.0\text{X}10^{-8}$
Quartz watch	$1.0\text{X}10^{-9}$
Atomic clock	$1.0\text{X}10^{-13}$

The Allan variance is particularly useful for oscillators because it converges for all types of noise processes. This is not the case for the classical variance

$$\sigma^2 = \lim_{m \to \infty} \frac{1}{m-1} \sum_{j=1}^{m} (y_n - \mathrm{E}[y])_j^2 \tag{2.11}$$

where m is the number of intervals measured and $\mathrm{E}[y]$ is the expected value, or mean of y. For certain noise processes (Equation 2.11) fails to converge. Strictly speaking, Allan variance specifies an average over all time. In a laboratory setting, this simply takes too long. Therefore, the Allan variance can only be approximated using finite sample sizes with equipment that has limited bandwidth. A number of improvements on the Allan variance have been proposed for different situations,

including the Modified Allan variance, the Time variance, the Hadamard variance, etc. [9, 10] Short term measures of variance are generally termed "jitter" and usually specified as the square root of the classical variance. The term "phase jitter" refers to the standard deviation of the short term phase error, which can easily be found using the square root of the time variance. Thus

$$\sigma_\phi = \frac{2\pi\sqrt{\sigma^2}}{T} \tag{2.12}$$

In general it is possible to compute variance or jitter given the power spectral density of phase noise, but it is not possible to go the other way without knowledge of the shape of the spectrum.

2.2.3 General comments on modelling

The tremendous variety of approaches to oscillator design has led to the development of a number of techniques for characterizing and analyzing their frequency stability. Broadly speaking, these techniques fall into four major areas:

Linear-time-invariant (LTI) Oscillators based on tank circuits, crystals, or other high-Q resonators are often treated using LTI techniques.

Linear time-variant (LTV) Linear time-variant models improve on LTI techniques by more accurately portraying the variations of the circuit elements and noise processes in time.

Time-domain Perturbation Here the oscillator is considered mainly in the time domain as a series of time delays. Noise processes directly affect the delays of circuit elements or change the times that limits or thresholds are met.

Stochastic Perturbation In this approach the oscillator is described by a system of integro-differential equations perturbed by random noise processes. These complicated techniques are more general and can be applied to a wider array of oscillator architectures.

In the following sections of this chapter some of the most pertinent results in some of these different areas will be reviewed.

2.3 Linear Time-invariant Analysis

2.3.1 Leeson

D.B. Leeson's paper on "linear feedback oscillator[s]" [11] seems to be one of the most referenced papers in the phase noise literature. Deciphering it requires a sound understanding of the fundamentals above, but provides significant insight into the nature of frequency stability in oscillators in general, so it is a good place to start. Leeson was considering an oscillator of the form shown in Figure 2.2, where the feedback network is a single resonator, *e.g.* a crystal or an LC tank. The feedback

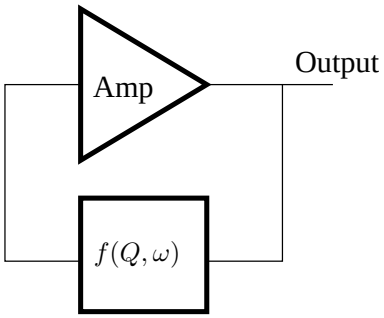

Figure 2.2: Leeson's feedback oscillator.

network has a bandwidth, $2B$, that depends on the loaded Q of the resonator. The center of the band is the operating frequency of the oscillator, ω_c. Thus the loaded Q is:

$$Q = \frac{\omega_c}{2B} \tag{2.13}$$

as we would expect. At frequencies that are far from ω_c, the feedback network is an open circuit, so the power spectral density of any phase error at the input of the amplifier should be the same as the power spectral density of the phase error at its output.

$$S_\phi(\omega) = S_{\Delta\Theta}(\omega) \tag{2.14}$$

This uses Leeson's notation for the power spectral density of the phase error at the input of the oscillator and its power spectral density as $\Delta\Theta$ and $S_{\Delta\Theta}(\omega)$ respectively.

At frequencies that are close to ω_c (*ie.* $|\omega - \omega_c| < B$) the feedback network comes into play. Phase errors at the input will result in frequency changes at the output, depending on the phase response of the feedback network. This is extremely important to understand. It means that in this regime the power spectral density of the frequency, shaped by the phase response of the feedback network, is related to the power spectral density of the phase error at the input:

$$S_{\dot{\phi}}(\omega) = \left(\frac{\omega_c}{2Q}\right)^2 S_{\Delta\Theta}(\omega) \tag{2.15}$$

Substituting using Equation (2.5) we get

$$S_{\phi}(\omega) = \left(\frac{\omega_c}{2Q\omega}\right)^2 S_{\Delta\Theta}(\omega) \tag{2.16}$$

The important result here is that the resonator effectively changes the shape of the input noise spectrum, and the shaping factor varies as ω^{-2}.

Leeson applied this analysis to two sources of noise: 1) white noise near the oscillator frequency and 2) flicker noise at frequencies away from the oscillator frequency which are mixed into the vicinity of the oscillator frequency by nonlinearities in the amplifier. Thus, using Leeson's notation style, he broke $S_{\Delta\Theta}(\omega)$ into two components:

$$S_{\Delta\Theta}(\omega_m) = \beta + \frac{\alpha}{\omega_m} \tag{2.17}$$

where ω_m represents the frequency, β represents the contribution of the white noise component, and α/ω_m represents the contribution of the component due to mixing. The white noise component is due to resistive loss in the resonator and the noise figure of the amplifier. At temperature T with a noise figure of F and an input signal level of P_s, $\beta = 2FkT/P_s$. (k is Boltzman's constant.) The term α/ω_m is meant to represent flicker noise at very low frequencies which has been mixed up to the vicinity of ω_c. Here α is an empirical constant that depends on the architecture of the amplifier. Leeson used Equations (2.14) (2.16) and (2.17) to create an asymptotic plot of $S_{\phi}(\omega_m)$ for a 100 MHz crystal oscillator with the specifications given in Table 2.2 Leeson's analysis shows that the flicker noise components, which vary as ω^{-1}, contribute to close-in phase noise that varies as ω^{-3}. At medium offset frequencies, where the flicker

Table 2.2: Parameters for Leeson's oscillator

feedback BW	=	16 kHz
P_s	=	-4 dBm
F	=	9 dB

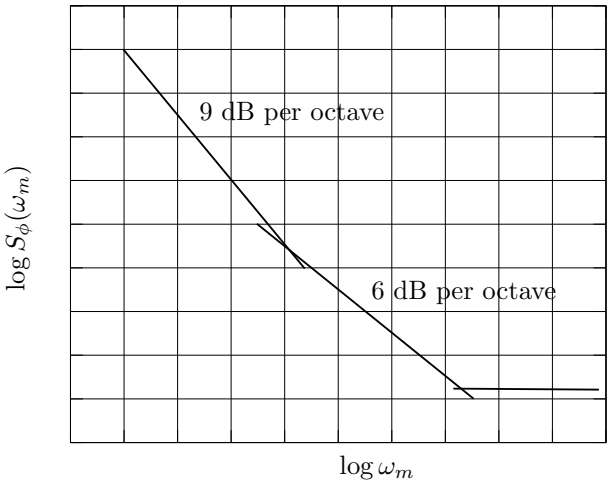

Figure 2.3: Shape of output spectrum for Leeson's oscillator

noise has become negligible, but the shaping effect of the resonator phase response is still in effect, the phase noise varies as ω^{-2}. At larger offset frequencies the response is flat. The "knees" in the graph are not predictable without knowing the empirical constants α and F.

Reverting back to our more familiar notation in equations in Section 2.2.1 above, the power spectral density of the fractional frequency deviation $S_y(\omega)$ in practical oscillators can often be modeled accurately [7] using an equation of the form

$$S_y(\omega) = h_\alpha \omega^\alpha \tag{2.18}$$

where $\alpha \in \{-4, -3, -2, -1, 0, 1, 2, \ldots\}$. Noise processes can be associated with the different values of α. The most common associations are shown in Table 2.3. As we

Table 2.3: α for common noise processes

Process	α	$S_y(\omega)$
Random run of frequency	-4	$h_{-4}\omega^{-4}$
Flicker walk of frequency	-3	$h_{-3}\omega^{-3}$
Random walk of frequency	-2	$h_{-2}\omega^{-2}$
Flicker frequency noise	-1	$h_{-1}\omega^{-1}$
White frequency noise	0	h_0
Flicker phase noise	1	$h_1\omega^1$
White phase noise	2	$h_2\omega^2$

know from Equations (2.5) and (2.6), we can translate from $S_y(\omega)$ to $S_\phi(\omega)$. The equivalent table for $S_\phi(\omega)$ is shown in Table 2.4.

2.3.2 Razavi

Razavi [12, 13] applied LTI analysis to CMOS ring oscillators. A block diagram of a three stage ring oscillator is shown in figure 2.4. In Razavi's case the inverters were implemented as CMOS differential amplifiers with active NMOS loads and variable tail current. Razavi's approach was to use a linearized model of the system which treated each of the inverters in shown in Figure 2.4 as a linear gain stage with an RC load and a noise source representing the thermal noise of the stage as shown in Figure

Table 2.4: Table 2.3 translated for $S_\phi(\omega)$

Process	α	$S_\phi(\omega)$
Random run of frequency	-4	$\omega_c^2 h_{-2}\omega^{-6}$
Flicker walk of frequency	-3	$\omega_c^2 h_{-2}\omega^{-5}$
Random walk of frequency	-2	$\omega_c^2 h_{-2}\omega^{-4}$
Flicker frequency noise	-1	$\omega_c^2 h_{-1}\omega^{-3}$
White frequency noise	0	$\omega_c^2 h_0\omega^{-2}$
Flicker phase noise	1	$\omega_c^2 h_1\omega^{-1}$
White phase noise	2	$\omega_c^2 h_2$

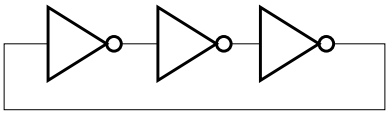

Figure 2.4: Block diagram of Razavi's oscillator

2.5. Assuming that the oscillator is linear — it isn't really — it can be described by

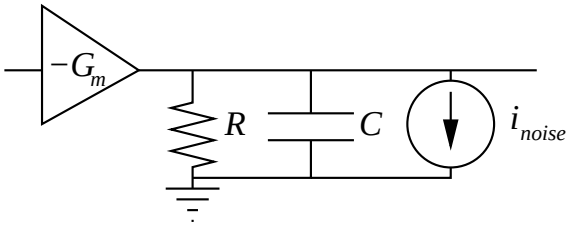

Figure 2.5: One stage of Razavi's linearized model

an equation of the form

$$\frac{Y}{X}(j\omega) = \frac{H(j\omega)}{1 + H(j\omega)} \tag{2.19}$$

representing the linear system shown in Figure 2.6 with X representing a noise source and Y the output. The closed-loop transfer function $\frac{Y}{X}(j\omega)$ is infinite at the oscillator frequency. $H(j\omega)$ is the open-loop transfer function. The sensitivity of the closed loop transfer function to changes in frequency can be assessed by evaluating $\frac{Y}{X}(j(\omega+\Delta\omega))$. For a small change in frequency $\Delta\omega$ the power spectral density of the phase is given

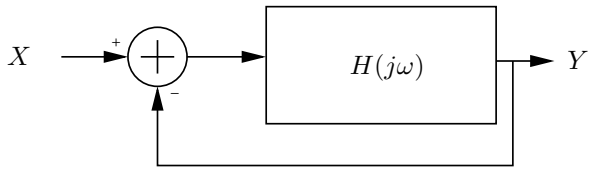

Figure 2.6: Linearized version of oscillator

by

$$S_\phi(\omega) = \frac{1}{(\Delta\omega)^2 \left|\frac{dH}{d\omega}\right|^2} \tag{2.20}$$

This is very similar to the Leeson case. The feedback system produces a phase change when the output frequency is wrong. Ideally, the feedback system is very sensitive to small changes in frequency near the operating frequency of the oscillator so it won't get far off the track. The shape of phase response to frequency errors determines the stability of the oscillator. When the open-loop transfer function of a simple resonator is substituted into Equation 2.20 the results are the same as Leeson's:

$$S_\phi(\omega) = \left(\frac{\omega_0}{2Q\omega}\right)^2 \tag{2.21}$$

The open-loop transfer function of Razavi's oscillator was

$$H(j\omega) = \frac{-8}{\left(1 + j\sqrt{3}\frac{\omega}{\omega_0}\right)^3} \tag{2.22}$$

where ω_0 is the operating frequency of the oscillator. Each stage of the oscillator had an output resistance R and a load capacitance C. Thus, $\omega_0 = \sqrt{3}/(RC)$, and the assumed voltage gain of each stage was near unity. Razavi used Equation 2.22 to show that the power spectrum resulting from a single current noise source i_{noise} is shaped as:

$$\left|\frac{V_1}{i_{noise}}[j(\omega_0 + \Delta\omega)]\right|^2 = \frac{R^2}{27}\left(\frac{\omega_0}{\Delta\omega}\right)^2 \tag{2.23}$$

Thus, similar to Leeson, Razavi used this relation to determine the noise output due to different input noise processes. The first noise source considered was thermal noise due to the resistivity of the channel in the transistors on one side of each differential

amplifier. Because $g_m R = 2$ the noise current on one side of each differential amplifier is given by

$$\overline{I_n^2} = 16kT(g_m)/3 \approx 8kT/R. \tag{2.24}$$

Razavi estimated that the differential pairs turn off for less than 10% of the period of oscillation so simply combining the contribution of the three stages and shaping with Equation (2.23) gives

$$|V_n|^2 = \frac{8kTR}{9}\left(\frac{\omega_0}{\Delta\omega}\right)^2 \tag{2.25}$$

Razavi notes that the result in Equation (2.25) needs to be doubled because of non-linearities. Since the second harmonic of the oscillator is large, noise at $\omega_0 + \omega_n$ results in components at $\omega_0 \pm \omega_n$ when mixed with $2\omega_0$. [1] Finally, noise in the tail transistor is considered. Noise in the tail transistor modulates the impedance of the active load transistors. To account for this, Razavi argues that a noise current $I_n = I_{n0}\cos(\omega_n t)$ results in two components $\frac{\sqrt{3}}{4} I_{n0}\cos(\omega_0 \pm \omega_n)t$ and applying Equation (2.23) results in

$$|V_n|^2 = \frac{R^2}{48}\left(\frac{\omega_0}{\Delta\omega}\right)^2 |I_n|^2 \tag{2.26}$$

2.3.3 Jing Zhang

Jing Zhang has applied LTV modeling to distributed oscillators [20] [21]. A generalized distributed oscillator is shown in Figure 2.7.

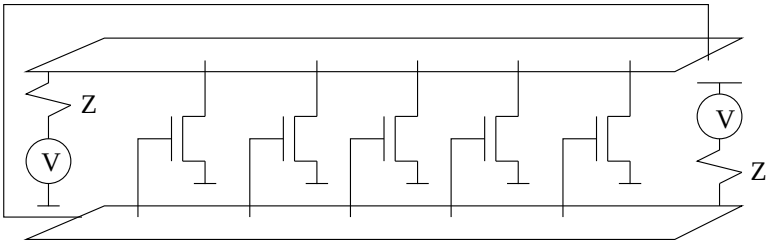

Figure 2.7: Generalized distributed oscillator

[1] Razavi only considers 2ω because of his particular oscillator. In general, the same folding mechanism occurs for all integer harmonics.

It consists of a transmission line along which a number of transistors are distributed. The transistors increase the power in a wave on the transmission line that is travelling from the left to the right. Assuming the total power gain available along the line is greater than unity, connecting the output end of the transmission line to the input end of the transmission line can result in sustained oscillation.

Zhang applies the Leeson/Razavi noise shaping function given in Equation 2.20 above to the open-loop transfer function of the distributed oscillator:

$$H(j\omega) = -g_m \frac{Z_0}{2} \frac{e^{-\alpha_x nl} - e^{-\alpha_d nl}}{\alpha_d l(l - \alpha_g)} e^{-j\beta nl} \tag{2.27}$$

Here the α and β variables are the real and imaginary parts of the complex propagation constant of the various transmission line segments, and Z_0 is the characteristic impedance of the transmission line. Zhang considers the thermal noise contributed by the transistors and the doubling required due to intermodulation, just as in Razavi described above. Since all of the thermal noise in the drains of the transistors travels down the transmission line to the input node, he combines them to give a total noise power density at the input node of

$$\left|V_{\text{in,noise}}\right|^2 = I_n^2 \left(Z_0/2\right)^2 \left(e^{-\alpha_d l} + e^{-\alpha_d 3l} + ... + e^{-\alpha_d (2n-1)l}\right) \tag{2.28}$$

Here the α terms account for the attenuation of the noise as it traverses the line, and the noise current I_n^2 is the same as Razavi's in Equation 2.24 above. Thus the output noise power density that results (after accounting for doubling due to intermodulation) is

$$\left|V_{\text{out,noise-total}}\right|^2 = \frac{I_n^2 \left(Z_0/2\right)^2 \left(e^{-\alpha_d l} + e^{-\alpha_d 3l} + ... + e^{-\alpha_d (2n-1)l}\right)}{(\Delta\omega)^2 \left[\frac{dH(j\omega)}{d\omega}\right]} \tag{2.29}$$

where $\frac{dH(j\omega)}{d\omega}$ is given in 2.27. After some simplifying assumptions 2.29 is reduced to a managable form and the phase noise for a constant carrier power is estimated to be

$$\mathcal{L}(\Delta\omega) \propto 10 \log \left[\frac{g_m R_0}{n \left(C + C_{in}\right)^2}\right] \tag{2.30}$$

In this equation C is the parasitic capacitance of one section of the transmission

line, and C_{in}, R_0, g_m are the input capacitance, output resistance, and transconductance, respectively, of the transistors that make up the distributed amplifier used to implement the oscillator.

2.4 Linear Time-variant Analysis

2.4.1 Hajimiri and Lee

Hajimiri and Lee [15, 16, 17, 18, 19] argue that in general, since practical oscillators have inherent amplitude-limiting mechanisms, amplitude changes due to noise tend to settle out whereas phase changes due to noise persist. They assert that treating the oscillator as a linear system for noise analysis purposes is defensible because noise is just a small perturbation of a nominally stable system. They argue that the assumption of time invariance is unjustified because for example, the response of the system to an impulse depends on the arrival time of the impulse. For example, consider Razavi's ring oscillator. In Figure 2.8 we see an artist's rendition of one side of the output of one stage, for a portion of the oscillation cycle. In (a), we see the unperturbed, idealized negative edge at the output of one stage of the oscillator. In (b) the same edge is considered, but at time τ_1 a current noise impulse is applied to the output node. The output rises temporarily, but the excess charge on the node just causes a temporary reduction in the current from the NMOS pull-up and the amplitude is fairly quickly restored to the original value. In (c) the noise pulse is not applied until time τ_2. But at this point in the waveform the disruption causes the time that the output crosses the input threshold of the next stage to be delayed by $\Delta\phi$. This delay persists for all time. Hajimiri and Lee proposed a mapping function $\Gamma(\omega_0\tau)$ which quantifies the $\Delta\phi$ of Figure 2.8 as a function of position of noise injection in the oscillator cycle.

$$\Delta\phi = \Gamma(\omega_0\tau)\frac{\Delta q}{q_{\max}} \tag{2.31}$$

Here Δq is a test charge which is small compared to $q_{\max}$, the highest charge level reached by the node under test. $q_{\max}$ is easily found by multiplying the total capacitance at the node by the maximum voltage at that node. This is the impulse

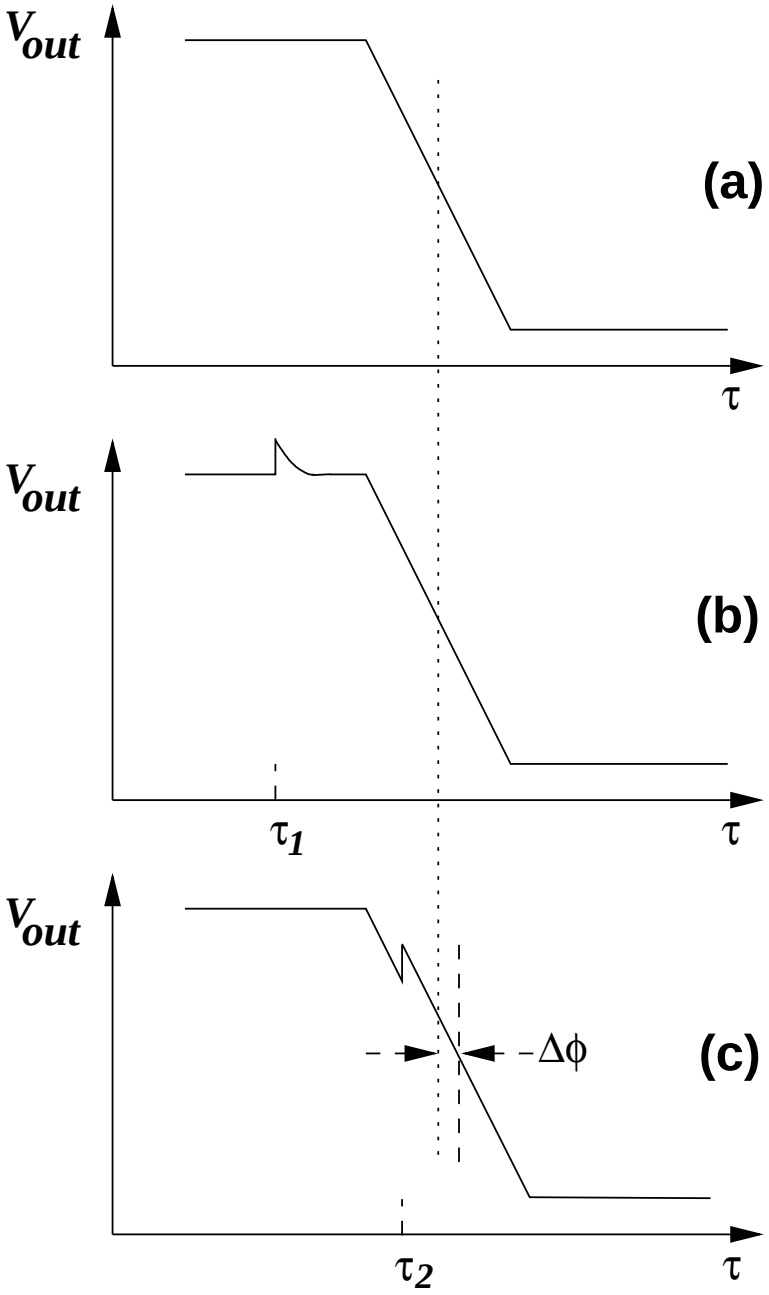

Figure 2.8: Hypothetical impulse response of ring oscillator.

sensitivity function (ISF). Unfortunately, the ISF can only be found analytically for extremely simple oscillators. For practical oscillators it is necessary to use measurements or simulations to derive it. To find $\Gamma(\omega_0\tau)$ by simulation, the oscillator is simulated repeatedly varying the point in time that charge is injected into a node under test and noting the resulting $\Delta\phi$ after the oscillator has stabilized.

Since, from a noise standpoint, we are considering the oscillator to be a linear system we can use convolution to calculate the phase response of the system from the impulse response of the phase. The normalized phase response $h_\phi(t,\tau)$ to an impulse is derived from Equation (2.31).

$$h_\phi(t,\tau) = \frac{\Gamma(\omega_0\tau)}{q_{\max}} u(t-\tau) \tag{2.32}$$

It is normalized in the sense that dividing by $q_{\max}$ makes it independent of the signal amplitude. Here $u(t-\tau)$ is the usual unit step function. Given $h_\phi(t,\tau)$ it is straightforward to find the output phase error due to an arbitrary input $i(t)$ as a function of time

$$\phi(t) = \int_{-\infty}^{t} \frac{\Gamma(\omega_0\tau)}{q_{\max}} i(\tau) d\tau \tag{2.33}$$

The ISF is a dimensionless periodic function, so it can be expanded into a Fourier series

$$\Gamma(\omega_0,\tau) = c_0 + \sum_{n=1}^{\infty} c_n \cos(n\omega_0\tau) \tag{2.34}$$

It turns out that knowledge of the RMS value of $\Gamma(\omega_0\tau)$ is enough to calculate the phase noise spectrum for an oscillator in the f^{-2} region

$$\mathcal{L}(\Delta\omega) = 10\log\left(\frac{\Gamma_{\text{rms}}^2}{q_{\max}^2}\frac{\overline{i_n^2}/\Delta f}{2\Delta\omega^2}\right) \tag{2.35}$$

and this information coupled with c_0, the DC component of $\Gamma(\omega_0\tau)$, is sufficient information to calculate the frequency that the slope changes to f^{-3} as

$$\Delta\omega_{f^{-3}} = \omega_{f^{-1}}\left(\frac{c_0}{\Gamma_{\text{rms}}}\right)^2 \tag{2.36}$$

where $\omega_{f^{-1}}$ is the flicker noise knee frequency. The ISF can handle the complication of a cyclostationary noise input. If the cyclostationary white noise input $i(t)$ is represented as the product of a stationary white input $i(t)$ and a periodic function $\alpha(\omega_0 t)$,

Equation 2.33 changes to

$$\phi(t) = \int_{-\infty}^{t} \frac{\Gamma(\omega_0\tau)}{q_{\max}} \alpha(\omega_0\tau) i(\tau) d\tau \tag{2.37}$$

Hajimiri and Lee have named the function $\alpha(\omega_0\tau)$ the noise modulating function (NMF). It leads to a modified impulse sensitivity function:

$$\Gamma_{\text{NMF}}(x) = \Gamma(x)\alpha(x) \tag{2.38}$$

A similar modification to Equation 2.33 yields the capacitance modulating function (CMF)

$$\beta(\omega_0 t) = \frac{C(\omega_0 t)}{C_{\max}} \tag{2.39}$$

where $C(\omega_0 t)$ represents periodic variation in the capacitance at the test node. In this case the modified impulse sensitivity function is given by

$$\Gamma_{\text{CMF}}(x) = \frac{\Gamma(x)}{\beta(x)} \tag{2.40}$$

Combining multiple noise sources using the ISF technique is somewhat complicated in general, but for stationary noise sources and time-invariant circuit elements it is fairly straightforward. In their many publications on this topic, Hajimiri and Lee have applied this technique to several different types of oscillators with great success. Here the only result we will have time to consider will be their work on ring oscillators [19].

In [19] Hajimiri and Lee apply the ISF technique to differential and single-ended ring oscillators. First they develop an approximate Γ_{rms} for a general ring oscillator based on simulations of a number of different ring oscillator configurations. The reason for this is that generating the ISF for a single noise source is expensive. To generate an n-point ISF for a single noise source test point requires n transient simulations. For m noise sources nm transient simulations are required. The approximation they came up with was

$$\Gamma_{\text{rms}} = \sqrt{\frac{2\pi^2}{3\eta^3}} \frac{1}{N^{1.5}} \tag{2.41}$$

where N is the number of stages and η is the product of the normalized stage delay and the maximum slope of the oscillator's output waveform. The output waveform

edges are assumed symmetric so $\Gamma_{\rm dc}$ (c_0 in Equation 2.34) is nominally zero. They point out that asymmetry in the rise and fall times of the oscillator will increase $\Gamma_{\rm dc}$ and that while the $N^{-1.5}$ term in Equation (2.41) suggests that increasing the number of stages will improve the stability of the oscillator, it also increases the potential number of noise sources.

Using the approximate $\Gamma_{\rm rms}$, and assuming that $\Gamma_{\rm dc}$ is minimized by keeping rise and fall times symmetrical, they find general expressions for phase noise and jitter. Because $\Gamma_{\rm dc}$ is minimized, upconversion of f^{-1} noise is limited, so it is acceptable to ignore the flicker noise components of the transistors and the transistor noise is calculated as:

$$\frac{\overline{i_n^2}}{\Delta f} = 4kT\gamma\mu g_{d0} \tag{2.42}$$

where k is Boltzman's constant, T is temperature, γ is a channel-length proportionality constant, and g_{d0} is the conductance of the transistor at zero bias. They show that for ring oscillators comprised of single-ended inverters the lower bound of the phase noise is inversely proportional to power dissipation and proportional to the square of the frequency of operation. They also show that the increase in the number of noise sources effectively cancels the $N^{-1.5}$ dependence of the number of stages, so the phase noise for single-ended ring oscillators is essentially independent of the number of stages.

For differential ring oscillators, noise contributed by the tail transistor should be a complication because tail current noise at low frequencies and at even multiples of the oscillation frequency will be converted to vicinity of the oscillation frequency. However, Hajimiri and Lee argue that the upconversion of the low-frequency components is minimized similar to the suppression of upconversion of f^{-1} components due to the low $\Gamma_{\rm dc}$ mentioned above, and that the higher frequency components are easily filtered. Thus it is acceptable to consider only the noise contribution of the differential transistors and their loads. They show that unlike the single-ended oscillator, the phase noise does increase with the number of stages.

2.4.2 White, Hajimiri, Wu

In 2002 White and Hajimiri applied Ham and Hajimiri's virtual damping analysis [22] to distributed oscillators [23].

Virtual damping analysis is based on the assumption that the phase noise of an oscillator that is only acted upon by white noise can be described by considering the ensemble average of an infinite number of identical oscillators. Thus, using our standard notation for the output waveform of the oscillator:

$$V(t) = V_0 \cos(\omega_0 t + \phi(t)) \tag{2.43}$$

the variance due to white noise is

$$\left\langle \phi^2(t) \right\rangle = 2Dt \tag{2.44}$$

where D is a "phase diffusion constant" which depends on the rate at which the phase drifts for the oscillator. Thus the ensemble average of $V(t)$ of an infinite number of oscillators experiencing white noise is given by

$$\langle V(t) \rangle = V_0 e^{-Dt} \cos(\omega_0 t) \tag{2.45}$$

Applying this theory to a lumped LC oscillator the phase shift due to a small injected current (*c.f.* Figure 2.8) results in a phase shift given by

$$\Delta\theta = \frac{q}{CV} \sin\theta \tag{2.46}$$

while for the distributed oscillator the analysis results in

$$\Delta\theta = \frac{(q(Z_0 v/2nl))}{V} \sin\theta \tag{2.47}$$

Since the total capacitance of the transmission line is $2nl/vZ_0$ the argument is that the resulting phase noise for a distributed oscillator experiencing pure white noise with power spectral density $i_n^2/\Delta f$ will track and is therefore

$$\mathcal{L}(\Delta\omega) = 10 \log \left(\frac{1/2}{(2nl/Z_0 v)^2 V^2} \frac{i_n^2/\Delta f}{2\Delta\omega^2} \right) \tag{2.48}$$

They go on to show that given a calculation of the ISF of the oscillator, it is possible to augment this equation to handle more complicated noise sources.

2.5 Time-domain Perturbation Analysis

2.5.1 Weigandt, Kim, and Gray

Weigandt used time-domain techniques to study ring oscillators used in phase-locked loops (PLLs) [24]. Weigandt's delay cell consisted of a differential CMOS amplifier with active load and an NMOS current mirror tail nominally pulling a current of I_{SS}. With an output swing of V_{PP} at each active load and capacitive loading per stage of C_L, the delay per stage is approximately

$$t_d \approx V_{PP}\left(\frac{C_L}{I_{SS}}\right) \tag{2.49}$$

Using the zero crossing of the differential output voltage as the beginning of the switching time it is possible to approximate how noise voltages and currents introduce randomness into the delay between switching times. Small noise voltages shift the slewing waveform slightly positive or negative as shown in Figure 2.9. Thus the

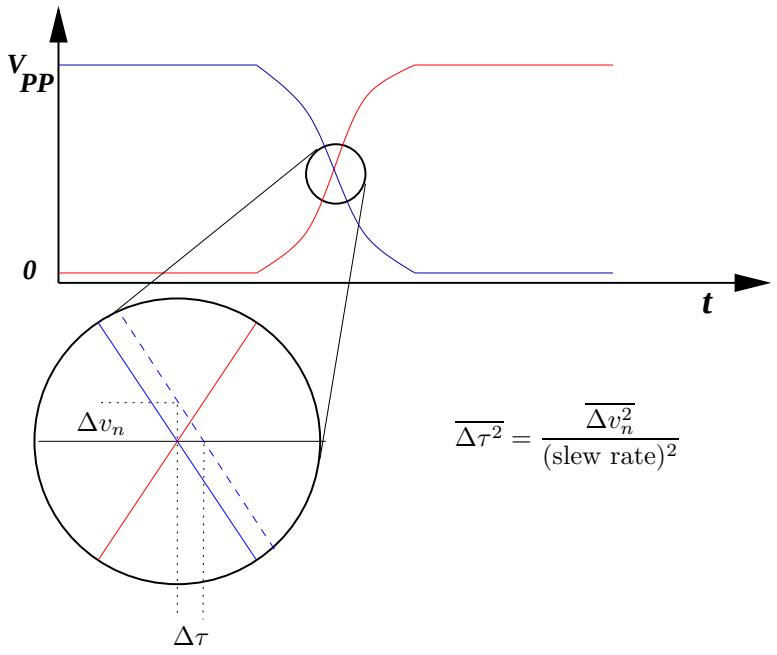

Figure 2.9: Effect of noise voltage Δv_n on delay time

variance of the timing error is approximately

$$\overline{\Delta\tau^2} \approx \overline{\Delta v_n^2}\left(\frac{C_L}{I_{SS}}\right)^2 \tag{2.50}$$

Here, $\overline{\Delta v_n^2}$ is the variance of the noise voltage. Weigandt first shows a first-order analysis which assumes that $\overline{\Delta v_n^2}$ is due to thermal noise in the MOSFET drains and is time-invariant. In this case the noise current is given by

$$\overline{i_d^2} = 4kT\left(\frac{2}{3}g_m\right)\Delta f \tag{2.51}$$

where k is Boltzman's constant, T is the temperature, g_m is the transconductance of the transistor, and Δf is the frequency band. Here Weigandt is assuming that g_m is constant and Δf is determined by the RC time constant formed by the output impedance of the stage and the input capacitance of the next stage. Using these simplifications he determines that the normalized jitter is

$$\frac{\Delta\tau_{\text{rms}}}{t_d} = \frac{1}{V_{PP}}\sqrt{\frac{2kT}{C_L}}\sqrt{1+\frac{2}{3}a_v} \tag{2.52}$$

In this equation, a_v is the small-signal gain of the inverter. In general $\overline{\Delta v_n^2}$ is time-variant because g_m changes with the operating point of the transistor. To accommodate this Weigandt provides a second order analysis. In the second order analysis tail current noise seen at the output of a stage is excluded during the switching transients but included when one side is fully on and the other side is fully off. The idea is that the current noise generated by the tail transistor affects both sides of the differential amplifier equally when the amplifier is "balanced" and thus does not contribute any differential output voltage. When the amplifier is "unbalanced," with one side turned on and the other turned off, noise current in the tail transistor leads directly to noise voltage in the differential output. This is a factor because the balanced and unbalanced regimes tend to overlap from stage to stage in a multistage oscillator. In this case determining $\overline{\Delta v_n^2}$ is more complicated. In this case the noise processes must be analyzed in the time domain and combined using convolution. The resulting normalized jitter is given by

$$\frac{\Delta\tau_{\text{rms}}}{t_d} = \frac{1}{V_{PP}}\sqrt{\frac{2kT}{C_L}}\sqrt{1+\frac{2}{3}a_v(1-e^{-t/\tau})+\frac{2\sqrt{2}}{3}a_v e^{-t/\tau}} \tag{2.53}$$

In this equation the time constant $\tau \approx t_d$, so Weigandt argues that the exponentials $e^{-t/\tau}$ decay quickly and that this indicates that the design parameter with the most impact on noise is the gain a_v. Extending this analysis to the point that multiple stages are simultaneously in the active amplification regime, $\overline{\Delta v_n^2}$ is increased by a factor of $a_v^2/2$. In this case $\overline{\Delta \tau^2}$ is proportional to a_v and inversely proportional to $V_{GS} - V_T$. This implies that increasing the supply current and reducing gain are factors that can be used to minimize the timing jitter.

2.5.2 McNeill

McNeill studied time-domain analysis of differential bipolar ring oscillators for use in phased-locked loops (PLLs) [25, 26, 27]. The 1997 paper [25] is a good summary of this work. In this paper he shows the development of a design figure of merit, κ, which can be used to relate circuit level noise to system level jitter in PLLs. As discussed in Section 2.2.2, the square root of the classical variance is a common method of specifying jitter and for many noise processes the classical variance diverges. This is because time uncertainty in one cycle of oscillation generally affects the uncertainty of the next cycle and this effect is cumulative. McNeill asserts that for many noise processes the jitter varies proportionally to the square root of the time interval over which it is accumulated, with a κ as the constant of proportionality

$$\sigma_\tau = \kappa\sqrt{\tau} \tag{2.54}$$

where τ is the measurement interval. This is because the jitter measured over the interval τ is the sum of a number of independent jitter contributions that accumulate during the measurement interval. As the length of the interval increases the number of contributions increases. The standard deviation of the sum increases as the square root of the number of contributions.[2] McNeill notes that κ is independent of the number of stages in the ring. This is because a little jitter is accumulated at each delay stage but only while that delay stage is transitioning. So the total jitter accumulated over the time interval τ depends on how many transitions occurred, not on how many

[2]Hajimiri and Lee [15] point out that for correlated noise sources the jitter increases proportionally to τ.

complete cycles around the ring occurred. Thus, κ can be predicted by determining the jitter contribution of each stage.

McNeill's delay stage is shown in Figure 2.10. He focused mainly on the following

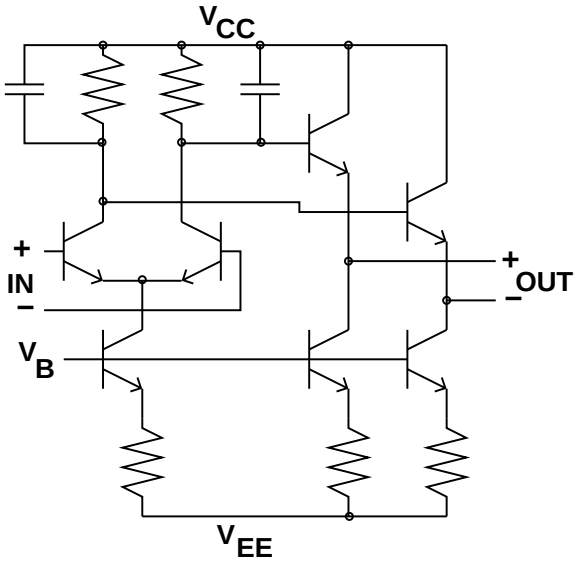

Figure 2.10: McNeill's delay stage.

noise sources:

- Thermal noise in the differential pair's load resistors
- Tail current noise
- Sampling noise at the differential pair input

He made a number of simplifying assumptions. First, that stage delay was strictly due to the RC time constant of the loads of the differential amplifier. Second, that the differential signal amplitude was large compared to the thermal voltage V_T. Finally, that all noise sources are much smaller than the differential signal voltage and uncorrelated.

His analysis is similar to Weigandt's in Section 2.5.1 in that he determines the time shift in the edge of a stage's output voltage from the magnitude of the noise. In calculating the shift due to thermal noise in the differential pair load resistors he assumes that the differential pair works like an ideal switch. Thus the output voltage

on each half of the differential pair looks like an ideal exponential decay or rise with a slope at the time of zero crossing, denoted t_{zc}, given by

$$\left.\frac{dv_o}{dt}\right|_{t=t_{zc}} = \frac{I_{EE}}{C} \tag{2.55}$$

In this equation v_o is the output voltage, I_{EE} is the tail current, and C is the capacitive load. Thus small errors in the output voltage are related to time via

$$\frac{\sigma_v}{\sigma_t} = \frac{I_{EE}}{C} \tag{2.56}$$

The two pullup resistors combined contribute thermal noise of $\sigma_v = \sqrt{2kT/C}$, and the average delay of the stage is $T_d = RC \ln 2$. Obviously, κ for one stage is $\sigma_t/\sqrt{T_d}$, so combining all this information yields κ_R, the κ due to the thermal noise in the pullup resistors:

$$\kappa_R = \sqrt{\frac{2}{\ln 2}}\sqrt{\frac{kT}{I_{EE}^2 R}} \tag{2.57}$$

In analyzing the tail current noise, McNeill runs into the same difficulty we have already seen in Sections 2.3.2 and 2.5.1. When the stage is near the switching point, the effect of noise in the tail is reduced because it affects both sides of the differential amplifier equally. He simplifies the situation by assuming that the differential pair switches instantaneously. Thus the voltage that is seen at the bottom of the pullup resistors just before the switching time can be calculated from i_{EE_n} the noise current in the tail. The noise voltage is band-limited because of the RC time constant of the pullup system. His result is that the standard deviation of the output voltage error at the time of zero crossing is given by

$$\sigma_v = \frac{i_{EE_n}}{2}\sqrt{\frac{R}{C}} \tag{2.58}$$

where i_{EE_n} is the tail current noise. Thus, using equation (2.56) and the fact that $T_d = RC \ln 2$ we find that

$$\kappa_{tail} = \frac{1}{\sqrt{2 \ln 2}}\frac{i_{EE_n}}{I_{EE}} \tag{2.59}$$

Using this relation it is possible to evaluate κ_{tail} for whichever noise process is of interest.

Finally, we consider McNeill's modelling of the noise due to the parasitic base resistance in the input transistors and the output resistance of the preceding stage. This noise is presented to the input of the amplifier, and the resulting effect on the output of the amplifier is complicated because the gain of the amplifier is not constant. Since we are interested in the σ_v at the zero crossing, it is possible to estimate the incremental gain at this point. If the input noise is lumped into a single white noise voltage e_n, σ_v at the zero crossing is given by

$$\sigma_v = \frac{e_n}{2}\sqrt{\frac{I_{EE}}{3CV_T}} \tag{2.60}$$

Combining this with Equation (2.56) and the fact that $T_d = RC\ln 2$ we have

$$\kappa_{samp} = \frac{e_n}{2\sqrt{3\ln 2}}\sqrt{\frac{1}{I_{EE}RV_T}} \tag{2.61}$$

The term e_n can be replaced with the appropriate base resistance or output impedance thermal noise term as required.

Since all noise sources are assumed independent it is possible to combine them as a root sum of squares (RSS):

$$\kappa = \sqrt{\kappa_R^2 + \kappa_{tail}^2 + \kappa_{samp}^2} \tag{2.62}$$

2.5.3 Leung

In 2004, Bosco Leung attempted to improve on the models of Weigandt and McNeill by more accurately modelling the trajectory of the input stage voltage near the threshold [28]. Leung's theory is that the transition time of a stage should be measured at the time that the output voltage of that stage makes its last transition in the direction opposite of the input voltage. The argument is that this point truly marks the time that the stage has actually transitioned. This "last passage time" model is supposed to provide a more accurate per-stage $\Delta\tau$ (see Figure 2.9) and thus a more accurate jitter for the oscillator as a whole.

Leung's technique consists essentially of modelling the thermal noise as the derivative of a Brownian motion process. That is, the delay cell can be modeled as a

capacitor that is being charged by a current source

$$I = I_{dc} + I_n \tag{2.63}$$

where I_n is due to white thermal noise. Substituting I_n with $\sigma_I B'_t$ where σ_I is the variance of I_n and B'_t is the derivative of a Brownian motion process results in a stochastic differential equation for the charging of the capacitance

$$I = I_{dc} + \sigma_I B'_t \tag{2.64}$$

This can be solved to yield

$$V = \frac{I_{dc}}{C} t + \frac{\sigma_I}{C} B_t \tag{2.65}$$

where I_{dc}/C and σ_I/C correspond to parameters needed to determine the pdf of a random variable describing the last passage time. Thus it is possible to numerically determine the expected value of the last passage time and more accurately determine the jitter of the oscillator.

2.6 Summary

This chapter has introduced the concepts from linear time-invariant analysis, linear time-varying analysis, and time-domain perturbation analysis that are fundamental to building an understanding of the noise mechanisms in DLTWOs. The concept of relating voltage shifts to slew rates as described in Sections 2.5.1 and 2.5.2 is applied in Chapter 3 in the analysis of the frequency stability of the DLTWO. The introduction of McNeill's figure of merit κ in Section 2.5.2 is important to the understanding of the results in Chapter 4. Hajimiri and Lee's impulse sensitivity analysis introduced in Section 2.4.1 is extremely important in generating the simulation results of Chapter 4 that are used to verify the validity of the analysis of frequency stability in DLTWOs which is presented in Chapter 3.

Chapter 3

Phase Noise and Jitter in DLTWOs

3.1 Introduction

Figure 3.1 shows a snapshot of an instant in time in the operation of a simplified DLTWO which is supporting a wavefront that is travelling from left to right in the diagram. The diagram is simplified mainly in that the DLTWO pictured consists of only six "stages," or inverter pairs. The position of the wavefront is shown schematically by the plus and minus signs to be about halfway across the ring at the time of the snapshot. At the time shown in the diagram all of the latches are latched except

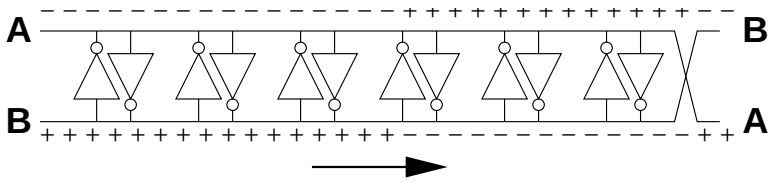

Figure 3.1: Simplified six-stage DLTWO

for the fourth one from the left, which is changing state. In fact, at any given time, all of the latches in a DLTWO except a very small number of latches that are at varying points in the process of changing state as the wavefront traverses them will be latched. Simulations show that it is common for the wavefront to only straddle one or two stages at a time, depending on the rise time, the number of stages in the ring, and the nature of the transmission line. For example Figure 3.2 shows typical

simulated output at three adjacent stages of a realistic DLTWO. As a result, due to

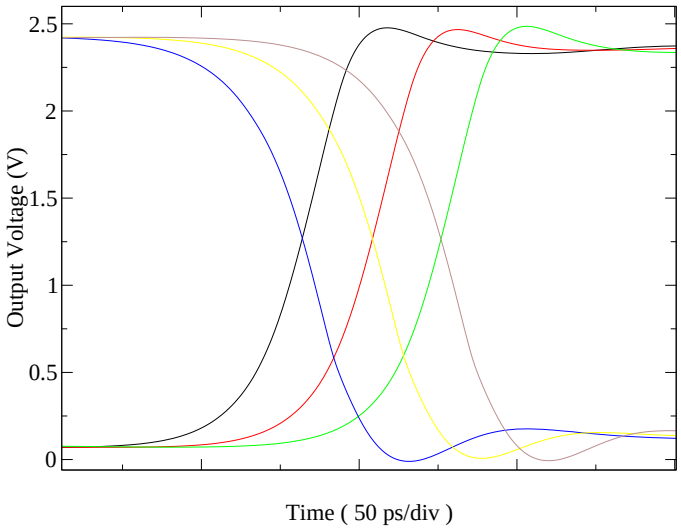

Figure 3.2: Simulation output showing waveforms of three adjacent DLTWO stages

rotational symmetry Figure 3.1 effectively describes the state of the DLTWO at any given time.

This situation makes phase noise and jitter analysis of the DLTWO particularly interesting. Figure 3.3 shows a closeup and more detailed view of a short section of a much larger ring. In Figure 3.3 the wavefront (depicted in the same manner as in Figure 3.1) is about to arrive at the latch that is shown schematically as an inverter pair. To simplify the discussion, the assumption is that the edge rate is such that the voltage on the differential transmission lines is only transitioning in the vicinity of the expanded latch. Only one latch is transitioning and the surrounding latches are therefore securely latched one way or the other, so they are shown as low impedances clamping the differential transmission line to the appropriate rail depending on the state of the latch. Typical on-resistance for these clamps will be in the small tens of ohms or less, depending on the technology selected for the implementation of the oscillator. In parallel they represent a resistance on the order of 1 Ω. Consider the top differential transmission line conductor, labeled A in Figure 3.3. To the left of the transitioning latch there is a bank of low impedance devices pulling the line to the

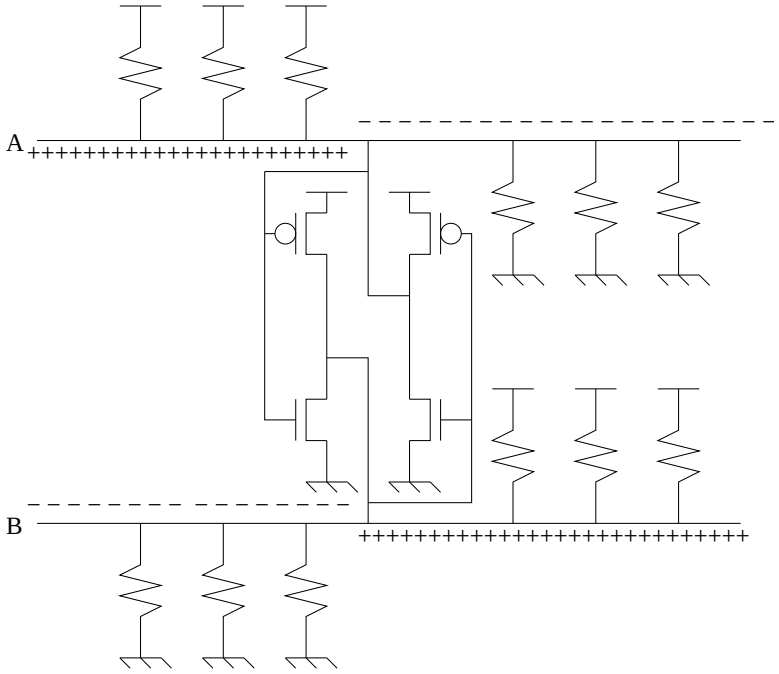

Figure 3.3: Closeup of DLTWO stage with wavefront passing

positive rail V_{dd}. To its right, a very similar situation exists with another determined bank trying to pull the same line to the negative rail V_{ss}. Between them is a very short length of a very conductive metal. Clearly there is potential in this situation for a lot of current to flow.

The wavefront is flowing from left to right in Figure 3.3. The expanded latch is initially in the same state as the latches to its right. As the wavefront approaches, energy in the wavefront causes positive charge to build up on conductor A and negative charge to build up on conductor B near the expanded latch in such a way as to cause it to flip state. At first, the P-channel transistor on the left and the N-channel transistor on the right resist this tendency. Early in the transition only one transistor in each latch is a factor. At this point the latch is actually impeding the forward progress of the wavefront. However, the low characteristic impedance of the transmission line allows the wavefront to deliver charge faster than the latch transistors can drain it off, so the voltages on the conductors of the differential transmission line start to transition. As the transition nears the point where the transmission line conductors are about a V_T away from the rails the latch stops resisting as it enters its negative resistance region. Here the latch becomes an amplifier, and it starts to aid in the transition. When the transitioning voltages pass the threshold voltages at the other end of the transition the latch returns to the resistive state with one transistor fully on and one transistor fully off until the next time the wavefront comes around. This scenario explains the shape of the wavefronts shown in Figure 3.3. At the beginning of both the negative-going and positive-going transitions, the latch is fighting against the voltage change on the differential transmission line, so the slope of the transition is more gradual. Once the latch enters into its negative resistance region, all of the energy in the wavefront goes toward charging the transmission line and slope of the transition is maximized all the way to the rail.

3.2 Perturbation Analysis

Obviously the action of the latches has an effect on the progress of the wavefront. In a perfect DLTWO each latch will react to the approaching wavefront the same way every time it comes by. As the wavefront nears, it will initially provide exactly the same amount of resistance to the progess of the wavefront every time and it will always enter the negative resistance (amplification) region and capitulate at the same point in the transition. In the amplification region it will always amplify the wavefront by the same amount at any given point of the transition. In this case the forward progress of the wavefront will be constant and no phase error will be introduced.

In a real DLTWO noise on the wavefront will cause changes in the action of the latch which will affect the speed of propagation of the waveform. Consider three different stages in the passing of the wavefront:

Stage 1: the wavefront is near the latch but there is no voltage change on the transmission line in the vicinity of the latch.

Stage 2: the wavefront has reached the latch but it is resisting.

Stage 3: the latch has capitulated and become an amplifier.

These three stages are shown schematically in Figure 3.4. To keep this figure less cluttered only one side of the differential transmission line is shown. Clearly, a differential noise voltage on the latch has no effect on the speed of the transition during Stage 1. Stage 2 is different. At the beginning of Stage 2 the latch is most strongly resisting a change in the voltage on the transmission line. This resistance to the voltage change represented by the wavefront also applies to voltage change caused by noise. As Stage 2 progresses the latch's resistance to voltage change steadily decreases. As a result, voltage noise can change the local slope of the waveform. The situation is very similar to the situation described in the discussion of the work of Weigandt in Section 2.5.1. At the very end of Stage 2, as the latch transitions into Stage 3, it amplifies the noise voltage. Thus it is possible for noise voltage to either prematurely end Stage 2 or artificially prolong Stage 2 depending on the instantaneous polarity

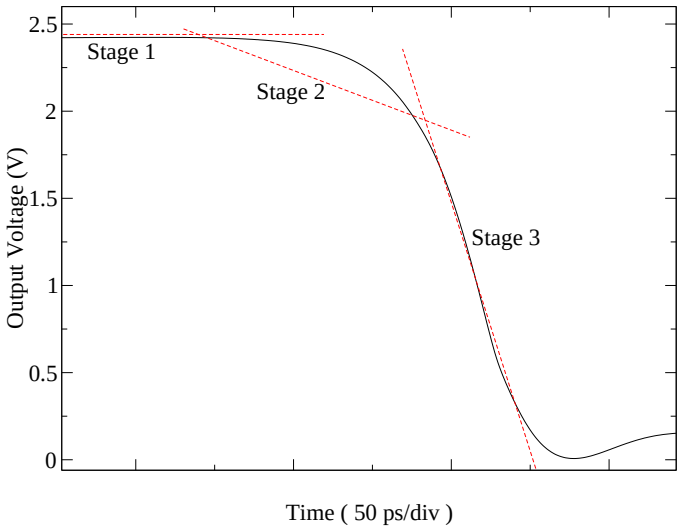

Figure 3.4: Three stages in the perturbation analysis

of the noise. Once Stage 3 is entered the noise continues to be amplified throughout most of the transition. During Stage 2, the slope of the transitioning waveform is more gradual than it is in Stage 3. Thus as suggested by Weigandt's results, voltage shifts due to noise will have a greater impact on the instantaneous phase of the waveform during Stage 2 than they will during Stage 3. Figure 3.5 shows the ramifications of this analysis. During Stage 1 noise in the latch has zero effect on the progress of the wavefront. Noise affecting the latch can expand or contract the length of Stage 2 effectively moving the time at which the slope changes to the maximum slew rate forward or backward. This effect is most pronounced with noise that arrives at the end of Stage 2 when the noise is being amplified. The combination of amplification of the noise with the slow slew rate during this part of the waveform exaggerates the effect of noise that arrives at this time. Noise that arrives during Stage 3 changes the instantaneous slope, but since the slope is at its maximum here the effect on phase change is not as great. The final result is that the we should expect the DLTWO stage to be totally immune to noise that does not arrive during a transition, and that the DLTWO stage should be most vulnerable to noise that arrives just as the latch is entering Stage 3, the amplification stage.

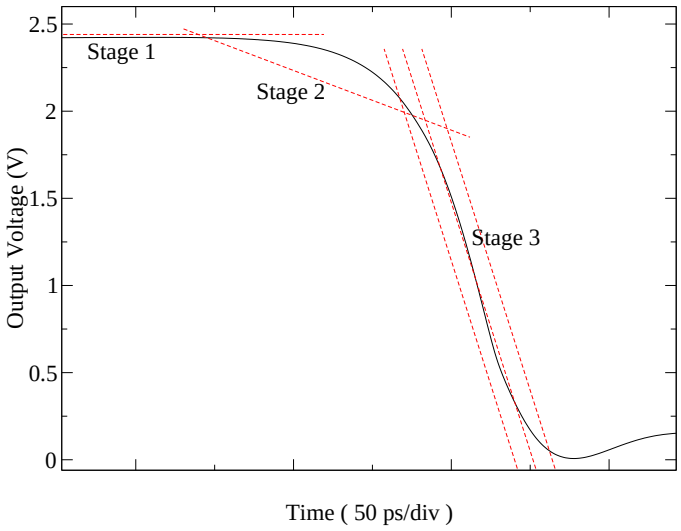

Figure 3.5: Ramifications of changes in the duration of Stage 2 for phase error.

Quantitative analysis of the situation is fraught with problems. The main problem is that the stages do not work strictly serially like the stages in Weigandt's oscillators. In a practical DLTWO it is quite typical for two neighboring stages to be experiencing a transition simultaneously, albeit in different Stages. For example, back in Figure 3.2 the waveforms for three adjacent stages are shown. When the first stage to switch is in the middle of Stage 3, the second stage is just leaving Stage 3. Thus at this point the amplification of the noise voltage is much larger than it would be for either of the latches independently. In essence these two stages are working almost in parallel, but not quite. Depending on the number of stages used to implement a DLTWO, the transistor sizing, loading, and a number of other parameters, the level of cooperation between any two stages is very difficult to quantify accurately. Thus accurate frequency stability analysis of the DLTWO requires simulation.

Nevertheless, it is possible to use simplified quantitative analysis to get ballpark numbers as a sanity check. The first step is to determine the sources of noise. In spite of the large number of active devices that comprise the DLTWO, the architecture is extremely simple. In one sense, the entire oscillator only consists of two nodes other than the supply rails. At least if the oscillator is broken up into stages, one

latch per stage, there are only two nodes per stage outside the rails, one node for each transmission line conductor. The obvious suspects for noise sources are the MOSFETs that make up the latches. The beautiful symmetry of the DLTWO makes it very easy to combine these noise sources into an effective differential voltage noise seen across the latch. As shown in Figure 3.3, each inverter input is connected to one of the differential transmission line conductors and the rest of the conductor is tied to the rail through low-impedance transistor pull-ups or pull-downs. Thus the noise on each rail is essentially due to a large distributed transistor so the total noise on each transmission line conductor is simply

$$\overline{v_n^2} = \frac{kT}{C} \tag{3.1}$$

where C is the total capacitance on the conductor and the total noise presented to the stage is the root sum of squares of the noise produced on the two conductors. This noise source is so large for a typical DLTWO that it is possible to completely ignore the noise current generated by the transistors in the transitioning latch without changing the results appreciably. Another possible source of noise is the supply rails, but the architecture of the DLTWO makes it very resistant to this. This phenomenon is covered in Chapter 5.

Given the noise on each of the transmission lines in Equation 3.1 the question is how to estimate the excess phase. One way to do this is to consider the model shown in Figure 3.6. The situation shown in the figure corresponds to a wavefront approaching a latch from the left side, with the A side of the differential transmission line positively charged on the left and discharged on the right. The opposite situation exists on the B conductor. The arrows indicate the direction of current flow. For the wavefront to propagate, the charge in C_{ll} must be flipped, C_{ls_A} must be charged and C_{ls_B} must be discharged. To do this, a total of

$$Q_{total} = 2 * V_{dd} \left(C_{ll} + C_{ls} \right) \tag{3.2}$$

Coulombs of charge must be transferred to the capacitors from the wavefront assuming that $C_{ls} = C_{ls_A} = C_{ls_B}$. Noise voltage across the latch causes a variance in the charging/discharging current. For most of the transition (from a voltage standpoint)

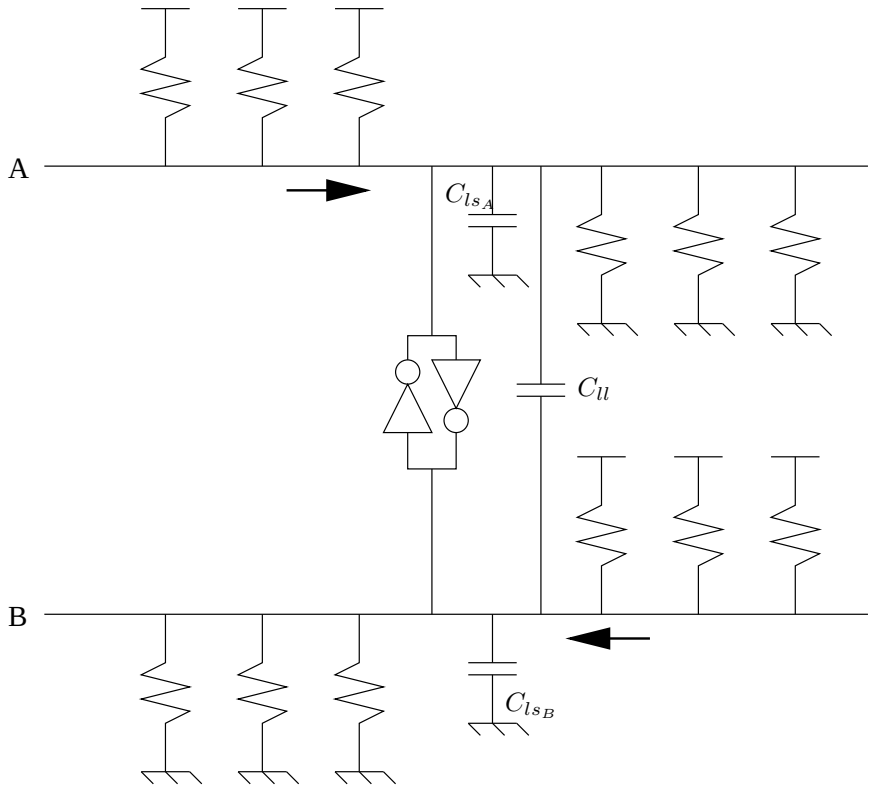

Figure 3.6: Model for estimating excess phase due to noise

the differential transconductance of the latch amplifies this noise voltage, resulting in a noise current

$$\overline{i_n^2} = {g_M}^2 \overline{v_n^2} \tag{3.3}$$

where $\overline{v_n^2}$ is the noise voltage from Equation 3.1 and g_M is the differential transconductance of the stage. The ratio of $\overline{i_n^2}$ to Q_{total} is a direct estimate of the time variance of charging/discharging. Once this is done per stage the combination of the effects of multiple stages and calculation of κ are the same as described in the Section on McNeill (Section 2.5.2).

Of course, this analysis is complicated by the fact that g_M is hard to determine analytically. It is easiest to get an accurate idea of g_M via simulation. In the case of the DLTWO this is especially true because it may be necessary to consider multiple stages simultaneously.

3.3 Multiple rings

Consider two rings connected as shown in Figure 3.8. A closeup of the intersection of

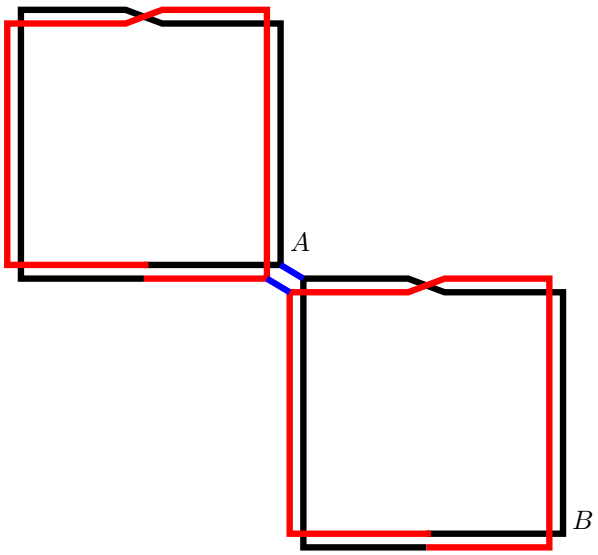

Figure 3.7: Connection of two DLTWO rings

the two rings is depicted in Figure 3.8 The arrows in Figure 3.8 represent incoming

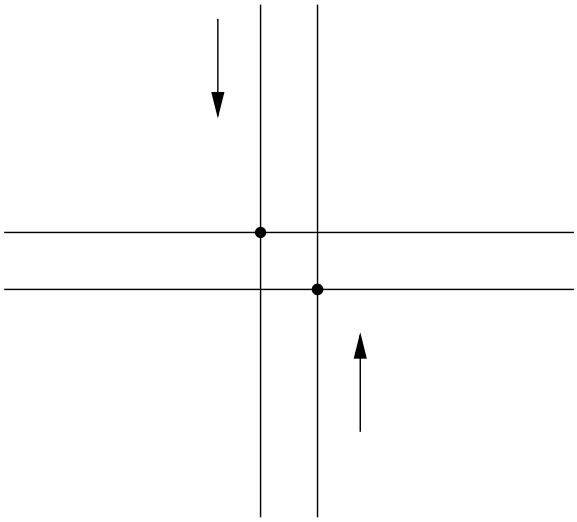

Figure 3.8: Closeup of intersection of two rings from Figure 1.4

wavefronts on the two rings. If the rings are perfectly synchronized, the junction of the transmission line appears to be perfectly matched to each wavefront. If one of the wavefronts arrives earlier than the other, the earlier–arriving wavefront sees a much different situation than it does in the synchronized case. In this case the earlier–arriving wavefront slows down a little because it is suddenly faced with three times its normal load. The later–arriving wavefront arrives very soon afterward (because we are assuming very small time variances due to noise sources) and reinforces the work done by the earlier–arriving waveform. It is effectively sped up a little. Thus, connecting a ring with phase error with a ring that has no phase error produces a system with two rings, each of which have the half the phase error. This principle applies linearly with the number of rings. Unfortunately, increasing the number of rings also increases the number of noise sources linearly, so the RMS phase error of a system with n rings is the RMS sum of the phase error of n rings divided by n. Simulations bearing out this analysis are discussed in Chapter 4.

3.4 Summary

Phase error in the DLTWO can be analyzed in much the same way as the differential ring oscillators analyzed by Weigandt and McNeill as described in Sections 2.5.1 and 2.5.2 using time-domain perturbation analysis techniques. The key is to understand the noise processes unique to the DLTWO and to understand how the unique architecture of the DLTWO is affected by these noise processes. The main novel contribution of this chapter is a detailed description of the genesis of phase error in the DLTWO stage as the wavefront traverses it. Specifically, excess phase is introduced when noise generated in the non-transitioning stages is communicated to the transitioning stage via the differential transmission line which links all the stages together. At the beginning of the transition the latch in the transitioning stage resists the change of state on the differential transmission line reducing the slew rate. As the transition proceeds the latch enters its negative-resistance region becoming an amplifier. The combination of amplification with reduced slew rate causes the effect of noise on the edge to introduce maximum phase error at this point in the transition. Outside of the transition time no excess phase is introduced by any stage.

An important benefit of the identification of the noise sources unique to the DLTWO is that this information is required for performing ISF analysis of the oscillator.

The final contribution of this chapter is an analysis of the effects of linking multiple rings together. The analysis predicts that the averaging effect expected for multiple rings will be effectively offset by the attendant increase in the number of noise sources.

All of the analysis in this chapter is verified by simulation and measurement in Chapter 4.

Chapter 4

Simulations and Measurement Results

This chapter presents the details of the simulation techniques used and their results and compares this information with measurement data from actual DLTWO test chips.

4.1 Test chip

Figure 4.1 shows a die photograph of the test chip used to help verify the validity of the models proposed here. This chip, first described in [1], was the first working on-chip DLTWO. Provided by Multigig, it was fabricated at MOSIS using their 0.25 μm 2.5 V CMOS process. The main characteristics of the process are shown in Table 4.1. The test chip contains two independent DLTWOs. The first is a single ring DLTWO,

Table 4.1: Test chip process characteristics

Technology	0.25 μm CMOS
Metal layers	5
Metallization	Al/Cu

which consists of a 12000 μm differential transmission line comprised of two coplanar 60 μm conductors on a 120 μm pitch. This transmission line has 128 62.5 μm/25 μm inverter pairs distributed along its length. This DLTWO is highlighted in Figure 4.1. The 12000 μm single-ring DLTWO worked very well but not perfectly.

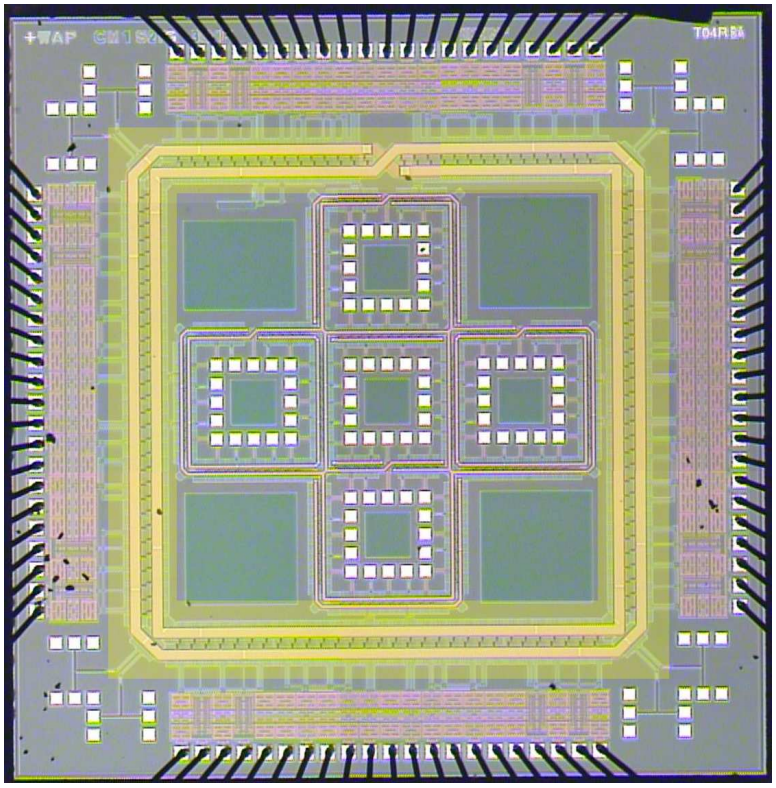

Figure 4.1: Die photo of test chip. 12000 μm DLTWO highlighted yellow.

The second DLTWO consists of four interconnected smaller rings. This DLTWO is easily seen in Figure 4.1 as being inside the single-ring DLTWO. The way that the rings are interconnected makes the oscillator appear to have five rings but it is really more accurate to think of it as having four. These rings use 20 μm conductors on a 40 μm pitch. Each ring's coplanar differential transmission line is 3200 μm long and each ring has only 40 125 μm/50 μm inverters distributed around it. This DLTWO functioned, but its performance was hampered by a layout flaw which was described in a previous paper [4] so it was not really useful for jitter or phase noise measurements.

4.2 Simulations and ISF analysis

4.2.1 Introduction

Proper simulation of the DLTWO is a complicated problem. The main tools used here are the SPICE [30] circuit simulator, a capacitance extraction tool named FASTCAP [31], and an inductance calculator named FASTHENRY [32].

Many variants of SPICE are available, and of course entire libraries exist on the subject of semiconductor device modelling with SPICE. The results presented here were calculated using HSPICE Version 2001.2 from Synopsys Inc. using binned BSIM [33] version 3 MOSFET models provided by the foundry that built the test circuits. These models are proprietary and therefore are not included in this document. Many simulations have also been performed using NG-SPICE [34], an open-source SPICE variant and non-quasi-static BSIM version 3.3 MOSFET models. NG-SPICE simulations were used extensively to test theories and work out details because (at the time) NG-SPICE could be run on much faster computers and on more computers at North Carolina State University and the results tracked with the HSPICE results very well.

4.2.2 Basic DLTWO simulation techniques

No unusual techniques are required to perform accurate simulations of DLTWO circuits. Standard semiconductor simulation practices are adequate. This is not to say

that it is particularly easy; there is a lot of painstaking work required to get an adequate model. The purpose of this subsection, however, is simply to document the level of attention required, and to foster more confidence in the results to be reported later.

The basic approach used in the simulations was to break each ring into a number of stages, with each stage consisting of one of the distributed latches embedded in a short length of transmission line. This approach does not mesh well with the traditional parasitic extraction technique because the extractor software typically doesn't handle transmission line modelling. Instead, parasitics were calculated analytically. This approach is also attractive because it makes generalizing the simulation techniques easier.

Transmission lines were modelled as lumped LC segments. This is reasonable because the stages are small compared to the wavelength of the energy propagating along the transmission line. Figure 4.2 shows the layout of one stage of the 12000 μm single-ring DLTWO, which is about 1/40 of a wavelength long at the fundamental frequency of the oscillator. For a DLTWO with more stages the ratio would be even more favorable for modeling the transmission line segment as a lumped LC segment. Even for energy at high multiples of the oscillator frequency this is not a problem.

FASTCAP and FASTHENRY were used to extract capacitance and inductance models for the line-to-line elements of differential transmission line segments. General-purpose scripts (cf. Section A.10) were written for automatically doing the discretizations required to run these tools on several different types of transmission line segments. Figure 4.3 shows an example of a typical FASTCAP discretization for the crossing of two differential transmission line segments. More examples are provided in Section A.10.1 of the Appendix. Extensive experimentation showed that FASTCAP discretizations based on dividing the conductor width by 10 and keeping the segments relatively square provided accurate extractions at high speed. In order to accurately model skin and proximity effects a fairly fine FASTHENRY discretization of 9 was used in the horizontal direction while the thickness of the conductor allowed a

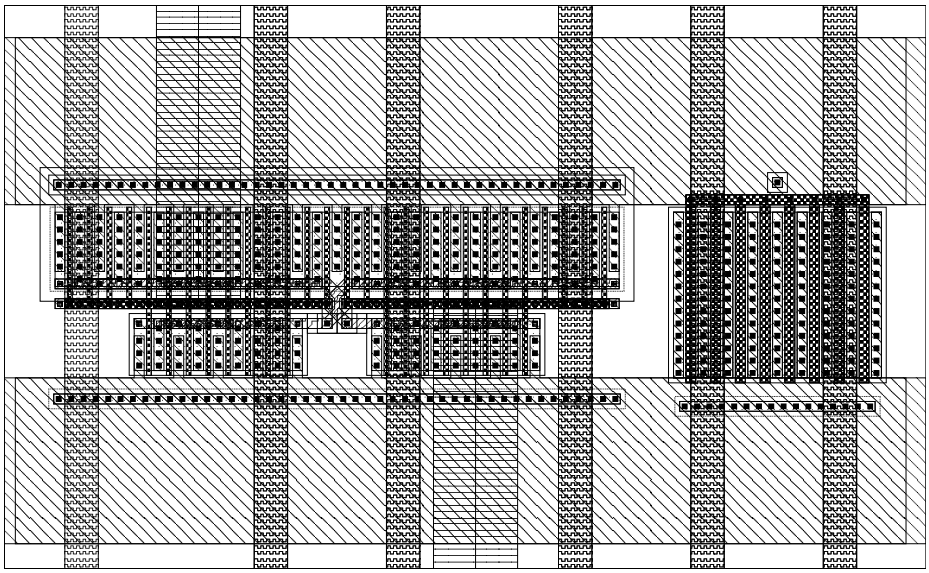

Figure 4.2: Typical stage layout of single-ring DLTWO

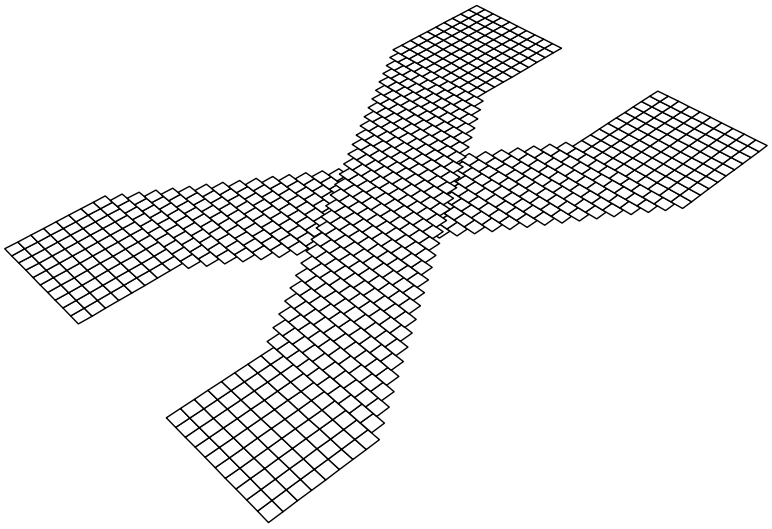

Figure 4.3: Fastcap discretization showing crossing of transmission line

more reasonable discretization of 3 to be used in the vertical direction. Simple formulas were used to handle the line-to-substrate elements of the differential transmission lines.

Parasitic capacitances and resistances of the transistors that make up the distributed latches of the oscillator are easily calculated from the geometry of the devices. In the HSPICE simulations these parasitics are included in the binned BSIM models provided that the appropriate parameters are included in the corresponding HSPICE MOSFET instantiation. Because the BSIM models available were binned it was necessary to break the transistors into smaller transistors in parallel. Once again, scripting was used to make this task more managable (cf.Section A.10). The resulting generic SPICE model is shown in Figure 4.4. Referring back to Figure 4.2, the large devices on the left side of the Figure correspond to the inverters in Figure 4.4 while the large device on the right hand side is a power supply decoupling capacitor that corresponds to c_{pg} in Figure 4.4.

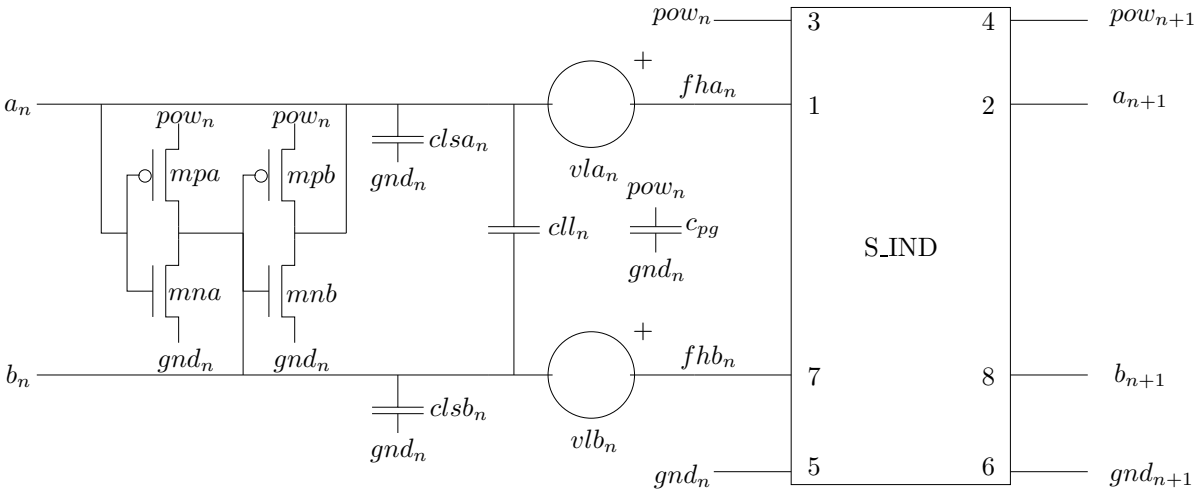

Figure 4.4: Generic SPICE model for one DLTWO stage

The FASTHENRY subcircuit S_IND in Figure 4.4 is automatically generated by running a single-frequency FASTHENRY simulation and then using FASTHENRY's

MakeLcircuit function. Since this is somewhat esoteric and there is no documentation for MakeLcircuit an example is in order. The most complicated situation is a straight differential transmission line segment parallel to the supply rails. A FASTHENRY zbuf rendition of this situation is shown in Section A.10.1 of the Appendix. The MakeLcircuit model that results from this structure has the following somewhat complicated form:

```
LZ_0 1 int_node0_2 1.44292e-10
LZ_1 3 int_node1_2 2.27127e-10
LZ_2 5 int_node2_2 2.2712e-10
LZ_3 7 int_node3_2 1.44274e-10
KZ_1_0 LZ_1 LZ_0 0.472477
KZ_2_0 LZ_2 LZ_0 0.377978
KZ_2_1 LZ_2 LZ_1 0.53121
KZ_3_0 LZ_3 LZ_0 0.338529
KZ_3_1 LZ_3 LZ_1 0.378395
KZ_3_2 LZ_3 LZ_2 0.47246
RZ_0_0 int_node0_2 int_node0_3 0.193044
HZ_0_1 int_node0_3 int_node0_4 Vam_1 0.000697493
HZ_0_2 int_node0_4 int_node0_5 Vam_2 0.00525218
HZ_0_3 int_node0_5 int_node0_6 Vam_3 -0.00181598
Vam_0 int_node0_6 2 dc=0v
HZ_1_0 int_node1_2 int_node1_3 Vam_0 0.000697493
RZ_1_1 int_node1_3 int_node1_4 1.10766
HZ_1_2 int_node1_4 int_node1_5 Vam_2 0.0320446
HZ_1_3 int_node1_5 int_node1_6 Vam_3 0.00508799
Vam_1 int_node1_6 4 dc=0v
HZ_2_0 int_node2_2 int_node2_3 Vam_0 0.00525218
HZ_2_1 int_node2_3 int_node2_4 Vam_1 0.0320446
RZ_2_2 int_node2_4 int_node2_5 1.10776
HZ_2_3 int_node2_5 int_node2_6 Vam_3 0.000623543
Vam_2 int_node2_6 6 dc=0v
HZ_3_0 int_node3_2 int_node3_3 Vam_0 -0.00181598
HZ_3_1 int_node3_3 int_node3_4 Vam_1 0.00508799
HZ_3_2 int_node3_4 int_node3_5 Vam_2 0.000623543
RZ_3_3 int_node3_5 int_node3_6 0.19286
Vam_3 int_node3_6 8 dc=0v
```

A schematic representation of this model is shown in Figure 4.5. Figure 4.6 shows a typical simulated DLTWO waveform resulting from using these techniques.

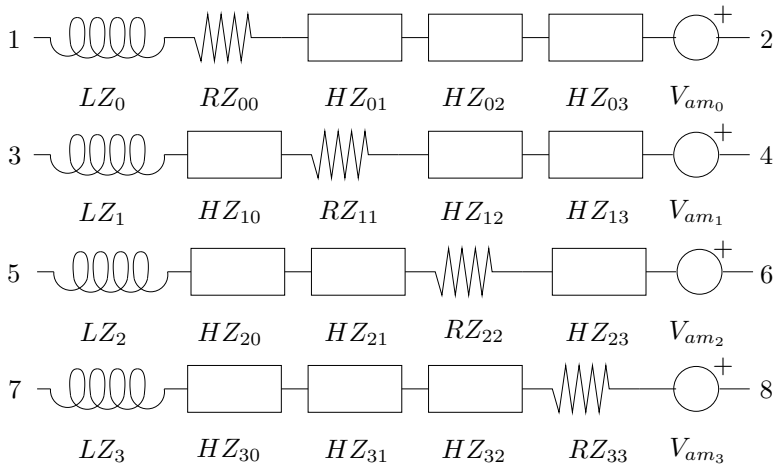

Figure 4.5: Schematic representation of the example MakeLcircuit output

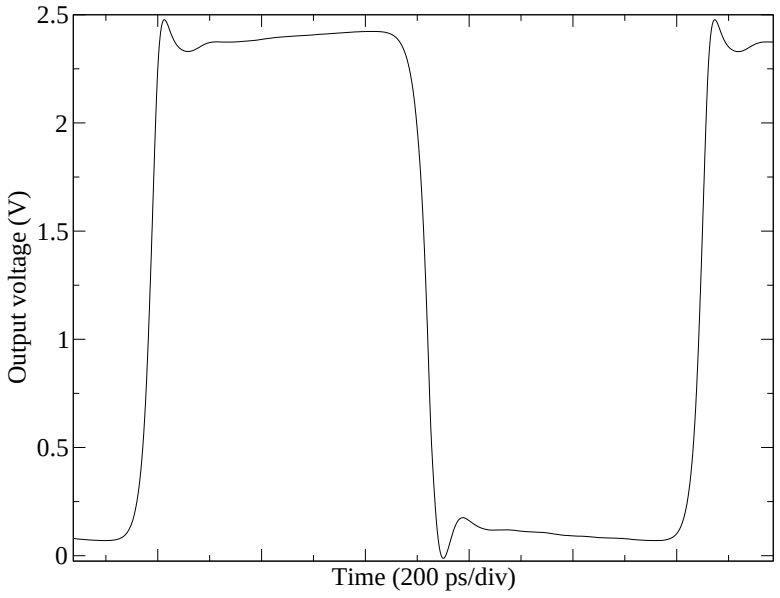

Figure 4.6: SPICE simulation showing typical DLTWO waveform

4.2.3 DLTWO ISF calculation

Implementing ISF analysis on the DLTWO is decidedly non-trivial. Hajimiri and Lee's 1999 paper on ring oscillators [19] is largely about estimating the ISF for ring oscillators because it is so painful to do. The SPICE model for the average DLTWO is much more complicated than that of the average ring oscillator because there are many more active devices and modeling the transmission line components adds a lot of extra nodes. Thus the simulations take a relatively long time.

The most accurate way to calculate the ISF is described in [15]:

1. simulate the circuit to establish a baseline waveform
2. insert a small impulse current source into the circuit to model a current noise source
3. simulate the circuit n times, each time shifting the time that the impulse is applied by $1/n*$period of oscillator. (Here n determines the frequency resolution of the ISF.)
4. for each of the simulations above determine the excess phase introduced by the current source.
5. Repeat 2,3 and 4 above for each noise source in the circuit.

Thus, if the circuit has n noise sources in it and it is desired to determine the ISF at m points distributed across its period of oscillation, $nm + 1$ simulations are required. A typical DLTWO can easily have 160 transistors per ring, each one of which might properly be considered to be a noise source. To distribute a clock across even a small IC it could have 20 rings. To find the ISF at 50 points distributed over the period of oscillation by brute force simulation would therefore require $160 * 20 * 50 + 1 = 160001$ simulations of a circuit with 3200 transistors, 3200 inductors, and 2400 capacitors. Obviously it is necessary to get this down to a more reasonable number if ISF analysis is to be useful.

Fortunately, the symmetry, simplicity, and operational characteristics of the DLTWO architecture make it possible to argue for drastically reducing the number of required

simulations. Consider the generic single-ring DLTWO with n stages. In general, it will be designed as n identical stages or $n/2$ stages in one style alternating with $n/2$ stages in another style. E.g. $n/2$ stages in which the differential transmission line conductors cross and $n/2$ stages in which they are straight. While it *is* possible to vary transistor sizes from stage to stage it is certainly not necessary and probably not desirable, so it is not unreasonable to assume that the transistor sizes are constant in all stages. If all of the stages are truly identical, the number of simulations required in the brute force ISF recipe are reduced by a factor of n. If $n/2$ alternating pairs of truly identical stages are used the number of simulations can be reduced by at least a factor of $n/2$.

The stages of the DLTWO are quite simple in that outside of the rails each stage only has two real nodes which correspond to the local connection to the transmission line. If they are perfectly symmetrical it is really only necessary to consider noise current injected into one at a time. Figure 4.7 shows the excess phase introduced into an $N = 40$ 12000 μm single-ring DLTWO as a result of injecting 750 fC of charge into one conductor of the differential transmission line at various points during the period of oscillation. The decision to use 750 fC of charge was based on a preliminary analysis of the sensitivity of the DLTWO to injected charge. Assuming that the maximum sensitivity of the DLTWO to injected charge would occur sometime near the flattest part of an edge, a sweep was performed over wide range of injected currents, the center of which is depicted in Figure 4.8. The goal was to choose a value that would produce a small but measurable phase shift without driving any of the internal nodes of the circuit to unreasonable voltages. Since only very small perturbations are induced the assumption of linearity applies so it is acceptable to superimpose the results. The graph shows that 750 fC shouldn't induce much more than approximately 1/100 of a radian or about 2/10 of one percent of a cycle of phase shift. Figure 4.7 shows that the DLTWO is *extremely* insensitive to noise pulses that do not coincide with the voltage transition. The excess phase induced during the flat parts of the waveform are typically buried in numerical noise of the simulations. To further bring this point home, Figure 4.9 shows two simulated waveforms superimposed upon each other, one

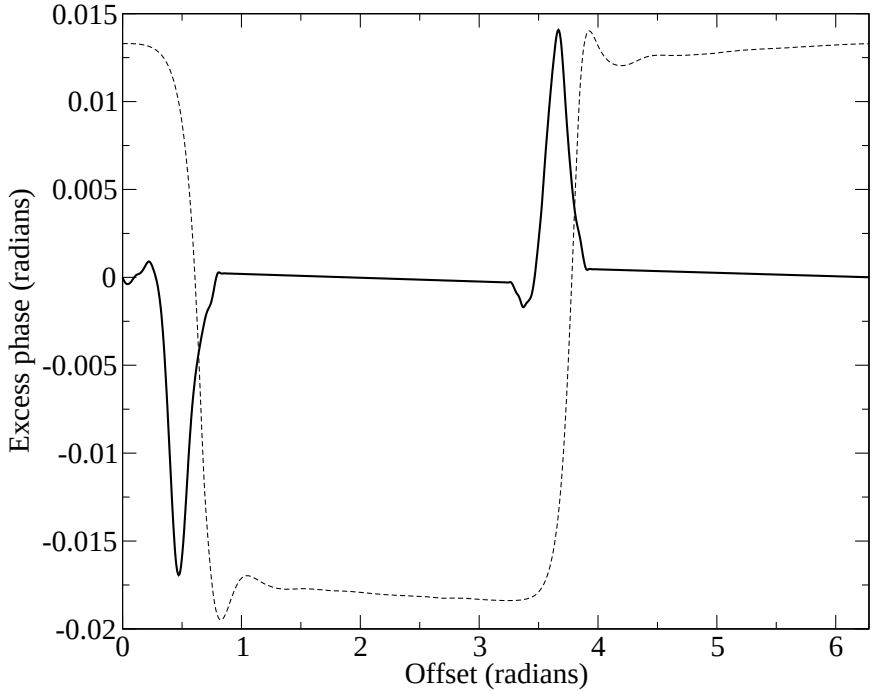

Figure 4.7: Excess phase resulting from injecting 750 fC of charge at various times during one cycle of oscillation.

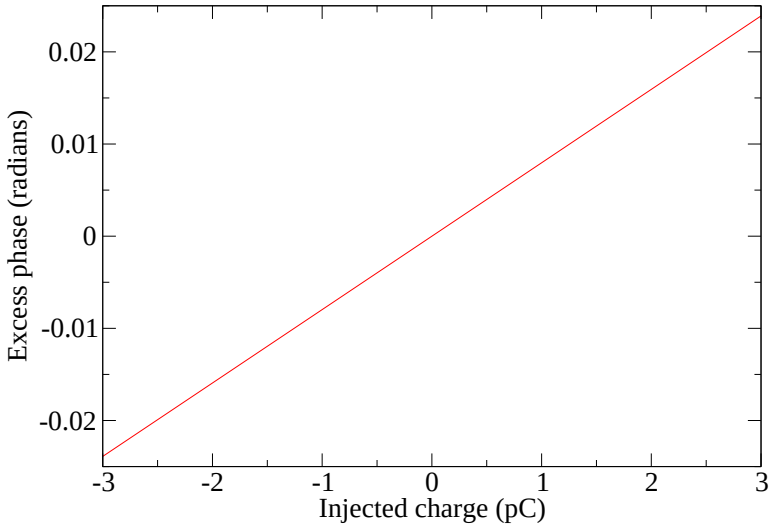

Figure 4.8: Excess phase vs. injected charge

of which is perturbed by a huge noise spike in the middle of the flat section of the waveform. Note that no excess phase is introduced into the victimized waveform. Everything that is of interest in calculating the ISF of a DLTWO therefore takes place during the transitions, as expected. Figure 4.10 is a closeup of a typical DLTWO simulation which shows a cross at the point of maximum sensitivity. As expected, the peak of sensitivity for each edge is coincident with the transition of the local inverter entering the negative resistance region of operation. This suggests two techniques for dramatically reducing the number of simulations required for calculating the ISF for the DLTWO:

41-simulation model Assume the ISF is zero outside the transition and do 20 simulations spread across each edge, plus one base simulation with no injected charge.

3-simulation model Do one simulation to establish a base waveform. Do two more simulations injecting charge into a stage at the time that the latch is entering the negative resistance region, once in each direction. Build the resulting ISF as a piecewise-linear approximation that is zero outside the edges and triangular

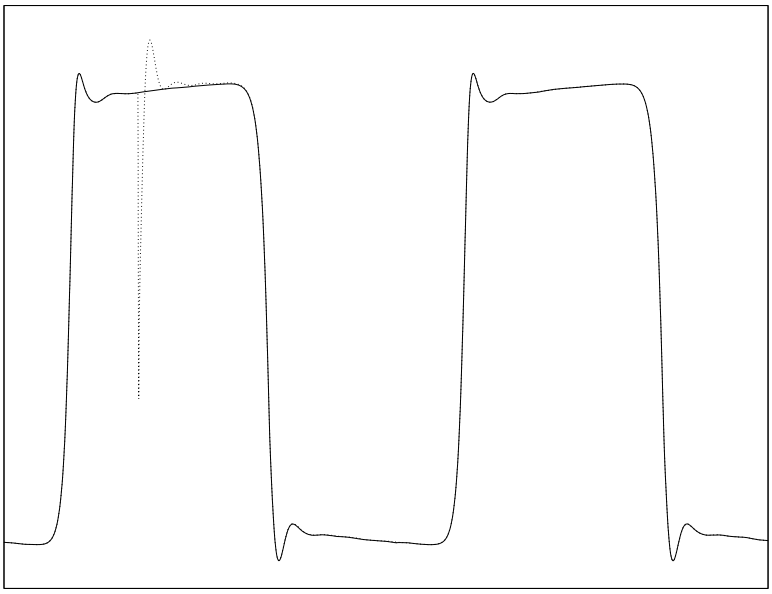

Figure 4.9: Demonstration of insensitivity of DLTWO to huge noise spike outside of transition time

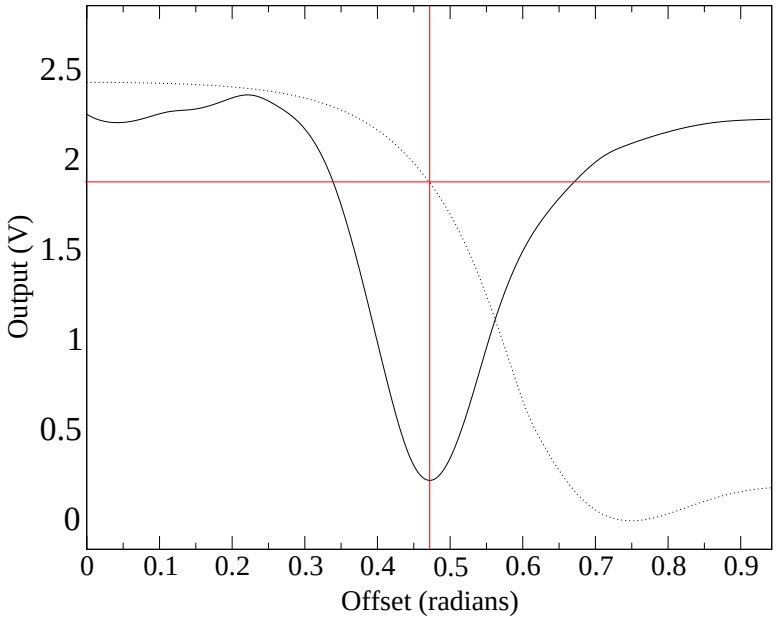

Figure 4.10: Closeup showing point of maximum sensitivity.

inside the edges with a maximum determined by the last two simulations.

Both ISF approaches rely on the fact that all of the stages are the same and mirrored across the differential transmission line. As a result the total ISF for a DLTWO consisting of n stages is $2n$ times the ISF of a single stage (because the waveform traverses the ring twice for each cycle) with a single noise source that is the root sum of squares of the noise sources on the two rings as described in Chapter 3. These approaches reduce the thousands of simulations required to do a brute force ISF calculation down to a managable number.

4.3 Laboratory measurements

4.3.1 Probing structures

Close-in phase noise measurements and jitter measurements were performed on the single ring DLTWO on the Multigig test chip. As described above, this was a 12000 μm differential transmission line comprised of two coplanar 60 μm conductors on a 120 μm pitch with 128 62.5 μm/25 μm inverter pairs distributed along its length to support the propagation of the wave. At each corner of the DLTWO, ground-signal-ground probe points were set up to allow probing of the waveform. Figure 4.11 shows the layout of the pads. These probe pads make it possible to attach either a high-impedance probe or a 50 Ω passive coaxial probe such as the GGB Model 40A or the Cascade Microtech ACP40. The equivalent circuit of the probe pad is shown in Figure 4.12. In the figure the ground-signal-ground (GSG) probe pad is shown on the left. The signal pad is connected to a point on the DLTWO differential transmission line through a 950 Ω polysilicon resistor. The idea here is that a 50 Ω probe and associated 50 Ω measurement system will present a total load of 1000 Ω and provide a 20:1 attenuation of the measurement system. This allows us to connect a spectrum analyzer or other 50 Ω instrument to the DLTWO without unduly loading it. If we connect a GGB Model 35 or other high-impedance probe to the signal pad we get the full signal.

Figure 4.13 shows a GGB Model 40A GSG probe with a pitch of 100 μm contacting

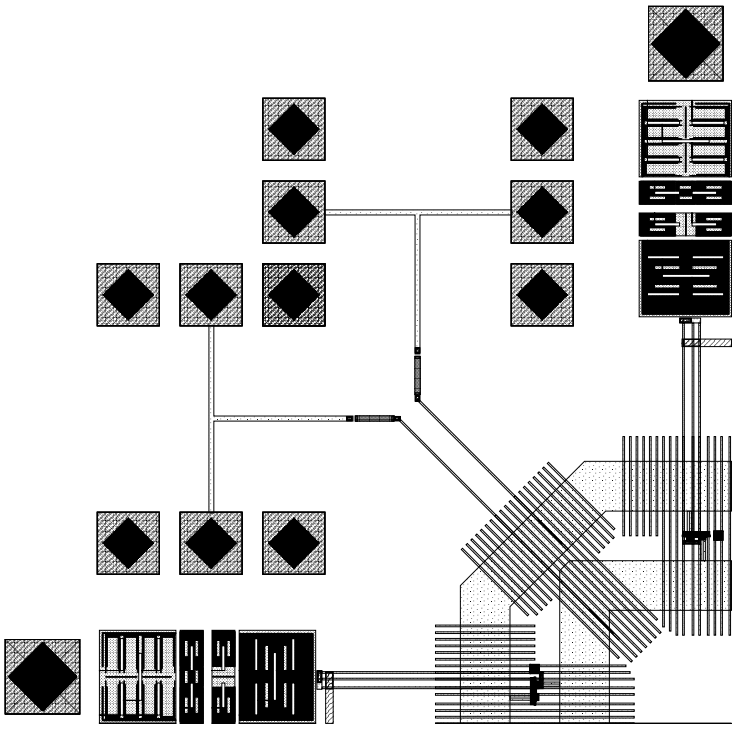

Figure 4.11: Probepads used to probe single-ring 12000 μm DLTWO

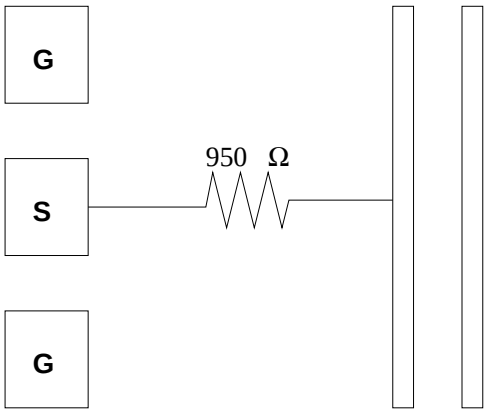

Figure 4.12: Circuit used for GSG probe pads

the pads in one corner of the test chip. The next picture, shown in Figure 4.14 provides a close-up view through the microscope of the probe tip as it contacts the pads.

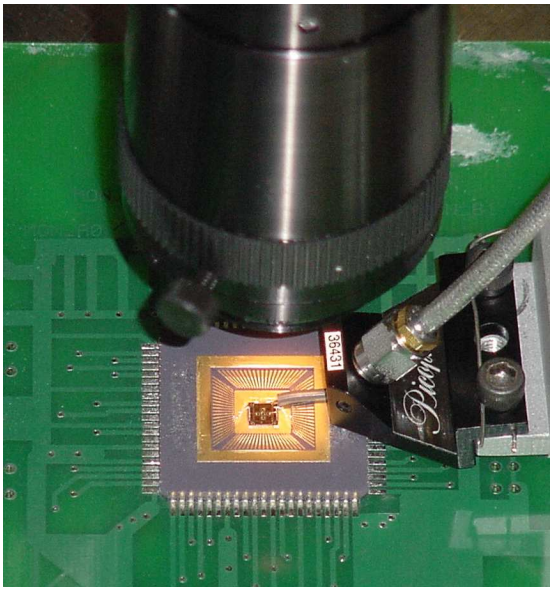

Figure 4.13: GGB Model 40A Probe measuring test chip

4.3.2 Basic characterization

Several basic characteristics of the single-ring 12000 μm DLTWO were measured in the laboratory to help verify that the SPICE modelling techniques used were basically sound. Accuracy in modelling such things as the frequency of operation, the rise and fall times, and the general waveshape is a good indicator that the modelling technique is on the right track. In this case the results were very encouraging.

Figure 4.15 shows the measured waveshape of a typical waveform for the single-ring 12000 μm DLTWO at the nominal V_{dd} of 2.5 V compared to simulations. To create this graph it was necessary to properly simulate the parasitics of the probing structure. The charging of the parasitic capacitance of two highly capacitive metal 1 test pads and probe tip through the 950 Ω resistor slows down the transition appreciably. Since the parasitics of the probe are variable, depending on exact positioning of the probe,

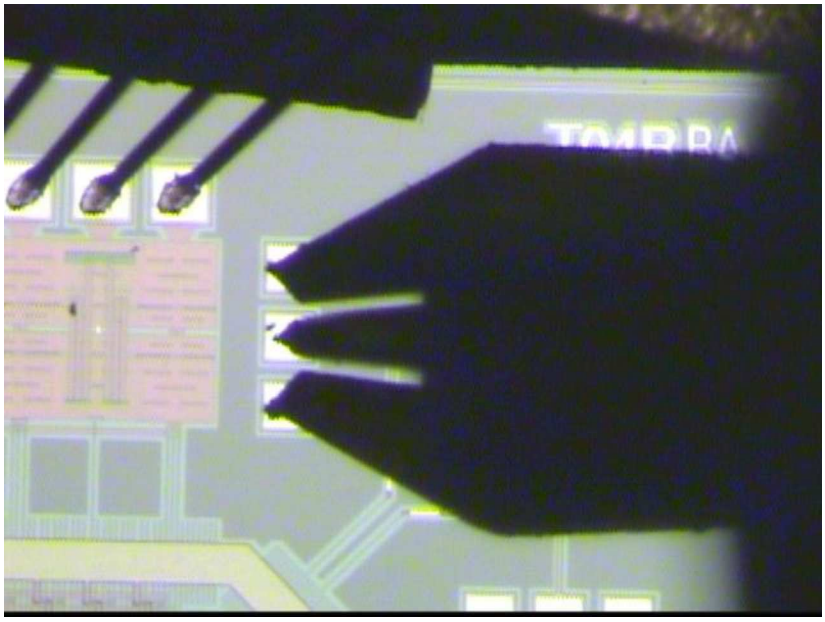

Figure 4.14: Close-up of GGB Model 40A Probe tip on pads

quality of probe contact, etc, it is probably not realisitic to expect perfect agreement here.

Figure 4.16 shows the measured vs. simulated graph of oscillation frequency f vs. V_{dd}. This is an important relation because changes in f obviously effect the instantaneous phase of the oscillator. The oscillator functioned down to a V_{dd} of 1.1 V but the shape of the waveform degraded significantly below 1.9 V, so data presented here will primarily be in the range above 1.9 V.

Figure 4.17 shows measured vs. simulated results for 20-80% rise time and fall times vs V_{dd}. Unfortunately the measured rise and fall times were affected by the imperfect probing structure used to monitor the operation of the circuit so the agreement between measured and simulated results is not very good. (Agreement for 10-90% rise and fall times was even worse....) This is true even though the simulated results presented in Figure 4.17 include the parasitics of the probe pads and estimated parasitics of the probe itself. While the rise and fall time numbers do not agree very well with the measurements, the agreement in general waveshape shown in Figure 4.15,

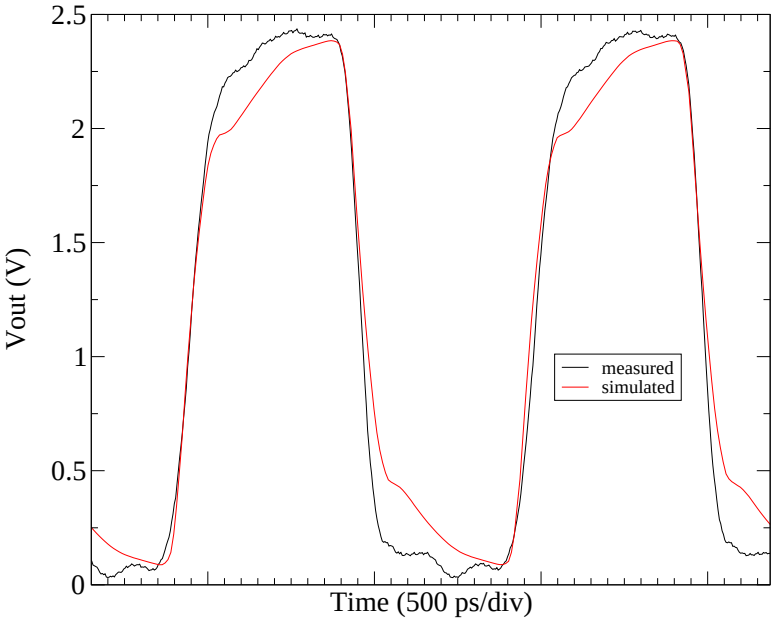

Figure 4.15: Typical measured waveform for single-ring 12000 μm DLTWO

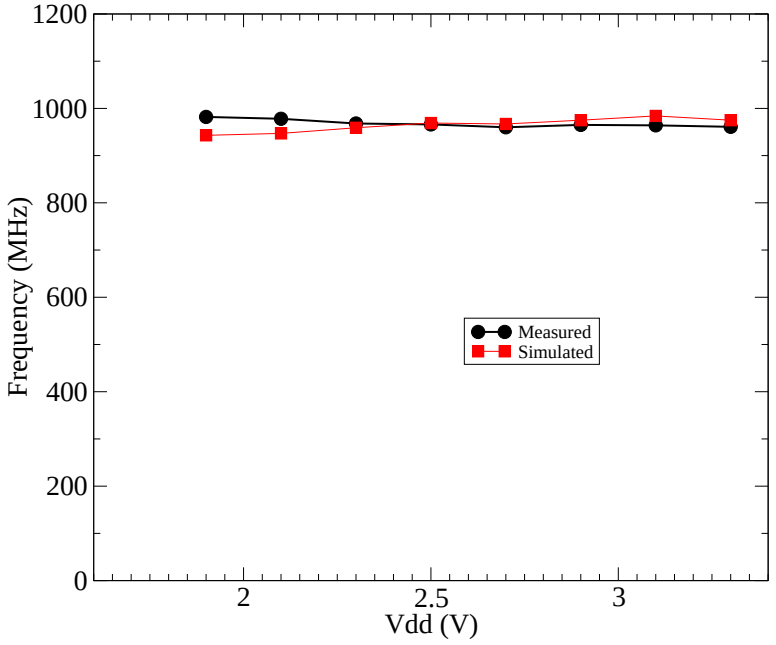

Figure 4.16: f vs. V_{dd} for single-ring 12000 μm DLTWO

suggests that the simulations do model the circuit very well. Subtle errors introduced by the probing structure, the poly resistor, and probe parasitics that have a small effect on the waveshape near the rails can have a large effect on the measured rise and fall times because the slope of the waveform is smaller in this region.

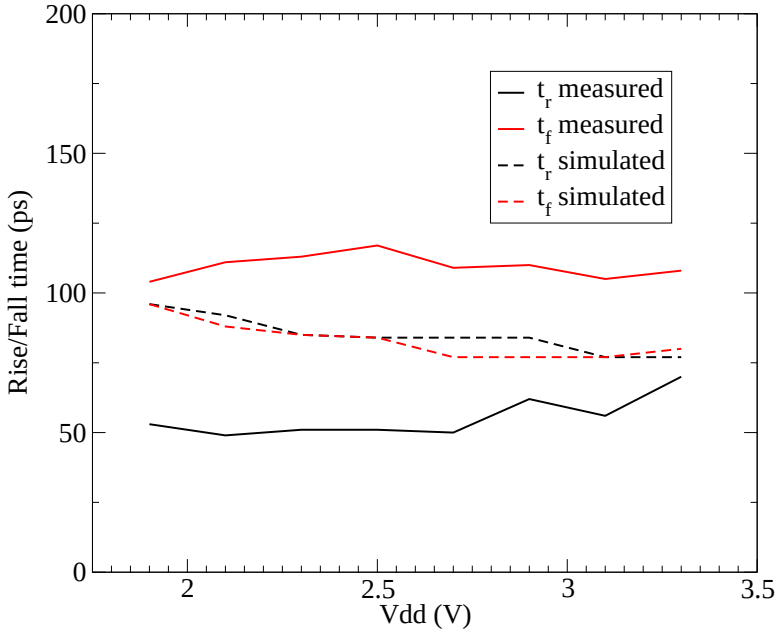

Figure 4.17: Rise and fall times vs. V_{dd} for single-ring 12000 μm DLTWO

4.3.3 Jitter measurements of single-ring 12000 μm DLTWO

Jitter measurements were performed on the single-ring 12000 μm DLTWO using a Tektronix TDS8000B digital sampling oscilloscope (DSO) with an 80E03 sampling head. The pertinent specifications of the instrument and sampling head are shown in Table 4.2.

The original estimate of the jitter reported in [1] turns out to be high. This was just a reading of the jitter measured by the scope at an arbitrary offset from the trigger and did not take trigger jitter or timebase jitter of the measurement system into account. The data reported here are based on measurements using the techniques

Table 4.2: Important specifications of DSO

Rise time	17.5 ps
Bandwidth	20 GHz
Displayed noise	≤ 1.2 mV_{RMS}
Ext. direct trig. delay jitter	790 fs RMS + 5 ppm horiz. pos.

suggested in [15] and [35]. Specifically, the setup shown in Figure 4.18 was used. The

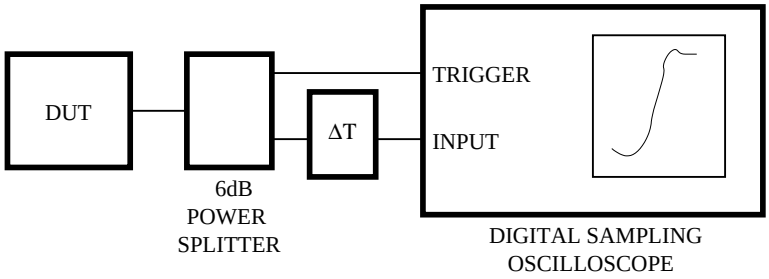

Figure 4.18: Setup used for jitter characterization

idea behind Figure 4.18 is simple. Since the DSO trigger is inherently delayed, it is normally not possible to see the samples that make up the triggering edges of the waveform. By splitting the oscillator output and delaying the sampled waveform, it is possible to display the triggering edge of the waveform. The jitter in this displayed edge is the jitter of the triggering system alone. In the case of the TDS8000B used for these measurements the triggering delay was approximately 19.2 ns. Thus, a 24 ns delay line was used for the block labeled ΔT in Figure 4.18. The jitter measured on the triggering edge was 765 fs which matches very well with the specifications for the instrument. Having established this the cycle-to-cycle jitter can be calculated as

$$\sigma_{ctc} = \sqrt{\sigma_{edge1}^2 - \sigma_{min}^2} \tag{4.1}$$

where σ_{edge1} is the RMS jitter measured at the next edge after the triggering edge and σ_{min} is the RMS jitter of the trigger system. The cycle-to-cycle jitter for the single-ring 12000 μm DLTWO was calculated to be 1.3 ps RMS. To extend this process for

longer periods the timebase jitter of 5 ppm of the horizontal offset has to be added to σ_{min}. The effective jitter due to the oscillator for an arbitrary offset τ is given by the equation

$$\sigma_{\tau,eff} = \sqrt{\sigma^2_{\tau,meas} - \sigma^2_{\tau,min}} \tag{4.2}$$

as given in [15] although they don't mention the timebase jitter correction explicitly. Figure 4.19 shows the resulting estimate of the measured RMS jitter vs τ for the

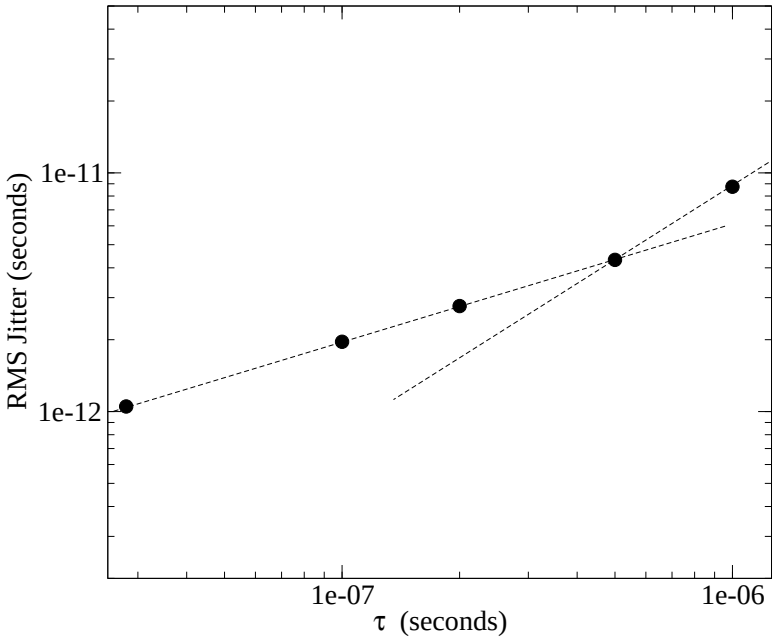

Figure 4.19: Measured RMS jitter vs τ for single-ring 12000 μm DLTWO

single-ring 12000 μm DLTWO. This graph shows the both the characteristic $\sqrt{\tau}$ region with $\kappa = 4.8e-9\sqrt{\text{sec}}$ as discussed by McNeill (see equation 2.54) as well as a section with a slope of 1. The section of the graph with a slope of 1 is probably due to correlation of low-frequency noise across many cycles. [15] Equation 4.3 shows that κ can also be estimated using the ISF

$$\kappa = \frac{\Gamma_{\text{rms}}}{q_{\max}\omega_0}\sqrt{\frac{1}{2}\frac{\overline{i_n^2}}{\Delta f}} \tag{4.3}$$

where all of the variables are the same as in Equation 2.35 [15]. Using this equation

with ISF simulation data using the both the 3-simulation technique and the 41-simulation results in the following estimates for κ:

Estimate	κ ($\sqrt{\text{sec}}$)
Measured	4.8e-9
41 simulations	4.9e-9
3 simulations	6.1e-9

Obviously the results for the 41-simulation method are superior, but the very simple three-simulation technique does provide a pretty good ballpark number, and the more precise technique provides a very good estimate of κ.

4.3.4 Close-in phase noise measurement

It is also possible to estimate $\mathcal{L}(\Delta\omega)$ using Equation 2.35. To check this and to quantify the DLTWO's $1/f^3$ corner the close in phase noise was measured in the lab. In the absence of a phase noise measurement system the close in phase noise was estimated by measuring its spectrum with an HP8565E spectrum analyzer. This spectrum analyzer was chosen because it is capable of a 1 Hz resolution bandwidth (RBW) and the because the phase noise of its local oscillator is much lower than the expected phase noise of the DLTWO being tested. While this measurement is not perfect it gives a pretty good idea of the close-in phase noise. Figure 4.20 shows the close-in phase noise for the single-ring 12000 μm DLTWO with at the nominal V_{dd} of 2.5 V.

Once again the 41-simulation and 3-simulation techniques were compared to the measured result:

Estimate	$\mathcal{L}$(100kHz)
Measured	-95 dB
41 simulations	-93 dB
3 simulations	-90 dB

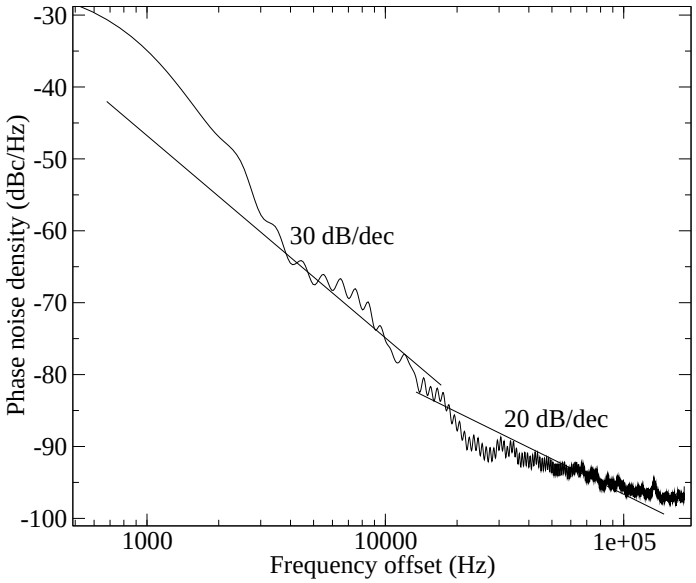

Figure 4.20: Close-in phase noise measurement of DLTWO

Once again the 41-simulation technique is superior and reasonably close to the measurement. However, a more precise analysis would require a better measurement technique.

Superimposed onto Figure 4.20 are lines showing the 9 dB per octave and 6 dB per octave sections of the phase noise characteristic. The $1/f^3$ corner is at approximately a 20 kHz offset.

4.3.5 Simulations of multi-ring DLTWOs

In order to verify the theory presented in Section 3.3 ISFs were calculated for a number of different multi-ring implementations of the single-ring 12000 μm DLTWO. Figures 4.21 through 4.24 show the different configurations for which complete ISFs were calculated. The four-ring DLTWO shown in Figure 4.23 is of academic interest only. Any multi-ring DLTWO intended for low-skew clock distribution will use the tightest, most efficient coupling possible. Real multi-ring DLTWOs will almost certainly look more like Figures 4.25 and 4.26. The starting point for studying the

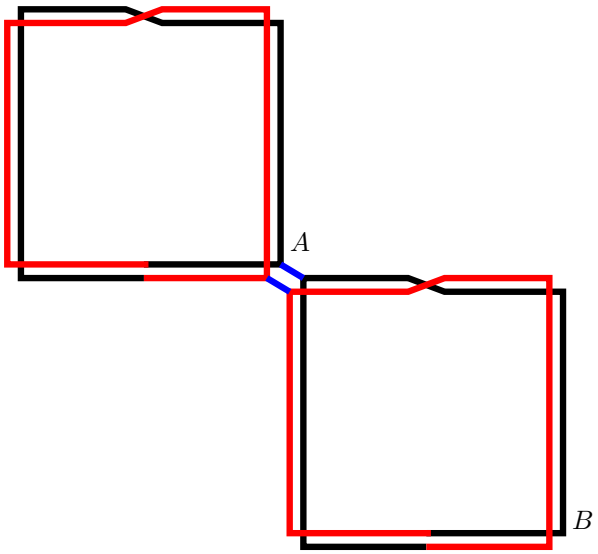

Figure 4.21: Two connected DLTWO rings

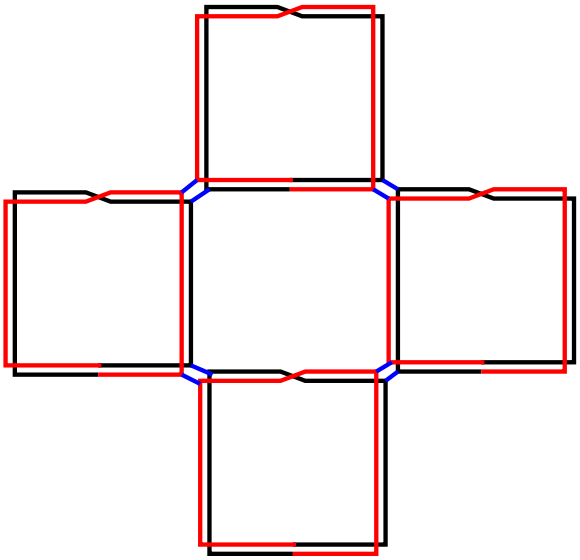

Figure 4.22: Most likely approach to connecting four DLTWO rings

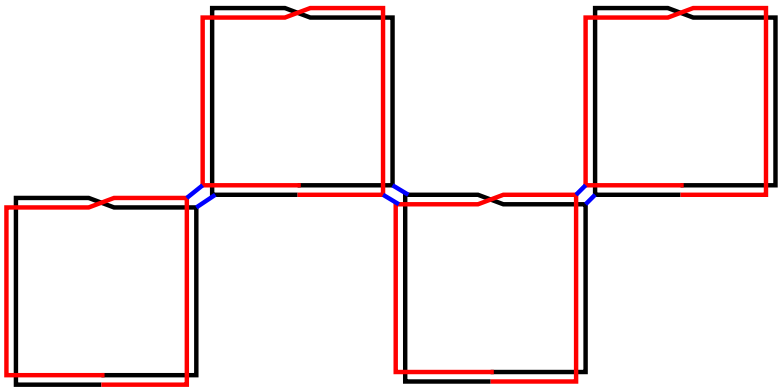

Figure 4.23: Less likely approach to connecting four DLTWO rings

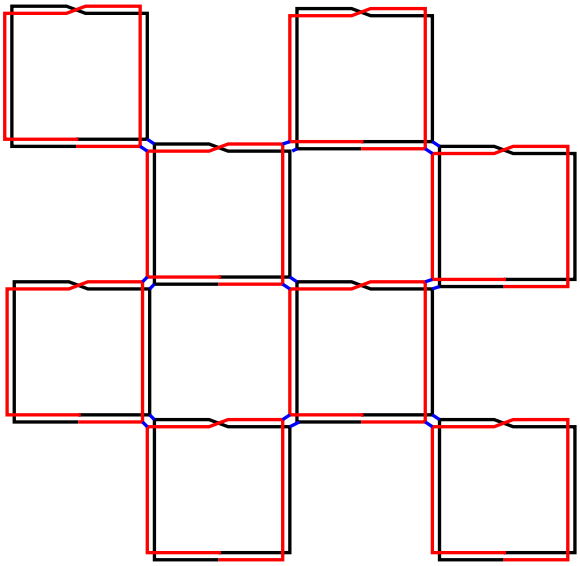

Figure 4.24: Eight connected DLTWO rings

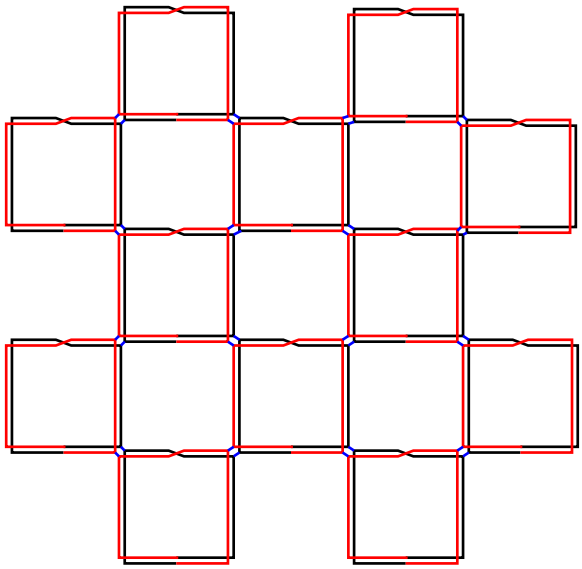

Figure 4.25: Twelve connected DLTWO rings

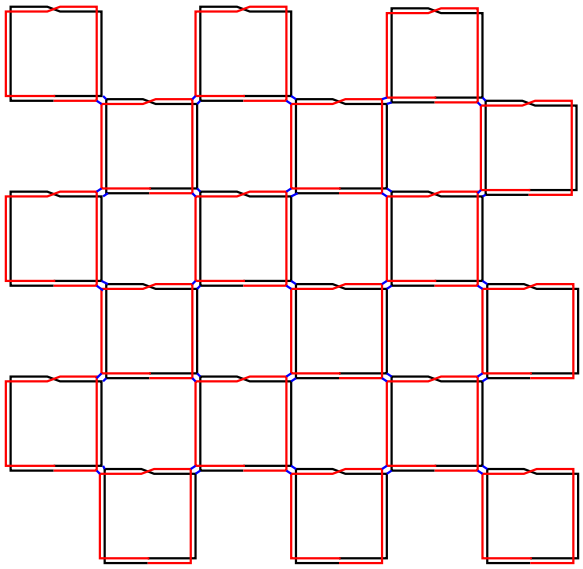

Figure 4.26: Sixteen connected DLTWO rings

interconnection of DLTWOs is to connect two of them together. Figure 4.21 shows the interconnection of two rings. The two main questions that arise are:

1. Does the averaging effect of multiple rings really reduce the excess phase induced, and if so, by how much?

2. How does the positioning of the injection of a noise pulse affect the results of question 1? I.e., is the result different if the noise pulse is injected at the point labeled A in Figure 4.21 rather than B, and if so, what is the difference?

The answer to question 2 is the simplest. ISF simulations were run for numerous positionings of the pulse injection both near and away from the junction and the results were always almost exactly the same. It is possible that the difference could be characterized precisely, but the practical result is that there is not enough difference to make this worth doing.

Figure 4.27 shows a closeup of the normalized ISF concentrating on sensitivity during the rising edge calculated for various numbers of interconnected rings. Clearly the impulse sensitivity decreases linearly with the number of rings used. This suggests

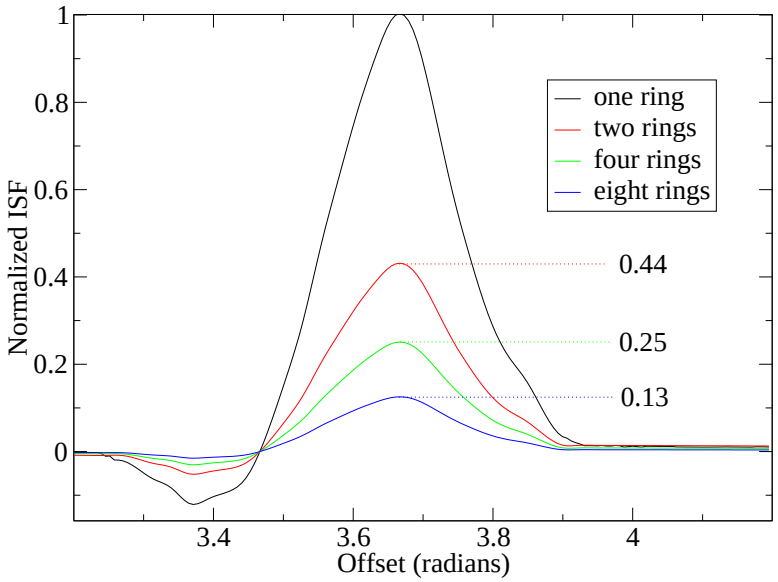

Figure 4.27: Excess phase vs. number of rings

that the averaging theory of multiple rings is correct.

4.4 Summary

This chapter has provided several important results. First, it shows that it is possible to accurately simulate the behavior of DLTWO oscillators using standard industry techniques. This is shown by comparing simulations of oscillation frequency, rise and fall times, and waveshape with measurements. While this is not a novel contribution, it does provide a foundation for trusting the ISF simulation results. A second important result of this chapter is a validation of the time-domain perturbation analysis provided in Chapter 3. Specifically, Figure 4.9 proves that the DLTWO stage introduces zero excess phase outside of the transition time and Figure 4.10 proves that the point of maximum noise sensitivity is indeed the moment that the DLTWO stage enters its negative-resistance region of operation. Figures 4.19 and 4.20 show that the noise model suggested in Chapter 3 provides accurate results. Figure 4.27 shows that the analysis of multiple rings provided in Chapter 3 is correct.

The novel contribution of this chapter is an analysis of the aspects of the architecture of the DLTWO that allow for a drastic reduction in the number of simulations required for performing ISF analysis. The validity of this analysis is shown by the comparison of ISF simulation data with measured data.

Chapter 5

Design Guidelines for Frequency Stability

5.1 Introduction

The design of a DLTWO is a very complicated problem because there is such a large number of design variables. The purpose of this chapter is to outline the problem and to show how different choices for some of the design variables might affect the jitter and phase noise of the resulting oscillator.

First, consider this list of design characteristics that may or may not be variable by the designer:

- Process characteristics
    1. Technology (minimum dimensions, etc.)
    2. Metal type (Al,AlSi,Cu,Au)
    3. Metal thicknesses
    4. Metal layer spacing
- Circuit characteristics
    1. Operating frequency
    2. Clock load
    3. Clock load distribution
- DLTWO characteristics
    1. Stages/ring

2. Number of rings
3. Configuration of rings
4. Ring length
5. Number of crossovers
6. Inverter size
7. P/N ratio

Changing any one of these characteristics of the design can change the frequency of operation and the waveshape of the DLTWO. Many of these characteristics will not be variable and most will have only an indirect effect on frequency stability. In the next few sections the characteristics most likely to impact frequency stability and most likely to be easily variable will be considered.

5.2 Sensitivity of jitter to number of stages

Phase error is introduced during the transition time of the DLTWO oscillation cycle. For a given cycle time, faster edge rates will reduce the percentage of cycle time that the oscillator spends transitioning. Thus, for a given frequency of oscillation a DLTWO with faster edge rates will accumulate less jitter.

Figure 5.1 shows the simulated edge rates on the ring for the 12000 μm DLTWO with the number of stages varied but all other variables kept the same. This shows that dividing the DLTWO into more stages improves the edge rate dramatically. Figure 5.2 shows the resulting positive impact of increasing the number of stages. This figure was created by simulating the excess phase of the 12000 μm DLTWO using different numbers of stages but keeping the frequency of operation constant. Unfortunately, increasing the number of stages also increases the number of transitions that take place during a given cycle of the waveform. The result is that, holding all other variables constant, increasing the number of stages decreases Γ_{rms} by the same factor by which the transition time percentage is reduced, and increases Γ_{rms} by the same factor as the increase in the number of transitions per cycle. κ increases linearly

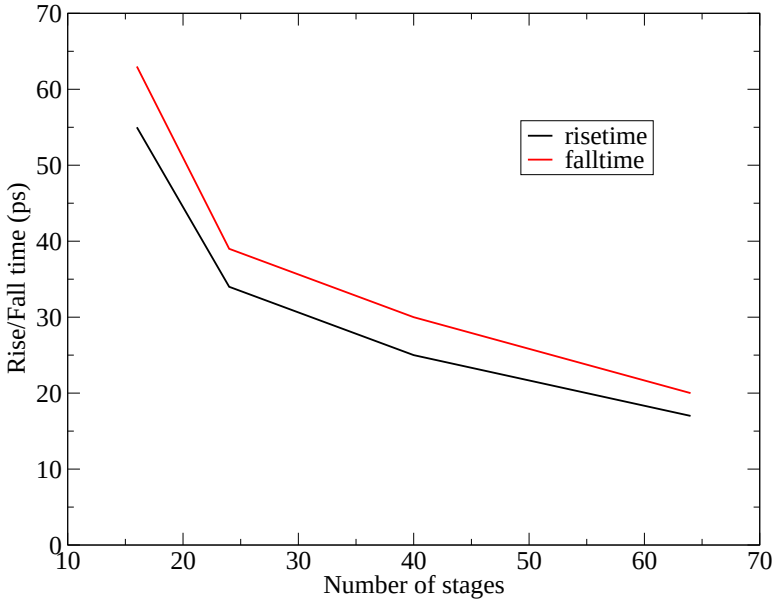

Figure 5.1: Rise and fall times vs. number of stages

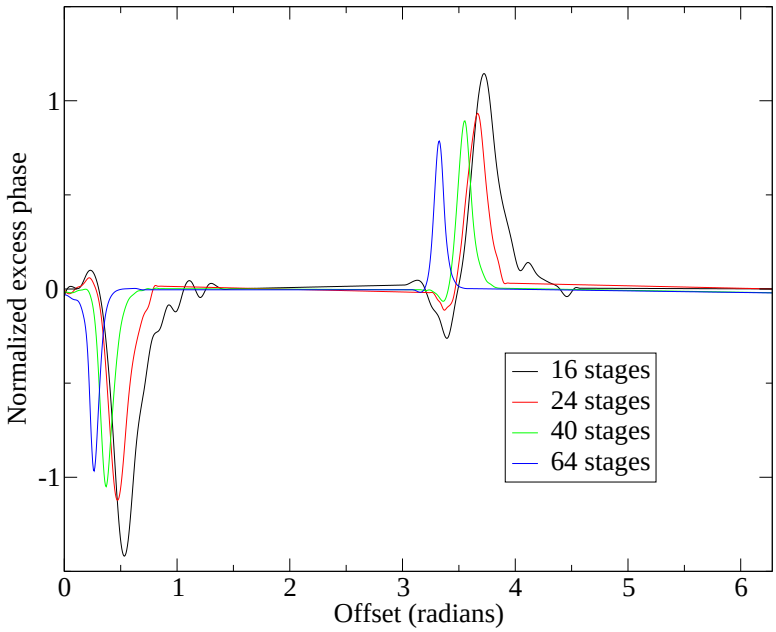

Figure 5.2: Effect of the number of stages on impulse sensitivity, normalized to 40 stages

with Γ_{rms}. Judging from the data in Figure 5.2, increasing the number of stages above that required to provide a solid waveform was a bad tradeoff in the case of the 12000 μm DLTWO. Since 40-stage versions of this oscillator produced good edge rates, many of the generalized DLTWO studies done as part of this work used 40-stage models, which is why the above data is normalized to 40 stages.

5.3 Sensitivity of jitter to V_{dd}

As V_{dd} is increased the oscillator output voltage swing increases. The power in the wavefront increases with the square of the voltage swing. The power of the noise sources does not increase so we expect the oscillator to become more stable.

Figure 5.3 shows the simulated ISF for several different values of V_{dd} for the single-ring 12000 μm DLTWO. Clearly the sensitivity of the oscillator to noise is reduced with increasing V_{dd}.

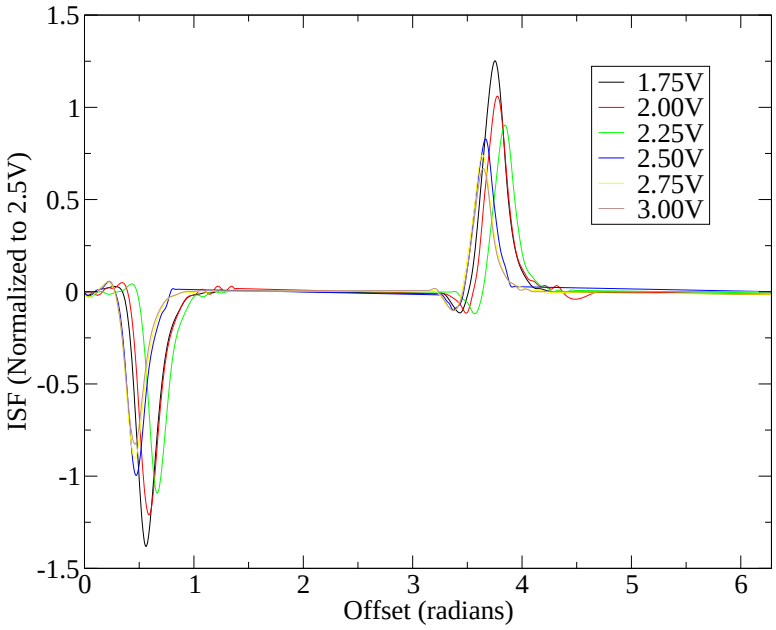

Figure 5.3: Impulse sensitivity to changes in V_{dd}

Phase jitter, and therefore timing jitter, is proportional to Γ^2_{RMS} [15]. The dashed

curve in Figure 5.4 shows the simulated effect of the Γ^2_{RMS} data for the curves in Figure 5.3 on the measured cycle-to-cycle RMS jitter measurements for the single-ring 12000 μm DLTWO. The predicted trend of decreasing jitter with increasing V_{dd} matches very well with the measurements.

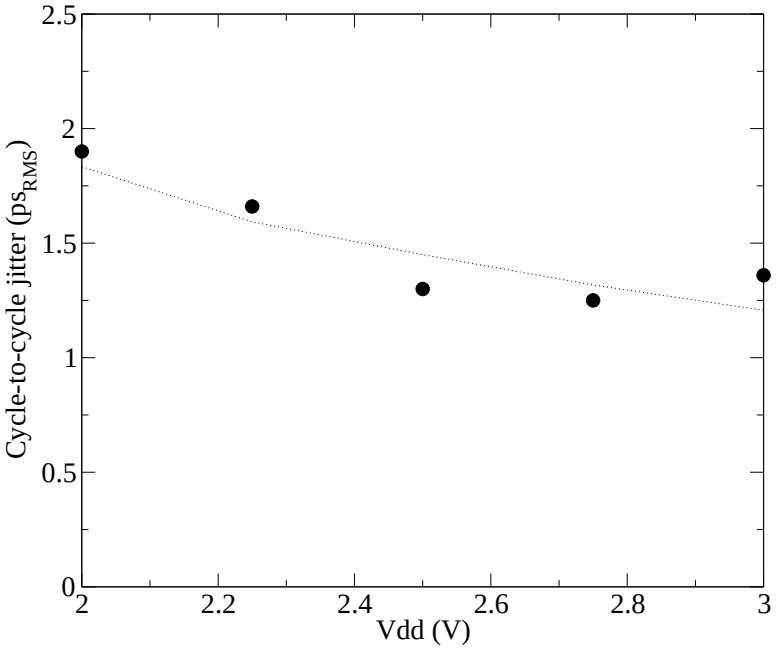

Figure 5.4: Measured RMS jitter vs V_{dd} for single-ring 12000 μm DLTWO

Another issue related to V_{dd} is sensitivity to supply noise. Because of the differential nature of the DLTWO, it is very resistant to supply noise. Noise seen on the supply and/or ground rails will generally affect each side of a transitioning stage equally if the layout is properly symmetric. If there is an imbalance due to, for example, greater coupling from a supply or ground rail to one side of the oscillator, this noise can be treated as an additional noise source and added to the ISF analysis. In general noise on the power supply rails will be cyclostationary in nature, but the ISF technique can handle this using the techniques described in [15].

As part of this study a number of experiments were performed to verify the stability of the DLTWO in the face of supply noise. First ISF simulations were run, injecting

charge into the local supply node for a stage. This required unreasonably large charge injection so this approach was abandoned. Instead of using charge impulses a number of simulations were performed using long-term V_{dd} pulses raising the entire V_{dd} system or parts of it for entire cycles or parts of cycles. To get phase shifts on the order of 1 picosecond it was necessary to shift V_{dd} by hundreds of millivolts at the nominal voltage of 2.5 V.

The results are:

1. Increasing V_{dd} increases the stability of the DLTWO because it maximizes carrier power while keeping noise power relatively unchanged.

2. The differential nature of the DLTWO makes it quite resistant to power supply noise. Power supply noise large enough to result in significant phase noise is highly implementation-specific and therefore not amenable to general analysis. Since power supply disturbances of this magnitude are generally cyclostationary it may be possible to analyze them on a case-by-case basis using ISF analysis.[15]

5.4 Sensitivity of jitter to waveform symmetry

Waveform symmetry is a very important goal to strive for in the design of DLTWOs. If the rising edge in one half of the differential circuit is a mirror image of the falling edge in that same circuit half a cycle later, low-frequency components of noise will cause excess phase in each edge that tend to cancel out the excess phase in the opposite edge.

This has obvious ramifications for the design of the inverters that make up the DLTWO ring and perhaps less obvious ramifications for the strategy of load distribution. The obvious consideration is that the normal attempts at equalization of pull-up and pull-down strength through appropriate transistor sizing is especially important in the design of DLTWOs if low phase noise is required. Probably the best way to tune the P/N ratio is to perform a series of ISF simulations to determine the P/N ratio that provides the minimum Γ_{dc}.

The less obvious consideration might be that loading only one side of the DLTWO at first blush seems to induce an unnecessary asymmetry that could lead to increased phase noise. Consider Figure 5.5. This is a simplified example of how one might lay

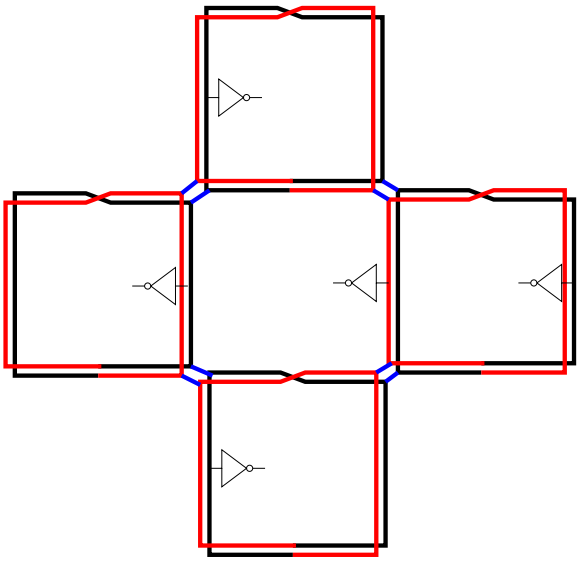

Figure 5.5: Simplified example of suspicious loading

out a small IC using a DLTWO to distribute a low-skew single-phase clock. Since the single-phase clock only needs one side of the DLTWO, the buffers (or possibly just a direct load on the clock) only load one side of the DLTWO which is an introduction of asymmetry. However, since the load is only presented to one half of the differential circuit the same loading presents itself during alternate transitions so the DC component of the ISF does not increase. The result is that unbalanced loading (in this sense) does not harm phase noise performance in the DLTWO. Putting dummy loads on the other half of the transmission line to offset the asymmetry is probably counterproductive in that it increases power dissipation without improving Γ_{dc}. Of course, such loads would reduce the noise bandwidth fractionally as described in Equation 3.1 but this effect would be minimal.

5.5 Optimization of Q

While the DLTWO looks a lot like a ring oscillator, it is possible to look at it very much like a classical Leeson-style resonator-based oscillator, with a resonator formed by an active transmission line and an amplifier with precisely unity gain.

In general, having the highest Q possible provides the best frequency stability. To maximize the Q it is necessary to minimize losses. One way to determine the effective loaded Q of the resonator is to compare the amount of energy that is effectively stored in the resonator to the amount provided to the load or lost:

$$Q = 2\pi \frac{\text{stored energy}}{\text{energy to load or lost}} \tag{5.1}$$

The stored energy is fairly easily calculated by considering the model of a single section of the oscillator shown in Figure 5.6

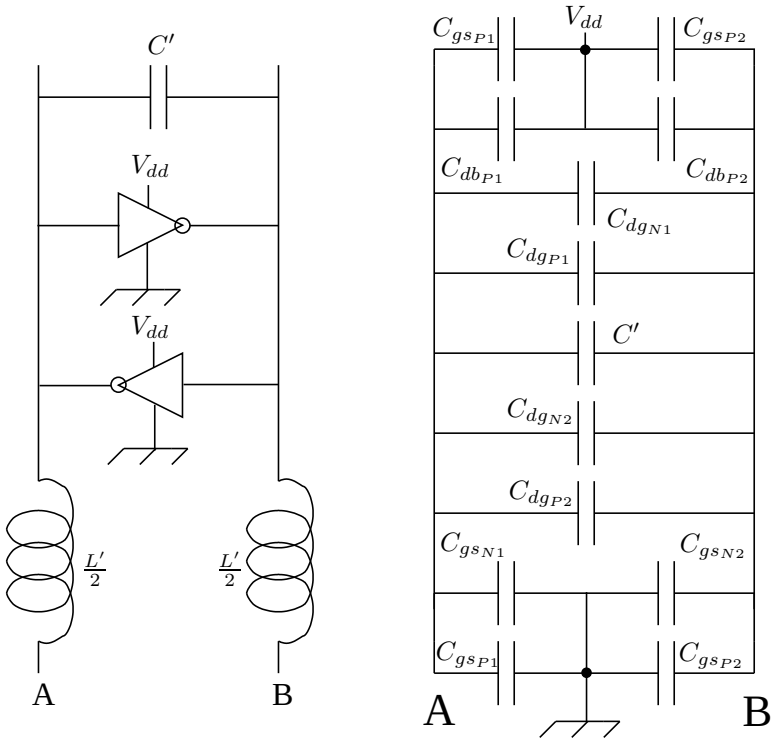

Figure 5.6: Circuit schematic of DLTWO section.

Clearly, to the first order, the charging and discharging of the parasitic capacitances only stores energy for the propagation of the wave along the transmission line. Thus if we assume the inverters used in the oscillator are all identical, the total stored energy is given by

$$W = \frac{nC_{total}V^2}{2} \tag{5.2}$$

where n is the number of segments, V is the magnitude of the oscillation, and C_{total} is given by

$$\begin{aligned} C_{total} &= \frac{C_{gsP1} + C_{gsP2}}{4} + \frac{C_{dbP2} + C_{dbP1}}{4} \\ &\quad + C_{dgN1} + C_{dgP1} + C' + C_{dgN2} + C_{dgP2} \\ &\quad + \frac{C_{dbN2} + C_{dbN1}}{4} + \frac{C_{dbN2} + C_{dbN1}}{4} + \frac{C_{load}}{4} \end{aligned} \tag{5.3}$$

Losses, assuming a purely capacitive load, are primarily the result of I^2R losses in the interconnect and transistors. Proper modelling of this loss is essential for accurate prediction of the power consumption of the oscillator as well as for determining the effective loaded Q of the transmission line. There are a variety of ways to approach this problem depending on the frequency of operation of the oscillator and the size of the conductors and transistors. It is of course possible to model all of the resistances as frequency-independent lumped constants at the cost of some accuracy. At the frequencies at which typical DLTWOs will be used, it is almost certainly more accurate to consider the frequency dependence of the resistance in the interconnects at the very least when determining the effective resistance to use.

Resistance in the conductors that make up the differential transmission line ($r_{tl_{eff}}$ in Figure 5.7) will probably be the biggest loss factor. Assuming that the oscillator consists of a ring comprised of n segments the total effective resistance in each conductor loop is given by $R_{eff} = nr_{tl_{eff}}$. The power transfer around the loop is given by

$$P = \frac{|V|^2}{2Z_0} \tag{5.4}$$

Thus the loss due to the series resistance in the transmission line is given by

$$P = \frac{V_{dd}^2}{Z_0^2} R_{eff} \tag{5.5}$$

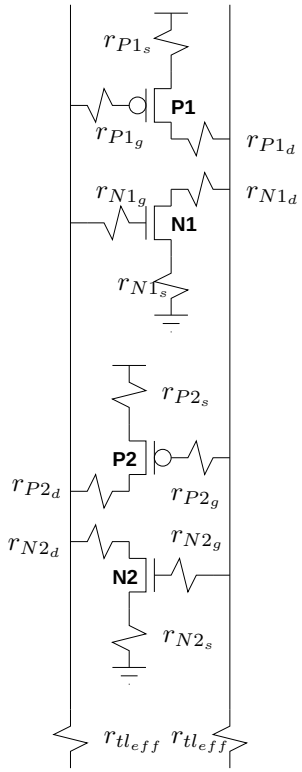

Figure 5.7: Resistances that lead to loss in one segment of a DLTWO

Loss in the series resistances in the transistors only manifests itself as the wavefront traverses them. As usual it is possible to model the power dissipated in these resistances with arbitrary accuracy but it can be argued that a first order model will probably be sufficient because this does not account for a major portion of the total losses. It should be sufficient to consider the leading edge of the wavefront as a straight line with rise (or fall) time t_r. In this case, as the wavefront passes a given transistor, the power dissipated in the parasitic resistances is approximately given by

$$P = \frac{V_{dd}^2 t_r}{2\left(r_s + r_d\right)} \tag{5.6}$$

Where r_s and r_d are the effective source and drain resistances of the transistor respectively. Assuming the two inverters are matched the total power lost in the source and drain resistances during one cycle – remembering that the wavefront passes each stage twice during a single cycle of the oscillator – is therefore

$$P = \frac{V_{dd}^2 t_r}{2}\left(\frac{1}{\left(r_{P1_s} + r_{P1_d}\right)} + \frac{1}{\left(r_{N2_s} + r_{N2_d}\right)}\right) \tag{5.7}$$

Thus the average power lost to these resistances in a DLTWO ring with n stages consisting of matched inverters and operating at a frequency f is approximately

$$P = \frac{nV_{dd}^2 t_r f}{2}\left(\frac{1}{\left(r_{P1_s} + r_{P1_d}\right)} + \frac{1}{\left(r_{N2_s} + r_{N2_d}\right)}\right) \tag{5.8}$$

Summing the two main components leads to a total resistive loss given by

$$P = V_{dd}^2\left(\frac{nt_r f}{2\left(r_{P1_s} + r_{P1_d}\right)} + \frac{nt_r f}{2\left(r_{N2_s} + r_{N2_d}\right)} + \frac{R_{eff}}{Z_0^2}\right) \tag{5.9}$$

To find the effective loaded Q we divide this into the equation for stored energy:

$$Q = \frac{nC_{total}}{\left(\frac{nt_r f}{\left(r_{P1_s} + r_{P1_d}\right)} + \frac{nt_r f}{\left(r_{N2_s} + r_{N2_d}\right)} + \frac{2R_{eff}}{Z_0^2}\right)} \tag{5.10}$$

The main loss mechanism in the DLTWO itself is I^2R loss in the differential transmission line. Thus, lowering the resistance of the differential transmission line is the first thing to consider. The most convenient way to lower this resistance is to use a lower-resistance conductor if possible. Table 5.1 the conductivity of some of the

Table 5.1: Available interconnect conductivities

Silver	Ag	$6.2X10^7$S/m
Copper	Cu	$5.8X10^7$
Gold	Au	$4.1X10^7$
Aluminium	Al	$3.5X10^7$
Tungsten	W	$1.8X10^7$

metals that might be used. Aluminum is probably still the most common metal used for most CMOS interconnect, but the table shows that the conductivity of copper is about 65% higher.

Barring the availability of lower resistivity materials, the next easiest way to reduce this resistance is to change the geometry of the conductors. If enough metal layers are available it may be possible to double the thickness of the conductors. Alternatively the conductors might be widened. In either case, this will change the parasitic capacitance of the lines and hence the frequency of operation of the ring. Obviously this is a complicated design trade-off that depends heavily on other factors in the design.

The second most important loss factor for the ring is probably I^2R loss in the MOSFETs themselves. Thus choosing a process with the smallest possible channel length and always using the minimum channel length are important. This also affects the number of stages used in the oscillator. Using more stages reduces the through current in the inverters and improves the edge rate considerably, but as we saw in Section 5.2 increasing the number of stages also increases the number of transitions/cycle which can lead to more phase noise.

Chapter 6

Conclusions

Generally speaking, the frequency stability of DLTWOs is comparable to that of well-designed ring oscillators, but DLTWOs provide the added benefit of a built-in clock distribution scheme and the ability to deliver very low skew, multiphase clocks, or both for no additional cost. Therefore it's quite likely that they will be of interest to designers of digital systems. The main goals of this work were to determine the mechanisms that give rise to frequency instability in DLTWOs, to explore the implications of these mechanisms on the design of DLTWOs, and to develop techniques for estimating the frequency stability of DLTWOs during the design phase. The following sections summarize the results of this work in these areas.

6.1 Frequency stability in DLTWOs

The dual nature of the DLTWO complicates the analysis of its frequency stability. On the one hand it is very much like a classical Leeson-style oscillator in which the output of an amplifier is tied to the input via a resonator that provides just the right amount of phase shift to support oscillation. Unfortunately, in the case of the DLTWO the amplifier is distributed throughout the resonator which makes it impossible to separate the two entities mathematically as Leeson did. On the other hand, while the DLTWO looks very much like a ring oscillator, the separation of the oscillator into discrete stages is difficult because, depending on the details of the particular

DLTWO, the "stages" may work more or less serially. That is, for a DLTWO with many "stages" two or three of them might be considered to be working more in parallel than serially, while for a DLTWO with very few stages it is more obvious that you can treat them serially. It turns out that for practical DLTWOs, the stages do work fairly independently with at most two or three in various stages of transition while all of the remaining stages are latched. This makes it possible to analyze the DLTWO using time-domain perturbation analysis. Time-domain perturbation analysis shows that the excess phase is mainly introduced when voltage noise originating in the latched stages is amplified in the transitioning latches. It is possible to estimate the excess phase introduced in the transitioning latch using a technique suggested by the work of Weigandt and to combine the contributions of the various stages to estimate a figure of merit κ as suggested by McNeill. This analysis is compicated by the non-linear nature of the latch and the fact that in general it is impossible to determine the extent of cooperation between adjacent stages without simulation, but it is possible to get a ballpark result.

6.2 ISF simulation of DLTWOs

To get a more accurate estimate of the frequency stability of a DLTWO it is necessary to do at least some simulation. The best approach for evaluating frequency stability using simulation is probably Hajimiri's impulse sensitivity function (ISF) technique. The problem with this technique is that it is extremely expensive, especially for DLTWOs which have a large number of active devices. This work shows that it is possible to dramatically reduce the number of required simulations and still achieve accurate results with a managable number of simulations. This can save tremendous amounts of design time reducing the overall design cycle appreciably. The specific techniques applied here reduce the number of simulations required by at least two orders of magnitude, and can provide fair accuracy with as few as three simulations. 41 simulations should provide very good accuracy for virtually any DLTWO. The simulations required do not rely on any specialized SPICE techniques. Standard

simulation techniques as used in industry are all that is required.

Thousands of simulations of various DLTWO configurations were performed in the process of completing this work. The results show very clearly that the time-domain perturbation analysis presented in Chapter 3 is correct in predicting that the DLTWO stage is most susceptible to noise just as its latch is entering the amplification region. The validity of the simulation techniques described is demonstrated by comparison with measurements of the first actual DLTWO integrated circuits to have been implemented in silicon.

6.3 Design Implications

Using the results of the time-domain analysis and simulations, a number of design insights have been developed. While the circuit designer is usually confined to a given process, there are certain parameters of the DLTWO design that are likely to be variable by the designer. Several of the most likely parameters have been considered here.

Probably the most surprising result in the area of design implications is that the averaging effect of tying together multiple rings is largely offset by the resulting increase in the number of noise sources and the number of transitioning latches integrating the noise. This is true because the averaging effect is linear even when the rings are multiply connected, which is not an obvious result. Configurations in which each ring is connected to two adjacent rings derive the same benefit as configurations in which each ring is connected to only one adjacent ring; the averaging factor seems to be determined solely by the total number of rings in the implementation.

A less surprising result is that frequency stability improves with an increase in V_{dd}. This is mainly due to the fact that the energy in the propagating wavefront on the DLTWO increases with the square of the supply voltage while the noise is practically independent of the supply. On a related note the DLTWO is shown to be extremely resistant to noise on the supply rails largely due to its differential nature.

A very important design parameter in the design of the DLTWO is the distribution

of the inverters around the ring. The P/N ratio and the sizing of the transistors that make up the latches has a dramatic impact on the speed of the wavefront as it traverses the ring. The main result here is that, holding the frequency of operation constant, the excess phase introduced in each stage decreases as the number of stages is increased. This is because the increase in the number of stages increases the edge rate, reducing the percentage of time that the wavefront is susceptible to noise. Unfortunately, this effect is negated because the ratio of improvement in edge rate to increase in the number of stages is generally less than unity. The overall result is that – from a frequency stability standpoint – the best approach is probably to use just enough stages to produce an adequate edge rate for the load that the oscillator will see and no more.

6.4 Suggestions for future work

Possible extensions of this work might include:

Analysis of non-CMOS DLTWOs Since DLTWOs are probably of greatest interest in multi-GHz clocking systems, it is natural to consider the possibility of implementing them using more exotic processes such as SiGe.

Study of improvements to SPICE simulation techniques While the SPICE simulation techniques used here are mainly industry-standard techniques, there may be improvements in efficiency if a built-in transmission line model is used instead of the LRC model used here. There are certainly other aspects of the SPICE simulation process that may likewise be improved, such as using lumped transistor models instead of binned models.

Study of changes to latches to improve performance Since the amplification of the noise by transitioning latches is the main source of excess phase in the DLTWO stage, it is natural to wonder if it might be possible to deliberately reduce the gain of the latch without destroying the overall performance of the oscillator.

List of References

[1] J. Wood, S. Lipa, P. Franzon, M. Steer, "Multi-gigahertz low-power low-skew rotary clock scheme," in *Proc. ISSCC*, San Francisco, CA, Feb 2001, pp. 400–401,470.

[2] PCT/GB00/00175 John Wood, MultiGiG Ltd, assigned.

[3] C. Walker, *Capacitance, Inductance, and Crosstalk Analysis,* Artech House, 1990.

[4] J.Wood, T.C. Edwards, S. Lipa, "Rotary traveling-wave oscillator arrays: a new clock technology," *IEEE Journal of Solid-State Circuits*, Volume 36, Issue 11, Nov 2001, pp. 1654 -1665.

[5] I Bernstein, "On fluctuations in the neighbourhood of periodic motion of an auto-oscillating system," *C.R. Acad. Sci. USSR*, Volume XX, no. 1, 1938, pp. 11-16.

[6] L.S. Cutler, C.L. Searle, "Some aspects of the theory and measurement of frequency fluctuations in frequency standards," *Proc. IEEE*, Volume 54, Feb 1966, pp. 136-154.

[7] J.A. Barnes, A.R. Chi, L.S. Cutler D.J. Healy, D.B. Leeson, T.E. McGunigal, J.A. Mullen, W.L. Smith, R.L. Sydnor, R.F.C. Vessot, G.M.R. Winkler, "Characterization of frequency stability," *IEEE Trans. Instrum. Meas.*, Volume IM-20, May 1971, pp. 105-120.

[8] D.W. Allan, "Statistics of atomic frequency standards," *Proc. IEEE*, Volume 54, Feb 1966, pp. 221-230.

[9] D.W. Allan, N. Ashby, C.C. Hodge, "The science of timekeeping," Hewlett-Packard Application Note 1289, 1997.

[10] D.A. Howe, T.K. Peppler, "Definitions of "total" estimators of common time-domain variances," *Proc. 2001 IIFCS*, 2001, pp. 127-132

[11] D. B. Leeson, "A simple model of feedback oscillator noise spectrum," *Proc. IEEE*, Volume 54, Feb 1966, pp. 329-330.

[12] B. Razavi, "A study of phase noise in CMOS oscillators," *IEEE JSSC*, Volume 31, No. 3, Mar 1996, pp. 331-343.

[13] B. Razavi, "Analysis, modeling, and simulation of phase noise in monolithic voltage-controlled oscillators," *Proc. IEEE 1995 CICC*, 1995, pp. 323-326.

[14] E. Hafner, "The effects of noise in oscillators," *Proc. IEEE*, Volume 54, Feb 1966, pp. 179-198.

[15] A. Hajimiri, T.H. Lee, *The Design of Low Noise Oscillators,* Kluwer Academic Publishers, 1999.

[16] T.H. Lee, A. Hajimiri, "Oscillator phase noise: a tutorial," *IEEE JSSC*, Volume 35, No. 3, Mar 2000, pp. 326-336.

[17] A. Hajimiri, T.H. Lee, "A general theory of phase noise in electrical oscillators," *IEEE JSSC*, Volume 33, No. 2, Feb 1998, pp. 179-194.

[18] A. Hajimiri, T.H. Lee, "Design Issues in CMOS differential LC oscillators," *IEEE JSSC*, Volume 34, No. 5, May 1999, pp. 717-724.

[19] A. Hajimiri, S. Limotyrakis, T.H. Lee, "Jitter and phase noise in ring oscillators," *IEEE JSSC*, Volume 34, No. 6, Jun 1999, pp. 790-804.

[20] J. Zhang, H. Mei, T. Kwasniewski, "Prediction of phase noise in CMOS distributed oscillators," *Proc. SBMO/IEEE MTT-S Int. Microwave and Optoelectronics Conf.*, Volume 1, Sep 2003, pp. 157-162.

[21] J. Zhang, T. Kwasniewski, H. Mei, "A computational method in predicting distributed oscillator phase noise," *Proc. 5th Int. Conf. on ASIC*, Volume 2, Oct 2003, pp. 1025-1028.

[22] D. Ham, A. Hajimiri, "Virtual damping and Einstein relation in oscillators," *IEEE JSSC*, Volume 38, No. 3, Mar 2002, pp. 407-418.

[23] C.J. White, A. Hajimiri, "Phase noise in distributed oscillators," *Electronics Letters*, Volume 38, No. 23, Nov 2002, pp. 1453-1454.

[24] T.C. Weigandt, B. Kim, P.R. Gray, "Analysis of timing jitter in CMOS ring oscillators," *Proc. ISCAS*, Volume 4, Jun 1994, pp. 27-30.

[25] J.A. McNeill, "Jitter in ring oscillators," *IEEE JSSC*, Volume 32, No. 6, Jun 1997, pp. 870-879

[26] J.A. McNeill, "Jitter in ring oscillators," *Proc. 1994 ISCAS*, Volume 6, May 1994, pp. 201-204

[27] J.A. McNeill, "Jitter in ring oscillators," Ph.D. dissertation, Boston University, 1994.

[28] B. Leung, "A novel model on phase noise of ring oscillator based on last passage time," *IEEE Trans. Circuits and Systems - I*, Volume 31, No. 3, Mar 2004, pp. 471-482.

[29] A. B. Grebene, *Bipolar and MOS Integrated Circuit Design,* John Wiley and Sons, 1984.

[30] L.W. Nagel, "SPICE2: a computer program to simulate semiconductor circuits," Memo ERL-M520, University of California, Berkeley, 1975.

[31] K. Nabors, J. White, "FastCap: a multipole accelerated 3D capacitance extraction program," *IEEE Trans. on CAD*, Volume. 10, No. 11, pp. 1447–1459, 1991.

[32] M. Kamon et. al., "FASTHENRY: a multipole accelerated 3-D inductance extraction program," *IEEE Transactions on Microwave Theory and Techniques*, Volume 42, No. 9, September 1994.

[33] Y. Cheng et. al., *BSIM 3v3 Manual*, University of California, Berkeley, 1995,1996.

[34] NG-SPICE, http://ngspice.sourceforge.net

[35] B. Drakhlis, "Calculate oscillator jitter by using phase-noise analysis," *Microwaves & RF*, Feb 2001, pp. 109-119.

[36] Octave, http://www.octave.org

[37] Grace, http://plasma-gate.weizmann.ac.il/Grace/

Appendix A

Scripts

This study of DLTWOs has relied heavily on the matrix analysis tool Octave [36] and the plotting tool Grace [37]. In addition to perl scripts written to automate the simulation process for DLTWO design and analysis, I have written a large number of scripts for processing and presenting lab and simulation data for this project using these tools. These scripts represent a very useful toolbox for general lab work and are presented in this appendix along with some of the general purpose DLTWO scripts.

A.1 capacitance.m - Octave script

```
# Usage: capacitance(z11 list)
#
# This function takes a z11 vs. frequency list and computes
# a capacitance vs. frequency list.  The input list is simply
# a list of vectors in the form
#
#         frequency z11
#
# where the z11 values are complex impedances.
#
# The return value is a list of vectors in the form
#
#         frequency C L
#
# If the reactance at a given frequency is negative the
# equivalent capacitance is returned in C (otherwise zero)
# and if it is positive the equivalent inductance is returned
# in L (otherwise zero).

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = capacitance (A)
  retval = 0;
  if (nargin != 1)
    usage ("capacitance (z11 list)");
  endif
  if (is_matrix (A) && (columns(A) == 2))
    if (imag(A(1,2)) < 0)
      z = [real(A(1,1)) , -1/(2*pi*A(1,1)*imag(A(1,2))),0];
    else
      z = [real(A(1,1)) , 0 , imag(A(1,2))/(2*pi*A(1,1))];
```

```
    endif
    for idx = 2:rows(A)
      if (imag(A(idx,2)) < 0)
        z = [z;real(A(idx,1)) , -1/(2*pi*A(idx,1)*imag(A(idx,2))),0];
      else
        z = [z;real(A(idx,1)) , 0 , imag(A(idx,2))/(2*pi*A(idx,1))];
      endif
    endfor
    retval = z;
  else
    error ("capacitance: expecting z11 list");
  endif
endfunction
```

A.2 cascade.m - Octave script

```
## Usage:   cascade(A, B, Zo)
##
## This function returns a two-port s-parameter matrix S, which is
## equivalent to the result of cascading two two-ports A and B in
## series:
##
##      -----            -----      -----
##     |     |          |     |    |     |
##   --|  S  |--  =   --|  A  |----|  B  |--
##     |     |          |     |    |     |
##      -----            -----      -----
##
##                      'first'    'second'
##
## Of course, the order is important. A represents the first in
## line and B represents the second in line.

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = cascade (X,Y,z0)
  retval = 0;
  threshold = 1;
  if (nargin < 2)
    usage ("cascade (A,B,[Zo])");
  endif

  if (nargin < 3)
    z0 = 50;
  endif

  if (is_matrix (X) && is_matrix (Y) && (rows(X) == rows(Y)))
```

```
    A=stot(X,z0);
    B=stot(Y,z0);
    empty_list_elements_ok = 1;
    t = [];

    for idx = 1:rows(A)

       if (abs(A(idx,1) - B(idx,1)) > threshold)
         error("cascade: frequency columns must be the same for each list!");
       endif

       t1 = [A(idx,2) A(idx,4);A(idx,3) A(idx,5)];
       t2 = [B(idx,2) B(idx,4);B(idx,3) B(idx,5)];
       t3 = t1 * t2;

       t = [t; A(idx,1) t3(1,1) t3(2,1) t3(1,2) t3(2,2)];

    endfor

    s=ttos(t,z0);
    retval = s;

  else
    error ("cascade: expecting two two-port s-parameter matrices with \
            equal numbers of rows");
  endif
endfunction
```

A.3 constants.m - Octave script

```
# Fundamental Constants ##################
#
#Boltzman's constant      k  = 1.3806503e-23 J/K = 8.617342e-5 eV/K
#Stefan-Boltzmann const.  sigma = 5.670400e-8 W/m^2/K^4
#Speed of light           c  = 2.99792458e8 m/s
#Planck's constant        h  = 6.62606876e-34 J s = 4.13566727e-15 eV s
#                         hbar  = 1.054571596e-34 J s = 6.58211889e-16 eV s
#Charge on electron       qe = 1.602176462e-19 Coloumbs
#Mass of electron         Me = 9.10938188e-31 kg
#Mass of proton           Mp = 1.67262158e-27 kg
#Mass of neutron          Mn = 1.67492716e-27 kg
#Unified AMU              u  = 1.66053873e-27 kg
#Standard gravity         g  = 9.80665 m/s^2 = 32.17405 ft/s^2
#Gravitational constant   G  = 6.673e-11 m^3/kg/s^2
#
#Permittivity of vacuum   e0 = 8.854187817e-12 F/m
#Permeability of vacuum   u0 = 1.2566370614e-6 H/m
#Faraday constant         F  = 96485.3415 C/mol
#Z of vacuum              Zo = 376.730313461 ohms
#Gyromagnetic ratio       gamma = 1.759e11 C/Kg (for g=2)
#
#Gas constant             R  = 8.314472 J/mol/K
#Standard Atmosphere         = 101325 Pa
#
#pi = 3.14159265358979323846264338327950288419716939937510
#
# Conductivities   (S/m) #################
#
# Aluminum        3.54e7
# Brass           2.564e7
# Bronze          1.00e7
# Chromium        3.846e7
# Copper          5.813e7
# Distilled H2O   2e-4
# Gold            4.098e7
# Graphite        7.0e4
# Iron            1.03e7
# Mercury         1.04e6
# Lead            4.56e6
# Nichrome        1.0e6
# Nickel          1.449e7
# Platinum        9.52e6
# Quartz          2e-17
# Rubber          1e-15
# Silver          6.173e7
# Soil (dry)      1e-5
```

```
# Steel (Si)      2e6
# Stainless Steel 1.1e6
# Water (Dist.)   2e-4
# Water (Sea)     3 - 5
#
# Dielectric Constants  ##################
#
#                     1kHz        1MHz         3GHz       10GHz
#
# Air                 1.0059
# Alumina                                                10.2
# Bakelite            5.0
# Beeswax                                                2.35
# Beryllia                                               6.4
# Ceramic (A-35)                                5.6
# Ceramic (C0G)       45
# Ceramic (X7R)       3000
# Ceramic (Z5U)       6000
# Ceramic filed teflon                                   10.2
# Delrin              3.6          3.6
# Epoxies             3-5
# GaAs                13.2                               13
# Germanium           16
# Glycerin            42
# Glass (pyrex)                                 4.82
# Glass Epoxy FR4,5   4.4          4.7                   4.5
# Glass Melamine G9   7.2          7.6
# Glass Phenolic G3   5.5
# Glass Silicone G7   4.2          4.7
# InP                 12.4
# Mica                5.4-8.7
# Nylon               3.6          3.2
# Nylon (610)                                   2.84
# Oil (Mineral)       2.2
# Oil (Silicone)      2.7
# Paper               2-4
# Parafin                                                2.24
# Plexiglass                                    2.6
# Polycarbonate       2.6-2.9
# Polyethylene        2.3-2.4      2.3-2.4               2.25
# Polyester           3.1-3.3
# Polyimide           3.6          3.6
# Polyolefin (irrad.)                                    2.32
# Polyphenylene oxide (PPO)                              2.55
# Polypropylene       2.1-2.3
# Polystyrene         2.1-2.5                            2.54
# PVC (rigid)         3-3.8
# PVC (flexible)      5-7          5-7
```

```
# Quartz              3.8-4.4
# Quartz (fused)                                        3.78
# Rexolite (1422)                               2.54
# Rubber              2.3-4.0
# Saphire                                               11.0
# Silicon                                               11.9
# Silicon Dioxide     3.9
# Silicon Nitride      7
# Silicone Rubber     3.2
# Soil (dry)          3-4
# Styrofoam                                     1.03
# Teflon PFA                      2.1
# Teflon PTFE                     2.1                   2.08
# Tefzel ETFE                     2.6
# Vaseline                                              2.16
# Water               78.5
# Water (Distilled)                             76.7
#
# Densities          ##################
# (at standard temp and pressure)
#
# Hydrogen            8.99e-2 kg/m^3
# Helium              0.1785 kg/m^3
# Nitrogen            1.250 kg/m^3
# Air                 1.293 kg/m^3
# Oxygen              1.429 kg/m^3
# Gasoline            660-690 kg/m^3
# Alcohol             806 kg/m^3
# Water               1000 kg/m^3
# Aluminum            2700 kg/m^3
# Iron                7860 kg/m^3
# Copper              8960 kg/m^3
# Lead                11400 kg/m^3
# Mercury             13600 kg/m^3
```

A.4 cutoff.m - Octave script

```
## Usage: cutoff(mode m, mode n, w(cm), h(cm), relative permittivity)
#
## This function returns the cutoff frequency for the given
## mode for a rectangular waveguide of given dimensions (given
## in cm) filled with dielectric with the given relative
## permittivity.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2003
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = cutoff (m,n,a,b,eeff)
  retval = 0;
  if (nargin != 5)
    usage ("cutoff (mode m, mode n, w(cm), h(cm), relative permittivity)");
  endif

  mye = 8.854187817e-12*eeff;
  myu0 = 1.2566370614e-6;
  mypi = 3.14159265358979;
  speed = 1/(sqrt(myu0*mye));

  retval = speed/(2*mypi)*sqrt((m*mypi/(a*1.0e-2))^2+(n*mypi/(b*1.0e-2))^2);

endfunction
```

A.5 dBm2V.m - Octave script

```
## Usage: dBm2V(dBm,[Zo])
#
## This function converts dBm to Volts RMS.  If Zo is
## not specified it is assumed to be 50 ohms.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.


function retval = dBm2V (dBm,z0)
  retval = 0;
  if ((nargin != 2) && (nargin != 1))
    usage ("dBm2V (dBm,[Zo])");
  endif

  if (nargin == 1)
    z0 = 50;
  endif

  empty_list_elements_ok = 1;

  outmat = [];
  if (is_matrix(dBm))
    for idx1 = 1:rows(dBm)
      outrow = [];
  for idx2 = 1:columns(dBm)
        outrow = [outrow sqrt(z0*0.001*10^(dBm(idx1,idx2)/10))];
      endfor
  outmat = [outmat; outrow];
endfor
  retval = outmat;
```

```
  endif

endfunction
```

A.6 dBm2W.m - Octave script

```
## Usage: dBm2W(dBm,[Zo])
#
## This function converts dBm to Watts.  If Zo is
## not specified it is assumed to be 50 ohms.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = dBm2W (dBm,z0)
  retval = 0;
  if ((nargin != 2) && (nargin != 1))
    usage ("dBm2V (dBm,[Zo])");
  endif

  if (nargin == 1)
    z0 = 50;
  endif

  retval = 0.001*10^(dBm/10);

endfunction
```

A.7 decascade.m - Octave script

```
## Usage:   decascade(S,A [or B],[sel])
##
## This function takes a two-port s-parameter matrix S, which is
## assumed to be the result of cascading two two-ports in series,
## and determines the s-parameters of one of the constituent
## two-ports given the s-parameters of the other one.
##
##      -----            -----      -----
##     |     |          |     |    |     |
##   --|  S  |--  =   --|  A  |----|  B  |--
##     |     |          |     |    |     |
##      -----            -----      -----
##
##                      'first'    'second'
##
## The s-parameter matrix for the first constituent two-port is
## called A, and the s-parameter matrix of the second constituent
## two-port is called B.  One of these must be specified as the
## second parameter.  The third parameter (sel), if present,
## indicates which constituent two-port network the second
## parameter represents.  Allowed values are 'first' and 'second'.
## This parameter defaults to 'first' if it is not present.
##
## The s-parameter matrix for the other constituent two-port
## is returned.

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = decascade (X,Y,seltext)
  retval = 0;
  threshold = 1;
```

```
 if ((nargin < 2) || (nargin > 3))
   usage ("decascade (S,A [or B],[sel])");
 endif

 if (!(is_matrix (X) && is_matrix (Y) && (rows(X) == rows(Y))))
 error("decascade: first two parameters must be two-port \
               s-parameter matrices with equal numbers of rows!");
 endif

 if (nargin == 2)
     seltext = 'first';
 elseif (nargin == 3)
 if (isnumeric(seltext))
seltext = 'first';
     else
        if  (!(strcmp(seltext,'first') || strcmp(seltext,'second')))
  error ("decascade: sel must be 'first' or 'second'.");
        endif
     endif
 endif

 empty_list_elements_ok = 1;
 A=stot(X);
 B=stot(Y);
 t = [];

 for idx = 1:rows(A)

    if (abs(A(idx,1) - B(idx,1)) > threshold)
      error("decascade: frequency columns must be the same for each list!");
    endif

    t1 = [A(idx,2) A(idx,4);A(idx,3) A(idx,5)];
    t2 = [B(idx,2) B(idx,4);B(idx,3) B(idx,5)];
    t4 = inv(t2);
    if (strcmp(seltext,'first'))
       t3 = t4 * t1;
    else
       t3 = t1 * t4;
    endif

    t = [t; A(idx,1) t3(1,1) t3(2,1) t3(1,2) t3(2,2)];

 endfor

 s=ttos(t);
 retval = s;
```

```
endfunction
```

A.8 delay.m - Octave script

```
## Usage: delay(S,t,p)
##
## This function applies a delay of t seconds to one or more ports,
## p, of an s-parameter matrix, S.  For 1-port s-parameter data the
## value of p is ignored.  For 2-port data p can be either 1, 2, or
## "ALL" (in quotes).  If 1 or 2 is specified the result is equivalent
## to the port extension feature on a VNA. If "ALL" is specified the
## result is equivalent to the electrical delay feature on a VNA.
##
## The delayed s-parameter matrix is returned.

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = delay (S,t,p)

  empty_list_elements_ok = 1;

  retval = [];

  if (nargin != 3)
    usage ("delay (S,t,p)");
  endif

  PI = 3.141592653589793238;

  if  (is_matrix (S) && (columns(S) == 5))

      s_ans = [];
      for x=1:rows(S)
        phase_offset = 2*PI*S(x,1)*t;
```

```
if (isstr(p))
           offset_matrix = [exp(-1*i*phase_offset)         0;
                0               exp(-1*i*phase_offset)];
elseif (p == 1)
           offset_matrix = [exp(-1*i*phase_offset)         0;
                0                          1           ];
elseif (p == 2)
           offset_matrix = [    1                         0;
                0               exp(-1*i*phase_offset)];
else
   error("delay: for two-ports p must be 1,2, or \"ALL\"");
endif
        stmp = [S(x,2) S(x,4); S(x,3) S(x,5)];
sprime = offset_matrix * stmp * offset_matrix;
        s_ans = [s_ans;
                 S(x,1) sprime(1,1) sprime(2,1) sprime(1,2) sprime(2,2)];
      endfor
      retval = s_ans;

  elseif   (is_matrix (S) && (columns(S) == 2))

      s_ans = [];
      for x=1:rows(S)
        phase_offset = 2*PI*S(x,1)*t;
stmp = S(x,2)*exp(-2*i*phase_offset);
        s_ans = [s_ans; S(x,1) stmp];
      endfor
      retval = s_ans;

  else
      error ("delay:S must be an s-parameter matrix!");
  endif
endfunction
```

A.9 effective dielectric constant.m - Octave script

```
## Usage: effective_dielectric_constant(relative dielectric \
constant, fill factor)
#
## This function returns the effective dielectric constant given a
## relative dielectric constant, and fill factor (the ratio of the dielectric
## area to the total area that would be used in stripline)
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.


function retval = effective_dielectric_constant (er,q)
  retval = 0;
  if ((nargin < 1))
    usage ("effective_dielectric_constant (relative dielectric \
            constant,fill factor)");
  endif

  retval = 1+q*(er-1);

endfunction
```

A.10 fsweep.pl - perl script

```
#!/usr/local/bin/perl5
#This perl script automatically generates a series of SPICE
#input decks and invokes HSPICE on them, sweeping over a parameter
#of choice.  It was originally used to study the effect of
#different parameters on frequency, hence the name.  It is
#offered as an example of the level of detail required for
#good DLTWO simulation.  The proprietary foundry information
#has been edited out and it has been reformatted for printing.

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

########## foundry data #########################
$metgap = ?;
$ppgap = ?;
$resistivity = ?;
$metal_metal_cap = ?;
$n_length = ?;
$p_length = ?;
$pfactor = ?;
$afactor = ?;
$diel_height =?;
$fill_ratio =?;
$pp_height = ?;
$conductor_height =?;
$maxbin = ?;
$p_scale = ?;
$pmodnam = ?;
$nmodnam = ?;

################################################
```

```
$xcslist =    "X-----C---------C-----------C----------C-----";
$powerlist =  "PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP";
$groundlist = "GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG";

$ring_size = shift;
$numcells = shift;
$total_nwidth = shift;

$voltage = shift;
$conductor_width = shift;
$pitch = shift;
$fguess = shift;
$pnode = shift;
$load = shift;
$pp_pitch = shift;
$pp_offset = shift;
$pp_width = shift;
$x_offset = shift;
$vestige = shift;
$transtep = shift;
$trantotal = shift;

$lvar = shift;
$lvarstop = shift;
$lvarinc = shift;

$spice_file = "sim-$lvar";

select STDOUT;
$sim_num = 0;

for ( ; $$lvar < $lvarstop; $$lvar = $$lvar + $lvarinc) {
  $sim_num++;
  $spice_input_file = $spice_file."-$$lvar".".sp";
  $spice_output_file = $spice_file."-$$lvar".".out";
  open(OFIL, ">$spice_input_filefile".".sp") or die\
         "can't open $spice_input_file";
  $ring_length = $ring_size/1000.0;
  $x_margin -$conductor_width/2.0;
  $cell_length = $ring_length/$numcells;
  $cell_left = $cell_length/(-2.0);
  $cell_right = $cell_length/2.0;
  if ($cell_left > $x_margin) {
     printf RFIL "WARNING: cell length ($cell_length) is too short to\
 use crossover...\n";
  }
  $cap_per_len = $metal_metal_cap*$conductor_width*$fill_ratio/2.0;
  $line_line_cap = $cap_per_len * $cell_length;
```

```
  $line_line_cap = $line_line_cap + $load;
  $discrete = 10;
  $bh = $conductor_width/$discrete;
  $bw = $bh;

  $nwidth = $total_nwidth/$numcells;
  $total_pwidth = $pscale*$nwidth;

  if ($nwidth > $maxbin) {
    $n_num = abs(int($nwidth/$maxbin))+1.0;
    $n_width = 1e-6*$nwidth/$n_num;
  }
  else {
    $n_width = 1e-6*$nwidth;
    $n_num = 1;
  }

  $p_width = $n_width*$p_scale;
  $n_pd = $n_width*2+$pfactor;
  $n_ad = $n_width*$afactor;
  $p_pd = $p_width*2+$pfactor;
  $p_ad = $p_width*$afactor;

  print OFIL "* # file name temp.sp\n";

  open(TF, ">straight.qui") or die "can't open fc file";
  print TF "0 straight.qui\n";

  for ($cxp = 0; $cxp < $cell_length; $cxp = $cxp+$bw) {
    write_column(1,$cxp,0,0,TF);
  }

  for ($cxp = 0; $cxp < $cell_length; $cxp = $cxp+$bw) {
    write_column(2,$cxp,$pitch,0,TF);
  }

  close(TF);

  open(LF, ">straight.lst") or die "can't open fc file";

  print LF "*
*
C straight.qui 4.0  0.0 0.0 0.0\n";

  close(LF);

  $cmultiplier = 1.0;
  printf STDERR "Starting fastcap run for straight section\n";
```

```
  @fcout = ‘fastcap -lstraight.lst‘;
  foreach $fcline (@fcout) {
      if ($fcline =~ /1\%GROUP1/) {
      ($dum1, $dum2, $self_cap, $mut_cap) = split /\s+/,$fcline;
      }
      elsif ($fcline =~ /CAPACITANCE MATRIX/) {
      ($dum1, $dum2, $mult_string) = split /\s+/,$fcline;
         if ($fcline =~ /femto/) {
           $cmultiplier = 1e-15;
         }
         elsif ($fcline =~ /pico/) {
           $cmultiplier = 1e-12;
         }
         elsif ($fcline =~ /nano/) {
           $cmultiplier = 1e-9;
         }
      }

  }
  $cap_ltol_straight = -$mut_cap*$cmultiplier;
  $cap_ltos_straight = ($mut_cap+$self_cap)*$cmultiplier;
  printf STDERR "Cl_l for straight section is $cap_ltol_straight\n";
  printf STDERR "Cl_s for straight section is $cap_ltos_straight\n";

  $discrete = 10;
  $bh = $conductor_width/$discrete;
  $bw = $bh;

  $cturn = ($cell_length - $x_offset)/2.0+$bw;
  $cboxes = $x_offset/$bw-1;
  $boffset = $pitch/$cboxes;

  open(TF, ">cross.qui") or die "can’t open fc file";
  print TF "0 cross.qui\n";

  for ($cxp = 0; $cxp < $cturn; $cxp = $cxp+$bw) {
    write_column(1,$cxp,0,0,TF);
  }
  $cyp = 0;
  for ($kk=0; $kk< $cboxes; $kk++) {
    $cyp = $cyp+$boffset;
    write_column(1,$cxp,$cyp,0,TF);
$cxp = $cxp+$bw;
  }
  for (; $cxp < ($cell_length-$bw); $cxp = $cxp+$bw) {
    write_column(1,$cxp,$pitch,0,TF);
  }
```

```
  for ($cxp = 0; $cxp < $cturn; $cxp = $cxp+$bw) {
    write_column(2,$cxp,$pitch,$diel_height,TF);
  }
  $cyp = $pitch;
  for ($kk=0; $kk< $cboxes; $kk++) {
    $cyp = $cyp-$boffset;
    write_column(2,$cxp,$cyp,$diel_height,TF);
$cxp = $cxp+$bw;
  }
  for (; $cxp < ($cell_length-$bw); $cxp = $cxp+$bw) {
    write_column(2,$cxp,0,$diel_height,TF);
  }

  close(TF);

  open(LF, ">cross.lst") or die "can't open fc file";

  print LF "*
*
C cross.qui 4.0  0.0 0.0 0.0\n";

  close(LF);

  $cmultiplier = 1.0;
  printf STDERR "Starting fastcap run for cross section\n";
  @fcout = `fastcap -lcross.lst`;
  foreach $fcline (@fcout) {
      if ($fcline =~ /1\%GROUP1/) {
      ($dum1, $dum2, $self_cap, $mut_cap) = split /\s+/,$fcline;
  }
      elsif ($fcline =~ /CAPACITANCE MATRIX/) {
      ($dum1, $dum2, $mult_string) = split /\s+/,$fcline;
     if ($fcline =~ /femto/) {
   $cmultiplier = 1e-15;
 }
 elsif ($fcline =~ /pico/) {
   $cmultiplier = 1e-12;
 }
 elsif ($fcline =~ /nano/) {
   $cmultiplier = 1e-9;
 }
  }

  }
  $cap_ltol_cross = -$mut_cap*$cmultiplier;
  $cap_ltos_cross = ($mut_cap+$self_cap)*$cmultiplier;
  printf STDERR "Cl_l for cross section is $cap_ltol_cross\n";
  printf STDERR "Cl_s for cross section is $cap_ltos_cross\n";
```

```
#Estimate corner using straight sections
$cap_ltol_corner =  0.33*$cap_ltol_straight;
$cap_ltos_corner1 = 0.25*$cap_ltos_straight;
$cap_ltos_corner2 = 0.75*$cap_ltos_straight;

printf STDERR "Cl_l for corner section is $cap_ltol_corner\n";
printf STDERR "Cl_s1 for corner section is $cap_ltos_corner1\n";
printf STDERR "Cl_s2 for corner section is $cap_ltos_corner2\n";

for ($x = 0; $x < $numcells; $x++) {
  $tcell = $x;
  $ncell = $x+1;
  $powtype = substr $powerlist,$tcell,1;
  $gndtype = substr $groundlist,$tcell,1;
  if ($powtype =~ /P/) {
     print OFIL "rpstrap$tcell vdd pow$tcell 0.001\n";
  }
  if ($gndtype =~ /G/) {
     print OFIL "rgstrap$tcell gnd$tcell 0.001\n";
  }
  print OFIL "vla$tcell a$tcell fha$tcell 0\n";
  print OFIL "vlb$tcell b$tcell fhb$tcell 0\n";
  $celltype = substr $xcslist,$tcell,1;
  if ($celltype =~ /X/)  {
    print OFIL "xfh$tcell fha$tcell a$ncell fhb$tcell b$ncell x_ind\n";
    print OFIL "cll$tcell a$ncell b$ncell $cap_ltol_cross\n";
    print OFIL "clsa$tcell a$ncell gnd$tcell $cap_ltos_cross\n";
    print OFIL "clsb$tcell b$ncell gnd$tcell $cap_ltos_cross\n";
    print OFIL "vpc$tcell pow$tcell pow$ncell dc 0\n";
    print OFIL "vgc$tcell gnd$tcell gnd$ncell dc 0\n";
  }
  elsif ($celltype =~ /C/)  {
    print OFIL "xfh$tcell fha$tcell a$ncell fhb$tcell b$ncell c_ind\n";
    print OFIL "cll$tcell a$ncell b$ncell $cap_ltol_corner\n";
    print OFIL "clsa$tcell a$ncell gnd$tcell $cap_ltos_corner1\n";
    print OFIL "clsb$tcell b$ncell gnd$tcell $cap_ltos_corner2\n";
    print OFIL "vpc$tcell pow$tcell pow$ncell dc 0\n";
    print OFIL "vgc$tcell gnd$tcell gnd$ncell dc 0\n";
  }
  else {
    print OFIL "xfh$tcell fha$tcell a$ncell fhb$tcell b$ncell\
                pow$tcell pow$ncell gnd$tcell gnd$ncell s_ind\n";
    print OFIL "cll$tcell a$ncell b$ncell $cap_ltol_straight\n";
    print OFIL "clsa$tcell a$ncell gnd$tcell $cap_ltos_straight\n";
    print OFIL "clsb$tcell b$ncell gnd$tcell $cap_ltos_straight\n";

    for ($y = 0; $y < $n_num; $y++) {
```

```
         $clet = chr(97+$y);
         print OFIL "mna$tcell$clet b$tcell a$tcell\
         gnd$tcell gnd$tcell $nmodnam l=$n_length w=$n_width\n";
         print OFIL "+ad=$n_ad as=$n_ad\n";
         print OFIL "+pd=$n_pd ps=$n_pd\n";
         print OFIL "+m=1\n";
         print OFIL "mpa$tcell$clet b$tcell a$tcell\
         pow$tcell pow$tcell $pmodnam l=$p_length w=$p_width\n";
         print OFIL "+ad=$p_ad as=$p_ad\n";
         print OFIL "+pd=$p_pd ps=$p_pd\n";
         print OFIL "+m=1\n";
      }
      for ($y = 0; $y < $n_num; $y++) {
         $clet = chr(97+$y);
         print OFIL "mnb$tcell$clet a$tcell b$tcell gnd$tcell gnd$tcell\
         $nmodnam l=$n_length w=$n_width\n";
         print OFIL "+ad=$n_ad as=$n_ad\n";
         print OFIL "+pd=$n_pd ps=$n_pd\n";
         print OFIL "+m=1\n";
         print OFIL "mpb$tcell$clet a$tcell b$tcell pow$tcell pow$tcell\
         $pmodnam l=$p_length w=$p_width\n";
         print OFIL "+ad=$p_ad as=$p_ad\n";
         print OFIL "+pd=$p_pd ps=$p_pd\n";
         print OFIL "+m=1\n";
      }
    }
  }
  print OFIL "rx1 a0 b$ncell 0.001\n";
  print OFIL "rx2 b0 a$ncell 0.001\n";

  open(TF, ">sfh.inp") or die "can't open shf.inp for writing";
  $a_coord = $pitch/2.0;
  $b_coord = -$a_coord;
  $p_coord = $pp_pitch/2+$pp_offset;
  $g_coord = -$pp_pitch/2+$pp_offset;
  $negppgap = -$ppgap;
  print TF "*sfh.inp
.units m
.freq fmin=$fguess fmax=$fguess ndec=1
.default rho=$resistivity
.default nwinc=9 nhinc=3

NA0 x=$cell_left y=$a_coord z=$metgap
NA1 x=$cell_right y=$a_coord z=$metgap
NB0 x=$cell_left y=$b_coord z=$metgap
NB1 x=$cell_right y=$b_coord z=$metgap
NP0  x=$cell_left y=$p_coord z=-$ppgap
NP1  x=$cell_right y=$p_coord z=$negppgap
```

```
NG0  x=$cell_left y= $g_coord z=-$ppgap
NG1  x=$cell_right y= $g_coord z=$negppgap
E0 NA0 NA1  w=$conductor_width h=$conductor_height
E1 NB0 NB1  w=$conductor_width h=$conductor_height
E2 NP0 NP1  w=$pp_width h=$pp_height
E3 NG0 NG1  w=$pp_width h=$pp_height

.external NA0 NA1
.external NP0 NP1
.external NG0 NG1
.external NB0 NB1
.end\n";
  close(TF);

  system("fasthenry sfh.inp");
  system("MakeLcircuit Zc.mat > sfh.sp");

  print OFIL ".subckt s_ind 1 2 3 4 5 6 7 8\n";
  @spicelist = `cat sfh.sp`;
  print OFIL @spicelist;
  print OFIL ".ends\n";

  open(TF, ">xfh.inp") or die "can't open xfh.inp for writing";
  $a_coord = $pitch/2.0;
  $b_coord = -$a_coord;
  $x1_coord = $cell_left + $x_offset;
  $x2_coord = $cell_right - $x_offset;
  $p_coord = $pp_pitch/2+$pp_offset;
  $g_coord = -$pp_pitch/2+$pp_offset;
  $negppgap = -$ppgap;
  print TF "*xfh.inp
.units m
.default rho=$resistivity
.default nwinc=9 nhinc=3
.freq fmin=$fguess fmax=$fguess ndec=1

NA0 x=$cell_left y=$a_coord z=$metgap
NA1 x=$x1_coord y=$a_coord z=$metgap
NA2 x=$x2_coord y=$b_coord z=$metgap
NA3 x=$cell_right y=$b_coord z=$metgap
NB0 x=$cell_left y=$b_coord z=$metgap
NB1 x=$x1_coord y=$b_coord z=$metgap
NB2 x=$x2_coord y=$a_coord z=$metgap
NB3 x=$cell_right y=$a_coord z=$metgap

E0 NA0 NA1  w=$conductor_width h=$conductor_height
E1 NA1 NA2  w=$conductor_width h=$conductor_height
E2 NA2 NA3  w=$conductor_width h=$conductor_height
```

```
E3 NB0 NB1  w=$conductor_width h=$conductor_height
E4 NB1 NB2  w=$conductor_width h=$conductor_height
E5 NB2 NB3  w=$conductor_width h=$conductor_height

.external NA0 NA3
.external NB0 NB3
.end\n";
  close(TF);

  system("fasthenry xfh.inp");
  system("MakeLcircuit Zc.mat > xfh.sp");

  print OFIL ".subckt x_ind 1 2 3 4\n";
  @spicelist = `cat xfh.sp`;
  print OFIL @spicelist;
  print OFIL ".ends\n";

  open(TF, ">cfh.inp") or die "can't open cfh.inp for writing";
  $negvestige = -$vestige;
  print TF "*cfh.inp
.units m
.default rho=$resistivity
.default nwinc=9 nhinc=3
.freq fmin=$fguess fmax=$fguess ndec=1

NA0 x=$negvestige y=$pitch z=$metgap
NA1 x=0 y=$pitch z=$metgap
NA2 x=$pitch y=0 z=$metgap
NA3 x=$pitch y=$negvestige z=$metgap
NB0 x=$negvestige y=0 z=$metgap
NB1 x=0 y=0 z=$metgap
NB2 x=0 y=$negvestige z=$metgap
E0 NA0 NA1  w=$conductor_width h=$conductor_height
E1 NA1 NA2  w=$conductor_width h=$conductor_height
E2 NA2 NA3  w=$conductor_width h=$conductor_height
E3 NB0 NB1  w=$conductor_width h=$conductor_height
E4 NB1 NB2  w=$conductor_width h=$conductor_height

.external NA0 NA3
.external NB0 NB2
.end\n";
  close(TF);

  system("fasthenry cfh.inp");
  system("MakeLcircuit Zc.mat > cfh.sp");

  print OFIL ".subckt c_ind 1 2 3 4\n";
  @spicelist = `cat cfh.sp`;
```

```
  print OFIL @spicelist;
  print OFIL ".ends\n";

  @modinfo = `cat spicemodels`;
  print OFIL @modinfo;

  print OFIL ".options ingold=2 acct=0
vps_main vdd 0 pulse 0 $voltage 0 1e-9 1e-9 1
vpowconn vdd pow0 dc 0
vgndconn gnd0 0 dc 0

.tran $transtep $trantotal
.print tran v($pnode)
.end\n";

  close(OFIL);

  printf STDERR "Starting spice run $sim_num for $spice_input_file.\n";
  system("hspice $spice_input_file > $spice_output_file");
}

close(RFIL);
exit(0);


  sub write_column {
      my $cname = shift(@_);
      my $x = shift(@_);
      my $y = shift(@_);
      my $z = shift(@_);
  my $fhan = shift(@_);
  my $i;

  for ($i=0; $i<$discrete; $i++) {
        write_block($cname,$x,$y,$z,$fhan);
$y = $y+$bh;
      }

      return;
  }


  sub write_block {
      my $cname = shift(@_);
      my $x = shift(@_);
      my $y = shift(@_);
      my $z = shift(@_);
  my $fhan = shift(@_);
```

```
  printf $fhan "Q %d ",$cname;
  printf $fhan "%e ",$x;
  printf $fhan "%e ",$y;
  printf $fhan "%e ",$z;

  printf $fhan "%e ",$x+$bw;
  printf $fhan "%e ",$y;
  printf $fhan "%e ",$z;

  printf $fhan "%e ",$x+$bw;
  printf $fhan "%e ",$y+$bh;
  printf $fhan "%e ",$z;

  printf $fhan "%e ",$x;
  printf $fhan "%e ",$y+$bh;
  printf $fhan "%e\n",$z;

      return;
  }

__END__
```

A.10.1 FASTCAP and FASTHENRY discretization examples for fsweep.pl

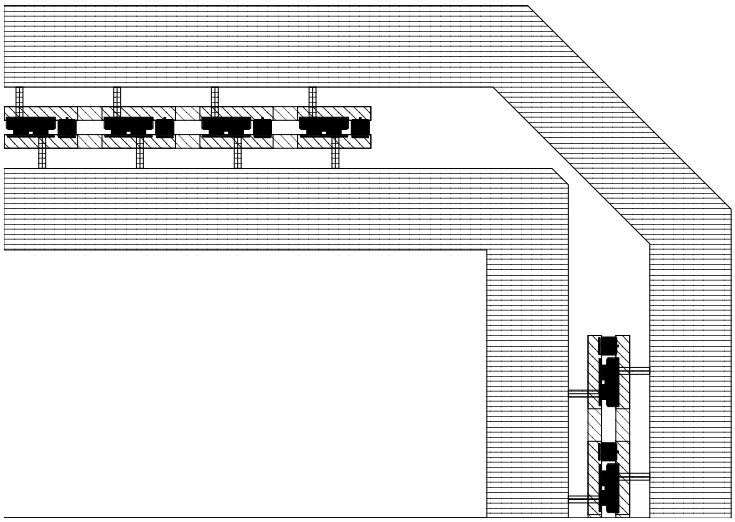

Figure A.1: Simplified view of layout style at the corner of a DLTWO

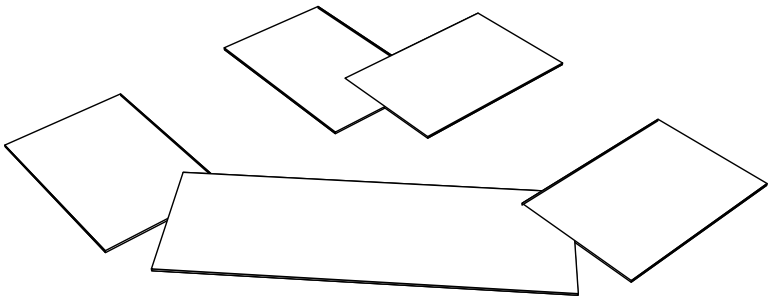

Figure A.2: FASTHENRY zbuf picture corresponding to corner in Figure A.1

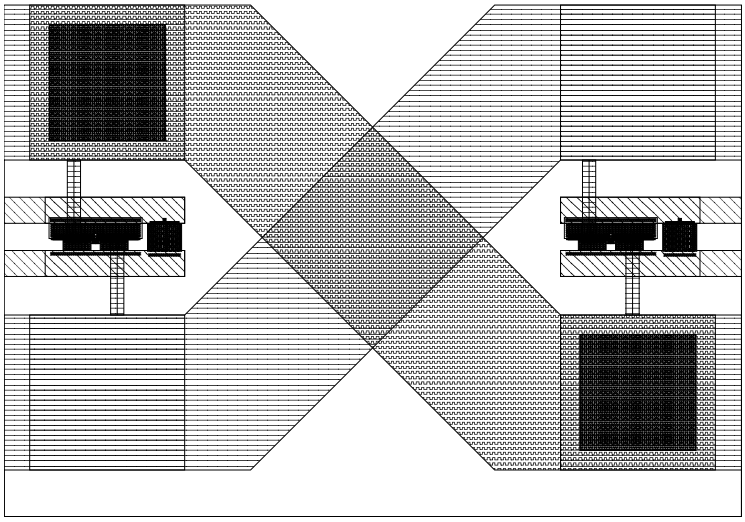

Figure A.3: Figure showing layout style for crossover on test chip.

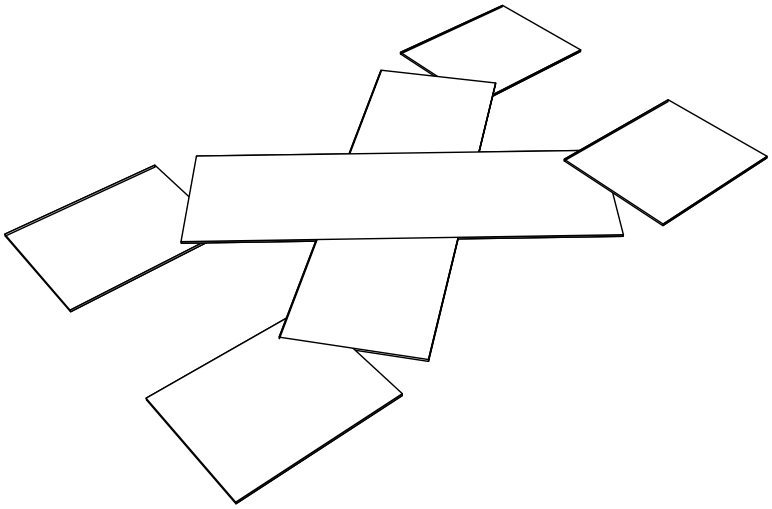

Figure A.4: FASTHENRY zbuf picture corresponding to Figure A.3

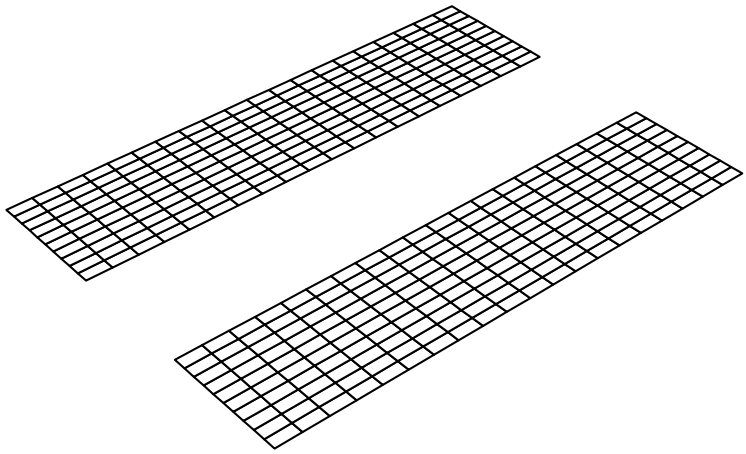

Figure A.5: FASTCAP discretization showing straight section of transmission line

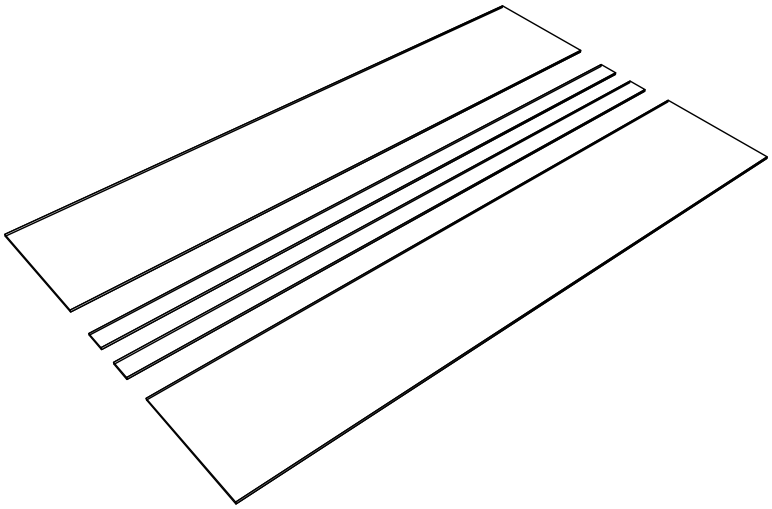

Figure A.6: FASTHENRY zbuf picture showing discretization used for straight part of differential transmission line.

A.11 gammazc.m - Octave script

```
## Usage:   gammazc(A,l,[Zo])
##
## This function takes a two-port s-parameter matrix (A) for a transmission
## line and the length of the line (l) and returns the propagation constant
## gamma (per unit length) and Zc of the line as a function of frequency.
## The line length is only used for gamma.  If length is given in meters,
## for example, the real part of gamma is in Np/m and the imaginary part
## is in rads/m.  The s-parameter matrix is a column of s-parameters vs.
## frequency in the following format:
##
##       frequency s11 s21 s12 s22
##
## The function also optionally accepts the characteristic impedance of the
## measurement system (Zo) which defaults to 50 ohms.
##
## The returned matrix is a column of rowvectors in the following format:
##
##       frequency  Zc  gamma
##
## where all numbers are complex
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = gammazc (X,l,z0)
  retval = 0;
  if (nargin < 2)
    usage ("gammazc (A,line length,[Zo])");
  endif

  if (nargin < 3)
```

```
    z0=50;
  endif

  if (l == 0)
    error("gammazc: Sorry, line length cannot be zero...");
  endif

  if (is_matrix (X) && (columns(X) == 5))
    T=stot(X,z0);

    empty_list_elements_ok = 1;
    t = [];

    for idx = 1:rows(T)
      Zc = sqrt(T(idx,4)/T(idx,3));
      gamma = (log2(T(idx,2)+sqrt(T(idx,2)*T(idx,2)-1)))/l;
      if (real(gamma) < 0.0)
        gamma = (log2(T(idx,2)-sqrt(T(idx,2)*T(idx,2)-1)))/l;
      endif

      t = [t; T(idx,1) Zc gamma];

    endfor

    retval = t;

  else
    error ("gammazc: expecting a two-port s parameter list");
  endif
endfunction
```

A.12 gend.m - Octave script

```
## Usage: gend([p])
##
## Ends a Grace plotting session.  This is the cleanest
## way to do it because it closes the anonymous pipes
## and lets Octave know that the pipe is no longer
## available.  If parameter p is specified, it chooses
## which element of __grace_pipe_data__ refers to the
## Grace process to terminate.  If p is not specified,
## the current Grace session is terminated and the
## variable __current_grace_pipe__ is decremented if it
## positive.    See also the help for grace.m

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
##  This code is freeware.  Everyone is permitted to use, copy,
##  modify, and distribute it for both personal and commercial
##  purposes.
##
##  Because the code is licensed free of charge, there is no
##  warranty for the code, to the extent permitted by applicable
##  law.   Except when otherwise stated in writing, the copyright
##  holder provides the code "as is" without warranty of any kind,
##  either expressed or implied, including but not limited to, the
##  implied warranties of merchantability and fitness for a particular
##  purpose. If the code proves to be defective, the user assumes the
##  cost of all necessary servicing, repair, or correction.

function retval = gend (p)
  global __current_grace_pipe__;
  global __grace_pipe_data__;

  retval = 0;

  if (!exist('__current_grace_pipe__') || !exist('__grace_pipe_data__'))
    error("gend: This function requires global variables that \
are not present. Try typing 'help grace'");
  endif

  if ((__current_grace_pipe__ < 1) || (length(__grace_pipe_data__) < 1))
    return;
  endif

  if (exist("p"))
    if (!isnumeric(p))
  error("gend: if specified, p must be a positive integer.");
    endif
```

```
if (p > length(__grace_pipe_data__))
      error("gend: you need to pick a smaller p!");
endif
  else
    p = length(__grace_pipe_data__);
  endif

  cgp = nth(__grace_pipe_data__,p);
  __grace_pipe_data__ = splice(__grace_pipe_data__,p,1,list());
  __current_grace_pipe__ = length(__grace_pipe_data__);
  fprintf(cgp.wfn,"EXIT\n");
  fflush(cgp.wfn);
  fclose(cgp.wfn);
  fclose(cgp.rfn);
  retval = __current_grace_pipe__;

endfunction
```

A.13 gpipe.cc - octfile c++ source

```
/*

FILE: gpipe.cc

Copyright (C) 2004 Steve Lipa

This is a slight modification of an excerpt of the file src/syscalls.cc
from the 2.1.36 distribution of Octave (see Copyright notice below)
This file provides a clone of the pipe() function called gpipe() which
returns slightly more information than the original function.

This is free software; you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the
Free Software Foundation; either version 2, or (at your option) any
later version.

This software is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
for more details.

You should have received a copy of the GNU General Public License
along with this software; see the file COPYING.  If not, write to the Free
Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

*/


/*

Copyright (C) 1996, 1997 John W. Eaton

This file is part of Octave.

Octave is free software; you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the
Free Software Foundation; either version 2, or (at your option) any
later version.

Octave is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
for more details.

You should have received a copy of the GNU General Public License
along with Octave; see the file COPYING.  If not, write to the Free
```

```
Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

*/

#include <octave/config.h>

#include <iostream.h>

#include <octave/defun-dld.h>
#include <octave/error.h>
#include <octave/oct-obj.h>
#include <octave/pager.h>
#include <octave/symtab.h>
#include <octave/file-ops.h>
#include <octave/file-stat.h>
#include <octave/oct-syscalls.h>
#include <octave/gripes.h>
#include <octave/lo-utils.h>
#include <octave/oct-map.h>
#include <octave/oct-stdstrm.h>
#include <octave/oct-stream.h>
#include <octave/sysdep.h>
#include <octave/syswait.h>
#include <octave/utils.h>
#include <octave/variables.h>

//The function is changed to dynamic and renamed gpipe

DEFUN_DLD (gpipe, args, ,
  "-*- texinfo -*-\n\
@deftypefn {Built-in Function} \
   {[@var{file_ids}, @var{err}, @var{msg}] =} gpipe ()\n\
Create a pipe and return the vector @var{file_ids}, which corresponding\n\
to the reading and writing ends of the pipe.\n\
\n\
If successful, @var{err} is 0 and @var{msg} is an empty string.\n\
Otherwise, @var{err} is nonzero and @var{msg} contains a\n\
system-dependent error message.\n\
@end deftypefn") {
  octave_value_list retval;

  retval(2) = std::string ();
  retval(1) = -1.0;
  retval(0) = Matrix ();

  int nargin = args.length ();

  if (nargin == 0) {
```

```
      int fid[2];
      double ipipe,opipe;
      std::string msg;

      int status = octave_syscalls::pipe (fid, msg);

      if (status < 0)
     retval(2) = msg;
      else {
       FILE *ifile = fdopen (fid[0], "r");
       FILE *ofile = fdopen (fid[1], "w");

       octave_stream is = octave_stdiostream::create \
                           (std::string (), ifile);
       octave_stream os = octave_stdiostream::create \
                           (std::string (), ofile);

        // the new function returns the file numbers ipipe and opipe

ipipe = (double) is.file_number();
opipe = (double) os.file_number();

       octave_value_list file_ids;

        file_ids(3) = ipipe;
       file_ids(2) = opipe;
       file_ids(1) = octave_stream_list::insert (os);
       file_ids(0) = octave_stream_list::insert (is);

       retval(1) = static_cast<double> (status);
        retval(0) = octave_value (file_ids);
  }
    }
  else
    print_usage ("gpipe");

  return retval;
}
```

A.14 gpset.m - Octave script

```
## Usage: gpset(p,[g,[s]])
##
## Sets the global variable __current_grace_pipe__ to p
## and optionally sets the global variables
##  __grace_pipe_data__[__current_grace_pipe__].graph  to  g and
##  __grace_pipe_data__[__current_grace_pipe__].set  to  g
## This is useful if you want to add a graph to a previous
## Grace session or if you want to add a data set to a graph
## in a previous Grace session.   PLEASE NOTE that the newgraph()
## and newset() routines assume that the entries in the
## __grace_pipe_data__ list reflect the last graph and set
## used for a given session.   Thus if you want to add a new
## graph to a session you should set the graph equal to a number
## at least as high as the highest numbered graph in the session.
## Similarly, if you want to add a data set to a graph, you need
## to set g to the number of the graph you want to add the set to,
## AND you need to set s to a number at least as high as the
## highest numbered set already plotted in that graph.
## Please note that if you want to specify s you MUST specify g.

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
##  This code is freeware.  Everyone is permitted to use, copy,
##  modify, and distribute it for both personal and commercial
##  purposes.
##
##  Because the code is licensed free of charge, there is no
##  warranty for the code, to the extent permitted by applicable
##  law.   Except when otherwise stated in writing, the copyright
##  holder provides the code "as is" without warranty of any kind,
##  either expressed or implied, including but not limited to, the
##  implied warranties of merchantability and fitness for a particular
##  purpose. If the code proves to be defective, the user assumes the
##  cost of all necessary servicing, repair, or correction.

function retval = gpset (p,g,s)
  global __current_grace_pipe__;
  global __grace_pipe_data__;

  retval = 0;
  if ((nargin < 1) || (nargin > 3))
    usage ("gpset (p,[g],[s])");
  endif

  __current_grace_pipe__=p;
  cgp = nth(__grace_pipe_data__,p);
```

```
  if (nargin > 1)
    cgp.graph = g;
    if (nargin > 2)
      cpp.set = s;
    endif
  __grace_pipe_data__ = splice(__grace_pipe_data__,p,1,list(cgp));
  endif

endfunction
```

A.15 grace command.m - Octave script

```
## Usage: grace_command("command text")
##
## Sends a command to the current grace pipe.  Mainly useful
## for scripts.  Here are some examples of how you might
## use this:
##
## grace_command("g0 on;focus g0;title \"Some fake S11 data\";redraw\n");
## grace_command("g0 on;focus g0;subtitle \"your subtitle here!\";redraw\n");
## grace_command("g0 yaxis label \"S11 mag (dB)\";redraw\n");
## grace_command("g0 xaxis label \"Frequency (GHz)\";redraw\n");
## grace_command("s0 on;s0 line color \"green\";redraw\n");
## grace_command("s0 on;s0 linewidth 4;redraw\n");
## grace_command("s0 on;s0 linestyle 3;redraw\n");
## grace_command("g0 on;focus g0;;g0.s0 legend \"s11 mag\";redraw\n");
## grace_command("g0 on;focus g0;world xmin 0.045;redraw\n");
## grace_command("g0 on;focus g0;world xmax 26.5;redraw\n");
## grace_command("g0 on;focus g0;xaxes scale logarithmic;redraw\n");
## grace_command("g0 on;focus g0;xaxes scale normal;redraw\n");
## grace_command("g0 on;focus g0;legend 0.92,0.87;redraw\n");
## grace_command("g1 yaxis label \"S11 phase (degrees)\";redraw\n");
## grace_command("g1 xaxis label \"Frequency (GHz)\";redraw\n");
## grace_command("s0 on;s0 line color \"magenta\";redraw\n");
## grace_command("s0 on;s0 linewidth 4;redraw\n");
## grace_command("s0 on;s0 linestyle 5;redraw\n");
## grace_command("arrange(2,1,0.1,0.1,0.4);redraw\n");

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
##  This code is freeware.  Everyone is permitted to use, copy,
##  modify, and distribute it for both personal and commercial
##  purposes.
##
##  Because the code is licensed free of charge, there is no
##  warranty for the code, to the extent permitted by applicable
##  law.   Except when otherwise stated in writing, the copyright
##  holder provides the code "as is" without warranty of any kind,
##  either expressed or implied, including but not limited to, the
##  implied warranties of merchantability and fitness for a particular
##  purpose. If the code proves to be defective, the user assumes the
##  cost of all necessary servicing, repair, or correction.

function retval = grace_command (command)
  global __current_grace_pipe__;
  global __grace_pipe_data__;

  retval = 0;
```

```
  if (nargin != 1)
    usage ("grace_command (\"command text\")");
  endif

  if (!isstr(command))
    error("the command was not a string!");
  endif

  cgp = nth(__grace_pipe_data__,__current_grace_pipe__);
  fprintf(cgp.wfn,"%s\n",command);
  fflush(cgp.wfn);

endfunction
```

A.16 grace.m - Octave script

```
## Usage: grace(x)
##
## Sends an n x 1 or n x 2 matrix (x) to Grace for plotting.  If
## number of columns is one, it is assumed to be the y data, if
## the number of columns is two, the first column is used for x
## and the second column is used for y.
##
## This function returns an index into the global variable
## __grace_pipe_data__ and sets the global variable
## __current_grace_pipe__ to the same value.  This return value
## can be used subsequently (after later Grace sessions are
## started) to come back to a previous Grace session to add data.
##
## To add another data set to an existing graph, use newset().
## To add a new graph to an existing Grace session, use newgraph();
## To send a command to the current Grace session use the
## grace_command() function.
##
## To use this function you need to define two global variables
## in your .octaverc file:
##
##    global __grace_pipe_data__;
##    global __current_grace_pipe__;
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
##  This code is freeware.  Everyone is permitted to use, copy,
##  modify, and distribute it for both personal and commercial
##  purposes.
##
##  Because the code is licensed free of charge, there is no
##  warranty for the code, to the extent permitted by applicable
##  law.   Except when otherwise stated in writing, the copyright
##  holder provides the code "as is" without warranty of any kind,
##  either expressed or implied, including but not limited to, the
##  implied warranties of merchantability and fitness for a particular
##  purpose. If the code proves to be defective, the user assumes the
##  cost of all necessary servicing, repair, or correction.

function retval = grace (x)
  global __grace_pipe_data__;
  global __current_grace_pipe__;

  retval = 0;
  if (nargin != 1)
```

```
    usage ("grace (x)");
  endif

  if (columns(x) == 1)
    x = [(1:rows(x))' x];
  endif

  bad_number_map = finite(x);
  good_number_total = sum(sum(bad_number_map));
  if (good_number_total < (2*rows(x)) )
    error ("grace: non-finite numbers in data. Grace cannot handle them!");
  endif

  pipe_temp = gpipe;
  cgp.rpipe = nth(pipe_temp,4);
  cgp.wfn = nth(pipe_temp,2)+0;
  cgp.rfn = nth(pipe_temp,1)+0;
  cgp.graph = 0;
  cgp.set = 0;

  if (exist("__grace_pipe_data__"))
    cgp.index = length(__grace_pipe_data__)+1;
    __grace_pipe_data__ = append(__grace_pipe_data__,cgp);
  else
    cgp.index = 1;
    __grace_pipe_data__ = list(cgp);
  endif

  fstr = strcat("/usr/local/bin/xmgrace -dpipe ",
                int2str(cgp.rpipe));
  fstr = strcat(fstr," &");
  system(fstr);

  fprintf(cgp.wfn,"g%d on\n",cgp.graph);
  fprintf(cgp.wfn,"g%d.s%d on\n",cgp.graph,cgp.set);
  fstr = strcat("g",int2str(cgp.graph));
  fstr = strcat(fstr,".s");
  fstr = strcat(fstr,int2str(cgp.set));
  fstr = strcat(fstr," point %.14g, %.14g\n");
  fprintf(cgp.wfn,fstr,x');
  fprintf(cgp.wfn,"autoscale;redraw\n");
  fflush(cgp.wfn);
  __current_grace_pipe__ = cgp.index;
  retval = cgp.index;

endfunction
```

A.17 htos.m - Octave script

```
## Usage:   htos(H, [Zo])
##
## This function takes an h-parameter matrix and returns an s-parameter
## matrix. An h-parameter matrix just a column of row vectors
## with the following format:
##
##       freq h11 [ h21 h12 h22 ]
##
## The returned s-parameter matrix is in the same format. Zo of the
## measurement system may also be specified.  It defaults to 50 ohms.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = htos (A,z0)
  retval = 0;
  if (nargin < 1)
    usage ("ztos (H,[Zo])");
  endif
  if (nargin < 2)
    z0 = 50;
  endif

  empty_list_elements_ok = 1;
  s = [];

  if (is_matrix (A) && (columns(A) == 5))

    for idx = 1:rows(A)
       v = A(idx,:);
       h11 = v(2);
```

```
      h21 = v(3);
      h12 = v(4);
      h22 = v(5);

      d = (h11+z0)*(h22+z0)-h12*h21;

      s11 = ((h11-z0)*(h22+z0)-h12*h21)/d;
      s12 = 2*h12*z0/d;
      s21 = -2*h21*z0/d;
      s22 = ((z0+h11)*(z0-h22)+h12*h21)/d;

      s = [s;A(idx,1) s11 s21 s12 s22];

    endfor

    retval = s;

  elseif (is_matrix (A) && (columns(A) == 2))

    s = [];

    for idx = 1:rows(A)
      v = A(idx,:);
      h11 = v(2);

      s11 = (h11-z0)/(h11+z0);

      s = [s;A(idx,1) s11];

    endfor

    retval = s;


  else
    error ("htos: expecting h-parameter matrix");
  endif
endfunction
```

A.18 koolen.m - Octave script

```
## Usage: koolen(dut,open,short,[Zo])
##
## This function applies Koolen et. al.'s  deembedding technique
## to a set of three s-parameter matrices: A dut, an open, and a short.
## Zo of the measurement system (default: 50 ohms) may also be specified.
##
## This routine is based on the work of Koolen et.al, "An improved
## de-embedding technique for on-wafer high-frequency characterization,"
## IEEE BCTM 1991 pp. 188-191  See paper for more information.
##
## The deembedded s-parameter matrix is returned.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = koolen (dut,open,short,z0)

  retval = 0;
  if (nargin < 3)
    usage ("koolen (dut,open,short, [Zo])");
  endif
  if (nargin < 4)
    z0 = 50;
  endif

  empty_list_elements_ok = 1;

  if  (is_matrix (dut) && (columns(dut) == 5) \
    && is_matrix (open) && (columns(open) == 5)  \
    && is_matrix (short) && (columns(short) == 5))
```

```
      yans = [];
      ydut = stoy(dut,z0);
      yshort = stoy(short,z0);
      yopen = stoy(open,z0);
      for x=1:1:length(ydut)
        yo = [yopen(x,2) yopen(x,4); yopen(x,3) yopen(x,5)];
        ys = [yshort(x,2) yshort(x,4); yshort(x,3) yshort(x,5)];
        yd = [ydut(x,2) ydut(x,4); ydut(x,3) ydut(x,5)];
ytrans = inv(inv(yd-yo)-inv(ys-yo));
        yans = [yans;
               ydut(x,1) ytrans(1,1) ytrans(2,1) ytrans(1,2) ytrans(2,2)];
      endfor
      retval = ytos(yans,z0);

  elseif   (is_matrix (dut) && (columns(dut) == 2) \
        &&  is_matrix (open) && (columns(open) == 2) \
        &&  is_matrix (short) && (columns(short) == 2))

      ydut = stoy(dut,z0);
      yshort = stoy(short,z0);
      yopen = stoy(open,z0);
      for x=1:1:length(ydut)
        yopen_act(x) = yopen(x,2);
        zshort_act(x) = 1/(yshort(x,2)-yopen_act(x));
        z_act(x)=(1-zshort_act(x)*
                  (ydut(x,2)-yopen_act(x)))/(ydut(x,2)-yopen_act(x));
        tempval(x) = 1/z_act(x);
      endfor
      yans = [ydut(:,1) tempval];
      retval = ytos(yans,z0);

  else
      error ("dut, open, and short must all be s parameter \
lists with the same number of ports!");
  endif
endfunction
```

A.19 magphase.m - Octave script

```
## Usage: magphase(s-parameter matrix)
##
## This function takes an s-parameter matrix and returns an
## s-parameter matrix in magnitude/phase form.  An s-parameter
## matrix is made up of an arbitrary number of rows of complex
## row vectors with the following format:
##
##       frequency s11 [ s21 s12 s22 ]
##
## The returned list is in the format
##
##       frequency s11m s11p [ s21m s21p s12m s12p s22m s22p ]
##
## where s11m is the magnitude, s11p is the phase, etc.   The phase is
## returned in degrees.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = magphase (A)
  retval = 0;
  PX = 57.2957795130823;
  if (nargin != 1)
    usage ("magphase (s-parameter matrix)");
  endif
  if (is_matrix (A) && (columns(A) == 5))

    v = A(1,:);
    s11m = sqrt(real(v(2))*real(v(2))+imag(v(2))*imag(v(2)));
    s11p = PX*atan2(imag(v(2)),real(v(2)));
    s21m = sqrt(real(v(3))*real(v(3))+imag(v(3))*imag(v(3)));
```

```
    s21p = PX*atan2(imag(v(3)),real(v(3)));
    s12m = sqrt(real(v(4))*real(v(4))+imag(v(4))*imag(v(4)));
    s12p = PX*atan2(imag(v(4)),real(v(4)));
    s22m = sqrt(real(v(5))*real(v(5))+imag(v(5))*imag(v(5)));
    s22p = PX*atan2(imag(v(5)),real(v(5)));

    y = [A(1,1) s11m s11p s21m s21p s12m s12p s22m s22p];

    for idx = 2:rows(A)
      v = A(idx,:);
      s11m = sqrt(real(v(2))*real(v(2))+imag(v(2))*imag(v(2)));
      s11p = PX*atan2(imag(v(2)),real(v(2)));
      s21m = sqrt(real(v(3))*real(v(3))+imag(v(3))*imag(v(3)));
      s21p = PX*atan2(imag(v(3)),real(v(3)));
      s12m = sqrt(real(v(4))*real(v(4))+imag(v(4))*imag(v(4)));
      s12p = PX*atan2(imag(v(4)),real(v(4)));
      s22m = sqrt(real(v(5))*real(v(5))+imag(v(5))*imag(v(5)));
      s22p = PX*atan2(imag(v(5)),real(v(5)));

      y = [y; A(idx,1) s11m s11p s21m s21p s12m s12p s22m s22p];

    endfor

    retval = y;

  elseif (is_matrix (A) && (columns(A) == 2))

    v = A(1,:);
    s11m = sqrt(real(v(2))*real(v(2))+imag(v(2))*imag(v(2)));
    s11p = PX*atan2(imag(v(2)),real(v(2)));

    y = [A(1,1) s11m s11p];

    for idx = 2:rows(A)
      v = A(idx,:);
      s11m = sqrt(real(v(2))*real(v(2))+imag(v(2))*imag(v(2)));
      s11p = PX*atan2(imag(v(2)),real(v(2)));

      y = [y; A(idx,1) s11m s11p];

    endfor

    retval = y;
  else
    error ("magphase: expecting s-parameter matrix");
  endif
endfunction
```

A.20 magrad.m - Octave script

```
## Usage: magrad(s parameter list)
##
## This function takes an s-parameter matrix and returns an s-parameter
## matrix in magnitude/phase form.  An s-parameter matrix is made up of
## an arbitrary number of rows of complex row vectors with the
## following format:
##
##       frequency s11 [ s21 s12 s22 ]
##
## The returned matrix is in the format
##
##       frequency s11m s11p [ s21m s21p s12m s12p s22m s22p ]
##
## where s11m is the magnitude, s11p is the phase, etc.   The phase is
## returned in radians.
##

## Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = magphase (A)
  retval = 0;
  if (nargin != 1)
    usage ("magphase (s parameter list)");
  endif
  if (is_matrix (A) && (columns(A) == 5))

    v = A(1,:);
    s11m = sqrt(real(v(2))*real(v(2))+imag(v(2))*imag(v(2)));
    s11p = atan2(imag(v(2)),real(v(2)));
    s21m = sqrt(real(v(3))*real(v(3))+imag(v(3))*imag(v(3)));
    s21p = atan2(imag(v(3)),real(v(3)));
```

```
    s12m = sqrt(real(v(4))*real(v(4))+imag(v(4))*imag(v(4)));
    s12p = atan2(imag(v(4)),real(v(4)));
    s22m = sqrt(real(v(5))*real(v(5))+imag(v(5))*imag(v(5)));
    s22p = atan2(imag(v(5)),real(v(5)));

    y = [A(1,1) s11m s11p s21m s21p s12m s12p s22m s22p];

    for idx = 2:rows(A)
      v = A(idx,:);
      s11m = sqrt(real(v(2))*real(v(2))+imag(v(2))*imag(v(2)));
      s11p = atan2(imag(v(2)),real(v(2)));
      s21m = sqrt(real(v(3))*real(v(3))+imag(v(3))*imag(v(3)));
      s21p = atan2(imag(v(3)),real(v(3)));
      s12m = sqrt(real(v(4))*real(v(4))+imag(v(4))*imag(v(4)));
      s12p = atan2(imag(v(4)),real(v(4)));
      s22m = sqrt(real(v(5))*real(v(5))+imag(v(5))*imag(v(5)));
      s22p = atan2(imag(v(5)),real(v(5)));

      y = [y; A(idx,1) s11m s11p s21m s21p s12m s12p s22m s22p];

    endfor

    retval = y;

  elseif (is_matrix (A) && (columns(A) == 2))

    v = A(1,:);
    s11m = sqrt(real(v(2))*real(v(2))+imag(v(2))*imag(v(2)));
    s11p = atan2(imag(v(2)),real(v(2)));

    y = [A(1,1) s11m s11p];

    for idx = 2:rows(A)
      v = A(idx,:);
      s11m = sqrt(real(v(2))*real(v(2))+imag(v(2))*imag(v(2)));
      s11p = atan2(imag(v(2)),real(v(2)));

      y = [y; A(idx,1) s11m s11p];

    endfor

    retval = y;
  else
    error ("magphase: expecting s parameter list");
  endif
endfunction
```

A.21 newgraph.m - Octave script

```
## Usage: newgraph(x)
##
## Sends an n x 1 or n x 2 matrix (x) to a new graph in the
## current existing Grace session for plotting.  If number
## of columns is one, it is assumed to be the y data, if the
## number of columns is two, the first column is used for x
## and the second column is used for y.
##
## To create a Grace session, use grace().
## To add a new set the current graph of the current Grace
## session, use newgraph();
## To send a command to the current Grace session use the
## grace_command() function.
##
## To use this function you need to define two global variables
## in your .octaverc file:
##
##    global __grace_pipe_data__;
##    global __current_grace_pipe__;
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
##  This code is freeware.  Everyone is permitted to use, copy,
##  modify, and distribute it for both personal and commercial
##  purposes.
##
##  Because the code is licensed free of charge, there is no
##  warranty for the code, to the extent permitted by applicable
##  law.   Except when otherwise stated in writing, the copyright
##  holder provides the code "as is" without warranty of any kind,
##  either expressed or implied, including but not limited to, the
##  implied warranties of merchantability and fitness for a particular
##  purpose. If the code proves to be defective, the user assumes the
##  cost of all necessary servicing, repair, or correction.

function retval = newgraph (x)
  global __current_grace_pipe__;
  global __grace_pipe_data__;

  retval = 0;
  if (nargin != 1)
    usage ("newgraph (x)");
  endif

  if (columns(x) == 1)
```

```
    x = [(1:rows(x))' x];
  endif

  bad_number_map = finite(x);
  good_number_total = sum(sum(bad_number_map));
  if (good_number_total < (2*rows(x)) )
    error ("grace: non-finite numbers in data. Grace cannot handle them!");
  endif

  cgp = nth(__grace_pipe_data__,__current_grace_pipe__);
  cgp.graph = cgp.graph+1;
  cgp.set = 0;
  __grace_pipe_data__ =
        splice(__grace_pipe_data__,__current_grace_pipe__,1,list(cgp));

  fprintf(cgp.wfn,"g%d on\n",cgp.graph);
  fprintf(cgp.wfn,"g%d.s%d on\n",cgp.graph,cgp.set);
  fstr = strcat("g",int2str(cgp.graph));
  fstr = strcat(fstr,".s");
  fstr = strcat(fstr,int2str(cgp.set));
  fstr = strcat(fstr," point %.14g, %.14g\n");
  fprintf(cgp.wfn,fstr,x');
  fprintf(cgp.wfn,"focus g%d;autoscale\n",cgp.graph);
  numgraphs = cgp.graph+1;
  ngstring = int2str(numgraphs);
  fstr = strcat("arrange(",ngstring);
  fstr = strcat(fstr,",1,0.1,0.1,0.4);redraw\n");
  fprintf(cgp.wfn,fstr);
  fflush(cgp.wfn);

endfunction
```

A.22 newset.m - Octave script

```
## Usage: newset(x)
##
## Sends an n x 1 or n x 2 matrix (x) to the current existing
## Grace session for plotting as a new data set on the current
## graph.  If number of columns is one, it is assumed to be the
## y data, if the number of columns is two, the first column is
## used for x and the second column is used for y.
##
## To create a Grace session, use grace().
## To add a new graph to an existing Grace session, use newgraph();
## To send a command to the current Grace session use the
## grace_command() function.
##
## To use this function you need to define two global variables
## in your .octaverc file:
##
##    global __grace_pipe_data__;
##    global __current_grace_pipe__;
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
##  This code is freeware.  Everyone is permitted to use, copy,
##  modify, and distribute it for both personal and commercial
##  purposes.
##
##  Because the code is licensed free of charge, there is no
##  warranty for the code, to the extent permitted by applicable
##  law.   Except when otherwise stated in writing, the copyright
##  holder provides the code "as is" without warranty of any kind,
##  either expressed or implied, including but not limited to, the
##  implied warranties of merchantability and fitness for a particular
##  purpose. If the code proves to be defective, the user assumes the
##  cost of all necessary servicing, repair, or correction.

function retval = newset (x)
  global __current_grace_pipe__;
  global __grace_pipe_data__;

  retval = 0;
  if (nargin != 1)
    usage ("newset (x)");
  endif

  if (columns(x) == 1)
    x = [(1:rows(x))' x];
```

```
  endif

  bad_number_map = finite(x);
  good_number_total = sum(sum(bad_number_map));
  if (good_number_total < (2*rows(x)) )
    error ("grace: non-finite numbers in data. Grace cannot handle them!");
  endif

  cgp = nth(__grace_pipe_data__,__current_grace_pipe__);
  cgp.set = cgp.set+1;
  __grace_pipe_data__ =
        splice(__grace_pipe_data__,__current_grace_pipe__,1,list(cgp));

  fprintf(cgp.wfn,"g%d on\n",cgp.graph);
  fprintf(cgp.wfn,"g%d.s%d on\n",cgp.graph,cgp.set);
  fstr = strcat("g",int2str(cgp.graph));
  fstr = strcat(fstr,".s");
  fstr = strcat(fstr,int2str(cgp.set));
  fstr = strcat(fstr," point %.14g, %.14g\n");
  fprintf(cgp.wfn,fstr,x');
  fprintf(cgp.wfn,"focus g%d;autoscale;redraw\n",cgp.graph);
  fflush(cgp.wfn);

endfunction
```

A.23 phasenoise.m - Octave script

```
## Usage: phasenoise(A)
#
## This function returns the phase noise L(f) given the right
## sideband noise in A with a carrier frequency of fc.  The
## routine assumes that the data in A starts at zero and
## continues to the maximum offset frequency.  It assumes that
## the data in A is given in dBm with a resolution bandwidth
## of 1Hz.   The returned matrix gives L(f) in dBc/Hz.

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = phasenoise (A)
  retval = 0;
  if (nargin != 1)
    usage ("phasenoise (A)");
  endif

  if (!ismatrix(A))
    error("first argument must be a two-column matrix");
  endif

  if (columns(A) != 2)
     error("first argument must be a two-column matrix");
  endif

  xinc = abs(A(2,1)-A(1,1));
  xmargin = xinc/20.0;
  total_power = 2*10^(A(1,2)/10)*xinc;
  pdata(1) = total_power;

  for i=2:rows(A)
```

```
     current_margin = abs(abs(A(i,1)-A(i-1,1))-xinc);
     if (current_margin > xmargin)
        error("there must be a constant frequency increment \
in the input matrix.");
     endif
     pdata(i) = 2*10^(A(i,2)/10)*xinc;
     total_power = total_power + pdata(i);
  endfor

  pdata = pdata/total_power;
  pout = 10*log10(pdata);
  retval=[A(:,1) pout];

endfunction
```

A.24 pretsl.m - Octave script

```
# Usage: pretsl (thru,line,Co,ldiff)
##
## This function calculates the error network for Kasten's TSL
## deembedding technique using the s-parameters of a thru, a
## line, the difference in length of the lines (in meters),
## and the capacitance per meter (Farads/meter) of the line.
##
## The s-parameters of the port 1 error network are returned.
##
## Example:
##        linetmp = touchstone('line.dat');
##        thrutmp = touchstone('thru.dat');
##        line = symmetrize(linetmp);
##        thru = symmetrize(thrutmp);
##        c0 = 1.621e-12/2500e-6;
##        error_network = pretsl(thru,line,c0,0.0005);
##        raw_dut = touchstone('dut.dat');
##        deembedded_dut = tsl(raw_dut,error_network);

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = pretsl (thru,line,c0,ldiff)

  retval = 0;
  if (nargin < 4)
    usage ("pretsl (thru, line, Co, length_diff)");
  endif
  if (nargin < 5)
    z0 = 50;
  endif
  empty_list_elements_ok = 1;
```

```
  if  (is_matrix (thru) && (columns(thru) == 5) \
      && is_matrix (line) && (columns(line) == 5))

      if (rows(thru) != rows(line))
        error("pretsl: thru and line must have equal number of rows...");
      endif

  garray = spgamma(thru,line,ldiff);
  zc = [];
  sline = [];
  sthru = [];
  for idx = 1:rows(thru)
    newzcelement = garray(idx,2)/(i*2.0*pi*line(idx,1)*c0);
    zc = [zc;newzcelement];
    sline = [sline;stos(line(idx,:),newzcelement,50)];
    sthru = [sthru;stos(thru(idx,:),newzcelement,50)];
  endfor
  en = trl(sthru,sline);
  serror = [];
  for idx = 1:rows(thru)
    serror = [serror;stos(en(idx,:),50,zc(idx))];
  endfor

  retval = serror;

  else
      error ("pretsl:thru and line must be two-port s parameter lists!");
  endif
endfunction
```

A.25 rtos.m - Octave script

```
## Usage: rtos(r parameter list)
##
## This function takes a a wave cascading (r) parameter list and
## returns an s parameter list.
##
## The r parameter list is just a column of vectors
## with the following format:
##
##       freq r11 r21 r12 r22
##
## The returned s parameter list is in the same format.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = rtos (A)
  retval = 0;
  if (nargin < 1)
    usage ("rtos (r parameter list)");
  endif

  if (is_matrix (A) && (columns(A) == 5))

    empty_list_elements_ok = 1;
    s = [];

    for idx = 1:rows(A)
       r11 = A(idx,2);
       r21 = A(idx,3);
       r12 = A(idx,4);
       r22 = A(idx,5);
```

```
      s11 = r12/r22;
      s12 = r11-r21*r12/r22;
      s21 = 1/r22;
      s22 = -r21/r22;

      s = [s;A(idx,1) s11 s21 s12 s22];

    endfor

    retval = s;

  else
    error ("rtos: expecting s parameter list");
  endif
endfunction
```

A.26 skindepth.m - Octave script

```
## Usage: skindepth(conductivity,permeability,frequency)
#
## This function returns skin depth given conductivity, permeability,
## and frequency.  Units are S/m, H/m, and Hz respectively.  The
## return value is in meters.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = skindepth (sigma,mu,f)
  retval = 0;
  if (nargin != 3)
    usage ("skindepth (conductivity(S/m), permeability(H/m), f(Hz))");
  endif

  retval = sqrt(2/(6.28318530717959*mu*sigma*f));

endfunction
```

A.27 smith chart.m - Octave script

```
## Usage: smith_chart (s-param-matrix, [column])
##
## This function creates a Postscript Smith chart given an
## s-parameter matrix.  If the s-parameter matrix represents
## two-port data you can select the column you want to plot
## using the optional column parameter.  This parameter
## defaults to 'S11' but can also be 'S21', 'S12', or 'S22'
## for two-port data.

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = smith_chart (A,coltext)
  retval = 0;

  if (nargin < 1)
    usage ("smith_chart (s-param-matrix, [column])");
  endif

  if (!is_matrix (A))
    error("smith_chart: first parameter must be s-parameter matrix.");
  endif

  if ((columns(A) != 2) && (columns(A) != 5))
    error("smith_chart: s-parameter data must be one-port or two-port");
  endif

  if (nargin < 2)
    scol = 2;
  else
    if (!isstr(coltext))
error("smith_chart: column must be 'S11', 'S21', 'S12', or 'S22'");
```

```
    endif
if ((columns(A) == 2) && !strcmp(coltext,'S11'))
   error("smith_chart: for 1-port data column must be 'S11'");
    endif
if (columns(A) == 2)
   scol = 2;
else
       if (strcmp(coltext,'S11'))
      scol = 2;
   elseif (strcmp(coltext,'S21'))
      scol = 3;
   elseif (strcmp(coltext,'S12'))
      scol = 4;
   elseif (strcmp(coltext,'S22'))
      scol = 5;
       else
 error("smith_chart: column must be 'S11', 'S21', 'S12', or 'S22'");
       endif
endif
  endif

  outfile_number = floor(294967296*rand(1));
  outfile_name = strcat("/tmp/octave_smith_tempfile.",getenv("USERNAME"));
  outfile_name = strcat(outfile_name,".");
  outfile_name = strcat(outfile_name,int2str(outfile_number));

  f = fopen(outfile_name,"w","native");
  for idx = 1:rows(A)
      v = A(idx,:);
      fprintf(f,"%e %e %e\n",real(v(1)),real(v(scol)),imag(v(scol)));
  endfor
  fclose(f);
  command_text = strcat("/usr/local/scripts/ss.pl ", outfile_name);
  system(command_text,"huh","async");
  sleep(1);
  unlink(outfile_name);

endfunction
```

A.28 sparam load.m - Octave script

```
# Usage: sparam_load (filename)
#
# This function loads a file of raw s-parameters of the form
#
#    freq s11r s11i [s21r s21i s12r s12i s22r s22i]
#    (where the r and i stand for real and imaginary parts.)
#
# and converts them into an s parameter list that the rest of the
# s parameter analysis routines can understand.  The function
# accepts data with three columns or nine columns only, representing
# any single port or full two port data.   Lines beginning with
# the exclamation point (!) or the pound sign (#) are ignored.
#
# The function returns a list of complex numbers in the form
#
#    freq Snn [Snn Snn Snn]
#
# where the Snn are complex numbers.
#

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = sparam_load (sfile)
  retval = 0;
  if (nargin != 1)
    usage ("sparam_load (filename)");
  endif

  datawidth = 0;

  sf = fopen(sfile,"r");
```

```
  if (sf != -1)
    nextline = fgetl(sf,1000);
    while (!isnumeric(nextline))
      thisrow = sscanf(nextline,"%e %e %e %e %e %e %e %e",1024)';
      if (datawidth == 0)
         if (columns(thisrow) != 0)
            if ((columns(thisrow) != 3) && (columns(thisrow) != 9))
               fclose(sf);
               error("sparam_load: invalid number of columns in \
file.  Must be 3 or 9.");
            endif
            datawidth = columns(thisrow);
            if (datawidth == 3)
               outrow = [thisrow(1) thisrow(2) + i * thisrow(3)];
            else
               outrow = [thisrow(1) thisrow(2)+i*thisrow(3)\
                                    thisrow(4)+i*thisrow(5)\
                                    thisrow(6)+i*thisrow(7)\
                                    thisrow(8)+i*thisrow(9)];
            endif
            retval = outrow;
         endif
      else
         if (columns(thisrow) != datawidth)
            fclose(sf);
            error("sparam_load: the file must have a consistent \
number of columns!");
         else
            if (datawidth == 3)
               outrow = [thisrow(1) thisrow(2) + i * thisrow(3)];
            else
               outrow = [thisrow(1) thisrow(2)+i*thisrow(3)\
                                    thisrow(4)+i*thisrow(5)\
                                    thisrow(6)+i*thisrow(7)\
                                    thisrow(8)+i*thisrow(9)];
            endif
            retval = [retval; outrow];
         endif
      endif
    nextline = fgetl(sf,1000);
    endwhile
    fclose(sf);
  else
    error ("sparam_load: problem opening file?");
  endif
endfunction
```

A.29 sparam load ma.m - Octave script

```
# Usage: sparam_load_ma (filename)
#
# This function loads a file of raw s-parameters of the form
#
#    freq s11m s11a [s21m s21a s12m s12a s22m s22a]
#    (where the m and a stand for magnitude and angle parts.)
#
#        THE ANGLE IS ASSUMED TO BE IN DEGREES
#
# and converts them into an s parameter list that the rest of the
# s parameter analysis routines can understand.  The function
# accepts data with three columns or nine columns only, representing
# any single port or full two port data.   Lines beginning with
# the exclamation point (!) or the pound sign (#) are ignored.
#
# The function returns a list of complex numbers in the form
#
#    freq Snn [Snn Snn Snn]
#
# where the Snn are complex numbers.

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = sparam_load_ma (sfile)
  retval = 0;
  if (nargin != 1)
    usage ("sparam_load_ma (filename)");
  endif

  datawidth = 0;
```

```
  sf = fopen(sfile,"r");
  if (sf != -1)
    nextline = fgetl(sf,1000);
    while (!isnumeric(nextline))
      thisrow = sscanf(nextline,"%e %e %e %e %e %e %e %e",1024)';
      if (datawidth == 0)
         if (columns(thisrow) != 0)
            if ((columns(thisrow) != 3) && (columns(thisrow) != 9))
               fclose(sf);
               error("sparam_load: invalid number of columns in file.\
 Must be 3 or 9.");
            endif
            datawidth = columns(thisrow);
            if (datawidth == 3)
               outrow = [thisrow(1) thisrow(2)*cos(thisrow(3)*pi/180)\
  + i * thisrow(2)*sin(thisrow(3)*pi/180)];
            else
               outrow = [thisrow(1) thisrow(2)*cos(thisrow(3)*pi/180)\
  + i * thisrow(2)*sin(thisrow(3)*pi/180)\
                                     thisrow(4)*cos(thisrow(5)*pi/180)\
  + i * thisrow(4)*sin(thisrow(5)*pi/180)\
                                     thisrow(6)*cos(thisrow(7)*pi/180)\
  + i * thisrow(6)*sin(thisrow(7)*pi/180)\
                                     thisrow(8)*cos(thisrow(9)*pi/180)\
  + i * thisrow(8)*sin(thisrow(9)*pi/180)];
            endif
            retval = outrow;
         endif
      else
         if (columns(thisrow) != datawidth)
            fclose(sf);
            error("sparam_load: the file must have a consistent number \
of columns!");
         else
            if (datawidth == 3)
               outrow = [thisrow(1) thisrow(2)*cos(thisrow(3)*pi/180)\
  + i * thisrow(2)*sin(thisrow(3)*pi/180)];
            else
               outrow = [thisrow(1) thisrow(2)*cos(thisrow(3)*pi/180)\
   + i * thisrow(2)*sin(thisrow(3)*pi/180)\
                           thisrow(4)*cos(thisrow(5)*pi/180)\
   + i * thisrow(4)*sin(thisrow(5)*pi/180)\
                           thisrow(6)*cos(thisrow(7)*pi/180)\
   + i * thisrow(6)*sin(thisrow(7)*pi/180)\
                           thisrow(8)*cos(thisrow(9)*pi/180)\
   + i * thisrow(8)*sin(thisrow(9)*pi/180)];
            endif
            retval = [retval; outrow];
```

```
        endif
      endif
    nextline = fgetl(sf,1000);
    endwhile
    fclose(sf);
  else
    error ("sparam_load_ma: problem opening file?");
  endif
endfunction
```

A.30 sparam load mr.m - Octave script

```
# Usage: sparam_load_mr (filename)
#
# This function loads a file of raw s-parameters of the form
#
#    freq s11m s11a [s21m s21a s12m s12a s22m s22a]
#    (where the m and a stand for magnitude and angle parts.)
#
#         THE ANGLE IS ASSUMED TO BE IN RADIANS
#
# and converts them into an s parameter list that the rest of the
# s parameter analysis routines can understand.  The function
# accepts data with three columns or nine columns only, representing
# any single port or full two port data.   Lines beginning with
# the exclamation point (!) or the pound sign (#) are ignored.
#
# The function returns a list of complex numbers in the form
#
#    freq Snn [Snn Snn Snn]
#
# where the Snn are complex numbers.

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = sparam_load_mr (sfile)
  retval = 0;
  if (nargin != 1)
    usage ("sparam_load_mr (filename)");
  endif

  datawidth = 0;
```

```
 sf = fopen(sfile,"r");
 if (sf != -1)
   nextline = fgetl(sf,1000);
   while (!isnumeric(nextline))
     thisrow = sscanf(nextline,"%e %e %e %e %e %e %e %e",1024)';
     if (datawidth == 0)
        if (columns(thisrow) != 0)
           if ((columns(thisrow) != 3) && (columns(thisrow) != 9))
              fclose(sf);
              error("sparam_load: invalid number of columns in file.\
Must be 3 or 9.");
           endif
           datawidth = columns(thisrow);
           if (datawidth == 3)
              outrow = [thisrow(1) thisrow(2)*cos(thisrow(3))\
   + i * thisrow(2)*sin(thisrow(3))];
           else
              outrow = [thisrow(1) thisrow(2)*cos(thisrow(3))\
   + i * thisrow(2)*sin(thisrow(3))\
                        thisrow(4)*cos(thisrow(5))\
   + i * thisrow(4)*sin(thisrow(5))\
                        thisrow(6)*cos(thisrow(7))\
   + i * thisrow(6)*sin(thisrow(7))\
                        thisrow(8)*cos(thisrow(9))\
   + i * thisrow(8)*sin(thisrow(9))];
           endif
           retval = outrow;
        endif
     else
        if (columns(thisrow) != datawidth)
           fclose(sf);
           error("sparam_load: the file must have a consistent number \
of columns!");
        else
           if (datawidth == 3)
              outrow = [thisrow(1) thisrow(2)*cos(thisrow(3))\
   + i * thisrow(2)*sin(thisrow(3))];
           else
              outrow = [thisrow(1) thisrow(2)*cos(thisrow(3))\
    + i * thisrow(2)*sin(thisrow(3))\
                        thisrow(4)*cos(thisrow(5))\
    + i * thisrow(4)*sin(thisrow(5))\
                        thisrow(6)*cos(thisrow(7))\
    + i * thisrow(6)*sin(thisrow(7))\
                        thisrow(8)*cos(thisrow(9))\
    + i * thisrow(8)*sin(thisrow(9))];
           endif
           retval = [retval; outrow];
```

```
        endif
      endif
    nextline = fgetl(sf,1000);
    endwhile
    fclose(sf);
  else
    error ("sparam_load_mr: problem opening file?");
  endif
endfunction
```

A.31 spgamma.m - Octave script

```
## Usage:   spgamma(thru,line,delta l,[Zo])
##
## Uses through and line measurements combined with difference in line
## length (in meters) and returns the propagation constant gamma of
## the line as a function of frequency.  The input S parameter list
## is a column of s parameters vs. frequency in the following format:
##
##       frequency s11 s21 s12 s22
##
## The function also requires the characteristic impedance of the measurement
## system, Zo.  If it is not specified, a value of 50 ohms is assumed.
##
## The returned list is a column of vectors in the following format:
##
##       frequency  gamma
##
## where all numbers are complex
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = spgamma (T,L,ldiff,z0)
  retval = 0;
  piover2 = pi/2.0;
  twobetac = 2.0*pi/ldiff;
  adflag = 0;
  adjust = 0;
  betam1 = 100.0;
  betam2 = 100.0;
  betam3 = 100.0;
  betam4 = 100.0;
```

```
  betam5 = 100.0;
  if (nargin < 3)
    usage ("spgamma (thru,line,delta length (m),[Zo])");
  endif

  if (nargin < 4)
    z0=50;
  endif

  if (ldiff == 0)
    error("spgamma: Sorry, delta length cannot be zero...");
  endif

  if ((is_matrix (T) && (columns(T) == 5)) &&
     (is_matrix (L) && (columns(L) == 5)) )
     T1M = stot(T,z0);
     T2M = stot(L,z0);

 empty_list_elements_ok = 1;
     g = [];

     for idx = 1:rows(T1M)

        A = (T1M(idx,2)*T2M(idx,5)+T2M(idx,2)*T1M(idx,5))-\
    (T1M(idx,3)*T2M(idx,4)+T1M(idx,4)*T2M(idx,3));
        argA = atan2(imag(A),real(A));
        radical = sqrt(A*A-4.0);

        if ((argA > 0) && (argA < piover2))
          egammal = (A+radical)/2.0;
        elseif ((argA > piover2) && (argA < pi))
          egammal = (A-radical)/2.0;
        elseif ((argA > -pi) && (argA < -piover2))
          egammal = (A-radical)/2.0;
else
          egammal = (A+radical)/2.0;
        endif
        gammal = log(egammal);
        gamster = gammal/ldiff;
beta = imag(gamster);
if ((beta < 0) && \
                    (adflag == 0) && \
                    (abs(beta) > twobetac*9.0/20.0))
           adjust = twobetac;
   adflag = 1;
elseif ((betam5 < 0) && (betam4 < 0) &&\
        (betam3 < 0) && (betam2 < 0) &&\
(betam1 < 0) && (beta > 0))
```

```
   adflag = 0;
endif
betam5 = betam4;
betam4 = betam3;
betam3 = betam2;
betam2 = betam1;
betam1 = beta;
        g_ans = real(gamster)+i*abs(beta+adjust);
        g = [g; T1M(idx,1) g_ans];

     endfor

     retval = g;

  else
    error ("spgamma: Usage:spgamma(thru,line,delta length(m), [Zo])");
  endif
endfunction
```

A.32 splot.pl - perl script

```
#!/usr/local/bin/perl5
# This script is a utility script that makes it easy to strip
# waveform data out of HSPICE output files so it can be used
# in Octave, Matlab, etc.

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

$flag = 0;
$infile = shift;
open (DF,$infile) or die "Can't open input file";
while ($_ = <DF>) {
   if (/transient analysis/) {$flag = 1;}
   if (/time/ && /voltage/ && ($flag == 1)) {
      $_ = <DF>;
      $vname = $_;
      $vname =~ s/\s+//g;
      printf STDERR "found $vname\n";
      open(CF,">$infile"."."."$vname".".dat");
      while ($_ = <DF>) {
        if (/^y/) {
            close(CF);
            last;
          }
          else {
          ($tpoint,$vpoint) = split;
          printf CF "%e %e\n",$tpoint,$vpoint;
          }
        }
     }
  }
  close(DF);
```

```
__END__
```

A.33 spplot.m - Octave script

```
## Usage: spplot (S,[c],[f])
##
## This function facilitates plotting data in s-parameter matrices.
## The input parameters are an s-parameter matrix (S), an optional
## column (c), and an optional format (f).  For 1-port data the
## column parameter is ignored if present.  The allowed values
## for the column parameter are 'S11', 'S21', 'S12', and 'S22'.
## The default value is 'S11'.  The allowed values for the format
## parameter are 'dB', 'mag', 'phase', 'rads', 'real', 'imag'.
## The default value is 'dB'.
##
## Examples:
##
##   grace(spplot(A))              // plots S11 in dB
##   grace(spplot(A,'phase'))      // plots S11 phase
##   grace(spplot(A,'S21','real')) // plots real part of S21
##
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = spplot (A,coltext,fmttext)
  retval = 0;

  coldata = "S11S21S12S22";
  fmtdata = "dBmagphaseradsrealimag";

  if ((nargin < 1) || (nargin > 3))
  usage ("spplot (S,[c],[f])");
  endif
```

```
  if (!is_matrix (A))
    error("spplot: first parameter must be s-parameter matrix.");
  endif

  if ((columns(A) != 2) && (columns(A) != 5))
    error("spplot: s-parameter data must be one-port or two-port");
  endif

  if (nargin == 1)
    scol = 2;
fmttext = 'dB';
  elseif (nargin == 2)
    if (!isstr(coltext))
       error("spplot: second parameter must be column or format.\
 See help!");
endif
    colcheck = index(coldata,coltext);
if (colcheck != 0)
      scol = ceil(colcheck/3)+1;
      fmttext = 'dB';
    else
   fmtcheck = index(fmtdata,coltext);
   if (fmtcheck == 0)
      error("spplot: second parameter must be \
column or format. See help!");
   else
   fmttext = coltext;
           scol = 2;
       endif
    endif
  else
    if (!isstr(coltext))
       error("spplot: second parameter must be \
column or format.  See help!");
endif
    colcheck = index(coldata,coltext);
if (colcheck)
      scol = ceil(colcheck/3)+1;
    else
      error("spplot: column must be 'S11', 'S21', 'S12', or 'S22'");
    endif
fmtcheck = index(fmtdata,fmttext);
if (fmtcheck == 0)
error("spplot: format must be 'dB', 'mag', 'phase', \
 'rads', 'real' or 'imag'");
    endif
  endif
```

```
  PX = 57.2957795130823;

  if (strcmp(fmttext,'dB'))
  retval = [real(A(:,1)) 20*log10(abs(A(:,scol)))];
  elseif (strcmp(fmttext,'mag'))
  retval = [real(A(:,1)) abs(A(:,scol))];
  elseif (strcmp(fmttext,'phase'))
  retval = [real(A(:,1)) PX*atan2(imag(A(:,scol)),real(A(:,scol)))];
  elseif (strcmp(fmttext,'rads'))
  retval = [real(A(:,1)) atan2(imag(A(:,scol)),real(A(:,scol)))];
  elseif (strcmp(fmttext,'real'))
  retval = [real(A(:,1)) real(A(:,scol))];
  elseif (strcmp(fmttext,'imag'))
  retval = [real(A(:,1)) imag(A(:,scol))];
  else
  error("spplot: Boy something went wrong!");
  endif

endfunction
```

A.34 sreverse.m - Octave script

```
## Usage: sreverse(two-port s parameter list)
##
## This function takes a two-port s parameter list and "reverses"
## it as follows:
##
##        s11 = s22
##        s21 = s21
##        s12 = s12
##        s22 = s11
##
## This function is used for tsl deembedding

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = sreverse (A)
  retval = 0;
  if (nargin < 1)
    usage ("sreverse (two-port s param list)");
  endif
  if (is_matrix (A) && (columns(A) == 5))

    empty_list_elements_ok = 1;

    rev = [];

    for idx = 1:rows(A)
       rev = [rev;A(idx,1) A(idx,5) A(idx,3) A(idx,4) A(idx,2)];
    endfor

    retval = rev;
```

```
  else
    error ("sreverse: expecting two-port S parameter list");
  endif
endfunction
```

A.35 ss.pl - perl script

```
#!/usr/bin/perl

## FILE: ss.pl perl script for generating Postscript Smith charts
##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

\fname = shift;

\timestamp = time;
(\sec,\min,\hour,\mday,\mon,\year,\wday,\yday,\isdst) = \
   localtime(\timestamp);

\tmpfile = "/tmp/octave_smith_\timestamp";
\x_location = 0.375;
\y_location = 0.5;
\total_size = 1.0;

\chart_radius = 2000*\total_size;
\origin_x = 7000*\x_location;
\origin_y = 5500*\y_location;

open SMC,">\tmpfile";

print SMC "\%!PS-Adobe-2.0
\%\%Title: \fname
\%\%Creator: Octave/smith_chart()
\%\%CreationDate: \wday \mon \mday \hour:\min:\sec \year
\%\%DocumentFonts: (atend)
\%\%BoundingBox: 50 50 554 554
\%\%Orientation: Landscape
\%\%Pages: (atend)
\%\%EndComments
```

```
/smithdict 256 dict def
smithdict begin
/Color false def
/Solid false def
/smithlinewidth 5.000 def
/userlinewidth smithlinewidth def
end
\%\%EndProlog
\%\%Page: 1 1
smithdict begin
gsave
50 50 translate
0.100 0.100 scale
90 rotate
0 -5040 translate
0 setgray
newpath
1.000 setlinewidth
(Helvetica) findfont 140 scalefont setfont\n";

printf SMC "%s %s %s 0 360 arc\n",
  \origin_x+chart_radius/2.0,\origin_y,\chart_radius+1;
print SMC "gsave\n";
print SMC "clip\n";
printf SMC "%s %s %s 0 360 arc\n",
  \origin_x+chart_radius/2.0,\origin_y,\chart_radius;
print SMC "stroke\n";

print SMC "1.000 setlinewidth\n";

print SMC "newpath\n";
printf SMC "%s %s moveto\n",\origin_x-\chart_radius,\origin_y;
printf SMC "%s 0 rlineto\n",\origin_x+\chart_radius;
print SMC "stroke\n";


foreach \r (0.2, 0.4, 0.6, 0.8, 1, 2, 5) {
  print SMC "newpath\n";
  printf SMC "%s %s %s 0 360 arc\n",
   \origin_x+\chart_radius*\r/(1+\r),\origin_y,\chart_radius/(1+\r);
  print SMC "stroke\n";
}

print SMC "(Helvetica) findfont 50 scalefont setfont\n";
print SMC "newpath\n";
foreach \r (0.2, 0.4, 0.6, 0.8, 1, 2, 5) {
  printf SMC "%s %s moveto\n",
   \origin_x+\chart_radius*\r/(1+\r)-\chart_radius/(1+\r)-25, \origin_y+25;
```

```
  print SMC "gsave\n";
  print SMC "90 rotate\n";
  printf SMC "(%d) show\n",50*\r;
  print SMC "grestore\n";
}
print SMC "stroke\n";


print SMC "newpath\n";
foreach \reac (0.25, 0.5, 1, 2, 5) {
  \x = \origin_x+\chart_radius;
  \y = \origin_y+\chart_radius/\reac;
  \rad = \chart_radius/\reac;
  printf SMC "%s %s %s 0 360 arc\n",\x,\y,\rad;
}
print SMC "stroke\n";

print SMC "newpath\n";
foreach \x (0.25, 0.5, 1, 2, 5) {
  printf SMC "%s %s %s 0 360 arc\n",
  \origin_x+\chart_radius,\origin_y-\chart_radius/\x,\chart_radius*(1/\x);
}
print SMC "stroke\n";
print SMC "grestore\n";

print SMC "newpath\n";
print SMC "(Helvetica) findfont 50 scalefont setfont\n";
foreach \reac (0.25, 0.5, 1, 2, 5) {
  \x = \origin_x+\chart_radius;
  \y = \origin_y+\chart_radius/\reac;
  \rad = \chart_radius/\reac;
  \a = (1+1/(\reac*\reac));
  \b = -2;
  \c = 1-(1/(\reac*\reac));
  \d = (\b*\b-4*\a*\c);
  \x = (-\b - sqrt(\b*\b-4.0*\a*\c))/(2.0*\a);
  \y = sqrt(1-\x*\x);
  \px = \origin_x+\chart_radius*\x;
  \py = \origin_y+\chart_radius*\y;
  \py2 = \origin_y-\chart_radius*\y;
  \phi = 57.3*atan2(\y,\x)+270.0;
  \phi2 = 57.3*atan2(-\y,\x)+270.0;
  printf SMC "%s %s moveto\n",\px,\py;
  print SMC "gsave\n";
  printf SMC "%d rotate\n",\phi;
  printf SMC "(+j%d) show\n",50*\reac;
  print SMC "grestore\n";
  printf SMC "%s %s moveto\n",\px,\py2;
```

```
  print SMC "gsave\n";
  printf SMC "%d rotate\n",\phi2;
  printf SMC "(-j%d) show\n",50*\reac;
  print SMC "grestore\n";
}
print SMC "stroke\n";
print SMC "3.000 setlinewidth\n";

open DF, "<\fname";

\_ = <DF>;
\count = 1;
(\f,\rd,\id) = split;
\rho = sqrt(\rd*\rd+\id*\id);
\theta = atan2(\id,\rd);
\xoffset = \origin_x + \chart_radius*\rho*cos(\theta);
\yoffset = \origin_y + \chart_radius*\rho*sin(\theta);
print SMC "newpath\n";
print SMC "\xoffset \yoffset moveto\n";


until (!\_) {

   (\f,\rd,\id) = split;
   \rho = sqrt(\rd*\rd+\id*\id);
   \theta = atan2(\id,\rd);
   \xoffset = \origin_x + \chart_radius*\rho*cos(\theta);
   \yoffset = \origin_y + \chart_radius*\rho*sin(\theta);
   print SMC "\xoffset \yoffset lineto\n";
   \_ = <DF>;
   \count = \count+1;
}

close DF;
print SMC "stroke\n";

print SMC "grestore
end
showpage
\%\%Trailer
\%\%DocumentFonts: Helvetica
\%\%Pages: 1";

close SMC;
system("gv -geometry 700x700+250+250 \tmpfile");
unlink(\tmpfile);

__END__
```

A.36 stoh.m - Octave script

```
## Usage: stoh(S, [Zo])
##
## This function takes an s-parameter matrix (S) and
## returns an h-parameter matrix. The s-parameter
## matrix is just a column of row vectors with the
## following format:
##
##       freq s11 [ s21 s12 s22 ]
##
## The returned h-parameter matix is in the same format.
## Zo of the measurement system may also be specified.
## It defaults to 50 ohms.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = stoh (A,z0)
  retval = 0;
  if (nargin < 1)
    usage ("stoh (two-port s parameter list,z0)");
  endif
  if (nargin < 2)
    z0 = 50;
  endif

  empty_list_elements_ok = 1;

  if (is_matrix (A) && (columns(A) == 5))

    z = [];
```

```
    for idx = 1:rows(A)
       v = A(idx,:);
       s11 = v(2);
       s21 = v(3);
       s12 = v(4);
       s22 = v(5);

       d = (1-s11)*(1+s22)+s12*s21;

       h11 = z0*((1+s11)*(1+s22)-s12*s21)/d;
       h12 = z0*(s12+s12)/d;
       h21 = -z0*(s21+s21)/d;
       h22 = z0*((1-s22)*(1-s11)-s12*s21)/d;

       z = [z;A(idx,1) h11 h21 h12 h22];

    endfor

    retval = z;

  elseif (is_matrix (A) && (columns(A) == 2))

    z = [];
    for idx = 1:rows(A)
      z  = [z; A(idx,1) z0 * (1 + A(idx,2))/(1 - A(idx,2))];
    endfor
    retval = z;

  else
    error ("stoh: expecting s parameter list");
  endif
endfunction
```

A.37 stor.m - Octave script

```
## Usage: stor(s parameter list)
##
## This function takes an s parameter list and returns a wave
## cascading (r) parameter list.
##
## The s parameter list is just a column of vectors
## with the following format:
##
##       freq s11 [ s21 s12 s22 ]
##
## The returned y parameter list is in the same format.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = stor (A)
  retval = 0;
  if (nargin < 1)
    usage ("stor (s parameter list)");
  endif

  if (is_matrix (A) && (columns(A) == 5))

    empty_list_elements_ok = 1;
    r = [];

    for idx = 1:rows(A)
       s11 = A(idx,2);
       s21 = A(idx,3);
       s12 = A(idx,4);
       s22 = A(idx,5);
```

```
      r22 = 1/s21;
      r21 = -s22/s21;
      r12 = s11/s21;
      r11 = s12-s11*s22/s21;

      r = [r;A(idx,1) r11 r21 r12 r22];

    endfor

    retval = r;

  else
    error ("stor: expecting s parameter list");
  endif
endfunction
```

A.38 stos.m - Octave script

```
## Usage: stos(s parameter list, Zo, Zref)
##
## This function converts an s parameter list in a system with
## characteristic impedance Zref to an s parameter list in a system
## with characteristic impedance Zo.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = stos (A,z0,zref)
  retval = 0;
  if (nargin < 1)
    usage ("stos (two-port s parameter list,Zo,Zref)");
  endif
  empty_list_elements_ok = 1;

  if (is_matrix (A) && (columns(A) == 5))

    s = [];

    for idx = 1:rows(A)
       v = A(idx,:);
       s11 = v(2);
       s21 = v(3);
       s12 = v(4);
       s22 = v(5);

       d = (1.0-s11)*(1.0-s22)-s12*s21;

       z11 = ((1.0+s11)*(1.0-s22)+s12*s21)/d;
       z21 = 2.0*s21/d;
```

```
        z12 = 2.0*s12/d;
        z22 = ((1.0-s11)*(1.0+s22)+s12*s21)/d;

    z11 = z11*zref/z0;
    z21 = z21*zref/z0;
    z12 = z12*zref/z0;
    z22 = z22*zref/z0;

    d = (1.0+z11)*(1.0+z22)-z12*z21;

    s11 = ((z11-1.0)*(z22+1.0)-z12*z21)/d;
    s21 = 2.0*z21/d;
    s12 = 2.0*z12/d;
    s22 = ((z11+1.0)*(z22-1.0)-z12*z21)/d;

        s = [s;A(idx,1) s11 s21 s12 s22];

     endfor

     retval = s;

  elseif (is_matrix (A) && (columns(A) == 2))

     s = [];
     for idx = 1:rows(A)

        v = A(idx,:);
        s11 = v(2);
        d = (1.0-s11)*(1.0-s22)-s12*s21;

        z11 = ((1.0+s11)*(1.0-s22)+s12*s21)/d;

    z11 = z11*zref/z0;

    d = (1.0+z11)*(1.0+z22)-z12*z21;

    s11 = ((z11-1)*(z22+1)-z12*z21)/d;

        s = [s;A(idx,1) s11];

     endfor
     retval = s;

  else
     error ("stos: expecting s parameter list");
  endif
endfunction
```

A.39 stot.m - Octave script

```
## Usage: stot(two-port s parameter list, [Zo])
##
## This function takes a two-port s parameter list and returns an ABCD
## transmission parameter list.
##
## The two-port s parameter list is just a column of vectors with the
## following format:
##
##       freq s11 s21 s12 s22
##
## The returned two-port ABCD parameter list is in the format:
##
##       freq  A   B   C   D
##
## Zo of the measurement system may also be specified.  It defaults
## to 50 ohms.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = stot (A,z0)
  retval = 0;
  if (nargin < 1)
    usage ("stot (two-port s param list,[Zo])");
  endif
  if (nargin < 2)
    z0 = 50;
  endif
  if (is_matrix (A) && (columns(A) == 5))

    y0 = 1/z0;
```

```
    v = A(1,:);
    s11 = v(2);
    s21 = v(3);
    s12 = v(4);
    s22 = v(5);

    d = s21+s21;

    t11 = ((1+s11)*(1-s22)+s12*s21)/d;
    t12 = z0*((1+s11)*(1+s22)-s12*s21)/d;
    t21 = ((1-s11)*(1-s22)-s12*s21)/(z0*d);
    t22 = ((1-s11)*(1+s22)+s12*s21)/d;

    t = [A(1,1) t11 t21 t12 t22];

    for idx = 2:rows(A)
       v = A(idx,:);
       s11 = v(2);
       s21 = v(3);
       s12 = v(4);
       s22 = v(5);

       d = s21+s21;

       t11 = ((1+s11)*(1-s22)+s12*s21)/d;
       t12 = z0*((1+s11)*(1+s22)-s12*s21)/d;
       t21 = ((1-s11)*(1-s22)-s12*s21)/(z0*d);
       t22 = ((1-s11)*(1+s22)+s12*s21)/d;

       t = [t;A(idx,1) t11 t21 t12 t22];

    endfor

    retval = t;

  else
    error ("stot: expecting two-port S parameter list");
  endif
endfunction
```

A.40 stoy.m - Octave script

```
## Usage: stoy(s parameter list,[Zo])
##
## This function takes an s parameter list and returns a y parameter list.
## The s parameter list is just a column of vectors
## with the following format:
##
##       freq s11 [ s21 s12 s22 ]
##
## The returned y parameter list is in the same format. Zo of the
## measurement system may be specified.  It defaults to 50 ohms.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = stoy (A,z0)
  retval = 0;
  if (nargin < 1)
    usage ("stoy (s parameter list,[Zo])");
  endif
  if (nargin < 2)
    z0 = 50;
  endif
  if (is_matrix (A) && (columns(A) == 5))

    y0 = 1/z0;

    v = A(1,:);
    s11 = v(2);
    s21 = v(3);
    s12 = v(4);
    s22 = v(5);
```

```
  d = (1+s11)*(1+s22)-s12*s21;

  y11 = y0*((1-s11)*(1+s22)+s12*s21)/d;
  y12 = y0*(-2*s12)/d;
  y21 = y0*(-2*s21)/d;
  y22 = y0*((1+s11)*(1-s22)+s12*s21)/d;

  y = [A(1,1) y11 y21 y12 y22];

  for idx = 2:rows(A)
     v = A(idx,:);
     s11 = v(2);
     s21 = v(3);
     s12 = v(4);
     s22 = v(5);

     d = (1+s11)*(1+s22)-s12*s21;

     y11 = y0*((1-s11)*(1+s22)+s12*s21)/d;
     y12 = y0*(-2*s12)/d;
     y21 = y0*(-2*s21)/d;
     y22 = y0*((1+s11)*(1-s22)+s12*s21)/d;

     y = [y;A(idx,1) y11 y21 y12 y22];

  endfor

  retval = y;

elseif (is_matrix (A) && (columns(A) == 2))

  y0 = 1/z0;

  v = A(1,:);
  s11 = v(2);

  y11 = -y0*(s11-1)/(s11+1);

  y = [A(1,1) y11];

  for idx = 2:rows(A)
     v = A(idx,:);
     s11 = v(2);

     y11 = -y0*(s11-1)/(s11+1);

     y = [y;A(idx,1) y11];
```

```
    endfor

    retval = y;

  else
    error ("stoy: expecting s parameter list");
  endif
endfunction
```

A.41 stoz.m - Octave script

```
## Usage: stoz(s parameter list, [Zo])
##
## This function takes an s parameter list and returns a z parameter list.
## The s parameter list is just a column of vectors with the
## following format:
##
##       freq s11 [ s21 s12 s22 ]
##
## The returned Z parameter list is in the same format. Zo of the
## measurement system may also be specified.  It defaults to 50 ohms.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = stoz (A,z0)
  retval = 0;
  if (nargin < 1)
    usage ("stoz (two-port s parameter list,z0)");
  endif
  if (nargin < 2)
    z0 = 50;
  endif

  empty_list_elements_ok = 1;

  if (is_matrix (A) && (columns(A) == 5))

    z = [];

    for idx = 1:rows(A)
       v = A(idx,:);
```

```
      s11 = v(2);
      s21 = v(3);
      s12 = v(4);
      s22 = v(5);

      d = (1-s11)*(1-s22)-s12*s21;

      z11 = z0*((1+s11)*(1-s22)+s12*s21)/d;
      z12 = z0*(s12+s12)/d;
      z21 = z0*(s21+s21)/d;
      z22 = z0*((1-s11)*(1+s22)+s12*s21)/d;

      z = [z;A(idx,1) z11 z21 z12 z22];

    endfor

    retval = z;

  elseif (is_matrix (A) && (columns(A) == 2))

    z = [];
    for idx = 1:rows(A)
      z  = [z; A(idx,1) z0 * (1 + A(idx,2))/(1 - A(idx,2))];
    endfor
    retval = z;

  else
    error ("stoz: expecting s parameter list");
  endif
endfunction
```

A.42 symmetrize.m - Octave script

```
## Usage: symmetrize (two-port s parameter list)
##
## This function takes a two-port s parameter list and returns a
## "symmetrized" two-port s parameter list with S11 = S22 and
## S12 = S21 by averaging.  The two-port s parameter list is just
## a column of vectors with the following format:
##
##       freq s11 s21 s12 s22
##
## The returned two-port s parameter list is in the same format.

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = symmetrize (A)
  retval = 0;
  if (nargin != 1)
    usage ("symmetrize (two-port s parameter list)");
  endif
  if (is_matrix (A) && (columns(A) == 5))

    v = A(1,:);
    s11 = (v(2)+v(5))/2;
    s21 = (v(3)+v(4))/2;
    s12 = s21;
    s22 = s11;

    y = [A(1,1) s11 s12 s21 s22];

    for idx = 2:rows(A)
       v = A(idx,:);
       s11 = (v(2)+v(5))/2;
```

```
      s21 = (v(3)+v(4))/2;
      s12 = s21;
      s22 = s11;

      y = [y; A(idx,1) s11 s21 s12 s22];
    endfor

    retval = y;

  else
    error ("symmetrize: expecting two-port s parameter list");
  endif
endfunction
```

A.43 touchstone.m - Octave script

```
# Usage: touchstone (filename)
#
# This function loads a touchstone file of s-parameters
# and converts it into an s-parameter matrix that the rest of the
# s-parameter analysis routines can understand.
#
# The function returns a complex matrix. Each row in the
# returned matrix is a complex row vector:
#
#    freq S11 [S21 S12 S22]
#
# The function does not handle touchstone files with included
# noise parameters.

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = touchstone (sfile)
  empty_list_elements_ok = 1;
  retval = [];
  PX = 1.74532925199433e-02;
  if (nargin != 1)
    usage ("touchstone (filename)");
  endif

  sf = fopen(sfile,"r");
  if (sf != -1)
    nextline = fgetl(sf,1000);
    while (!isnumeric(nextline))
      thisrow = sscanf(nextline,"%e %e %e %e %e %e %e %e",1024)';
  if (columns(thisrow) == 0)
        if (nextline(1) == "#")
```

```
   templine = "";
   last_char = "";
           for cidx = 1:length(nextline)
      if (last_char == " ")
    if (nextline(cidx) == " ")
else
      templine = strcat(templine,nextline(cidx));
  last_char = nextline(cidx);
endif
  else
    templine = strcat(templine,nextline(cidx));
last_char = nextline(cidx);
  endif
   endfor
         tempspaces = findstr(templine," ");
 if (columns(tempspaces) != 5)
            error("The specification line is malformed");
 endif

         touch_units = templine(tempspaces(1)+1:tempspaces(2)-1);
         if (strcmp(touch_units,"Hz"))
   fmult = 1.0;
         elseif (strcmp(touch_units,"GHz"))
   fmult = 1.0e9;
         elseif (strcmp(touch_units,"MHz"))
           fmult = 1.0e6;
         elseif (strcmp(touch_units,"KHz"))
   fmult = 1.0e3;
 else
           fclose(sf);
           error("touchstone: Units must be one of GHz, MHz, KHz, or Hz...");
 endif

         touch_type = templine(tempspaces(3)+1:tempspaces(4)-1);
         touch_R = templine(tempspaces(5)+1:length(templine));
 Zo = str2num(touch_R);
     endif

  else
     if (!exist("touch_type"))
            error("First uncommented line of Touchstone file \
must be specification line.");
 endif

 if (!exist("datawidth"))
            datawidth = columns(thisrow);
         endif
```

```
         if ((columns(thisrow) != datawidth) ||
     ((columns(thisrow) != 3) && (columns(thisrow) != 9)))
             fclose(sf);
 if (columns(thisrow) != datawidth)
               error("touchstone: all rows must have the \
same number of columns");
 else
               error("touchstone: file must have 3 or 9 data columns; \
see help");
 endif
         else
            if (datawidth == 3)
   if (touch_type == "RI")
                  outrow = [fmult*thisrow(1) thisrow(2) + i * thisrow(3)];
               elseif (touch_type == "MA")
                  outrow = [fmult*thisrow(1) thisrow(2)*cos(thisrow(3)*PX)\
       + i * thisrow(2)*sin(thisrow(3)*PX)];
               elseif (touch_type == "DB")
      mag2 = 10^(thisrow(2)/20.0);
                  outrow = [fmult*thisrow(1) mag2*cos(thisrow(3)*PX)\
       + i * mag2*sin(thisrow(3)*PX)];
               else
                 fclose(sf);
                 error("touchstone: Format must be one of MA, DB, RI");
   endif
           else
   if (touch_type == "RI")
                  outrow = [fmult*thisrow(1) thisrow(2)+i*thisrow(3)\
                                    thisrow(4)+i*thisrow(5)\
                                    thisrow(6)+i*thisrow(7)\
                                    thisrow(8)+i*thisrow(9)];
               elseif (touch_type == "MA")
                  outrow = [fmult*thisrow(1) thisrow(2)*cos(thisrow(3)*PX)\
       + i * thisrow(2)*sin(thisrow(3)*PX)\
                           thisrow(4)*cos(thisrow(5)*PX)\
       + i * thisrow(4)*sin(thisrow(5)*PX)\
                           thisrow(6)*cos(thisrow(7)*PX)\
       + i * thisrow(6)*sin(thisrow(7)*PX)\
                           thisrow(8)*cos(thisrow(9)*PX)\
       + i * thisrow(8)*sin(thisrow(9)*PX)];
               elseif (touch_type == "DB")
      mag2 = 10^(thisrow(2)/20.0);
      mag4 = 10^(thisrow(4)/20.0);
      mag6 = 10^(thisrow(6)/20.0);
      mag8 = 10^(thisrow(8)/20.0);
                  outrow = [fmult*thisrow(1) mag2*cos(thisrow(3)*PX)\
                      + i * mag2*sin(thisrow(3)*PX)\
                                  mag4*cos(thisrow(5)*PX)\
```

```
                    + i * mag4*sin(thisrow(5)*PX)\
                                mag6*cos(thisrow(7)*PX)\
                    + i * mag6*sin(thisrow(7)*PX)\
                                mag8*cos(thisrow(9)*PX)\
                    + i * mag8*sin(thisrow(9)*PX)];
              else
                fclose(sf);
                error("touchstone: Format must be one of MA, DB, RI");
   endif
          endif
          retval = [retval; outrow];
        endif
      endif
    nextline = fgetl(sf,1000);
    endwhile
    fclose(sf);
  else
    error ("touchstone: problem opening file?");
  endif
endfunction
```

A.44 trl.m - Octave script

```
## Usage: trl (thru,line,[Zo])
##
## This function applies Engen and Hoer's TRL deembedding technique
## to a set of two s-parameter matrices (a line and a thru).
## Zo of the measurement system may be optionally specified.
##
## The error network s-parameter matrix is returned.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = trl (thru,line)

  retval = 0;
  if (nargin < 2)
    usage ("trl (thru,line)");
  endif

  if  (  is_matrix (thru) && (columns(thru) == 5)  \
      && is_matrix (line) && (columns(line) == 5))

      if (rows(thru) != rows(line))
         error("trl: thru and line must have equal number of rows...");
      endif

      rthru = stor(thru);
      rline = stor(line);

      empty_list_elements_ok = 1;
      rerror = [];
```

```
      for idx=1:rows(thru)

         gammasc = thru(idx,2) - thru(idx,3);
         rt = [rthru(idx,2) rthru(idx,4); rthru(idx,3) rthru(idx,5)];
         rl = [rline(idx,2) rline(idx,4); rline(idx,3) rline(idx,5)];

         t = rl * inv(rt);
         tmpa = t(2,1);
         tmpb = t(2,2)-t(1,1);
         tmpc = -t(1,2);

         a1=(-tmpb+sqrt(tmpb**2-4*tmpa*tmpc))/(2*tmpa);
         a2=(-tmpb-sqrt(tmpb**2-4*tmpa*tmpc))/(2*tmpa);

         if (abs(a2) > abs(a1))
            ac = a2;
            b = a1;
         else
            ac = a1;
            b = a2;
         endif

         w1 = gammasc;
         a = (w1-b)/(-1.0*(1.0-w1/ac));
         c = a/ac;

         r22 =sqrt(-thru(idx,2)/(thru(idx,3)*(a*c-b)));

         rerror = [rerror ;thru(idx,1) a*r22 c*r22 b*r22 r22];

      endfor

      retval = rtos(rerror);

  else
      error ("trl:thru and line must be two-port s parameter matrices!");
  endif
endfunction
```

A.45 tsl.m - Octave script

```
## Usage: tsl (dut,serror)
##
## This function applies Kasten's TSL deembedding technique
## to a set s-parameters given the s-parameters of the error
## network as calculated by pretsl.
##
## The deembedded s-parameter matrix is returned.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = tsl (dut,serror)

  retval = 0;
  if (nargin < 2)
    usage ("tsl (dut,serror)");
  endif
  empty_list_elements_ok = 1;

  if  (is_matrix (dut) && (columns(dut) == 5) \
      && is_matrix (serror) && (columns(serror) == 5))

      if (rows(dut) != rows(serror))
        error("tsl: dut, line, and thru must all \
have equal number of rows...");
      endif

      srerror = sreverse(serror);
      terror = stot(serror);
      trerror = stot(srerror);
      tdut = stot(dut);
```

```
      empty_list_elements_ok = 1;
      tdmblist = [];

      for idx=1:rows(tdut)

         A   = [terror(idx,2)  terror(idx,4);\
                terror(idx,3)  terror(idx,5)];
         Ar  = [trerror(idx,2) trerror(idx,4);\
                 trerror(idx,3) trerror(idx,5)];
 dutparams = [tdut(idx,2) tdut(idx,4);\
                              tdut(idx,3) tdut(idx,5)];
         Ainv = inv(A);
         Arinv = inv(Ar);

         tmp1 = Ainv * dutparams;
         tdmb = tmp1 * Arinv;

         tdmblist = [tdmblist;\
                     tdut(idx,1) tdmb(1,1) tdmb(2,1) tdmb(1,2) tdmb(2,2)];

      endfor

      retval = ttos(tdmblist);

  else
      error ("tsl:dut and serror must be two-port s parameter lists!");
  endif
endfunction
```

A.46 ttos.m - Octave script

```
## Usage:  ttos(two-port ABCD parameter list, [Zo])
##
## This function takes a two-port ABCD parameter matrix and
## returns a two-port s-parameter matrix. The two-port ABCD
## parameter matrix is just a column of row vectors with the
## following format:
##
##       freq t11 t21 t12 t22
##
## The returned s-parameter matrix is in the same format.
## Zo of the measurement system may also be specified.  It
## defaults to 50 ohms.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = ttos (A,z0)
  retval = 0;
  if (nargin < 1)
    usage ("ttos (two-port ABCD param list, [Zo])");
  endif
  if (nargin < 2)
    z0 = 50;
  endif

  if (is_matrix (A) && (columns(A) == 5))

    v = A(1,:);
    t11 = v(2);
    t21 = v(3);
    t12 = v(4);
```

```
    t22 = v(5);

    td = t11+t12/z0+t21*z0+t22;

    s11 = (t11+t12/z0-t21*z0-t22)/td;
    s12 = 2*(t11*t22-t12*t21)/td;
    s21 = 2/td;
    s22 = (t12/z0-t11-t21*z0+t22)/td;

    s = [A(1,1) s11 s21 s12 s22];

    for idx = 2:rows(A)
       v = A(idx,:);
       t11 = v(2);
       t21 = v(3);
       t12 = v(4);
       t22 = v(5);

       td = t11+t12/z0+t21*z0+t22;

       s11 = (t11+t12/z0-t21*z0-t22)/td;
       s12 = 2*(t11*t22-t12*t21)/td;
       s21 = 2/td;
       s22 = (t12/z0-t11-t21*z0+t22)/td;

       s = [s;A(idx,1) s11 s21 s12 s22];

    endfor

    retval = s;

  else
    error ("ttos: expecting two-port ABCD parameter list");
  endif
endfunction
```

A.47 twoport.m - Octave script

```
## Usage: twoport(S)
##
## This function takes a one-port s-parameter matrix and returns a two port
## s-parameter matrix that can be used with the tsl function.  Both S11 and
## S22 of the returned matrix are set to the same value, that of the input
## matrix.  The S21 and S12 of the returned matrix are set to 1e-8.  This
## allows deembedding data from one-port measurements  using tsl.

## Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = twoport (A)
  retval = 0;
  if (nargin < 1)
    usage ("twoport (S)");
  endif

  empty_list_elements_ok = 1;

  if (is_matrix (A) && (columns(A) == 2))

    z = [];
    for idx = 1:rows(A)
      z  = [z; A(idx,1) A(idx,2) 1e-8  1e-8  A(idx,2)];
    endfor
    retval = z;

  else
    error ("twoport: expecting one-port s-parameter matrix");
  endif
endfunction
```

```
## Usage: V2dBm(V,[Zo])
#
## This function converts Volts RMS to dBm.  If Zo is
## not specified it is assumed to be 50 ohms.
##

function retval = V2dBm (V,z0)
  retval = 0;
  if ((nargin != 2) && (nargin != 1))
    usage ("V2dBm (V,[Zo])");
  endif

  if (nargin == 1)
    z0 = 50;
  endif

  retval = 10*log10(V*V/(0.001*z0));

endfunction
```

A.48 vprop.m - Octave script

```
## Usage: vprop(relative dielectric constant)
#
## This function returns velocity of propagation in meters/sec for a
## cable with a given relative dielectric constant.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = vprop (er)
  retval = 0;
  if ((nargin < 1))
    usage ("vprop (relative dielectric constant)");
  endif

  retval = 2.998e8/sqrt(er);

endfunction
```

A.49 W2dBm.m - Octave script

```
## Usage: W2dBm(W)
#
## This function converts Watts to dBm.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.


function retval = W2dBm (W,z0)
  retval = 0;
  if (nargin != 1)
    usage ("W2dBm (W)");
  endif

  if ((W == 0.0) || (W < 0.0))
    error("Power needs to be positive!");
  endif

  retval = 10*log10(W/0.001);

endfunction
```

A.50 wavelength.m - Octave script

```
## Usage: wavelength(f (Hz) ,[rel. dielectric constant])
#
## This function returns wavelength in meters for a given frequency
## and optional relative dielectric constant.    If the relative dielectric
## constant is not given it is assumed to be 1.
##

##  Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.


function retval = wavelength (f,er)
  retval = 0;
  if ((nargin < 1))
    usage ("wavelength (f,[rel. dielectric constant])");
  endif

  if (nargin < 2)
    er = 1;
  endif

  retval = 2.998e8/(f*sqrt(er));

endfunction
```

A.51 yanzhen.m - Octave script

```
## Usage: yanzhen (SA, SB, SC, SDUT, ea, eb)
##
## This function applies Yan-zhen's deembedding technique
## to a set of four one-port s-parameter matrices, SA,SB,
## SC, and SDUT.  SA and SB are measurements of known
## standards such as a short or open with known permittivities
## ea and eb.  The third deembedding standard SC is assumed
## to be acetone, and the formula
##
##           e = einf + (e0-einf)/(1+jm(tau))
##
## where einf = 1.9, e0 = 21.2 and tau = 3.3e-12 s, is used
## for its permittivity.
##
## e' and e'' vs. frequency for SDUT is returned  as a matrix
## made up of row vectors in following format:
##
##                     freq e' e''
##

## Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = yanzhen (SA,SB,SC,SDUT,ea,eb)

  retval = 0;
  if (nargin < 6)
    usage ("yan-shen (SA,SB,SC,SDUT,ea,eb)");
  endif

  einf = 1.9;
  eacetone0 = 21.2;
```

```
  tau = 3.3e-12;

  if  (  is_matrix (SA) && (columns(SA) == 2)  \
      && is_matrix (SB) && (columns(SB) == 2)  \
      && is_matrix (SC) && (columns(SC) == 2)  \
      && is_matrix (SDUT) && (columns(SDUT) == 2))

      ZA = stoz(SA);
      ZB = stoz(SB);
      ZC = stoz(SC);
  ZDUT = stoz(SDUT);

      empty_list_elements_ok = 1;
      epsilon = [];

      for idx=1:rows(SA)

     ec = einf+(eacetone0-einf)/(1+i*2*pi*SA(idx,1)*tau);
         comat = [1 1/ea ZA(idx,2);
          1 1/eb ZB(idx,2);
  1 1/ec ZC(idx,2)];
         comat_inv = inv(comat);
         x = [ZA(idx,2)/ea; ZB(idx,2)/eb; ZC(idx,2)/ec];
   delta = comat_inv * x;
           etmp = (ZDUT(idx,2)-delta(2))/(delta(1)+ZDUT(idx,2)*delta(3));
   e_prime = real(etmp);
   e_prime_prime = -imag(etmp);
         epsilon = [epsilon ; SA(idx,1)  e_prime e_prime_prime];

      endfor

      retval = epsilon;

  else
      error ("yanzhen: SA-SDUT must be one-port s-parameter matrices");
  endif
endfunction
```

A.52 ytos.m - Octave script

```
## Usage: ytos(Y, [Zo])
##
## This function takes a y-parameter matrix Y and returns an s-parameter
## matrix. A y-parameter matrix is just a column of row vectors
## with the following format:
##
##                   freq y11 [ y21 y12 y22 ]
##
## the returned s-parameter matrix is in the same format. Zo of the
## measurement system may also be specified.  It defaults to 50 ohms.
##

## Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = ytos (A,z0)
  retval = 0;
  if (nargin < 1)
    usage ("ytos (Y, [Zo])");
  endif
  if (nargin < 2)
    z0 = 50;
  endif
  if (is_matrix (A) && (columns(A) == 5))

    y0 = 1/z0;

    v = A(1,:);
    y11 = v(2);
    y21 = v(3);
    y12 = v(4);
    y22 = v(5);
```

```
  yd = (y11+y0)*(y22+y0) - y12*y21;

  s11 = ((y0-y11)*(y0+y22)+y12*y21)/yd;
  s12 = (-2*y12*y0)/yd;
  s21 = (-2*y21*y0)/yd;
  s22 = ((y0+y11)*(y0-y22)+y12*y21)/yd;

  s = [A(1,1) s11 s21 s12 s22];

  for idx = 2:rows(A)
     v = A(idx,:);
     y11 = v(2);
     y21 = v(3);
     y12 = v(4);
     y22 = v(5);

     yd = (y11+y0)*(y22+y0) - y12*y21;

     s11 = ((y0-y11)*(y0+y22)+y12*y21)/yd;
     s12 = (-2*y12*y0)/yd;
     s21 = (-2*y21*y0)/yd;
     s22 = ((y0+y11)*(y0-y22)+y12*y21)/yd;

     s = [s;A(idx,1) s11 s21 s12 s22];

  endfor

  retval = s;

elseif (is_matrix (A) && (columns(A) == 2))

  v = A(1,:);
  y11 = v(2);

  s11 = (1-y11*z0)/(1+y11*z0);

  s = [A(1,1) s11];

  for idx = 2:rows(A)
     v = A(idx,:);
     y11 = v(2);

     s11 = (1-y11*z0)/(1+y11*z0);

     s = [s;A(idx,1) s11];

  endfor
```

```
    retval = s;

  else
    error ("ytos: expecting y-parameter matrix");
  endif
endfunction
```

A.53 ztos.m - Octave script

```
## Usage:   ztos(Z, [Zo])
##
## This function takes a z-parameter matrix and returns an s-parameter
## matrix. A z-parameter matrix just a column of row vectors
## with the following format:
##
##       freq z11 [ z21 z12 z22 ]
##
## The returned s-parameter matrix is in the same format. Zo of the
## measurement system may also be specified.  It defaults to 50 ohms.
##

## Copyright (C) Steve Lipa <slipa@eos.ncsu.edu>  2002
##
## This script is free software; you can redistribute it and/or
## modify it under the terms of the GNU Library General Public
## License as published by the Free Software Foundation; either
## version 2 of the License, or (at your option) any later version.
##
## This script is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## Library General Public License for more details.
##
## You should have received a copy of the GNU Library General Public
## License along with this library; if not, write to the
## Free Software Foundation, Inc., 59 Temple Place - Suite 330,
## Boston, MA 02111-1307, USA.

function retval = ztos (A,z0)
  retval = 0;
  if (nargin < 1)
    usage ("ztos (Z,[Zo])");
  endif
  if (nargin < 2)
    z0 = 50;
  endif

  empty_list_elements_ok = 1;
  s = [];

  if (is_matrix (A) && (columns(A) == 5))

    for idx = 1:rows(A)
       v = A(idx,:);
       z11 = v(2);
```

```
      z21 = v(3);
      z12 = v(4);
      z22 = v(5);

      d = (z11+z0)*(z22+z0)-z12*z21;

      s11 = ((z11-z0)*(z22+z0)-z12*z21)/d;
      s12 = 2*z12*z0/d;
      s21 = 2*z21*z0/d;
      s22 = ((z11+z0)*(z22-z0)-z12*z21)/d;

      s = [s;A(idx,1) s11 s21 s12 s22];

    endfor

    retval = s;

  elseif (is_matrix (A) && (columns(A) == 2))

    s = [];

    for idx = 1:rows(A)
      v = A(idx,:);
      z11 = v(2);

      s11 = (z11-z0)/(z11+z0);

      s = [s;A(idx,1) s11];

    endfor

    retval = s;


  else
    error ("ztos: expecting two-port z-parameter matrix");
  endif
endfunction
```

A.54 cables.m - Octave script

```
# Cable and Waveguide data ##################
#
#                                  Attenuation per 100 ft.
#                                   vs frequency in GHz
#         Zo          ------------------------------------------------
# Cable  ohms   Er    0.5     1.0    3.0    5.0    10.0    12.0   18.0
#
# RG-6    75   2.3    4.4     6.3
# RG-8A   52   2.3    4.6     9.0   19      28     47
# RG-9A   51   2.3    4.6     9.0   19      28     47
# RG-11   75   2.3    2.7     3.9
# RG-58A  52   2.3   11      20     41
# RG-59A  75   2.3    6.7    11.5   25.5    41
# RG-62A  93          5.2     8.5   18.4    29.5
# RG-174  50   2.3   17.5    31     64.3    97    185
# RG-196  50   2.1   27.5    45     78     115    172
# RG-214  50   2.3    4.6     9     19      28     47
# RG-223  50   2.3    8.8    16.6   36      51     90
# 0.085sr 50   2.07  11.6    18.7   34      46     73.2    84    115
# 0.141sr 50   2.07   7.2    11.6   21.5    28.5   44.5    51     68
# 0.250sr 50   2.07   4.3     7.3   14      18.9   29      33
#
# NOTE: for attenuation/meter, divide by 30.48
#
#
# Rectangular Waveguides
#
#IEEE    Sugg.         fco      w        h     Military
#BAND   f range       (GHz)    (cm)     (cm)   band names
#
#L     1.12-1.70       0.908  16.51    8.225   L:1-2 GHz
#R     1.70-2.60       1.372  10.922   5.461
#S     2.60-3.95       2.078  7.214    3.404   S:2-4
#H(G)  3.95-5.85       3.152  4.755    2.215   C:4-8
#C(J)  5.85-8.20       4.301  3.485    1.580
#W     7.05-10.0       5.259  2.850    1.262
#X     8.20-12.4       6.557  2.286    1.016   X:8-12
#Ku(P) 12.4-18.0       9.486  1.580    0.790   K:12-18/27-40
#K     18.0-26.5      14.047  1.070    0.430
#Ka(R) 26.5-40.0      21.081  0.711    0.356   Ka:27-40
#Q     33.0-50.5      26.342  0.570    0.280
#U     40.0-60.0      31.357  0.480    0.240
#V     50.0-75.0      39.863  0.380    0.190   V:40-75
#E     60.0-90.0      48.350  0.310    0.150
#W     75.0-110.0     59.010  0.254    0.127   W:75-110
#F     90.0-140.0     73.840  0.203    0.102
```

```
#D    110.0-170.0     90.854  0.170    0.083   mm:110-300
#G    140.0-220.0    115.750  0.130    0.0648  u mm:300-3000
#
# NOTE: to find fco for a dielectric other than air
#       use cutoff(m,n,w,h,Er) for the m,n mode
#
```