

Modeling and analysis of telecom frontends: basic concepts

You insist that there is something a machine cannot do. If you tell me precisely what it is that a machine cannot do, then I can always make a machine which will do just that.
—John von Neumann

Good models and efficient methods for constructing and evaluating them are of utmost importance in making true top-down design of telecom frontends a reality. This book intends to contribute to the work in this area. Firstly, however, we must clearly define “models”, “good models”, “modeling” and “analysis”. You cannot realize something, e.g. a good model, if you cannot tell what it is you want to realize. Therefore, section 2.1 spends some time to elaborate these concepts.

Incorporating all relevant prior knowledge and experience is one of the main properties that characterize good (telecom building block) models. It makes models reliable in a sense that their behavior can be trusted to correspond with that of physical implementations. This is of great importance in avoiding redesigns. Furthermore, exploiting prior knowledge helps to improve, for example, simulation efficiency. It allows us to use “shortcuts” in capturing a system’s behavior. As this book mainly deals with telecom frontends and their building blocks, section 2.2 reviews some typical telecom frontend architectures and summarizes our prior knowledge of their behavior. Further chapters will often exploit this knowledge to simplify analysis.

In communications, the relevance of the different aspects of a building block’s behavior—and therefore the need to incorporate them into (good) models—is measured by their impact on the system’s overall performance in transmitting information. Physically, information is stored by modulating the properties of the waveforms that are transmitted, e.g. their amplitude and phase. Information is lost due to distortion of a waveform’s shape as it travels from sender to receiver. These shape distortions are, for example, due to linear, nonlinear and stochastic building block behavior. Predicting them is of great importance in estimating the probability of transmission errors to occur (bit error rate). Hence our need for accurate building block models.

2.1 Models, modeling and analysis

A major part of this text is about models, modeling and analysis. Before focusing on particular applications, it is useful to spend some time to clarify these concepts.

In what follows, “models”, “good models”, “modeling” and “analysis” are given a precise meaning. This provides the context for the methods presented in this text. We also stress the importance of having good models available to make top-down design truly possible.

2.1.1 Models: what you want or what you have

Modeling and analysis is all about manipulating models. So, the first topic to be addressed sounds: what is a model? Almost any introductory textbook on physics provides a definition. A fairly thorough and consistent treatment on the matter can be found in [Hest87]. There, the author defines:

Definition (Model–Physics): *A model is a conceptual representation of a real object.*

The means used for representing an object can be quite general: it may involve a verbal description, a set of mathematical equations, a collection of measurement data or combinations of the aforementioned. Note that for engineering purposes, the word “object” is often too general and abstract. It is often better to use “system” or “circuit” instead.

From an engineering point of view, the definition above does not satisfy. It highlights the fact that physics is mainly concerned with description and analysis of systems (objects) that exist. Engineers, on the other hand, are also concerned with the creation of new systems. Most of the time, they describe systems that they would like to have but that are not there yet. In [Hest87], such descriptions would be called *fictitious models*. Since this viewpoint puts unequal emphasis on objects that already exist, this text prefers a different, more balanced definition:

Definition (Model–Engineering): *A model is a conceptual representation of a system that you want to realize (specification model) or that you have realized (implementation model).*

This definition puts equal weight on both reasons for using models in engineering practice: the specification of a system’s behavior involves constructing a model for the system as we would like to have it; the verification of a particular implementation involves the construction of a model for a physically realized system as it is given to us. The first kind of models occur as we go down in the design hierarchy while the second kind occurs when going up.

Example (Low-noise amplifiers models): Consider the example in Fig. 2.1 where two people construct models for a low-noise amplifier (LNA). The system designer is responsible for the realization of a complete frontend architecture. He or she will construct a specification model that describes the LNA behavior as (s)he wants it. The circuit designer, on the other hand, realizes an LNA in terms of transistors, coils, capacitors and resistors. The resulting netlist represents an implementation model that describes the LNA as it is or will be physically available. Of course, the goal is to realize an LNA that meets the specifications. Phrased in the terminology introduced above: it must be possible to map the implementation model realized by the circuit designer on the specification model provided by the system designer. ▲

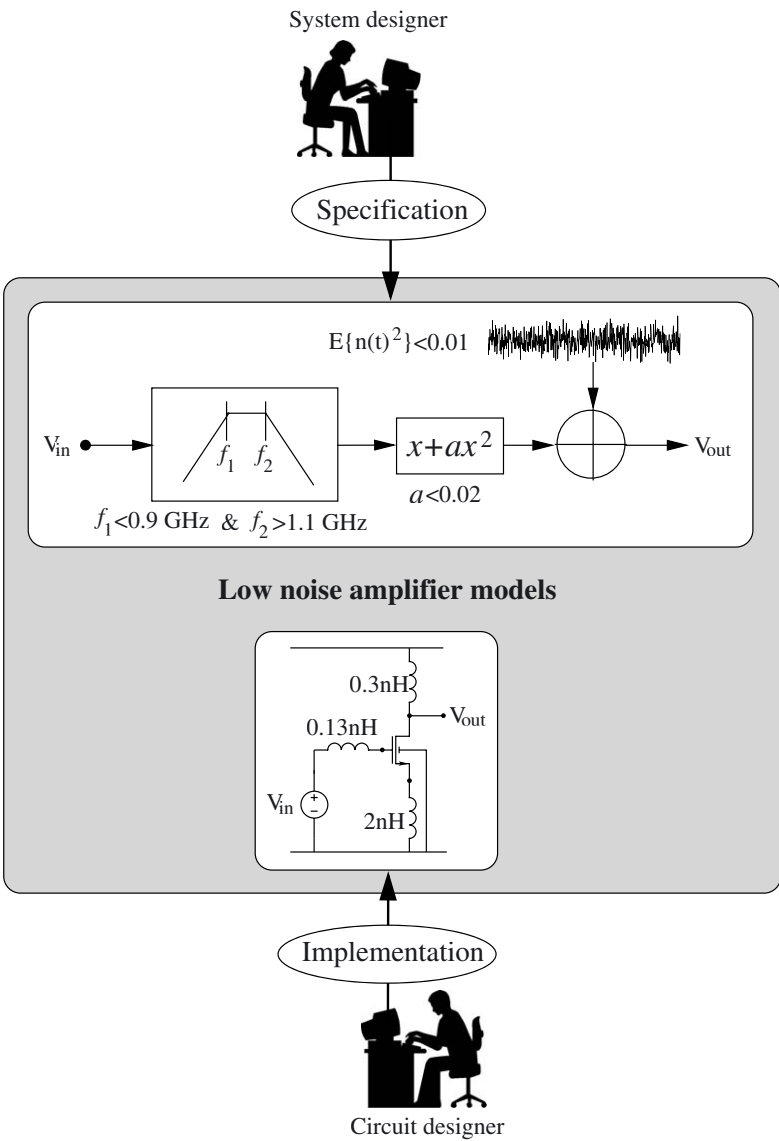


Figure 2.1: A system designer constructs LNA models as (s)he would like the LNA to behave. Through this model, the desired LNA behavior is specified, i.e. it represents the LNA as we would want it to be. A circuit designer realizes an LNA in terms of transistors, coils, etc. The resulting netlist represents a model for the LNA that is physically available, i.e. it represents the LNA as we have it.

2.1.2 Good models

In the example in Fig. 2.1, one may wonder why the system designer makes the LNA model so complicated. Basically, what (s)he really wants is represented by the first block: a simple bandpass filtering operation with some gain. No system designer wants nonlinear distortion or noise. So, why mention it in the specification model? The reason for this is that there is no use in living in utopia. If a system designer makes decisions based on building block models that do not correspond to reality, these decisions often result in implementations that fail to meet the overall performance requirements. A model is said to correspond to reality if it captures all (relevant) behavior exhibited by an actual implementation. Stated differently, it is possible to map the entire behavior of a physical implementation to the (specification) model. If there is no such correspondence, system performance might get ruined by (unwanted) behavior that was not accounted for. This results in costly redesigns.

Clearly, there is a catch in the previous discussion. How is it possible to construct (specification) models that account for the entire behavior of implementations *that may not yet exist*? Often, we don't even have a clue on how the implementation will look like. The answer, of course, is that capturing the *entire* behavior is impossible. The only thing we can do is bundling all our knowledge, gained from similar design experiences in the past. This should help us to suggest models that approximate reality as closely as possible. It is impossible to account for behavior that nobody is expecting at the time when a model is constructed. However, it would be a waste if system design fails because all prior knowledge was not exploited in constructing proper models. This brings us to the concept of a *good model*.

Definition (Good model): *A good model is a model that incorporates all relevant information and experience that we have on the system/circuit being modeled.*

It is clear that constructing good models requires retrieving information from previous (design) experience. Therefore, knowledge and experience should not be gathered and stored in an ad hoc manner. In order to avoid looking for a needle in a haystack of experience, knowledge should be structured and compacted. As outlined in chapter 1, this promotes a hierarchical classification of systems according to their characteristics and properties.

One way to ensure the “goodness” of —especially system-level— models is to show that, at least in theory, they can be derived from existing circuit-level implementations by means of a number of approximations. This links the model to reality while reducing model complexity. Moreover, it is well controlled which behavior is taken into account and which behavior is neglected. Transfer functions, for example, derive their status as a powerful concept to model and specify circuit behavior from the fact that they adequately describe the behavior of a circuit's small-signal approximation. This small-signal approximation linearizes circuit behavior in the neighborhood of a constant (DC) operating point. However, it neglects all nonlinearities. Hence, as illustrated in Fig. 2.2, transfer-function-based (specification) models link to (possible) circuit implementations through a number of well-controlled approximations. A likewise approach towards constructing good models is pursued in chapters 3 and 4.

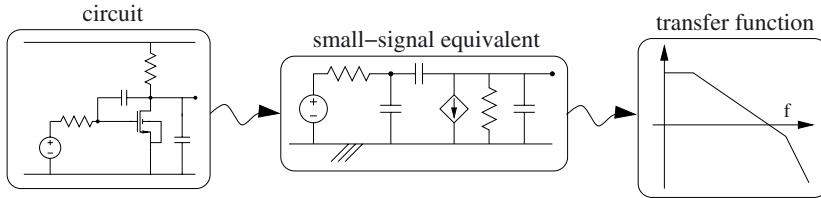


Figure 2.2: Transfer functions derive their status as a powerful concept to model and specify circuit behavior from the fact that they adequately describe the behavior of a circuit's small-signal approximation.

As a final note, it should be stressed that all good models should be as compact as possible. Overly detailed descriptions bear tedious computations and lengthy simulations with them. They do not contribute much to design efficiency. Irrelevant model behavior, for instance, involves non-dominant nonidealities¹ or high-frequency transients. Removing irrelevant details from a model is very important, e.g. for efficient optimization-based design.

2.1.3 The importance of good models in top-down design

As discussed in chapter 1, minimizing the time necessary to design analog frontends requires the introduction of hierarchy in the design process². As illustrated in Fig. 2.3, a hierarchical, top-down method initially tackles frontend design in terms of low-noise amplifiers, filters, mixers, oscillators, A/D converters, etc. In turn, these blocks are implemented using integrators, operational amplifiers, etc. In a next step, circuit- and transistor-level details are filled out. Beyond the circuit level, there is layout and manufacturing.

For CAD support of such a hierarchical flow, good models are of utmost importance. They are among the cornerstones in realizing each of the steps in the design flow hierarchy. The models are used as interfaces to communicate design decisions between the different levels in the design tree. Many actions in a top-down design flow result in specification and/or implementation models that are passed down/up the design hierarchy. As illustrated in Fig. 2.1, system engineers, responsible for the overall frontend design, communicate their desires by means of specification models for the behavior of the frontend building blocks. Creating these models involves synthesis, specification translation and system exploration. Circuit engineers create a transistor netlist. This netlist acts both as a specification model for layout and as an implementation model for building block verification at the architectural level. Here, in a verification step, one checks whether the implementation model can be mapped on the specification model that was initially passed down.

¹A nonideality is said to be non-dominant if there is no frequency band of interest in which the nonideality contributes the major part of the unwanted signal energy.

²For a more complete introduction to hierarchical design methods, see [Donn98].

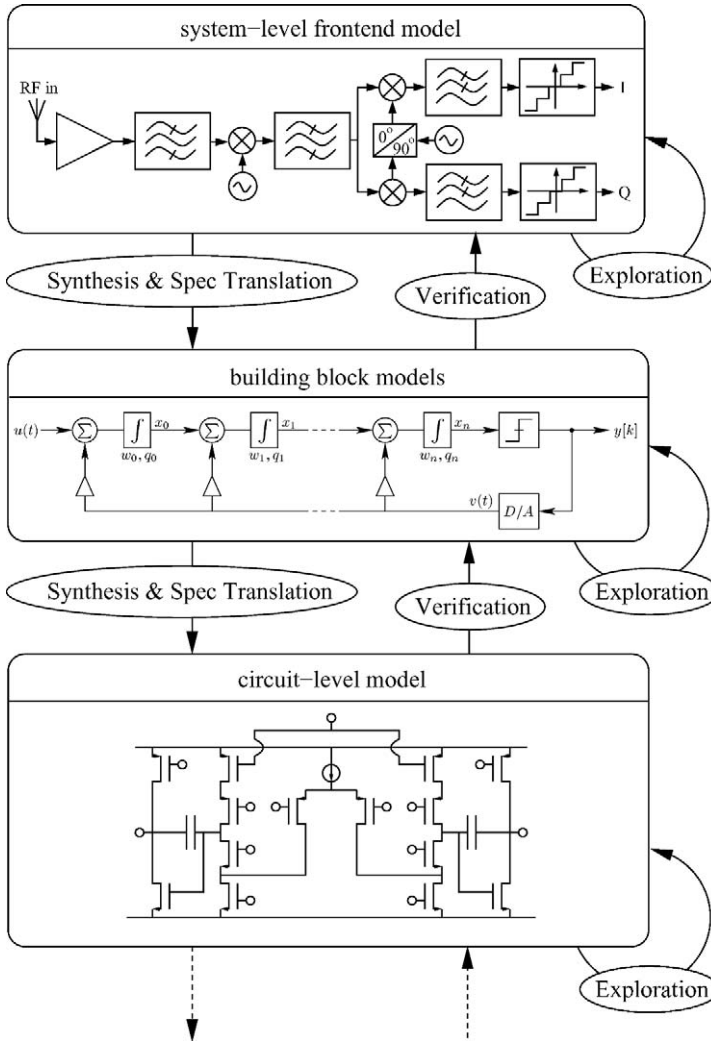


Figure 2.3: Minimizing design time requires the introduction of hierarchy in the design process. Frontend design is done in terms of LNAs, filters, mixers, A/D converters, etc. These building blocks (e.g. a $\Delta\Sigma$ A/D converter) are in turn implemented using integrators, operational amplifiers, etc. In a next step, transistor-level details are filled out. To support this hierarchical design process, good models are of utmost importance. They serve as interfaces used to communicate design decisions between the different levels in the design tree.

All models used in a top-down design flow need to be “good models”: they must accurately represent real-life system behavior. For example, a system-level mixer model should capture all of the relevant behavior that characterizes a mixer’s transistor-level implementation. Otherwise, wrong design decisions will be taken at the architectural level, them being based on analysis using a defective or incomplete mixer model. This often results in costly redesigns. As mentioned before, since we do not know all implementation details in advance, we must rely on prior knowledge and experience to construct realistic model templates. This especially holds for models constructed during the early stages of a design.

Finally, in realizing efficient (partially automated) hierarchical design methods, it is very important that models should be kept as simple as possible. Complex and overly detailed models compromise efficiency of analysis as they result in tedious computations and lengthy simulations. This in turn hampers architecture exploration or optimization-based synthesis and/or specification translation. Since these steps often require numerous evaluations of the building block models, complex models render it slow and sometimes even infeasible. This especially holds at the architectural level where a great number of building block models need to be evaluated simultaneously. Low-complexity models are therefore among the cornerstones of (semi-)automated design. In summary, capturing complex building block behavior in a manner that is as simple as possible is one of the great challenges in CAD.

2.1.4 Modeling languages

Describing system behavior requires a proper formalism, i.e. a language, to do so. Choice of the language is driven by the nature of the system we want to describe. Human behavior, for one, is best described in a spoken language like English. On the other hand, the quantitative nature of engineering problems renders mathematics a natural choice. Often, new language constructs (syntaxes) need to be developed to capture newly encountered objects and their behavior. For example, electrical netlists have driven the creation of the SPICE input syntax [Vlad94]. More recently, describing mixed-signal systems has brought about languages like VHDL-AMS [VHDL] and VERILOG-AMS [VERI]. It is, however, not the intention of this text to give a complete overview of such languages nor to discuss their strengths and shortcomings. In what follows, mathematics will be the main language of choice.

2.1.5 Modeling and analysis: model creation, transformation and interpretation

Models are almost never given to us in a manner that directly suits our needs. In the very beginning, there might even be no model available at all. There are only our observations or a vague idea that we have in mind. Therefore, we need techniques to create and manipulate models. This is where modeling and analysis come into play.

Definition (Modeling and Analysis): *Modeling and analysis concern the acts of creating, manipulating and interpreting models. Hereby, the aim of modeling is to create a model, i.e. models are considered a result of the operation. In carrying out analysis,*

models are just a means to gain the information necessary to make design decisions. Analysis involves interpreting models.

The main difference between both concepts comes down to whether models are considered a result of the act or a means to obtain results. Looked upon in this manner, analysis always involves modeling steps followed by the interpretation of the resulting models and the consequences they imply towards design.

One of the hardest parts of the modeling process is the creation of a starting point. We need to construct an initial specification or implementation model. This requires a formalized description of ideas or observations. There is often nothing to guide us but our past experience and our capability to detect similarities between different systems and problems. Here, again, a hierarchical classification of systems and their behavior can be of great help (see chapter 1).

Once an initial model is created, we can proceed by transforming one model into another. A circuit netlist, for example, can be transformed into a small-signal model. Symbolic modeling techniques [Fern98, Gie91, Gie94, Lin91, Wamb98b] in turn transform this small-signal model into a set of transfer functions. The oscillator modeling methods presented in chapter 5 proceed by gradually transforming a set of circuit equations into a more compact description. Even techniques for numerical simulation can be considered as model transformations. Here, the key observation is to recognize that simulation methods produce models that capture input-output behavior by means of (*input signal, output signal*) tuples. Hence, the models that result from numerical simulation can be described as

$$M_{sim} = \{(\mathbf{x}_1(t), \mathbf{y}_1(t)); (\mathbf{x}_2(t), \mathbf{y}_2(t)); \dots\} \quad (2.1)$$

where the $\mathbf{x}_k(t)$ represent vectors of input signals and the $\mathbf{y}_k(t)$ vectors of corresponding output signals. The example at the end of this section illustrates this model transformation process for a single-stage amplifier.

At each stage in the process of modeling and analysis, we often have numerous possible transformations to proceed with. Which one to choose? Choice of a proper model transformation should be driven by the target we have in mind. Which part of the system's behavior is of greatest interest to us? Which kind of input signals are we dealing with? For example, if we want the resulting model to be parameterized, e.g. to be of use for trade-off analysis, the transformations should—at least partially—be symbolic in nature. If we are interested in the response to a limited set of input signals, numerical simulation might be the way to go. Note that in almost all cases, we want the resulting model to be as simple as possible.

Example (Analysis of a single-stage amplifier): Fig. 2.4 illustrates modeling and analysis for a single-stage amplifier. The starting point is a parameterized netlist. New models are derived by transforming this netlist model. A typical transformation, for instance, involves dumping the modified nodal equations. These equations in turn serve as a starting point for further transformations.

The transformations that we select should be driven by the interest in mind. This interest corresponds to what we want to know about the amplifier and its behavior. When

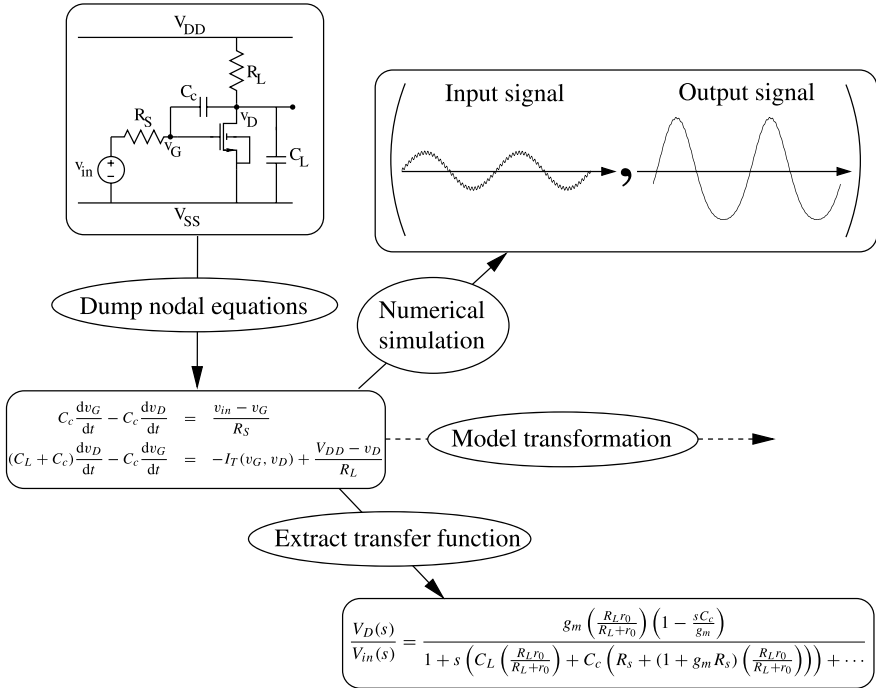


Figure 2.4: Analyzing the behavior of a single-stage amplifier involves transforming one model for the amplifier into another one. The idea is to select those transformations that yield the most simple results and/or that emphasize that part of the amplifier's behavior in which we are interested.

interested in the response to a particular input signal, a transformation involving numerical simulation is the way to go. When interested in the impact of the load capacitance C_L on the gain-bandwidth product, symbolic transformations present themselves as suitable candidates. Of course, when the symbolic expressions become too complicated, results are useless and we need to try a different transformation. In short, the art of system and circuit analysis comes down to selecting the proper sequence of model transformations. ▲

As a final note, we address automated modeling and analysis. Creating the initial model is often hard to automate. Coming up with a suitable circuit topology or an adequate elementary building block (transistor) model will always require some ingenuity. Transforming the models, however, can be automated if the transformations are formalized to the point that they can be written down as computer algorithms. This requires clear specification of the structure of the models that serve as an input. Furthermore, we must also state the conditions for the transformation to yield reliable results. Algorithms for symbolic analysis [Fern98], model-order reduction [Odab97] and numerical simulation [Kund97, Nag75] represent some well-known examples of automated model transformations.

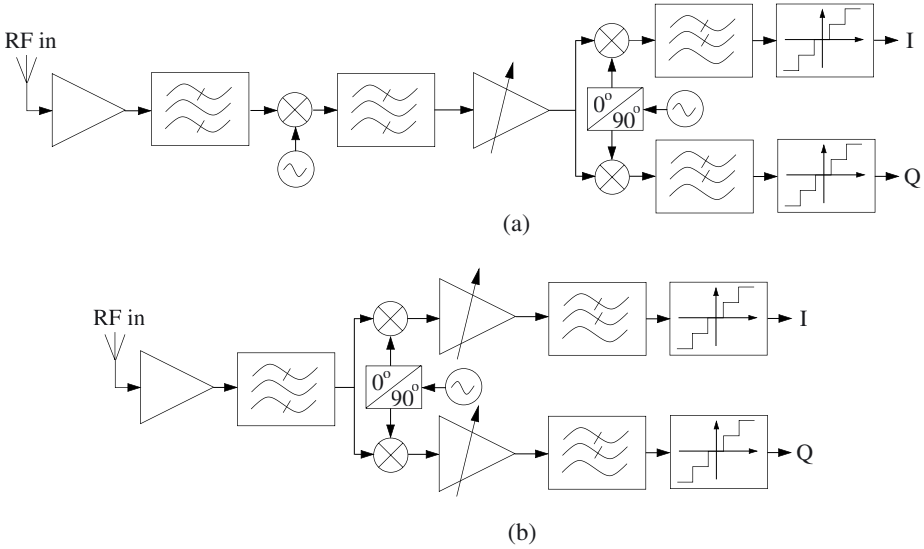


Figure 2.5: Frontend architectures of (a) a heterodyne receiver and (b) a low-IF receiver.

2.2 Good models for telecommunication frontends: Architectures and their behavioral properties

Having considered models, modeling and analysis from a global perspective, the main topic of this book can be phrased as: *the construction of good models for telecommunication systems and their building blocks*. As was outlined in section 2.1.2, the construction of good models requires us to incorporate all relevant prior knowledge and experience. This section summarizes this knowledge. It presents a brief review of common frontend architectures, their building blocks and their behavioral properties. It attempts to answer the question: what has experience taught us on the behavior of telecommunication frontends and their building blocks?

2.2.1 Frontend architectures and their building blocks

Fig. 2.5 depicts two commonly used receiver frontend architectures. The heterodyne receiver on top provides good performance in terms of channel selectivity and sensitivity, but typically requires surface-acoustic wave (SAW) bandpass filters in combination with additional IF circuitry. The low-IF architecture on the bottom requires less parts, but exhibits inherent problems such as self-mixing, $1/f$ noise and sensitivity. Self-mixing comes from the local oscillator (LO) signal making its way to the input of the mixer. This generates a DC component at the mixer output, possibly saturating the filters and gain amplifiers that follow. In [Crol97] a more rigorous overview of receiver (and transmitter) architectures is presented.

Observing both architectures, it is seen that their functioning relies on similar building blocks: low-noise amplifiers (LNAs), automatic gain control (AGC), filters, mixers, analog-to-digital converters (ADCs) and local oscillator (LO) signals. The latter are typically derived from a reference signal using a voltage-controlled oscillator (VCO) embedded in a phase-locked loop (PLL). Modeling an entire receiver frontend therefore requires us to have models for each of these building blocks. This book focuses on techniques that can be used to model the behavior of mixers, oscillators and PLLs.

Note that Fig. 2.5 only considers receiver architectures. Transmitters, however, have similar structures. In this case, the I - and Q -channels serve as inputs that are fed to digital-to-analog converters (DACs). The signals are then combined, upconverted to RF and transmitted through the antenna which is driven by a power amplifier (PA). Modeling DACs and PAs is, however, not a topic in this book³.

2.2.2 Properties of frontend building block behavior

The construction of good frontend building block models requires us to incorporate all relevant prior knowledge. As will be illustrated in subsequent chapters, exploiting this knowledge helps us to improve model quality. This, for instance, makes simulations run faster. In what follows, we give a brief overview of some properties in common to many building blocks that occur in telecommunication frontend architectures: their almost linear nature, the presence of widely spaced time constants and the presence of stochastic (noisy) components in their behavior. Note that subsequent chapters also exploit other properties. However, these properties are often tied to a single building block. It is therefore not relevant to discuss them here.

2.2.2.1 Almost linear

Most building blocks in telecommunication frontends are, by design, intended to behave linearly. Here, *linear* should be interpreted in its most general, time-varying setting. As will be discussed in depth in chapter 3, up- and downconversions, induced by a multiplication $y(t) = u(t) \times \cos(\omega_0 t)$ of an input signal $u(t)$ with some periodic carrier signal $\cos(\omega_0 t)$, are linear operations: the principle of superposition still holds. This stands contrary to popular belief that considers multiplying two non-constant signals as a nonlinear operation. The key issue to observe is that one of the signals in the product—the carrier signal—is *known at the time when the mixer is designed*: It does not depend on the the information (the data bits) that is transmitted or received. In chapter 3 and 4, it will be shown how linearity can be exploited to significantly facilitate analysis of up- and downconversion behavior.

However, in almost any real system, it also occurs that two information-carrying (data) signals $u_1(t)$ and $u_2(t)$ intermodulate. This happens when the system's output depends, for instance, on the product $u_1(t) \times u_2(t)$ of the two data signals. The intermodulation of two data signals is truly nonlinear: the principle of superposition no longer holds. Fig. 2.6 illustrates the difference between the linear intermodulation of a data signal

³With regard to PAs, it is possible to handle them using the HTM-formalism in chapter 3 as well as the separation of time constants methods in chapter 5.

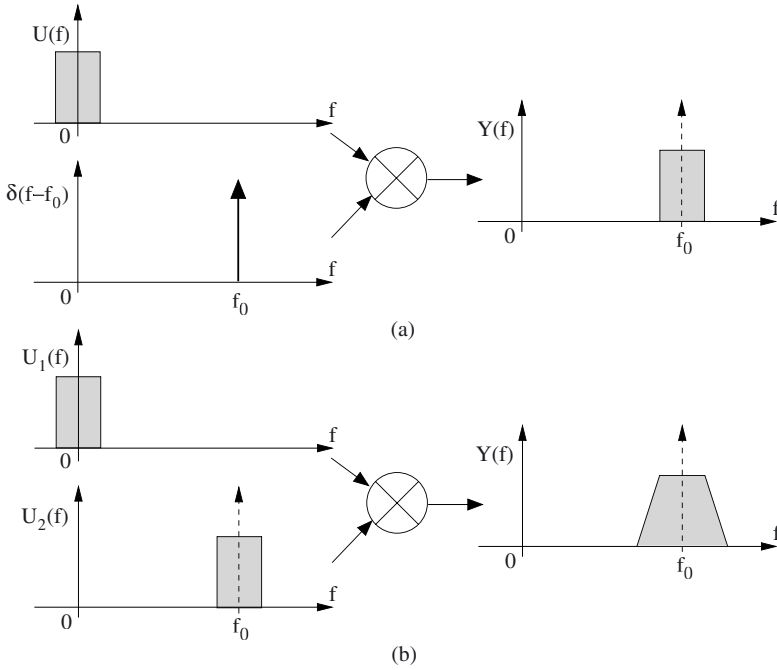


Figure 2.6: Difference between linear and nonlinear intermodulation. (a) Linear intermodulation involves the product of a (a priori unknown) data signal $u(t)$ and a (known) carrier. As a result, the information contained in $u(t)$ is shifted along the frequency axis. (b) Nonlinear intermodulation involves the product of two data signals $u_1(t)$ and $u_2(t)$. As a result, $u_1(t)$ is upconverted and, on top of that, its content is spread out over the frequency axis. Typically, this spectral spreading is not desired.

and a known carrier and the nonlinear intermodulation of two data signals.

Nonlinear intermodulation is in most cases undesired. Too large a contribution to the overall signal content may very well ruin the system's performance. In order to prevent this from happening, a good design should keep the signal components due to nonlinear intermodulation as small as possible. This means that their magnitude is well below that of the signal components generated by linear system behavior. Good design therefore requires most telecom building blocks to behave *almost linearly*. In traditional literature [Wamb98a], it is more common to say that the system behaves in a *weakly nonlinear* manner.

2.2.2.2 Widely spaced time constants

Many wireless applications transmit their information by upconverting it to the GHz frequency range or beyond. For example, GSM operates in the 900 MHz frequency

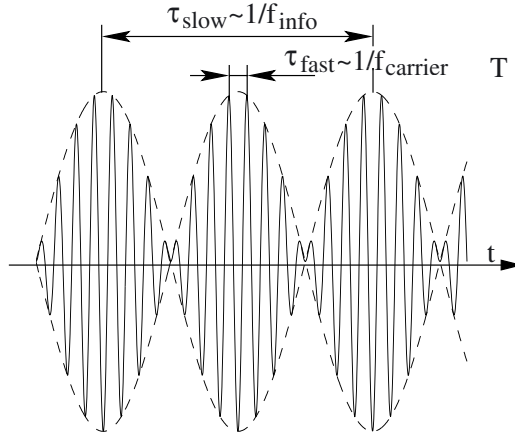


Figure 2.7: Many wireless applications transmit information by upconverting it to the GHz range. This results in signals containing widely spaced time constants. The carrier introduce a fast time constant τ_{fast} while the upconverted information induces a slow time constant τ_{slow} .

band, DCS1800 in the 1.8 GHz band while wireless LAN (WLAN) applications operate near 2.4 GHz (ISM band) or 5 GHz (military applications). Signal bandwidths, however, are often orders of magnitude smaller. GSM bands are, for instance, 200 kHz wide while WLAN bands occupy a few MHz. Hence, the rate at which information is transmitted is well below the carrier frequency, or

$$f_{info} \ll f_{carrier} . \quad (2.2)$$

The resulting signals, illustrated in fig. 2.7, can be described as rapid oscillations with a slowly modulated amplitude and phase. This kind of behavior can also be generated by a building block's internal transient dynamics. For example, both high-Q harmonic oscillators and PLLs settle at a rate that is much slower than that of their steady-state output oscillation.

Traditional SPICE-like simulators [Nag75] experience a great deal of trouble in evaluating this kind of behavior. As illustrated in Fig. 2.8, the time step τ_{sim} with which simulations proceed is inversely proportional to the highest signal frequency, or

$$\tau_{sim} \sim \frac{1}{f_{carrier}} . \quad (2.3)$$

The total time interval T_{sim} over which simulations are performed is typically a multiple of the length of a single information symbol, or

$$T_{sim} \sim \frac{N}{f_{info}} . \quad (2.4)$$

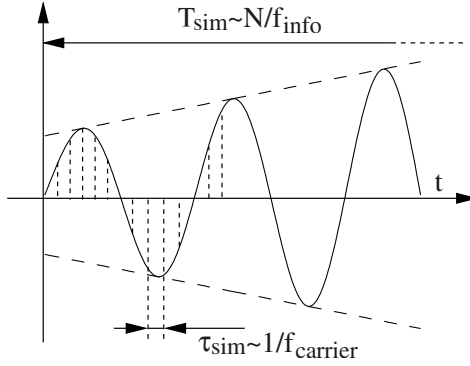


Figure 2.8: The time step τ_{sim} with which Spice-like simulations proceed is inversely proportional to the highest signal frequency. This time step is much smaller than the length T_{sim} of the entire simulation interval. As a consequence, the presence of widely spaced time constants forces SPICE-like simulation algorithms to take an enormous number of simulation steps.

Hence, to cover the entire simulation interval, a SPICE-like simulator needs to take

$$S = \frac{T_{sim}}{\tau_{sim}} \sim N \frac{f_{carrier}}{f_{info}} \quad (2.5)$$

steps. With $f_{carrier}/f_{info}$ often being over 1000 and with, for bit error rate simulations, N being a million in order of magnitude, SPICE-like simulators are forced to take over a billion steps to complete a single simulation run. As a result, the simulation of a complete system architectures may take days or even weeks to complete.

Both chapters 3 and 5 present methods that cope with this problem by means of models that keep slow- and fast-varying behavior explicitly separated. In this way, the situation in Fig. 2.8 is avoided. Wide separation of carrier and settling time constants is even a necessary condition for the oscillator modeling methods in chapter 5 to work properly. It is an excellent example on how building block characteristics that at first seem to hamper efficient analysis, can be exploited to speed up simulation.

2.2.2.3 Stochastic behavior

All physical systems, and, therefore, all telecom systems and their building blocks, have in common that they produce noise. For example, both thermal and $1/f$ noise occur in a wide variety of applications. Noisy signals, also called *stochastic processes*, are disturbances about which little information is available. Hence, they cannot be compensated for and, as a consequence, they cause data transmission errors. As it is the case for signal components due to nonlinear intermodulation, it is important to keep their magnitude small.

Contrary to deterministic signals, the shape of a stochastic process $n(t)$ is never known exactly. There is only a certain probability that a particular shape will occur. Stochastic processes are therefore characterized by probability density functions that quantify the likelihood that the process assumes a particular shape. However, it is often more practical to characterize signal stochastics by means of its moments or, equivalently, its cumulants [Mid87]. If we are mainly interested in a stochastic signal's energy content, then it is sufficient to know all moments up to second order, i.e. the signal's expected value $\mu(t) = E\{n(t)\}$ and autocorrelation $\Phi(t, \tau) = E\{n(t + \tau/2)n(t - \tau/2)\}$.

In order to cope with noise, any modeling framework for a particular class of building blocks must at least be able to characterize stochastic input-output behavior by means of the moments up to second order. Chapters 3, 4 and 5 outline how to perform noise computations for, respectively, LPTV systems and oscillators.

2.3 Conclusions

Models are descriptions of systems that we want to realize or that we have realized. In a top-down design flow they are used to communicate design decisions between the different levels of hierarchy. For example, a frontend system engineer will pass an LNA specification model to a circuit engineer. It describes the LNA as the system engineer would like it to behave. For the circuit engineer, it makes up the starting point for circuit-level synthesis, resulting in an implementation model, e.g. a Spice netlist. This model describes the LNA as it is realized. It is used by the system engineer for frontend-level verification.

In order for a top-down design flow to be successful, both specification and implementation models must be as accurate as possible in capturing the behavior of actual on-chip implementations. Defective and incomplete models are one of the main reasons for a redesign to be required. Of course, at any given stage of the design process, one only disposes of the knowledge gathered up to that point in time. This includes experience obtained from similar designs in the past. Models including all relevant prior knowledge and experience are called good models. The target of this work can therefore be phrased as “constructing good models for telecom frontends and their building blocks”.

Past designs of telecommunication frontends show that their behavior and that of their building blocks all have some important properties in common:

- Their behavior is almost linear (weakly nonlinear). Here, linearity must be interpreted in its most general, time-varying setting.
- They produce signals containing widely spaced time constants. This especially holds for the RF sections of, for example, wireless applications.
- They all produce noise, i.e. they produce signals containing random elements.

Subsequent chapters in this book exploit these and other properties to construct compact models that allow accurate and efficient evaluation of telecom frontends and their

building blocks. This assists us in their design and in the identification of performance bottlenecks.

Finally, models should only capture relevant behavior. Irrelevant details only burden analysis and simulation without deepening a designer's understanding of the system's behavior. In communications the relevance of building-block behavioral characteristics—and therefore the need to incorporate them into (good) models—is measured by their impact on the system's ability to transmit information without great loss. Unfortunately, it turns out that few building block behavior can be neglected. This especially holds when pushing the limits of performance. Hence, one of the great challenges for CAD is to capture complex building block behavior in a manner that is as compact as possible.

Systematic Modeling and Analysis of Telecom Frontends
and their Building Blocks

Vanassche, P.; Gielen, G.; Sansen, W.M.C.

2005, XX, 230 p., Hardcover

ISBN: 978-1-4020-3173-1