An Extensible Compact Model Description Language and Compiler

R. V. H. Booth, Agere Systems
1247 S. Cedar Crest Blvd., Allentown, PA 18103
(610) 712-2324 rvbooth@agere.com

Abstract

In this paper, we describe the Compact Model Compiler approach to developing and supporting compact device models. A single model specification and model compiler tool support circuit simulation, parameter extraction, and documentation. We present three examples of the model compiler approach: a compact transisitor model, a bandgap reference circuit, and a flip-flop.

Introduction

Developing, implementing, and providing support for a compact device model, which is to be used for analog/mixed-signal circuit simulation, requires significant time and resources. This is because the model must be represented in many different domains, including the development platform, the circuit simulation software, documentation, and parameter extraction tools.

For example, some analog/mixed-signal circuit simulation programs provide an application-program interface, such that user-specific models can be incorporated into the simulator [1]. Other packages provide high-level means for describing non-standard device models, albeit in simulator-specific language, such as VHDL-AMS and Verilog-AMS [2], and MAST [3]. Each of these circuit simulator targets for a compact device model requires special handling, including reformatting and possible re-coding. Whenever a revision to the model is implemented, whether major or minor, then all of the targets must be revisited and maintained.

In this paper, we describe the Compact Model Compiler approach to this problem. This approach consists of: (1) a single model specification, and (2) a model compiler tool.

The model specification supports all downstream representations, such as circuit simulation and parameter extraction tools. It includes generic specifications for all details of the model, including the equivalent circuit topology, model parameters, branch-limiting, element equations, and parameter extraction frames. The specification may also cover an entire model family, including variations for device polarity and model complexity.

The model compiler is used to compile, or in some cases translate, the model specification, for incorporation or use within the downstream application. This is typically done by generating code for an API which is supplied by the respective application. The model compiler must be extensible, so that as further applications are presented, it is straightforward to add target code to support the new platforms. It must also be straightforward to add new model specifications, as required, and new compiler code to support them.

The model compiler tool we have developed at Agere Systems, the **A**nalog **M**odel **C**ompiler, supports a variety of different platforms: circuit simulators, parameter extractors, and stand-alone solvers, and automatically generates model documentation. The AMC model compiler approach greatly improves model maintainability, since only one description file must be maintained, rather than source code for the different target simulation and extraction tools. We have used the model compiler approach to develop and support a large variety of device models, for example [4].

AMC provides a powerful compact-modeling development platform. Model development initially involves exercising the model using a stand-alone solver to verify that the correct behavior over bias, frequency and temperature has been achieved. Next, a parameter extractor is used to fit model parameters to measured data, and to develop a workable parameter extraction strategy. Finally, the behavior of the model in a circuit simulator must be investigated. All of these targets are generated using AMC and a single model description. Once the model is ready for distribution, model documentation can be automatically generated.

The AMC compiler/approach is a means for porting models to newly acquired circuit simulators. In this situation, no model specification need be rewritten for the new target, but rather only specific target code within the compiler. Once this step has been accomplished once, all models can be ported to the new simulator.

The model compiler approach provides an efficient way to answer specific questions about whether a particular processing problem gives rise to unusual circuit behavior. In this situation, special model implementations with the suspected processing peculiarity are implemented, but not distributed to the design community.

In addition to using AMC for representing basic physical device models, the Model Compiler approach can be used for higher-level descriptions of circuit building blocks, which can drastically speed up the simulation of large systems such as Phase-Locked-Loops, without sacrificing any accuracy. This analog behavioral approach, used in an analog simulator, complements the large-digital/small-analog approach for simulating large systems.

In the next two sections, we describe the architecture of AMC, and describe some details of the description language. In the following sections, we present three examples using the model compiler approach: a compact transisitor model, a bandgap reference circuit, and a flip-flop.

Model Compiler Architecture

The AMC system consists of a front-end and a back-end (Figure 1). The front-end is responsible for extracting and storing information about the model from the AMC compact-model description language file. The back-end is responsible for generating source-code or other target-specific text-files. In addition, if it is required for a specific target, the back-end proceeds to actually compile the AMC-generated source-code and completely generate an executable.

AMC is an object-oriented application – separate classes for different aspects of a compact model are responsible for storing the information (front-end), and reformatting the information (back-end).

The back-end of AMC is template driven. Generating source code for each target involves filling in one or several templates which are very specific. The templates are written with embedded code which requests information from the various classes which have stored the information from the model description. Therefore, the bulk of the work to build a new target involves writing templates. Within AMC, there are many internal methods which are required for the template requests for information. For example, dependency information for each set of element equations is developed for many of the targets which require this information. Presently, templates do not issue all of the commands which are required to build a target. The rest of the work is done in a Model class method. That is, the Model class method controls building the target, but allows the template a great deal of freedom in doing the job.

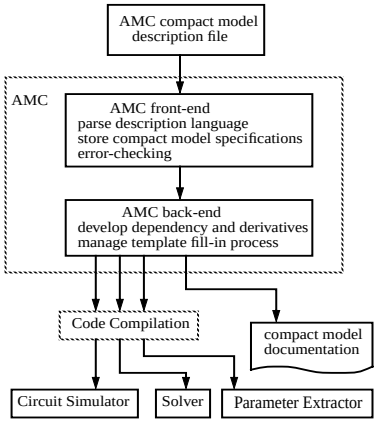


Figure 1. AMC model compiler block diagram.

Currently, AMC provides support for three circuit simulators: ADVICE and Celerity, which are Cadence tools which were originally developed at Bell Laboratories, and Cadence Spectre. AMC can translate the model into Verilog-AMS and VHDL-AMS format. Other targets are the stand-alone solver, parameter-extractor, and HTML-format model documentation. For the circuit simulators, AMC generates an initial input deck for exercising the model in the simulator with DC sweep, AC analysis, and noise analysis.

AMC description language

The AMC description language is actually Tcl-Tk code. This arrangement allows each description file to be an executable program, interpreted by Camelot [5], which is a Tcl-Tk application, incorporating Tcl-Tk 8.0, [incr Tcl] [6], and other extensions. There are many benefits to using Tcl /[incr Tcl] as the description language. Tcl is a scripting language which is already well-documented. Since Tcl is used to interpret the description file, no new parsing engine is required. Tcl can be used to structure the compact-model description in a programmatic way, such as partitioning the model into a hierarchy. Tcl provides extensive error-checking – many errors in the model description can be flushed out simply by responding to reported Tcl errors. Adding additional error-detection in the model description can easily be added at any point in the compiler. This early error-detection in the model description greatly improves the maintainability of the model description file.

The AMC language is essentially a set of model specification commands, each of which is a method of the [incr Tcl] Model class. The specification commands are mixed-case phrases. However, it would be trivial to completely change the specification command-set, should this be of interest, simply by changing the names of the respective Model class methods, and only slightly more difficult to modify the body of a specification, and would involve changing the details of the respective front-end procedure.

Simple MOSFET transistor model

Figure 2 shows a simple MOSFET transistor compact-model AMC description. In the description, the `compact-model` command takes two arguments: the model name and a list of specifications. Immediately following the `compact-model` command is the `amc-compile` command, which invokes the compiler with any user-supplied arguments. If no arguments are supplied, a dialog box is displayed, and the user can select models and targets.

The first specification of the `compact-model` command is `componentSpecs`, which establishes the name of the model component and its heritage, if applicable. Following this specification are the specifications for the topology of the equivalent circuit. Nodes are specified before branches and elements, and can be either external or inter-

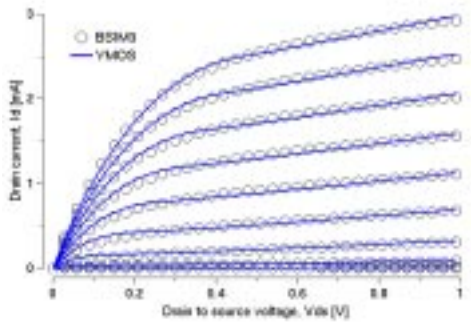
```
compact-model ymos {
  componentSpecs {
    family mosfet component ymos version 1.0
  }
  nodeSpecs external {
    d "Drain"; g "Gate"; s "Source"; b "Bulk"
  }
  branchSpecs {
    Vds d s "Drain to source voltage"
    Vgs g s "Gate to source voltage"
    Vbs b s "Bulk to source voltage"
  }
  elementSpecs {Ids d s}
  elementEqns Ids {
    rev  = ISLTZ(Vds)
    rsp  = sqrt(PHI-MIN(Vbs-Vds*rev,0))
    abc  = 1+GAMMA/(2*rsp)
    dib  = ETA-(1+2*ETA)*rev
    Vsth = SST/1151
    Vgse = Vgs-VTH-GAMMA*(rsp-sqrt(PHI))+dib*Vds
    Vgsl = Vsth*log(1+exp(Vgse/Vsth))
    Vdss = Vgsl/(abc+THETAC*Vgsl/2)
    Vdsl = MAX(MIN(Vds,Vdss),-Vdss)
    vmob = 1+THETAS*(Vgsl+2*GAMMA*rsp)
    hmob = 1+THETAC*ABS(Vdsl)
    cls  = 1+LAMBDA*ABS(Vds-Vdsl)
    Ids  = BETA*cls/(vmob*hmob)*\
           (Vgsl-abc*ABS(Vdsl)/2)*Vdsl
  }
  parameterSpecs process {
    {BETA   1e-3 1e-4 0.1
      A/V^2   "transconductance parameter"}
    {VTH    0.5  0.2  2.0
      V       "threshold voltage at Vbs=0"}
    {PHI    0.7  0.5  1.2
      V       "strong-inversion surface potential"}
    {GAMMA  0.7  0    2
      sqrt(V) "body-effect parameter"}
    {LAMBDA 0.01 0    0.5
      1/V     "output-conductance parameter"}
   {THETAS 0.01 0    5
      1/V     "verti. field mobility parameter"}
    {THETAC 0.10 0    5
      1/V     "horiz. field mobility parameter"}
    {ETA    0    0    0.2
      #       "dibl parameter"}
    {SST    80   20   200
      mV/dec  "subthreshold slope"}
  }
  extractorData ts {
    variables {Vd}
    interface {MODEL = Id*1E6}
    screen    {MEAST = Id*1E6}
    plots {xy Vd MEAST MODEL -xtitle Vds -ytitle Id}
  }
  extractorFrame TS1 {
    fit {ts}
    evaluate {ts th st}
    residual {conductance}
    include  {LAMBDA THETAC}
  }
}
eval amc-compile $argv
exit 0
```

Figure 2. Simple MOSFET transistor compact-model AMC description.

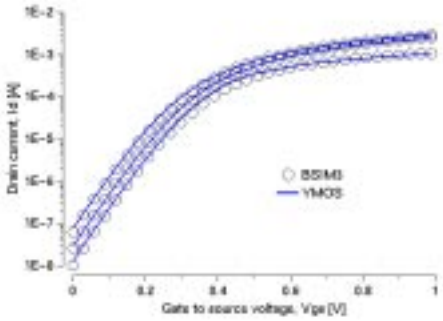
nal. Branches are aliases for voltage differences between two nodes, and can be used in the element equations. Elements are equivalent circuit elements connected between two nodes.

For each element in the equivalent circuit, equations are specified in an `elementEquations` specification. One of the lines in this specification must be an assignment to the variable by the same name as the element; in this case `Ids`.The YMOS model is symmetric about the MOSFET drain and source. If the source voltage is higher than the drain voltage, then the equations effectively turn the device around.

Parameters which appear in the equations are specified in the `parameterSpecs` block. In this case, all of the parameters do not depend on layout parameters, such as channel length or width, so they have been specified as `process` parameters, or ones which do not change with specified dimensions. Each specification in the parameter block configures each parameter with typical values, limits, units and descriptive text. For the parameter extractor target, the parameter limits are used to constrain the values the parameters are allowed to assume to realistic values during optimization. On the other hand, for a circuit simulator target, the parameter limits may be used for detecting out of range parameter values. The units and description fields are provided for the model-documentation target.



(a)



(b)

Figure 3. YMOS simplified MOSFET compact model playbacks versus data generated by simulating a 0.13 µm n-MOSFET using BSIM3 models for the TSMC CL013LV process. (a) Output characteristics (b) Subthreshold characteristics.

The parameter extractor target uses several specifications for the data sets which are to be fit, and for the extractor frames, or subroutines. In the listing in Figure 2, only one of each of these is shown.

A parameter extractor was constructed for the YMOS model and used to fit the model to simulated characteristics of a 0.13 µm channel-length n-MOSFET from the TSMC

CL013LV process using the foundry-supplied BSIM3 models. Playbacks of YMOS versus BSIM3 are shown in Figure 3. YMOS fits the output and subthreshold characteristics very well, even though a very simple output conductance model is employed. Although not shown in the figure, the bulk bias dependence is also well-modeled.

Bandgap reference circuit compact model

```
Macro ISGT {X Y} {
  ISGT = 0.5*(1-signum(Y-X))
}
Constant Tabs 273.16

compact-model bg {
  componentSpecs {
    family bg
    component bg
    version 1.0
  }
  nodeSpecs external {
    v "bandgap reference voltage"
    h "high reference"
    l "low  reference"
  }
  branchSpecs {
    Vsup h l
    Vv   v l
    Vq   v h
  }
  elementSpecs {
    Iv   v l
    Qv   v h
  }
  parameterSpecs process {
    {VF   1.2  0.5    2  V "flat voltage"}
    {TF    65  -50  125  C "flat temperature"}
    {A1     0  -20   20  # "temperature coefficient 1"}
    {A2    50    0  200  # "temperature coefficient 2"}
    {A3    50    0  200  # "temperature coefficient 3"}
    {VDO  2.0    0   20  V "drop-out voltage"}
  }
  parameterSpecs temperature_derived {VREF}
  temperatureEqns {
    tn = (TEMPERATURE-TF)/(TF+Tabs)
    VREF = VF*(1+1e-3*(A1+(-A2+A3*tn)*tn)*tn)
  }
  elementEqns Iv {Iv = Vv-VREF*ISGT(Vsup,VDO)}
  elementEqns Qv {Qv = Vq*1e-13}
}
eval amc-compile $argv
exit 0
```

Figure 4. Bandgap reference circuit compact-model AMC description.

Figure 4 shows the AMC description of a bandgap reference circuit compact model. At the top of the description is a macro definition. The macro `ISGT` returns a 1 if the argument X is greater than Y, and 0 otherwise. In addition, the constant `Tabs`, 0 degrees C in absolute temperature, is defined. There are only external nodes in the compact model. The bandgap reference voltage appears at the "output" node `v`. Nodes `h` and `l` establish the high and low reference voltages of the circuit. We make use of the high reference node to detect whether the supply voltage to the circuit is greater than a drop-out voltage, which is specified by the value of the `VDO` model parameter

Branches and elements of the equivalent circuit are specified in the `branchSpecs` and `elementSpecs` blocks of the description code. The main voltage-controlled current source, `Iv`, is connected between nodes `v` and `l`. The voltage-controlled charge element `Qv` is not strictly necessary, but is included in the description simply so that the simulation program does not complain that there are too few elements connected at node `h`.

The variable parameters of the model are specified in the first `parameterSpecs` block of description code. Just as in the MOSFET compact-model, these parameters are categorized as `process` parameters, since the model does not include a geometry mapping

In the second `parameterSpecs` block, another

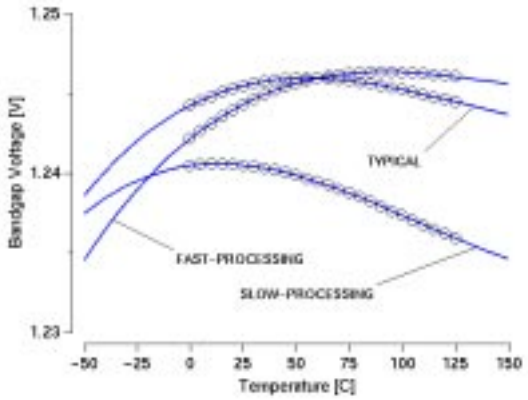


Figure 5. Output reference voltage versus temperature, from circuit simulation of bandgap reference compact-model. Three circuits representing typical, slow-processing, and fast-processing case corners are shown (solid lines). The original bandgap post-layout simulations are also shown (circles).

class of model parameters is listed. These `temperature derived` parameters are invisible to the user of the compact model in a circuit simulator. That is, the parameter value can not be specified on the model card or instance line. Rather, these parameters are calculated from other parameters and temperature in the simulator's temperature pre-calculation routine for the compact model. The pre-calculated value is then available to equivalent circuit elements which use the parameter.

The temperature pre-calculation routine is specified in the `temperatureEqns` block. Here, the reference voltage parameter, `VREF`, is calculated using the current simulation temperature, `TEMPERATURE`

Finally, the equivalent circuit element equations are specified in the `elementEqns` blocks. The reference voltage at node `Vv` is established by passing a current of `VREF` through a one ohm resistor. These elements are in parallel, and can be described with one equivalent element, `Iv`. The reference voltage is nulled out when the supply voltage, `Vsup`, is less than the drop-out voltage `VDO`.

We characterized an actual bandgap reference circuit using this model. To do this, a series of computer simulations of the actual bandgap reference circuit were performed over temperature. The compact model was then used to fit the bandgap voltage versus temperature data. The compact model was incorporated into a circuit simulation program and, using the resulting model parameters, played back over temperature. A comparison between the actual reference and the compact model is shown in Figure 5. The compact model can be used in place of the actual reference circuit, with no sacrifice in accuracy. When using the compact model, no start-up time is required, saving a significant amount of overhead time required to simulate a circuit incorporating the bandgap reference.

Flip-flop compact model

Figure 6 is the listing of a reset-able D-flip-flop AMC compact model description. The flip-flop model can be used to build up dividers and more complex circuitry. The model is derived as shown in Figure 7. The basic high-level flip-flop circuit consists of a master and slave section, each with a monitoring controlled source `Vm` and `Vs`, respectively, a hold capacitor, `Cm` and `Cs`, respectively, and a switch to sample the monitored voltages, `Gm` and `Gs`, respectively. `Vm` monitors the `D` input, while `Vs` monitors the output of the master section, `V(m)`. In the master section, we transform the series connected controlled source `Vm` and switch conductance `Gm` into a Norton equivalent parallel-connected current source `Im` and switch conductance `Gm`. The current source and conductance are further combined into a single current source `Im`, as shown in Figure 7(b).

In the model description, we use a hyperbolic tangent transformation for sampling the digital inputs. This is implemented in the `Dig` macro at the top of the description file. Rather than a sharp transition between 0 and Vdd, the transformation is a smooth function of the data, clock and reset inputs.

The master monitor supply `Vm`, from Figure 7(a) is

$$V_m = V_{dd} \cdot \mathbf{Dig}(V(d)) \quad (1)$$

where Vdd is the supply voltage, V(d) is the voltage at the data input. The master switch conductance is

$$G_m = G_{on} \cdot \mathbf{Dig}(V(c)) + G_{off} \cdot (1 - \mathbf{Dig}(V(c))) \quad (2)$$

where Gon and Goff are the on and off switch conductance, respectively, and V(c) is the voltage at clock input.

In Figure 7(b), the current source is

$$I_m = (V_m - V(m)) \cdot G_m \quad (3)$$

In the compact model description, Im is turned around,

```
# Flip-Flop compact model

Macro Dig {v vdd} {
  b   = v-vdd*0.5
  s   = signum(b)
  u   = exp(-460.5*s*b)
  Dig = 0.5*(1+s*(1-u)/(1+u))
}
Constant Glow 1e-8

compact-model ff {
  componentSpecs {
    family ff
    component ff
    version 1.0
  }
  nodeSpecs external {
    c "clock"
    d "data"
    r "reset"
    q "noninverted output"
    b "inverted    output"
    h "high reference"
    l "low  reference"
  }
  nodeSpecs internal {
    m "master output"
    s "slave  output"
  }
  branchSpecs {
    Vsup  h l
    Vc    c l
    Vd    d l
    Vr    r l
    Vm    m l
    Vs    s l
    Vqu   q h
    Vqd   q l
    Vbu   b h
    Vbd   b l
  }
  elementSpecs {
    Im    m l
    Is    s l
    Iqu   q h
    Iqd   q l
    Ibu   b h
    Ibd   b l
    Qm    m l
    Qs    s l
    Qc    c l
    Qd    d l
    Qr    r l
  }
  parameterSpecs instance {
    {DELAY 1e-9  1e-12 100  s "clock to Q delay"}
    {CIN   1e-14 0     1e-9 F "input capacitance"}
  }
  elementEqns Im {
    chi = Dig(Vc,Vsup)
    rhi = Dig(Vr,Vsup)
    gr = rhi+(1-rhi)*Glow
   Im =(chi+(1-chi)*Glow)*(Vm-Vsup*Dig(Vd,Vsup))+gr*Vm
  }
  elementEqns Is  {
   Is =((1-chi)+chi*Glow)*(Vs-Vsup*Dig(Vm,Vsup))+gr*Vs
  }
  elementEqns Iqu {
    shi = Dig(Vs,Vsup)
    gq  = shi+(1-shi)*Glow
    gb  = (1-shi)+shi*Glow
    Iqu = gq*Vqu
  }
  elementEqns Iqd {Iqd = gb*Vqd}
  elementEqns Ibu {Ibu = gb*Vbu}
  elementEqns Ibd {Ibd = gq*Vbd}
  elementEqns Qm  {Qm  = DELAY*1.5*Vm}
  elementEqns Qs  {Qs  = DELAY*1.5*Vs}
  elementEqns Qc  {Qc  = CIN*Vc}
  elementEqns Qd  {Qd  = CIN*Vd}
  elementEqns Qr  {Qr  = CIN*Vr}
}

eval amc-compile $argv

exit 0
```

Figure 6. Flip-flop compact-model description.

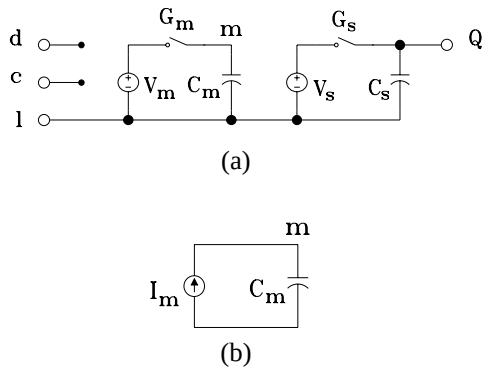


Figure 7. Formulation of the flip-flop compact model. (a) Master-slave sample and hold blocks, (b) Combined Norton-equivalent of master section.

and a reset switch is also added. Capacitors Qm and Qs are used to implement the specified clock to Q delay.

We performed a computer simulation of two instances of the ff D-flip-flop compact model hooked up as a ripple counter, as shown in Figure 8. The output waveforms are shown in Figure 9, where `Div2` and `Div4` are the outputs of the first and second flip-flop. When the circuit first starts up, no initial condition was applied, and since the dividers have not been reset, they start at half supply voltage.

In Figure 9(b), a close-up is shown at a clock transition showing the specified clock to Q delay of about 1ns between `Clock` and `Div2`, and between `Div2` and `Div4.`

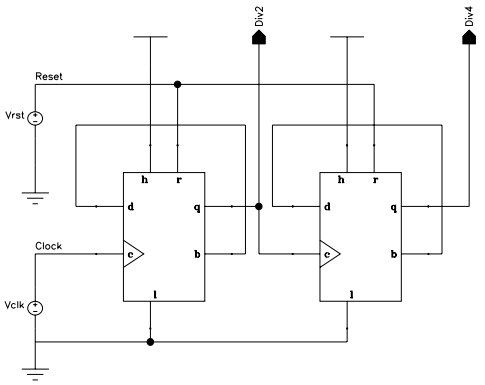
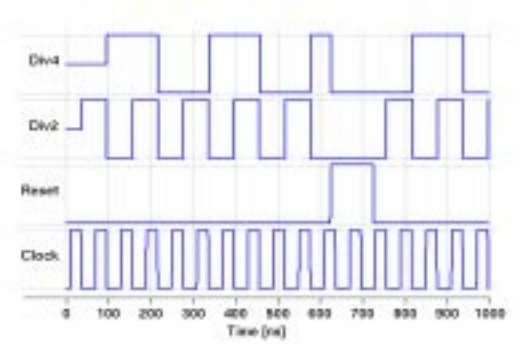


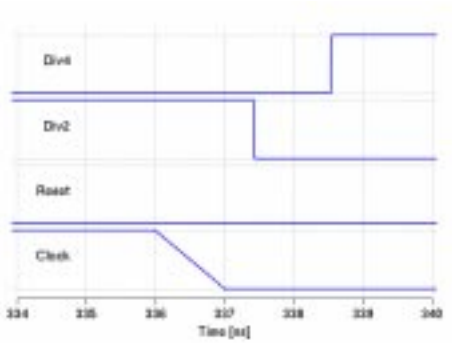
Figure 8. Ripple-counter composed of two ff instances.

Conclusions

The Compact Model Compiler methodology and technology implemented in AMC has been used to develop and support a wide range of device models for analog circuit simulation. In this paper we presented one device model and two high-level models for circuit building blocks. The high-level models can be used in an analog circuit simulator to speed up transient simulations of moderately sized modules such as phase-locked loops, which are composed of many such sub-blocks..The AMC extensible model description language and compiler provides a framework to develop new models and to support existing ones



(a)



(b)

Figure 9. Ripple-counter simulated waveforms.

Acknowledgments

The author acknowledges the work, help, and suggestions of Kishore Singhal, Sani Nassif, Colin McAndrew, David Lee, Jeffrey Hantgan, Karti Mayaram, Michael McLennan, Changlin Ma, Averill Bell, Shahriar Moininan, George Howlett, David Goldthorp, Russel Pierce, Margaret French, and Kathy Krisch.

References

[1] *Spectre Compiled-Model Interface Reference Manual*, Version 4.3.4, Cadence Design Systems, 1995.

[2] *Analog Behavioral Modeling with the Verilog-A Language,* D. Fitzpatrick, I. Miller, 1998, Kluwer.

[3] *Modeling with an Analog Hardware Description Language,* H.A. Mantooth, and M. Fiegenbaum, 1995, Kluwer.

[4] R.V.H. Booth and C.C. McAndrew, "A 3-terminal model for diffused and ion-implanted resistors," *IEEE Trans Electron Devices*, volume 44, number 5, May 1997.

[5] M. S. Toth and R. V. Booth, "A Designer-Customizable Design Environment for Analog/Mixed-Signal Circuit Design," presented at the *O'Reilly Open-Source Convention*, San Diego CA, July 23-27, 2001.

[6] *Tcl-Tk Tools*, M. Harrison, ed., O'Reilly, 1997.