Derivative Management in Verilog-A

19th Hicum Workshop 2019 at Infineon Technologies AG, AmCampeon 1-12 Neubieberg, Germany
13-14.05.2019

Letter Session

Zoltan Huszka, Kund Molnár
03.05.2019
Outline

Derivative Management in Verilog-A

- Verilog-A in compact modeling
- features
- evens and odds of the automatic derivative generation
- concept of suppressing derivatives
- unphysical derivatives
- weak derivatives
- summary
- references
Verilog-A in compact modeling

Derivative Management in Verilog-A

- Verilog-A (VA) is an analog subset of hardware description languages for defining compact models
- officially supported by a regularly updated Language Reference Manual e.g. [1]
- accepted and proposed by the CMC
- allows model providers to focus primarily on device physics
- numerical details are left for an additional program block the Verilog-A compiler
- this compiler translates the model instructions to C++ code which is the most common modeling interface to present simulators
Features

Derivative Management in Verilog-A

• Verilog-A describes physical systems by the physical network approach

• the system may be electrical, thermal, mechanical etc. depending on the selection of the through (e.g. Current) and across (e.g. Voltage) variables

• such systems obey Kirchhoff’s circuit rules particularly Kirchhoff’s Current Law (KCL) and Kirchhoff’s Voltage Law (KVL) for the defined through and across variables

• such systems can be solved for the across variables by the Newton-Raphson (NR) iteration scheme

• the system matrix is the Jacobian with elements as the partial derivatives of the through variables w.r.t. the across variables

• the derivatives are automatically generated by the VA compiler
Compact models typically include complicated equations. Manual derivative calculation is tedious and error prone. Former tools often suffered from slow or no convergence. Automatic derivative computation is therefore a significant achievement.

However:

- derivatives proved to be destructive at bipolar NQS modeling
- derivatives are non-physical and superfluos in some cases like smoothing functions
- “weak” derivatives with negligible impact pose computational burden w/o benefits

Automatic derivative generation can not be influenced from the coding domain. Methods for their suppression have not been proposed so far.
Possibilities of derivative removal are apparently restricted to
- making them zero
- making them the difference of two equal quantities

The first option can be achieved by “discretizing” the selected dependent variable [2].

Function $f(x)$ is discretized within the breakpoints implying zero derivative in that section.
From Verilog-AMS v.2.0 on, the standard functions have been extended by the `floor()` and `ceil()` functions corresponding to their C language counterparts. For the strong correlation between the two it is enough to focus only on `floor(x)`.

In C its returned value is defined as a double which is the largest integral value that is not greater than the argument.

For a proper precision, the variable is transformed to the range of the reciprocal computation accuracy which is approximately `1e15` at double precision computations. Select a model parameter or some other model constant `var0` “not too far” from the variable value. The transformation

\[
vnorm = 1e\cdot 15 \cdot \text{var0} \quad \text{var} = vnorm \cdot floor(\text{var}/vnorm)
\]

makes `var` match the value of `var` with zero derivatives.
Particularly, when \( \text{var0} \) is difficult to estimate a very small number can be used for normalization

\[
\text{var}_\_ = 1e^{-30} \cdot \text{floor}(1e30 \text{var})
\]

Extremely 30 could be increased to 307 which is the range limit of doubles in a 32 bit machine.

The transformation can be used inline or as a macro.

**Functional format**

```verbatim
analog function real dstrip;
    input x,vnorm;
    real x,vnorm;
    dstrip =
    vnorm*floor(x/vnorm);
endfunction
```

**Function call**

```verbatim
x_ = dstrip(x,vnorm);
```

If it were made a system function compilers could recognize it for not to include \( x_ \) in the derivative dependence tree
Derivatives can also be removed by real => integer conversion

```verilog
integer ivar;
.............
ivar = var/vnorm; //integer to real conversion
var_ = vnorm*ivar;
```

ADS delivers the following warning message on this construct:

“Warning: Variable 'ivar' isn't real-valued; derivatives will be ignored.”

Care must be taken at using this conversion. Range of integers in Verilog-A is \([-2147483648... 2147483648]\) so a larger normalization constant must be selected than before. This leads to some loss of accuracy but it still can be less than \(1e-5...1e-6\). It is generally accepted yet e.g. for a model-to-model deviation.
Integer conversion is incorrectly implemented in open simulators Qucs and QucsStudio. The integer inherits the derivatives of the real it is converted from. The problem could be fixed for \(0 \leq var < 1\) as

\[
i_{\text{var}} = \text{var};
\]
\[
\text{var}_\_ = \text{var} - i_{\text{var}};
\]

The conversion results in \(i_{\text{var}} = 0\). Thus the output returns the original argument with derivatives that are the difference of equal numbers due to the incorrect transparency of the type conversion. Obviously, this is more expensive than the former variants since the derivatives are computed instead of assigning them as zero. Having been originated from a bug, this example is given just for information.
Case study #1: vertical NQS

Derivative Management in Verilogo-A

The NQS phenomenon is modeled by the response of a 2nd order Bessel filter adjunct network. Using $k=1$ provides correct result in VBIC adopting a constant delay time TD. In Hicum TD is bias dependent and the same setup outputs severely distorted result.

It has been proved in [3] theoretically and by device simulations that the instance value of TD needs to be adopted for a correct NQS result.
Case study #2: vertical NQS, cont’d

Derivative Management in Verilog-A

Formerly [4] proposed a scaling factor

\[ k = t_0 / tf \]

for achieving the equivalent of an instance TD in the adjunct network. The move relocates TD from the dynamic to the static component of the system matrix where its derivatives do not create transcapacitance elements.

The single-minded \( k=1 \) (unscaled) implementation implies a huge error in the NQS phase. The scaled [4], and derivative stripped versions by the floor() function or by real=>integer conversion give the same correct result.
Case study #3, bias dependent PTF

Derivative Management in Verilog-A

The NQS parameter PTF was found to be bias dependent by device simulations in [5].

Polynomial approximation

TICCR is quite robust against model implementation conditions
The thermal voltage $VT$ occurs very often in the model equations:

- physical definitions e.g. in diode equations
- smoothing functions

Stars represent the case when $VT$ derivatives are stripped ($VT_{ds}$) only in the smoothing functions. Circles stand for the option when all occurrences of $VT$ are substituted by $VT_{ds}$.

The latter case convergence problems arise (missing points from the FT curve).
Case study #5, weak derivatives in rbi

Derivative Management in Verilog-A

rbi has the most pronounced effect on the unilateral power gain U

if(rbi0_t > 0.0) begin : HICRBI
  real Qz_nom,f_QR,ETA,Qz0,fQz;
  // Consideration of conductivity modulation
  // To avoid convergence problem hyperbolic smoothing u:
  f_QR = (1+fdqr0) * qp0_t;
  Qz0 = Qjei+Qci+Qf;
  Qz_nom = 1+Qz0/f_QR;
  fQz = 0.5*(Qz_nom+sqrt(Qz_nom*Qz_nom+0.01));
  vnorm = rbi0*1e-15;
  rbi = vnorm*floor(rbi0_t/fQz/vnorm);
  // Consideration of emitter current crowding
  if( ibei > 0.0) begin
    vnorm = rbi0*ibeis*1e-15;
    ETA = vnorm*floor(rbi*ibeis*fgeo/VT/vnorm);
    if(ETA < 1.0e-6) begin
      rbi = rbi*(1.0-0.5*ETA);
    end else begin
      rbi = rbi*ln(1.0+ETA)/ETA;
    end
  end
  // Consideration of peripheral charge
  if(Qf > 0.0) begin
    rbi = 1e-15*floor(1e15*rbi*(Qjei+Qf*fqi)/(Qjei+Qf));
  end
  end else begin
    rbi = 0.0;
  end

derivative strip in rbi (rb-ds) has no effect on the results
Case study #6, weak derivatives in crbi

Derivative Management in Veriloga

The original paper on LATNQS [6] demonstrated agreement with measurements by manual computations without derivatives. It is expected that the HICUM implementation neither exhibits derivative dependence.

```
//High-frequency emitter current crowding (lateral NQS
Cdei_ = 1e-30*floor(1e30*T_f0*itf/VT);
Cdci_ = 1e-30*floor(1e30*tr*itr/VT);
Cjei_ = 1e-30*floor(1e30*Cjei);
Cjci_ = 1e-30*floor(1e30*Cjci);
Crbi  = fcrbi*(Cjei_+Cjci_+Cdei_+Cdci_);
qrbi  = Crbi*V(br_bpbi_v);
```

Derivative strip in the lateral NQS section has no effect on the results. The Crbi components have been stripped of derivatives individually.
Summary

Derivative Management in Verilog-A

• methods have been proposed for making selected derivatives zero in Verilog-A

• use of the `floor()` function is preferred to the methods based on real=> integer conversion

• adopting the scheme to the Bessel type vertical NQS implementation was proved to give equivalent results with the former network scaling

• derivatives of VT for transition functions can be safely omitted

• complicated derivatives of rbi were shown not to affect U (and fmax) and can be removed. Derivatives of Crbi neither influence the results

• if the functional form was a standard Verilog-A function compilers could recognize the variables not to be put in the derivative dependence tree. Even in this state runtimes decrease due to sparing computation of selected derivatives.
References

Derivative Management in Verilog-A


Thank you!

Please visit our website
www.ams.com