Summary of Changes from BSIM-BULK107.0.0 to BSIM-BULK107.1.0:

BSIM Group, IIT Kanpur, UC Berkeley

Ayushi Sharma (sayushi@iitk.ac.in), Ravi Goel (gravi@iitk.ac.in), Chetan K. Dabhi (dabhi@berkeley.edu)
Girish Pahwa (pahwa@berkeley.edu)

A. Summary of Enhancements:

1. **2019enh7:** Abulk Implementation in I-V.
2. **2019enh8:** Flexibility in tuning Cgg.
3. **2019enh9:** Enhancing the fitting flexibility of Cgd at high Vds.
4. **2019enh10:** Add the flicker noise for both Rd and Rs.
5. **2020enh2:** Modeling of drift region with back bias in HV extension.
6. **2020enh3:** New model parameters for EDGEFET.
7. **2021enh1:** Flat band voltage scaling parameters for I-V.
8. **2021enh2:** Improvement to the implementation of body-drain diode.
9. **2021enh3:** Gate bias dependence in the depletion model of drift region.
10. **2021enh4:** Improved modeling of Id-Vd at high Vg and Vd for HV model.
11. **2022enh1:** Add QA test to check the missing parameters

B. Summary of bug-fixes:

12. **2020bug5**: ‘gamCV’ instead of ‘gam’ in VdsatCV equation.
13. **2020bug6:** Manual update (Temperature dependent parameter list update in BSIM-BULK 107 manual).
14. **2020bug7:** Change default values of model parameters related to CVMOD = 1
15. **2020bug8:** Minimize division by Rdrain and Rsource.
16. **2020bug9**: Issues raised by VAMPyRE
    - $error() preferred over $finish( )
    - Formatting issues (extra tab, space, or Incorrect indent)
    - Unused Parameters.

17. **2020bug10:** Improved flexibility in asymmetry mode.
18. **2020bug11:** IDEFF and ISEFF.
19. **2020bug12:** Convergence issue due to DVT2EDGE.
20. **2021bug1:** Clamping of gate electrode resistance in RGATEMOD=2.
21. **2021bug3:** Limit XRCRG1 & XRCRG2 parameters.
22. **2021bug4:** Revert IIMOD changes to BSIM-BULK107.0.0.
23. **2021bug5:** Missing multiplication factor in 2nd instance of T5 in Rdss=0.
24. **2021bug6:** Removing redundancy in the calculation of Rdss.
25. **2021bug7:** Improved smoothing of drift resistance in HV module.
26. **2021bug8:** Correct the description of RBPD parameter.
27. **2021bug9:** Issues raised by VAMPyRE
    - Warning on bias-dependent condition if (Rdss==0).
    - Warning on bias-dependent condition if (PCLM_a !=0).
28. **2021bug10:** Typo in the comment of Drain junction current (IBD).
29. **2021bug11:** Node order correction in the body-drain diode for HV model.
30. **2021bug12:** Removing extra contribution of Ibd in the HV model.
31. **2021bug13:** Drain-body junction current splitting in RBODYHVMOD.
32. **2021bug14:** Protection for junction capacitance grading coefficient (MJX) against 0/0 form.
33. **2021bug15:** Smoothing function for psiph.
34. **2021bug16:** Id-Vg shows spikes in transient simulation.
35. **2021bug17:** Default values of body-bias dependent parameters in HVMOD
36. **2021bug18**: Unit correction of source and drain resistances.
37. **2021bug19:** Operating point should include charge Qbdj_ext from external diode.

38. **2021bug20:** Changing lln to ln for junction capacitance calculation in VA code.
39. **2021bug21:** Clamping Nsat for non-saturation effect.
40. **2021bug22:** Correction in JunCap macro
41. **2021bug23:** VAMPyRE error in operating-point variable 'IDRIFTSATD'
42. **2021bug24:** Limit model parameter MDRIFT
43. **2022bug1:** Allow minr = 0
44. **2022bug2:** Model parameter ABULK added for backward compatibility
45. **2022bug3:** keyLetter x is added for Smartspice/ELDO simulator in QA pearl script
46. **2022bug4:** Encountered divide by zero error with node (g,b).
47. **2022bug5:** Typo Correction in gspr.
48. **2022bug6:** Divide by zero error when parameters DMCG / (DMCG+DMCI) are zero
49. **2022bug7:** Limit the layout-dependent parasitic model parameters
50. **2022bug8:** Code cleaning

Description of Enhancements:

1. 2019enh7: Abulk Implementation in I-V.

- To get the fitting flexibility in the saturation region, bias dependent Abulk term is added to Vdseff equation, which allows tuning flexibility in Id – Vd in the saturation region of operation.
- In BSIM-BULK107.0.0. the equation of Vdseff is given below.

$$\boldsymbol{Vdseff = Vds} * \left(\mathbf{1} + \left(\frac{\boldsymbol{Vds}}{(\boldsymbol{Vdssat})}\right)^{\frac{\mathbf{1}}{\boldsymbol{DELTA}}}\right)^{-\boldsymbol{DELTA}}$$

- Adding Abulk in Vdseff :

$$\boldsymbol{Vdseff = Vds} * \left(\mathbf{1} + \left(\frac{\boldsymbol{Vds}}{\left(\frac{\boldsymbol{Vdssat}}{\boldsymbol{Abulk}}\right)}\right)^{\frac{\mathbf{1}}{\boldsymbol{DELTA}}}\right)^{-\boldsymbol{DELTA}}$$

Where Abulk is given as :

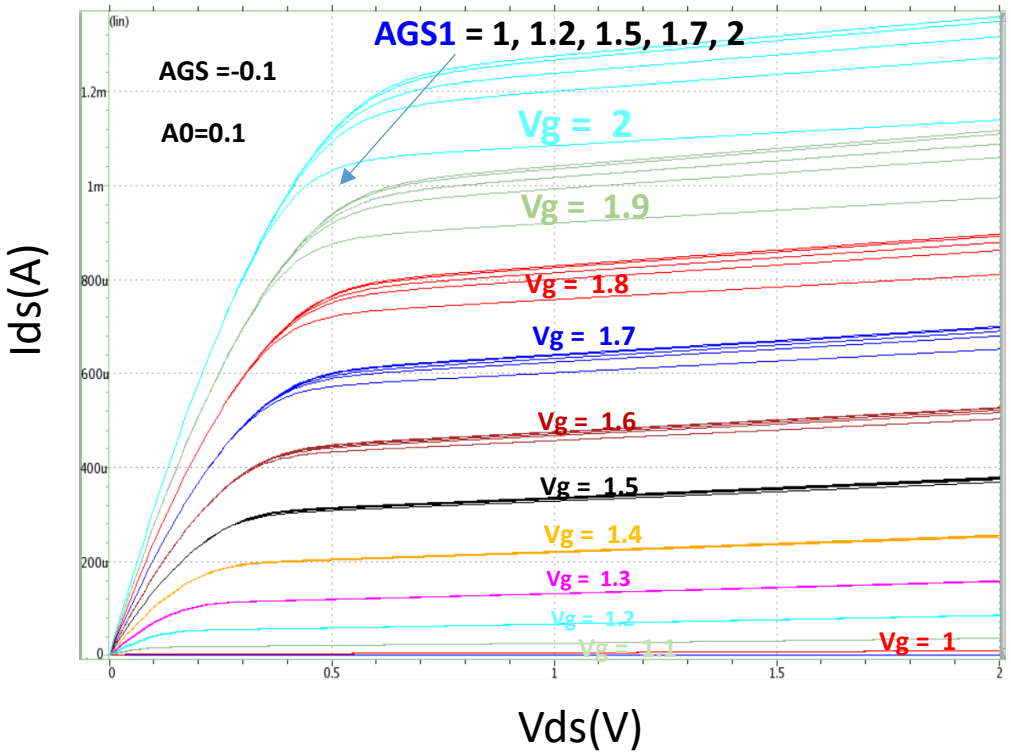
$$Abulk = 1 + \frac{A_0 * T1 - AGS * q_s^{\,AGS1} * V_T * T1}{1 + KETA * V_{bsx}}$$

Where, $T1 = \dfrac{L_{eff}}{L_{eff} + \sqrt{X_j} * XDEP}$

- $\boldsymbol{A_0}$, $\boldsymbol{AGS}$, $\boldsymbol{KETA}$, $\boldsymbol{AGS1}$ are the model parameters used for tunning.
- VA code with the updated implementation is given below:

```
T1    = Leff / (Leff + sqrt(XJ_i * Xdep));
Abulk = 1 + (A0 * T1 - AGS * T1 * pow(qs,AGS1) * nVt) / (1 + KETA * Vbsx);
`Smooth(Abulk, 0.1, 0.0005, Abulk)
`Smooth(Vdsat - Vs, 0.0, 1.0e-3, Vdssat)
Vdssat    = Vdssat / Abulk;
T7      = pow(Vds / Vdssat , 1.0 / DELTA_t);
T8      = pow(1.0 + T7, -DELTA_t);
Vdseff  = Vds * T8;
vdeff   = (Vdseff + Vs) * inv_nVt;
`BSIM_q(psip, phib_n, vdeff, gam, qdeff)
```

- Ids – Vds with the updated implementation is shown below:



2. 2019enh8: Flexibility in tuning Cgg.

- In BSIM-BULK107.0.0, ABULK was model parameter with default value 1, to allow fitting flexibility in strong inversion region of operation.

- BSIM-BULK107.1.0 allows more tuning flexibility in strong inversion, by replacing “ABULK” with a bias dependent “AbulkCV” equation, given below:

$$Vdseff = Vds * \left(1 + \left(\frac{Vds}{Vdssatcv \cdot Abulkcv}\right)^{\frac{1}{DELTA}}\right)^{-DELTA}$$

Bias dependent equation

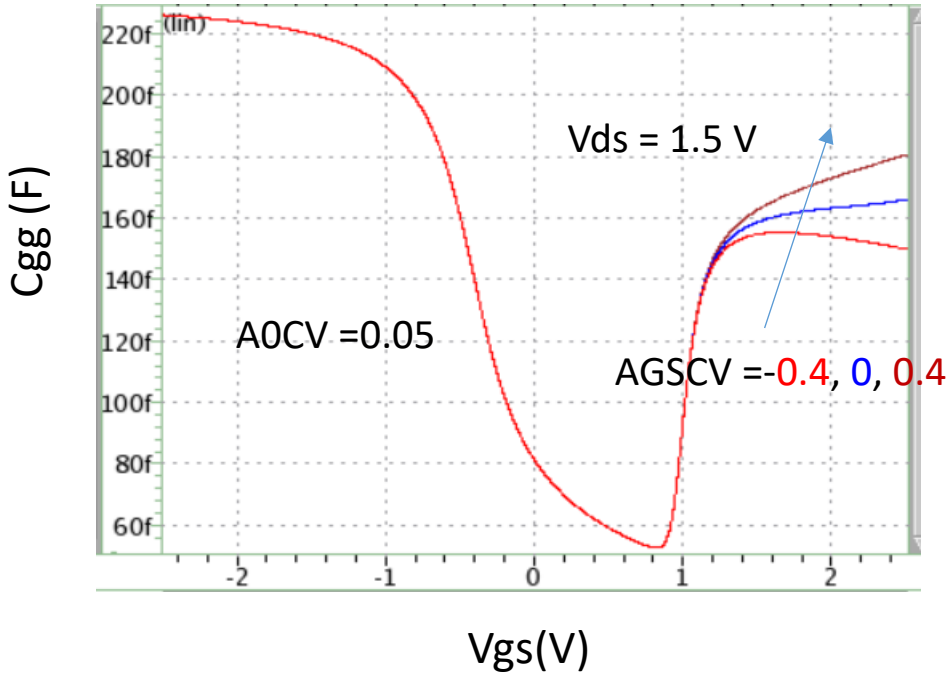
$$AbulkCV = 1 + \frac{A0CV * T1 - AGSCV * q_s * V_T * T1}{1 + KETACV * V_{bsx}}$$

BSIMBULK107.1.0

```
`Smooth(VdsatCV - Vs, 0.0, 1e-3, VdssatCV)
T1     = Leff / (Leff + sqrt(XJ_i * Xdep));
AbulkCV = 1 + (A0CV * T1 - AGSCV * T1 * qs * nVt) / (1.0 + KETACV * Vbsx);
`Smooth(AbulkCV, 0.1, 0.0005, AbulkCV)
VdssatCV     = VdssatCV / AbulkCV;
T7     = pow(Vds / VdssatCV , 1.0 / DELTA_t);
T8     = pow(1.0 + T7, -DELTA_t);
Vdseff = Vds * T8;
vdeff  = (Vdseff + Vs) * inv_Vt;
`BSIM_q(psip, phibCV, vdeff, gamCV, qdeff)
```

$$Q_i = \frac{n_q}{MDL}\left[(q_s + q_{deff}) + \frac{1}{3}(q_s - q_{deff})^2 \frac{AbulkCV * DVSAT}{MDL \cdot (1 + q_s + q_{deff})} + 2n_q(MDL - 1)q_{deff}\right] \quad (9.46)$$

- Cgg – Vgs with the updated implementation shows good tuning flexibility at high Vgs.



3. 2019enh9: Enhancing the fitting flexibility of Cgd at high Vds.

- In BSIM-BULK107.0.0, simulated Cgd cannot match the measured data at Vds =0 and at high Vds.
- BSIM-BULK107.1.0 improves the fitting flexibility to tune Cgd by improving the model for overlap capacitance.
- In this release Vgd,overlap in the below equation is replaced with *T6*.

At the drain side:

$$\frac{Q_{gd,ov}}{NF \cdot W_{effCV}} = CGDO \cdot V_{gd} + CGDL \cdot \left[V_{gd} - V_{fbsd} - V_{gd,overlap} - \frac{CKAPPAD}{2} \left(\sqrt{1 - \frac{V_{gd,overlap}}{CKAPPAD}} - 1 \right) \right]$$

T6

$$T6 = \frac{Vgd,overlap}{\left[1 + \left(\frac{-Vgd,overlap}{CKAPPAD1}\right)^{CKAPPAD2}\right]^{1/CKAPPAD2}}$$

At the Source side:

$$\frac{Q_{gs,ov}}{NF \cdot W_{effCV}} = CGSO \cdot V_{gs} + CGSL \cdot \left[V_{gs} - V_{fbsd} - V_{gs,overlap} - \frac{CKAPPAS}{2} \left(\sqrt{1 - \frac{4 V_{gs,overlap}}{CKAPPAS}} - 1 \right) \right]$$

T6

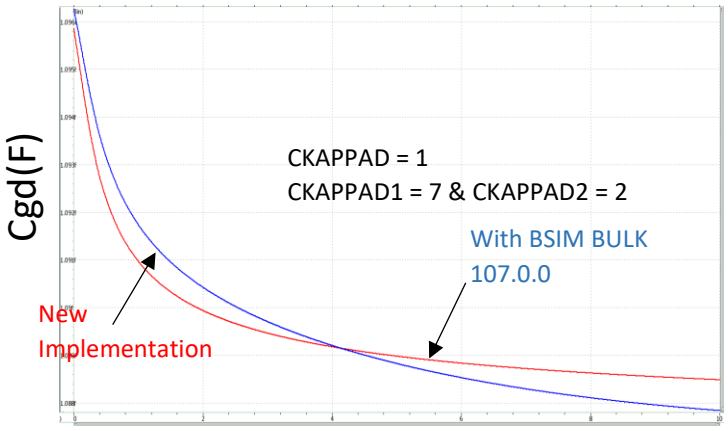
$$T6 = \frac{Vgs,overlap}{\left[1 + \left(\frac{-Vgs,overlap}{\boldsymbol{CKAPPAS1}}\right)^{\boldsymbol{CKAPPAS2}}\right]^{1/\boldsymbol{CKAPPAS2}}}$$

- Parameters **CKAPPAD1, CKAPPAD2, CKAPPAS1, CKAPPAS2** cannot be Zero or Infinite, otherwise it may cause convergence problem.

BSIMBULK107.1.0

```
if (COVMOD == 0) begin
    Qovs = -Wact * NF * Cgsof * Vgs_ov_noswap;
    Qovd = -Wact * NF * Cgdof * Vgd_ov_noswap;
end else begin
    T0    = sqrt((Vgs_ov_noswap - Vfbsdr + `DELTA_1) * (Vgs_ov_noswap - Vfbsdr + `DELTA_1) + 4.0 * `DELTA_1);
    Vgsov = 0.5 * (Vgs_ov_noswap - Vfbsdr + `DELTA_1 - T0);
    T6    = Vgsov / pow(1.0 + pow((-Vgsov / CKAPPAS1), CKAPPAS2), 1.0 / CKAPPAS2);
    T1    = sqrt(1.0 - 4.0 * T6 / CKAPPAS_i);
    Qovs  = -Wact * NF * (Cgsof * Vgs_ov_noswap + CGSL_i * (Vgs_ov_noswap - Vfbsdr - Vgsov - 0.5 * CKAPPAS_i * (-1.0 + T1)));
    T0    = sqrt((Vgd_ov_noswap - Vfbsdr + `DELTA_1) * (Vgd_ov_noswap - Vfbsdr + `DELTA_1) + 4.0 * `DELTA_1);
    Vgdov = 0.5 * (Vgd_ov_noswap - Vfbsdr + `DELTA_1 - T0);
    T6    = Vgdov / pow(1.0 + pow((-Vgdov / CKAPPAD1), CKAPPAD2), 1.0 / CKAPPAD2);
    T2    = sqrt(1.0 - 4.0 * T6 / CKAPPAD_i);
    Qovd  = -Wact * NF * (Cgdof * Vgd_ov_noswap + CGDL_i * (Vgd_ov_noswap - Vfbsdr - Vgdov - 0.5 * CKAPPAD_i * (-1.0 + T2)));
end
Qovb = -devsign * NF * Lact * CGBO * V(gm, bi);
Qovg = -(Qovs + Qovd + Qovb);
```

- Cgd vs Vds plot with and without new implementation.



4. 2019enh10: Add the flicker noise for both Rd and Rs.

- Flicker noise due to external drain and source resistance is added in BSIM-BULK107.1.0.
- Flicker noise due to External resistances is as follows:

$$s_{id}{}^2 = \frac{KFN * W * \left(\frac{I_d}{W}\right)^{AFN}}{f^{BFN}}$$

Where **AFNS**, **BFNS**, **KFNS and AFND**, **BFND,** and **KFND** are the parameters for the Source and Drain resistance respectively.

- VA code implementation in BSIM-BULK107.1.0 is given below:

```
//External Resistance flicker noise Model
    if (FNOIMOD == 0 && RDSMOD == 1) begin
      if(Rsource > 1.0e-3) begin
       I(s,si) <+ flicker_noise(KFNS * W * pow((ids / W), AFNS), BFNS ,"flicker");
       end
       if(Rdrain > 1.0e-3) begin
       I(d,di) <+ flicker_noise(KFND * W * pow((ids / W), AFND), BFND ,"flicker");
   end
 end
```

- Parameters **AFN** & **BFN** cannot be zero or negative, and **KFN** should be positive.

5. 2020enh2: Modeling of Drift region with back bias in HV extension.

- The body-bias dependency of the drift region as implemented in BSIM-BULK107.0.0. is not sufficient to obtain a good fit for the HV devices.
- In BSIM-BULK107.1.0 body bias dependency is added by the model equation as given below to obtain better fitting flexibility.

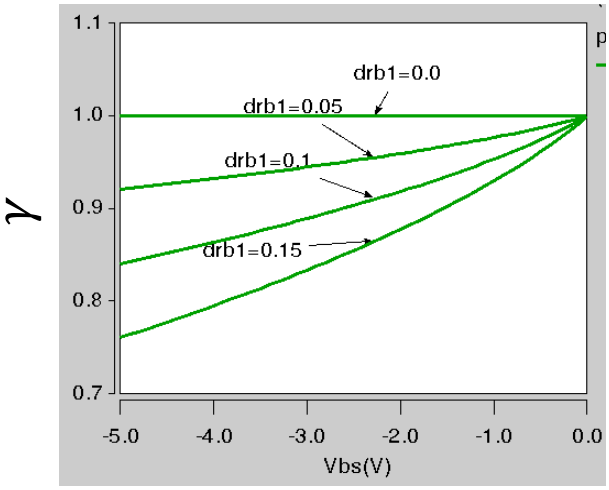
$$\gamma = 1 - DRB1 * (\sqrt{1 + V_{sb}/V_{bi}} - 1) + DRB2 * V_{bs}$$

- Parameter **DRB1** is used for nonlinear dependence and **DRB2** is used for the linear dependence.
- VA code with the updated implementation is shown below:

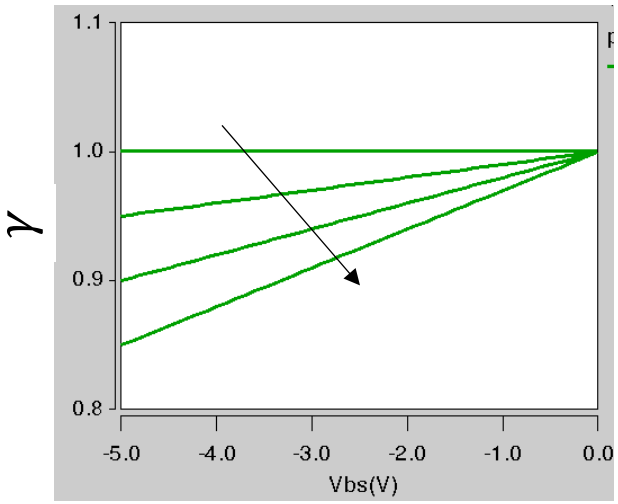
```
idrift_sat_d = T11 * NDRIFTD * T9;
T2 = 1 - Vbsx / vbi_drift;
`Smooth(T2, 0.0, 0.05, T2)
T6 = (1 - DRB1 * ( sqrt(T2) - 1) + DRB2 * Vbsx) ;
idrift_sat_d = T6 * idrift_sat_d;
```

$$\gamma = 1 - DRB1 * (\sqrt{1 + V_{sb}/V_{bi}} - 1) + DRB2 * V_{bs}$$

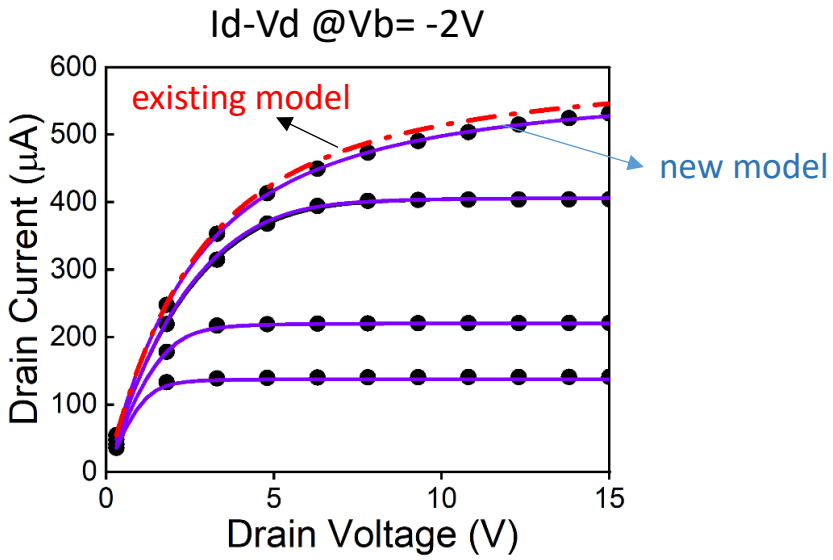
- DRB1 variation (nonlinear effect) at fixed Vgs & Vds.



- DRB2 variation (linear effect) at fixed Vgs & Vds.



- Id – Vd with and without new implementation.



6. 2020enh3: New Model Parameters for EDGEFET.

- Well proximity effect is added in EDGEFET module in BSIM-BULK107.1.0.
- **KVTH0EDGEWE**, **LKVTH0EDGEWE, WKVTH0EDGEWE, PKVTH0EDGWE** are new model parameters added in EDGEFET module.
- **K2EDGEWE**, **LK2EDGEWE, WK2EDGEWE, PK2EDGEWE** are new model parameters added in EDGEFET module.

1. **KVTH0EDGEWE_i = KVTH0EDGEWE** + BIN_L * **LKVTH0EDGEWE** + BIN_W * **WKVTH0EDGEWE** + BIN_WL * **PKVTH0EDGEWE;**
2. **K2EDGEWE_i = K2EDGEWE** + BIN_L * **LK2EDGEWE** + BIN_W * **WK2EDGEWE**+ BIN_WL * **PK2EDGEWE;**

vth0_well_edge = KVTH0EDGEWE_i * (local_sca + WEB * local_scb + WEC * local_scc)

k2_well_edge = K2EDGEWE_i * (local_sca + WEB * local_scb + WEC * local_scc)

Vth_shift = dvth_dibl - dvth_temp + dvth_sce + DVTEDGE + vth0_stress_EDGE - (K2EDGE_i + k2_well_edge) * Vbsx + vth0_well_edge;

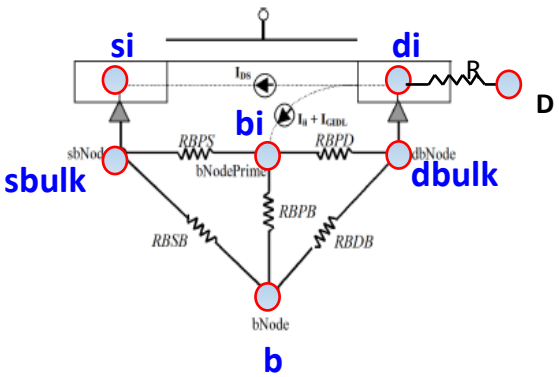
7. 2021enh1: Flat band voltage scaling parameters for I-V.

- Flat band voltage scaling parameters are added in this release.

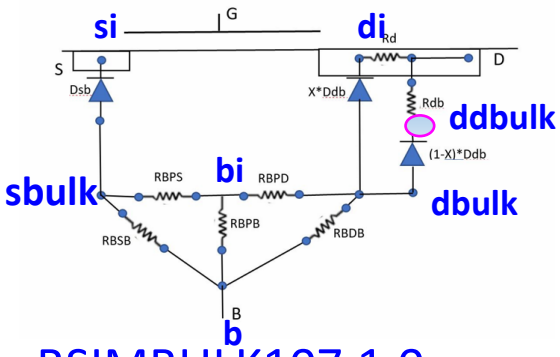
```
`MPRnb( VFBL      ,0.0     ,"m"    ,"Length dependence coefficient of VFBCV" )
`MPRoz( VFBLEXP   ,1.0     ,""     ,"Length dependence exponent coefficient of VFBCV" )
`MPRnb( VFBW      ,0.0     ,"m"    ,"Width dependence coefficient of VFBCV" )
`MPRoz( VFBWEXP   ,1.0     ,""     ,"Width dependence exponent coefficient of VFBCV" )
`MPRnb( VFBWL     ,0.0     ,"m^2   ,"Width-length dependence coefficient of VFBCV" )
`MPRoz( VFBWLEXP  ,1.0     ,""     ,"Width-length dependence coefficient of VFBCV" )
```

8. 2021enh2: Improvement to the implementation of body-drain diode.

- In BSIM-BULK107.0.0, FB series resistance is significantly over-estimated since part of the DB diode does not pass through the drain extension resistance Rd.
- In both .tran and .ac simulations, drain signal delay is too pessimistic.
- Body network in BSIM-BULK107.0.0.



- Body network in BSIM-BULK107.1.0.



BSIMBULK107.1.0

Resistor Rdb

```
if (RBODYMOD != 0 && RBODYHVMOD == 1) begin
    I(ddbulk, d) <+ V(ddbulk, d) * Grdb;
    I(ddbulk, d) <+ white_noise(Nt * Grdb, "rdb");
end else begin
    V(d, ddbulk)  <+ 0.0;
end
```

Diode Partitioning

```
// Diode currents and capacitances HV
if (RBODYMOD != 0 && RBODYHVMOD == 1) begin
    I(dbulk, di) <+ devsign * XPART * Ibd + V(dbulk, di) * gmin;
    I(ddbulk, dbulk) <+ devsign * (1 - XPART) * Ibd + V(ddbulk, dbulk) * gmin;
    I(dbulk, di) <+ devsign * ddt(XPART * Qbdj);
    I(dbulk, ddbulk) <+ devsign * ddt((1 - XPART) * Qbdj);
end else begin
    V(d, ddbulk)  <+ 0.0;
end
```

9. 2021enh3: Gate bias dependence in the depletion model of drift region.

- In BSIM-BULK107.0.0. there is no gate bias dependency in HV model.
- BSIM-BULK107.1.0 includes gate bias dependency in HV model in the Idrift as given below:

T1 = qs - PTWGHV1 ;

`Smooth(T1, 0.1, 2, T1)

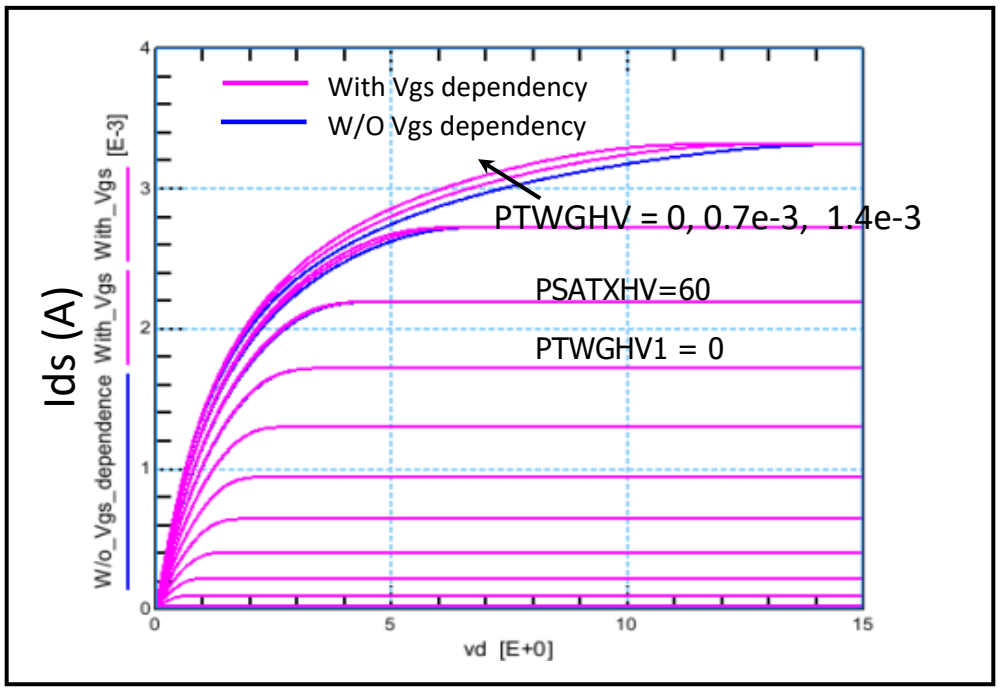
T2 = 10.0 * **PSATXHV** * T1 / (10.0 * **PSATXHV** + T1);

$$I_{driftsat} \propto (1 + \mathbf{PTWGHV} * T2)$$

- VA code with the updated implementation is given below:

```
if (RDSMOD == 1 && HVMOD == 1) begin
    T4  = 1 + PDRWB * Vbsx;
    T0  = ids ;
    T1  =  qs - PTWGHV1 ;
    `Smooth(T1, 0.1, 2, T1)
    T2  = 10.0 * PSATXHV * T1 / (10.0 * PSATXHV + T1);
    VDRIFTeff = VDRIFT_t * (1.0 + PTWGHV * T2);
    T11 = NF * Weff * `q  * VDRIFTeff ;
```

- **PSATXHV** and **PTWGHV1** parameters allow more tuning flexibility.
- Below shows Ids-Vds curve flexibility for the BSIM-BULK 107.1.0.



10. 2021enh4: Improved Modeling of Id-Vd at high Vg and Vd for HV model.

- In BSIM-BULK107.0.0. Idrift equation is modelled as:
- To enhance the fitting flexibility at high vg and high vd Idrift (which is called as “Deep Junction DIBL”) equation is updated in BSIM-BULK107.1.0.

$$I_{drift,sat} = q * NDRIFT * W * VDRIFT$$

$$I_{drift,sat} = q * NDRIFT * W * VDRIFT * T0$$

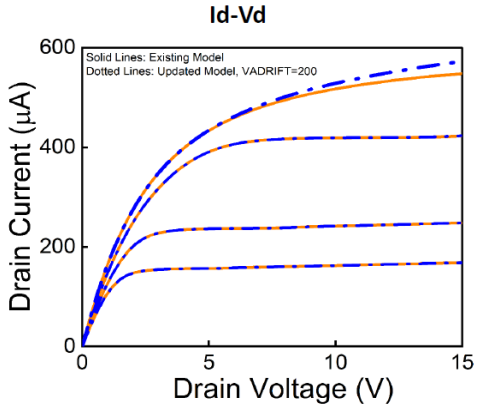
where,

$$T0 = 1 + GADRIFT * (V(d, di) - RDVDS)$$

- **GADRIDT** act as a model selector.

```
if (RDLCW != 0) begin
if (GADRIFT == 0) begin
   T9 = 1.0;
    end else begin
       `Smooth(devsign * V(d,di) - RDVDS, 0, 0.5, T10)
        T9 = 1 + T10 * GADRIFT;
end
    idrift_sat_d = T11 * NDRIFTD * T9;
```

- Below shows the Id-Vd flexibility at high vg and high vd.



11. 2022enh1: Add QA test to check the missing parameters

- A test with all the default parameters specified is added to QA of BSIM-BULK107.1.0. The purpose of this test is to catch the accidentally deleted parameters during the model development. For nmos this test is:

BSIMBULK107.1.0

```
//Backward Compatibility Check on Parameters
test                    044_DC_PARAM_Check
temperature             27
biases                  V(s)=0 V(b)=0 V(d)=1
biasSweep               V(g)=0, 1, 0.5
outputs                 I(d) I(g) I(s) I(b)
instanceParameters      RBODYMOD=0 RDB=50.0 RGEOMOD=0 IDS0MULT=1.0 NF=1 GEOMOD=0
instanceParameters      PS=0.0 RBPB=50.0 RBPD=50.0 DTEMP=0.0 SCA=0.0 SCC=0.0
instanceParameters      SCB=0.0 RBPS=50.0 PD=0.0 RBDB=50.0 NGCON=1 AD=0.0
instanceParameters      L=1e-05 DELVTO=0.0 SC=0.0 MINZ=0 VFBSDOFF=0.0 W=1e-05
instanceParameters      SB=0.0 SSLMOD=0 MULU0=1.0 RGATEMOD=0 RBSB=50.0 AS=0.0
instanceParameters      NRD=1.0 EDGEFET=0 NRS=1.0 XGW=0.0 SA=0.0 SD=0.0
modelParameters         parameters/nmosPARAM_Check
```

The file nmosPARAM_Check contains all the BSIM-BULK parameters with their default values. A similar test is added for the pmos.

Description of bug-fixes:

12. 2020bug5: ‘gamCV’ instead of ‘gam’ in VdsatCV equation.

BSIMBULK107.0.0

```
gamCV = sqrt(2.0 * `q * epssi * NDEPCV_i * inv_Vt) / Cox;
inv_gam = 1.0 / gamCV;
vdsatcv = psip - 2.0 * phibCV - (2.0 * qdsat + lln((qdsat * 2.0 * nq * inv_gam)
    * ((qdsat * 2.0 * nq * inv_gam) + (gam / (nq - 1.0)))));
```

BSIMBULK107.1.0

```
gamCV      = sqrt(2.0 * `q * epssi * NDEPCV_i * inv_Vt) / Cox;
inv_gam  = 1.0 / gamCV;
vdsatcv     = psip - 2.0 * phibCV - (2.0 * qdsat + lln((qdsat * 2.0 * nq * inv_gam)
    * ((qdsat * 2.0 * nq * inv_gam) + (gamCV / (nq - 1.0)))));
```

13. 2020bug6: Manual Update (Temperature dependent parameter list update in BSIM-BULK 107 manual).

- Temperature dependent parameter list in BSIM-BULK107.0.0 manual was not complete, e.g: EU1, IGT, TPB, TPBSW.

EU1 (b)	Temperature coefficient for EU	0.0

- BSIM-BULK107.1.0 manual is updated accordingly.

14. 2020bug7: Change default values of model parameters related to CVMOD=1

- In BSIM-BULK107.0.0. CV model parameters related to CVMOD =1 is not consistent with I-V model parameters.
- CV model parameters related to CVMOD =1 is made consistent with I-V model parameters.
- Full parameter list is given below:

	Previous implementation (BSIM-BULK107.0.0)	Updated implementation (BSIM-BULK107.1.0)
NDEPCV	,1e24	,NDEP
NDEPCVL1	,0.0	,NDEPL1
NDEPCVLEXP1	,1.0	,NDEPLEXP1
NDEPCVL2	,0.0	,NDEPL2
NDEPCVLEXP2	,2.0	,NDEPLEXP2
NDEPCVW	,0.0	,NDEPW
NDEPCVWEXP	,1.0	,NDEPWEXP
NDEPCVWL	,0.0	,NDEPWL
NDEPCVWLEXP	,1.0	,NDEPWLEXP
LNDEPCV	,0.0	,LNDEP
WNDEPCV	,0.0	,WNDEP
PNDEPCV	,0.0	,PNDEP
VFBCV	,-0.5	,VFB
LVFBCV	,0.0	,LVFB
WVFBCV	,0.0	,WVFB
PVFBCV	,0.0	,PVFB
VSATCV	,1e5	,VSAT
LVSATCV	,0.0	,LVSAT
WVSATCV	,0.0	,WVSAT
PVSATCV	,0.0	,PVSAT
VSATCVL	,0.0	,VSATL
VSATCVLEXP	,1.0	,VSATLEXP

	Previous implementation (BSIM-BULK107.0.0)	Updated implementation (BSIM-BULK107.1.0)
VSATCVW	,0.0	,VSATW
VSATCVWEXP	,1.0	,VSATWEXP
VSATCVWL	,0.0	,VSATWL
VSATCVWLEXP	,1.0	,VSATWLEXP
VFBCVL	,0.0	,VFBL
VFBCVLEXP	,1.0	,VFBLEXP
VFBCVW	,0.0	,VFBW
VFBCVWEXP	,1.0	,VFBWEXP
VFBCVWL	,0.0	,VFBWL
VFBCVWLEXP	,1.0	,VFBWLEXP
PCLMCV	,PCLM	,PCLM
PCLMCVL	,PCLML	,PCLML
PCLMCVLEXP	,PCLMLEXP	,PCLMLEXP
LPCLMCV	,LPCLM	,LPCLM
WPCLMCV	,WPCLM	,WPCLM
PPCLMCV	,PPCLM	,PPCLM

15. 2020bug8: Minimize division by Rdrain and Rsource.

- 1/ Rdrain and 1/ Rsource is replaced by gdpr and gspr in this release.

```
if ((SHMOD != 0) && (RTH0 > 0.0)) begin
    Pdiss = devsign * sigvds * ids * V(di, si);
    if (RDSMOD !=2 && RDrainGeo >0) begin
        Pdiss = Pdiss + V(d, di) * V(d, di) * gdpr;
    end
    if (RDSMOD !=2 && RSourceGeo >0) begin
        Pdiss = Pdiss + V(s, si) * V(s, si) * gspr;
    end
    Pwr(t) <+ delTemp1 * gth + ddt(delTemp1 * cth) - Pdiss;
end else begin
    Temp(t) <+ 0.0;
end
```

16.2020bug9: Issues raised by VAMPyRE

- During the VAMPyRE run, BSIM-BULK107.0.0. shows the following error:

1. $error() preferred over $finish( ).

```
if (K1_i < 0.0) begin
  $strobe("Fatal: K1_i = %e is negative.", K1_i);
  $finish(0);
end
```

- BSIM-BULK107.1.0 resolves this issue by replacing $finish().

```
if (K1_i < 0.0) begin
  $error("Fatal: K1_i = %e is negative.", K1_i);
end
```

2. Formatting issues (extra tab, space, or Incorrect indent)
3. Unused Parameters are removed in BSIM-BULK107.1.0, the list is given below:

   ESAUB, PDIBLCLR, VFBDRIFT, PTWGLR, PDIBLCLEXPR, PTWGLEXPR, CDSCDLR.

17.2020bug10: Improved flexibility in Asymmetry mode.

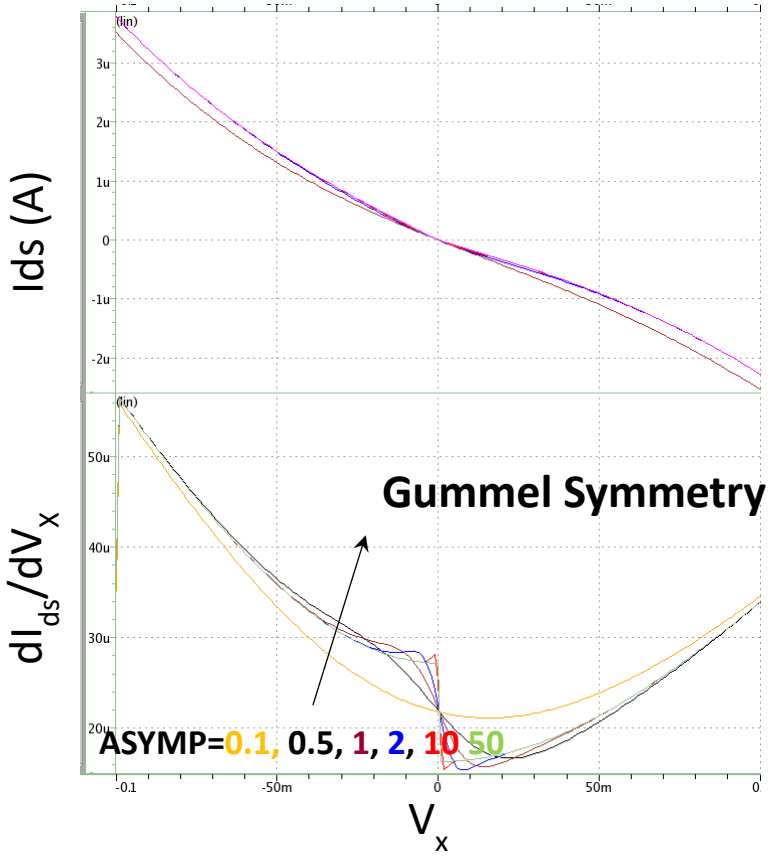
- In BSIM-BULK107.0.0, in asymmetry mode, if Vds is positive but small, e.g. 0.05V, wr is not 0 and wf is not 1, then CDSCD_a and ETA0_a would be “in between” “forward” and “reversed” model parameters.

```
// Asymmetry model
T0 = tanh(0.6 * Vds_noswap / Vtm);
wf = 0.5 + 0.5 * T0;
wr = 1.0 - wf;
if (ASYMMOD != 0) begin
   CDSCD_a  = CDSCDR_i * wr + CDSCD_i * wf;
   ETA0_a   = ETA0R_t * wr + ETA0_t * wf;
   ...
   ... ..
end ...
```

- BSIM-BULK107.1.0 fixes this bug by replacing constant = 0.6 to a parameter **ASYMP**.

```
T0 = tanh(ASYMP* Vds_noswap / Vtm)
```

- It passes the Gummel symmetry test, and the plots are shown below.



18.2020bug11: IDEFF and ISEFF.

- In BSIM-BULK107.0.0, IDEFF and ISEFF is the total Drain and Source current that does not include diode junction current.

BSIMBULK107.0.0

```
IDS   = devsign * ids;                        // Intrinsic drain-to-source current
IDEFF = IDS - (IGD + IGCD) + ISUB + IGIDL;    // Total drain current
ISEFF = -IDS - (IGS + IGCS) + IGISL;          // Total source current
IBS   = -devsign * Ibs;                          // Source junction current
IBD   = -devsign * Ibd;                          // Source junction current
```

- BSIM-BULK107.1.0 fixes this bug and diode junction current is added in IDEFF and ISEEF for the drain and source junction respectively.

BSIMBULK107.1.0

```
IDS    = devsign * ids;                          // Intrinsic drain-to-source current
IDEFF  = IDS - (IGD + IGCD) + ISUB + IGIDL + IBD;   // Total drain current
ISEFF  = -IDS - (IGS + IGCS) + IGISL + IBS;         // Total source current

IBS    = -devsign * Ibs;                          // Source junction current
IBD    = -devsign * Ibd;                          // Source junction current
```

19.2020bug12: Convergence issue due to DVT2EDGE.

- litl_edge = 0 can cause 1 by 0 error in BSIM-BULK107.0.0.
- In BSIM-BULK107.1.0, 1 by zero error is avoided by putting conditions on litl_edge.

BSIMBULK107.1.0

```
litl_edge = litl * (1.0 + DVT2EDGE * Vbsx);
if (litl_edge > 0) begin
  T0          = DVT1EDGE * Leff / litl_edge;
  if (T0 < 40.0) begin
    theta_sce_edge = 0.5 * DVT0EDGE / (cosh(T0) - 1.0);
  end else begin
    theta_sce_edge = DVT0EDGE * lexp(-T0);
  end
end else begin
  theta_sce_edge = 0;
end
```

20.2021bug1: Clamping of Gate electrode resistance RGATEMOD=2.

- In BSIM-BULK107.0.0, the bias independent gate-electrode resistance (Rgeltd) is not clamped in RGATEMOD = 2.

BSIMBULK107.0.0

```
if (RGATEMOD == 2) begin
    T11  = Grgeltd + Gcrg;
    Gcrg = Grgeltd * Gcrg / T11;
end
```

- BSIM-BULK107.1.0 fixes this bug by clamping the gate-electrode resistance (Rgeltd) to Minr if (Rgeltd < Minr) in RGATEMOD = 2.
- VA code for the updated implementation is given below:

BSIMBULK107.1.0

```
if (RGATEMOD == 2) begin
   Rgeltd = 1 / Grgeltd;
   if (Rgeltd < minr) begin
      Rgeltd = minr ;
      Grgeltd = 1 / Rgeltd;
   end
    T11  = Grgeltd + Gcrg;
    Gcrg = Grgeltd * Gcrg / T11;
end
```

21.2021bug3: Limit XRCRG1 & XRCRG2 parameters

$$\frac{1}{R_{ii}} = XRCRG1.NF \cdot \left(\frac{I_{ds}}{V_{dseff}} + XRCRG2 \cdot \frac{W_{eff}\mu_{eff}C_{oxeff}V_t}{L_{eff}} \right)$$

- 1/ Rii should always be positive.
- In this release, XRCRG1 and XRCRG2 are limited to be non-negative:

BSIMBULK107.0.0

```
`MPRnb( XRCRG1          ,12.0       ,""    ,"1st fitting parameter the bias-dependent Rg " )
`MPRnb( XRCRG2          ,1.0        ,""    ,"2nd fitting parameter the bias-dependent Rg " )
```

BSIMBULK107.1.0

```
`MPRcz( XRCRG1          ,12.0       ,""    ,"1st fitting parameter the bias-dependent Rg " )
`MPRcz( XRCRG2          ,1.0        ,""    ,"2nd fitting parameter the bias-dependent Rg " )
```

22. 2021bug4: Revert IIMOD changes to BSIM-BULK 107.0.0.

- In BSIM-BULK107.1.0 Beta1, we had removed IIMOD and changed the default values of ALPHDADR and BETADR, but they were not backward compatible.

- Therefore, we have reverted all the changes, that were made in the BSIM-BULK107.1.0 Beta1.

BSIM-BULK107.1.0. Beta1

```
`MPRcz( ALPHADR     ,0          ,"m/V"          ,"First parameter of Iii in the drift region" )
`MPRcz( BETADR      ,0          ,"1/V"          ,"Second parameter of Iii in the drift region" )
```

```
// Secondary impact ionization in the drift region
if (HVMOD == 1 && ALPHADR > 0) begin
    Ntot = DRII1 * ids/(NF * Weff * `q  * VDRIFT_t );
    Nextra = Ntot/NDRIFTD - 1;
    `Smooth(Nextra, 0, DELTAII, Nextra)
    Nextra = NDRIFTD * Nextra;
```

- Now, in BSIM-BULK107.1.0, IIMOD is same as in BSIM-BULK107.0.0.

BSIMBULK107.1.0

```
`MPRnb( ALPHADR     ,ALPHA0     ,"m/V"          ,"First parameter of Iii in the drift region" )
`MPRnb( BETADR      ,BETA0      ,"1/V"          ,"Second parameter of Iii in the drift region" )
```

```
// Secondary impact ionization in the drift region
if (HVMOD == 1 && IIMOD == 1) begin
    Ntot = DRII1 * ids/(NF * Weff * `q  * VDRIFT_t );
    Nextra = Ntot/NDRIFTD - 1;
    `Smooth(Nextra, 0, DELTAII, Nextra)
    Nextra = NDRIFTD * Nextra;
```

23. 2021bug5: Missing multiplication factor in 2nd instance of T5 in Rdss=0.

- In BSIM-BULK107.1.0 Beta1, (-2*LambdaC) multiplication factor was missing from the second instance of T5.

- There was a typo in else condition of T5 equation in both the instances. Now, it is corrected to -2*LambdaC (T1/T2).

BSIM-BULK107.0.0.

```
// qdsat for external Rds
if (Rdss == 0) begin
    // Accurate qdsat derived from consistent I-V
    T0 = 0.5 * LambdaC * (qs * qs + qs) / (1.0 + 0.5 * LambdaC * (1.0 + qs));
    T1 = 2.0 * LambdaC * (qs - T0);
    T2 = sqrt(1.0 + T1 * T1);
    ln_T1_T2 = asinh(T1);
    if (T1 != 0.0) begin
        T3 = T2 + (1.0 / T1) * ln_T1_T2;
    end else begin
        T3 = T2 + (1.0 / T2);
    end
    T4 = T0 * T3 - LambdaC * ((qs * qs + qs) - (T0 * T0 + T0));
    if (T1 != 0.0) begin
        T5 = -2.0 * LambdaC * (T1 * T2 - ln_T1_T2) / (T1 * T1);
    end else begin
        T5 = -2.0 * LambdaC * (T1/T2) * (T1/T2) *(T1/T2);
    end
    T6 = T0 * T5 + T3 + LambdaC * (2.0 * T0 + 1.0);
    T0 = T0 - (T4 / T6);
    T1 = 2.0 * LambdaC * (qs - T0);
    T2 = sqrt(1.0 + T1 * T1);
    ln_T1_T2 = asinh(T1);
    if (T1 != 0.0) begin
        T3 = T2 + (1.0 / T1) * ln_T1_T2;
    end else begin
        T3 = T2 + (1.0 / T2);
    end
    T4 = T0 * T3 - LambdaC * ((qs * qs + qs) - (T0 * T0 + T0));
    if (T1 != 0.0) begin
        T5 = -2.0 * LambdaC * (T1 * T2 - ln_T1_T2) / (T1 * T1);
    end else begin
        T5 = (T1 / T2) * (T1 / T2) * (T1 / T2);
    end
    T6    = T0 * T5 + T3 + LambdaC * (2.0 * T0 + 1.0);
    qdsat = T0 - (T4/T6);
```

BSIMBULK107.1.0

```
if (Rdss == 0) begin
    // Accurate qdsat derived from consistent I-V
    T0 = 0.5 * LambdaC * (qs * qs + qs) / (1.0 + 0.5 * LambdaC * (1.0 + qs));
    T1 = 2.0 * LambdaC * (qs - T0);
    T2 = sqrt(1.0 + T1 * T1);
    ln_T1_T2 = asinh(T1);
    if (T1 != 0.0) begin
        T3 = T2 + (1.0 / T1) * ln_T1_T2;
    end else begin
        T3 = T2 + (1.0 / T2);
    end
    T4 = T0 * T3 - LambdaC * ((qs * qs + qs) - (T0 * T0 + T0));
    if (T1 != 0.0) begin
        T5 = -2.0 * LambdaC * (T1 * T2 - ln_T1_T2) / (T1 * T1);
    end else begin
        T5 = -2.0 * LambdaC * (T1/T2);
    end
    T6 = T0 * T5 + T3 + LambdaC * (2.0 * T0 + 1.0);
    T0 = T0 - (T4 / T6);
    T1 = 2.0 * LambdaC * (qs - T0);
    T2 = sqrt(1.0 + T1 * T1);
    ln_T1_T2 = asinh(T1);
    if (T1 != 0.0) begin
        T3 = T2 + (1.0 / T1) * ln_T1_T2;
    end else begin
        T3 = T2 + (1.0 / T2);
    end
    T4 = T0 * T3 - LambdaC * ((qs * qs + qs) - (T0 * T0 + T0));
    if (T1 != 0.0) begin
        T5 = -2.0 * LambdaC * (T1 * T2 - ln_T1_T2) / (T1 * T1);
    end else begin
        T5 = -2.0 * LambdaC * (T1/T2);
    end
    T6    = T0 * T5 + T3 + LambdaC * (2.0 * T0 + 1.0);
    qdsat = T0 - (T4/T6);
```

24. 2021bug6: Removing redundancy in the calculation of Rdss.

- In BSIM-BULK107.1.0 Beta1, Rdss is calculated twice for RDSMOD =2.

BSIM-BULK107.0.0.

```
if (RDSMOD == 1) begin
    Rdss = 0.0;
end else begin
    T0   = 1.0 + PRWG_i * qis;
    T1   = PRWB_i * (sqrtPhistVbs - sqrtPhist);
    T2   = 1.0 / T0 + T1;
    T3   = T2 + sqrt(T2 * T2 + 0.01);
    Rdss = (RDSWMIN_i + RDSW_i * T3) * WeffWRFactor * NF * rdstemp;
    if (RDSMOD == 2) begin
        Rdss = (RSourceGeo + (RDSWMIN_i + RDSW_i * T3) * WeffWRFactor * NF + RDrainGeo) * rdstemp;
    end
end
```

- BSIM-BULK107.1.0, removes the redundant calculation of Rdss by changing the if-else condition.

BSIMBULK107.1.0

```
if (RDSMOD == 1) begin
    Rdss = 0.0;
end else begin
    T0   = 1.0 + PRWG_i * qis;
    T1   = PRWB_i * (sqrtPhistVbs - sqrtPhist);
    T2   = 1.0 / T0 + T1;
    T3   = T2 + sqrt(T2 * T2 + 0.01);
    if (RDSMOD == 0) begin
        Rdss = (RDSWMIN_i + RDSW_i * T3) * WeffWRFactor * NF * rdstemp;
    end else begin
        Rdss = (RSourceGeo + (RDSWMIN_i + RDSW_i * T3) * WeffWRFactor * NF + RDrainGeo) * rdstemp;
    end
end
```

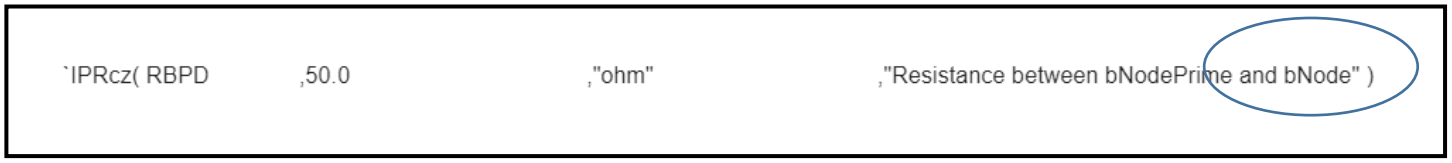
25. 2021bug7: Improved smoothing of drift resistance in HV module

- BSIM-BULK107.1.0 Beta1 for HV module, shows unphysical trends in Ids and gds plots at high Vds.
- In the BSIM-BULK107.1.0 Beta2, clamping of T5 equation has been improved. But this update is not a part of BSIM-BULK107.1.0., because later, drift resistance implementation was made voltage dependent.

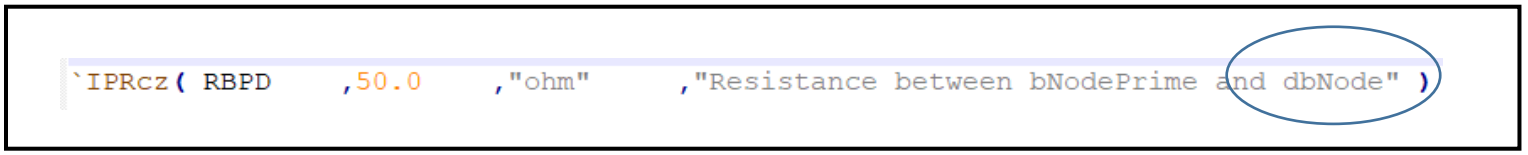
26. 2021bug8: Correct the description of RBPD parameter.

BSIM-BULK107.0.0.

- There was a typo in the description of RBPD parameter.



BSIMBULK107.1.0



27.2021bug9: Issues raised by VAMPyRE

a. Warning on bias-dependent condition if (Rdss==0).

- During the VAMPyRE run, BSIM-BULK107.1.0. Beta1 shows warning due to if (Rdss==0) condition.

BSIM-BULK107.0.0.

```
if (Rdss == 0) begin
    // Accurate qdsat derived from consistent I-V
    T0 = 0.5 * LambdaC * (qs * qs + qs) / (1.0 + 0.5 * LambdaC * (1.0 + qs));
    T1 = 2.0 * LambdaC * (qs - T0);
    .

end else begin
    // Accurate qdsat derived from consistent I-V
    T11 = Weff * 2.0 * nq * Cox * nVt * VSAT_a;
    T12 = T11 * LambdaC * Rdss / (2.0 * nVt);
    T0  = 0.5 * LambdaC * (qs * qs + qs) / (1.0 + 0.5 * LambdaC * (1.0 + qs));
```

- BSIM-BULK107.1.0 removes this VAMPyRE warning by changing the if condition to (Rdss >0), as Rdss can never be negative.

BSIMBULK107.1.0

```
if (Rdss > 0) begin
    // Accurate qdsat derived from consistent I-V
    T11 = Weff * 2.0 * nq * Cox * nVt * VSAT_a;
    T12 = T11 * LambdaC * Rdss / (2.0 * nVt);
    T0  = 0.5 * LambdaC * (qs * qs + qs) / (1.0 + 0.5 * LambdaC * (1.0 + qs));
    .

end else begin
    T0 = 0.5 * LambdaC * (qs * qs + qs) / (1.0 + 0.5 * LambdaC * (1.0 + qs));
    T1 = 2.0 * LambdaC * (qs - T0);
```

b. Warning on bias-dependent condition if (PCLM_a!=0) .

- BSIM-BULK107.1.0. Beta1 shows warning due to if (PCLM_a!=0) condition.

BSIM-BULK107.0.0.

```
if (PCLM_a != 0.0) begin
    if (PCLMG < 0.0) begin
        T1 = PCLM_a / (1.0 - PCLMG * qia / EsatL) / Fp;
    end else begin
        T1 = PCLM_a * (1.0 + PCLMG * qia / EsatL) / Fp;
    end
    MdL = 1.0 + T1 * lln(1.0 + diffVds / T1 / Vasat);
end else begin
    MdL = 1.0;
end
Moc = Moc * MdL;
```

- In BSIM-BULK107.1.0, this VAMPyRE warning is removed by changing the if condition to (PCLM_a > 0), as PCLM_a can never be negative.

- For PCLM_a = 0, we have applied L'Hopital rule to get the updated equation in the else condition as MDL = 1+T1.

BSIMBULK107.1.0

```
if (PCLM_a > 0.0) begin
    if (PCLMG < 0.0) begin
        T1 = PCLM_a / (1.0 - PCLMG * qia / EsatL) / Fp;
    end else begin
        T1 = PCLM_a * (1.0 + PCLMG * qia / EsatL) / Fp;
    end
    MdL = 1.0 + T1 * lln(1.0 + diffVds / T1 / Vasat);
end else begin
    if (PCLMG < 0.0) begin
        T1 = PCLM_a / (1.0 - PCLMG * qia / EsatL) / Fp;
    end else begin
        T1 = PCLM_a * (1.0 + PCLMG * qia / EsatL) / Fp;
    end
    MdL = 1.0 + T1 ;
end
Moc = Moc * MdL;
```

28.2021bug10 (ADI): Typo in the comment of Drain junction current (IBD).

BSIM-BULK107.0.0

```
IBS    = -devsign * Ibs;                    // Source junction current
IBD    = -devsign * Ibd;                    // Source junction current
```

BSIMBULK107.1.0

```
IBS    = -devsign * Ibs;                    // Source junction current
IBD    = -devsign * Ibd;                    // Drain junction current
```

29. 2021bug11: Node order correction in the body-drain diode for HV model.

- We have added external body drain diode in Beta1, and its current direction is from dbulk to ddbulk.
  But in BSIM-BULK107.1.0. Beta2 we found that the node orders are not correct.
- In BSIM-BULK107.1.0, node orders are corrected according to the direction of current flow.

BSIM-BULK107.1.0 Beta2

```
if (RBODYMOD != 0 && RBODYHVMOD == 1) begin
    I(dbulk, di) <+ devsign * XPART * Ibd + V(dbulk, di) * gmin;
    I(ddbulk, dbulk) <+ devsign * (1.0 - XPART) * Ibd + V(ddbulk, dbulk) * gmin;
    I(dbulk, di) <+ devsign * ddt(XPART * Qbdj);
    I(dbulk, ddbulk) <+ devsign * ddt((1.0 - XPART) * Qbdj);
end else begin
    V(d, ddbulk)  <+ 0.0;
end
```

BSIMBULK107.1.0

```
I(dbulk, ddbulk) <+ devsign * Ibd_ext + XPART * V(dbulk, ddbulk) * gmin;
```

30. 2021bug12: Removing extra contribution of Ibd in the HV model.

- In BSIM-BULK107.1.0. Beta2 when RBODYMOD!= 0, then Ibd is contributed twice.
- We have corrected this bug in BSIM-BULK107.1.0.

BSIM-BULK107.1.0 Beta2

```
if (RBODYMOD != 0) begin
    I(sbulk, si) <+ devsign * Ibs + V(sbulk, si) * gmin;
    I(dbulk, di) <+ devsign * Ibd + V(dbulk, di) * gmin;
    I(sbulk, si) <+ devsign * ddt(Qbsj);
    I(dbulk, di) <+ devsign * ddt(Qbdj);
end else begin
    I(bi, si) <+ devsign * Ibs + V(bi, si) * gmin;
    I(bi, di) <+ devsign * Ibd + V(bi, di) * gmin;
    I(bi, si) <+ devsign * ddt(Qbsj);
    I(bi, di) <+ devsign * ddt(Qbdj);
end
if (RBODYMOD != 0 && RBODYHVMOD == 1) begin
    I(dbulk, di) <+ devsign * XPART * Ibd + V(dbulk, di) * gmin;
    I(ddbulk, dbulk) <+ devsign * (1.0 - XPART) * Ibd + V(dbulk, ddbulk) * gmin;
    I(dbulk, di) <+ devsign * ddt(XPART * Qbdj);
    I(dbulk, ddbulk) <+ devsign * ddt((1.0 - XPART) * Qbdj);
end else begin
    V(d, ddbulk)  <+ 0.0;
end
```

BSIMBULK107.1.0

```
if (RBODYMOD != 0) begin
    I(sbulk, si) <+ devsign * Ibs + V(sbulk, si) * gmin;
    I(sbulk, si) <+ devsign * ddt(Qbsj);
    if (RBODYHVMOD == 0) begin
        I(dbulk, di) <+ devsign * Ibd + V(dbulk, di) * gmin;
        I(dbulk, di) <+ devsign * ddt(Qbdj);
    end
end else begin
    I(bi, si) <+ devsign * Ibs + V(bi, si) * gmin;
    I(bi, di) <+ devsign * Ibd + V(bi, di) * gmin;
    I(bi, si) <+ devsign * ddt(Qbsj);
    I(bi, di) <+ devsign * ddt(Qbdj);
end
if (RBODYMOD != 0 && RBODYHVMOD == 1) begin
    I(ddbulk, d) <+ V(ddbulk, d) * Grdb;
    I(ddbulk, d) <+ white_noise(Nt * Grdb, "rdb");
end else begin
    V(d, ddbulk)  <+ 0.0;
end
```

31.2021bug13: Drain-body junction current splitting in RBODYHVMOD.

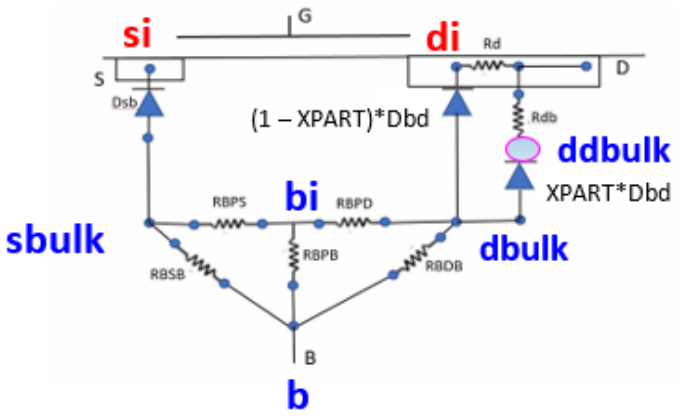
- We have added external body diode in Beta2 and Ibd was the function of V(dbulk, di), which has been partitioned.
- In BSIM-BULK107.1.0, Ibd is made a function of V(dbulk, di), and Ibd_ext is a function of V(dbulk, ddbulk).

Ibd = (1-XPART) * F(V(dbulk,di)) , Ibd_ext = XPART * F(V(dbulk,ddbulk))

BSIM-BULK107.1.0 Beta2

```
if (RBODYMOD != 0) begin
    I(sbulk, si) <+ devsign * Ibs + V(sbulk, si) * gmin;
    I(dbulk, di) <+ devsign * Ibd + V(dbulk, di) * gmin;
    I(sbulk, si) <+ devsign * ddt(Qbsj);
    I(dbulk, di) <+ devsign * ddt(Qbdj);
end else begin
    I(bi, si) <+ devsign * Ibs + V(bi, si) * gmin;
    I(bi, di) <+ devsign * Ibd + V(bi, di) * gmin;
    I(bi, si) <+ devsign * ddt(Qbsj);
    I(bi, di) <+ devsign * ddt(Qbdj);
end
if (RBODYMOD != 0 && RBODYHVMOD == 1) begin
    I(dbulk, di) <+ devsign * XPART * Ibd + V(dbulk, di) * gmin;
    I(ddbulk, dbulk) <+ devsign * (1.0 - XPART) * Ibd + V(dbulk, ddbulk) * gmin;
    I(dbulk, di) <+ devsign * ddt(XPART * Qbdj);
    I(dbulk, ddbulk) <+ devsign * ddt((1.0 - XPART) * Qbdj);
end else begin
    V(d, ddbulk)  <+ 0.0;
end
```

BSIMBULK107.1.0



Drain-body current = Drain-side junction current + tunneling current

1. Drain-side junction current partitioning:

```
if (Isbd > 0.0) begin
    if (OneMinusXpart > 0) begin
        if (Vbd_jct < VjdmRev) begin
            T0  = Vbd_jct / Nvtmd;
            T1  = lexp(T0) - 1.0;
            T2  = IVjdmRev + DslpRev * (Vbd_jct - VjdmRev);
            Ibd = OneMinusXpart * T1 * T2;
        end else if (Vbd_jct <= VjdmFwd) begin
            T0  = Vbd_jct / Nvtmd;
            T1  = (BVD + Vbd_jct) / Nvtmd;
            T2  = lexp(-T1);
            Ibd = OneMinusXpart * Isbd * (lexp(T0) + XExpBVD - 1.0 - XJBVD * T2);
        end else begin
            Ibd = OneMinusXpart * (IVjdmFwd + DslpFwd * (Vbd_jct - VjdmFwd));
        end
    end else begin
        Ibd = 0.0;
    end
    if (XPART > 0.0 && RBODYHVMOD == 1) begin
        if (Vbd_ext < VjdmRev) begin
            T0  = Vbd_ext / Nvtmd;
            T1  = lexp(T0) - 1.0;
            T2  = IVjdmRev + DslpRev * (Vbd_ext - VjdmRev);
            Ibd_ext = XPART * T1 * T2;
        end else if (Vbd_ext <= VjdmFwd) begin
            T0  = Vbd_ext / Nvtmd;
            T1  = (BVD + Vbd_ext) / Nvtmd;
            T2  = lexp(-T1);
            Ibd_ext = XPART * Isbd * (lexp(T0) + XExpBVD - 1.0 - XJBVD * T2);
        end else begin
            Ibd_ext = XPART * (IVjdmFwd + DslpFwd * (Vbd_ext - VjdmFwd));
        end
    end else begin
        Ibd_ext = 0.0;
    end
end else begin
    Ibd = 0.0;
    Ibd_ext = 0.0;
end
```

2. Tunneling current partitioning:

```
if (JTSD_t > 0.0) begin
    if ((VTSD - Vbd_jct) < (VTSD * 1.0e-3)) begin
        T0  = -Vbd_jct / Vtm0 / NJTSD_t;
        T1  = lexp(T0 * 1.0e3) - 1.0;
        Ibd = Ibd - OneMinusXpart * ADeff * JTSD_t * T1;
    end else begin
        T0  = -Vbd_jct / Vtm0 / NJTSD_t;
        T1  = lexp(T0 * VTSD/ (VTSD - Vbd_jct)) - 1.0;
        Ibd = Ibd - OneMinusXpart * ADeff * JTSD_t * T1;
    end
end
if (JTSSWD_t > 0.0) begin
    if (PDeff > Weffcj * NF) begin
        T2 = OneMinusXpart * (PDeff - Weffcj * NF) * JTSSWD_t;
    end else begin
        T2 = OneMinusXpart * PDeff * JTSSWD_t;
    end
    if ((VTSSWD - Vbd_jct) < (VTSSWD * 1.0e-3)) begin
        T0  = -Vbd_jct / Vtm0 / NJTSSWD_t;
        T1  = lexp(T0 * 1.0e3) - 1.0;
        Ibd = Ibd - T2 * T1;
    end else begin
        T0  = -Vbd_jct / Vtm0 / NJTSSWD_t;
        T1  = lexp(T0 * VTSSWD / (VTSSWD - Vbd_jct)) - 1.0;
        Ibd = Ibd - T2 * T1;
    end
end
if (JTSSWGD_t > 0.0) begin
    if ((VTSSWGD - Vbd_jct) < (VTSSWGD * 1.0e-3)) begin
        T0  = -Vbd_jct / Vtm0 / NJTSSWGD_t;
        T1  = lexp(T0 * 1.0e3) - 1.0;
        Ibd = Ibd - Weffcj * NF * JTSSWGD_t * T1;
    end else begin
        T0  = -Vbd_jct / Vtm0 / NJTSSWGD_t;
        T1  = lexp(T0 * VTSSWGD / (VTSSWGD - Vbd_jct)) - 1.0;
        Ibd = Ibd - Weffcj * NF * JTSSWGD_t * T1;
    end
end

 if (XPART > 0) begin
     if (JTSD_t > 0.0) begin
         if ((VTSD - Vbd_ext) < (VTSD * 1.0e-3)) begin
             T0  = -Vbd_ext / Vtm0 / NJTSD_t;
             T1  = lexp(T0 * 1.0e3) - 1.0;
             Ibd_ext = Ibd_ext - XPART * ADeff * JTSD_t * T1;
         end else begin
             T0  = -Vbd_ext / Vtm0 / NJTSD_t;
             T1  = lexp(T0 * VTSD/ (VTSD - Vbd_ext)) - 1.0;
             Ibd_ext = Ibd_ext - XPART * ADeff * JTSD_t * T1;
         end
     end
     if (JTSSWD_t > 0.0) begin
         if (PDeff > Weffcj * NF) begin
             T2 = (XPART * (PDeff - Weffcj * NF) + Weffcj * NF) * JTSSWD_t;
         end else begin
             T2 = XPART * PDeff * JTSSWD_t;
         end
         if ((VTSSWD - Vbd_ext) < (VTSSWD * 1.0e-3)) begin
             T0  = -Vbd_ext / Vtm0 / NJTSSWD_t;
             T1  = lexp(T0 * 1.0e3) - 1.0;
             Ibd_ext = Ibd_ext - T2 * T1;
         end else begin
             T0  = -Vbd_ext / Vtm0 / NJTSSWD_t;
             T1  = lexp(T0 * VTSSWD / (VTSSWD - Vbd_ext)) - 1.0;
             Ibd_ext = Ibd_ext - T2 * T1;
         end
     end
 end
```

Total drain-body junction current partitioning implementation:

```
if (RBODYMOD != 0) begin
    I(sbulk, si) <+ devsign * Ibs + V(sbulk, si) * gmin;
    I(sbulk, si) <+ devsign * ddt(Qbsj);
    if (RBODYHVMOD == 0) begin
        I(dbulk, di) <+ devsign * Ibd + V(dbulk, di) * gmin;
        I(dbulk, di) <+ devsign * ddt(Qbdj);
    end
end else begin
    I(bi, si) <+ devsign * Ibs + V(bi, si) * gmin;
    I(bi, di) <+ devsign * Ibd + V(bi, di) * gmin;
    I(bi, si) <+ devsign * ddt(Qbsj);
    I(bi, di) <+ devsign * ddt(Qbdj);
end

if (RBODYMOD != 0 && RBODYHVMOD == 1) begin
    I(dbulk, di) <+ devsign * Ibd + (1.0 - XPART) * V(dbulk, di) * gmin;
    I(dbulk, ddbulk) <+ devsign * Ibd_ext + XPART * V(dbulk, ddbulk) * gmin;
    I(dbulk, di) <+ devsign * ddt(Qbdj);
    I(dbulk, ddbulk) <+ devsign * ddt(Qbdj_ext);
end else begin
    V(d, ddbulk)  <+ 0.0;
end
```

Capacitance partitioning:

```
Czbd       = OneMinusXpart * CJD_t * ADeff;
if (PDeff > Weffcj * NF) begin
    if (XPART > 0) begin
       Czbdsw     = OneMinusXpart* CJSWD_t * (PDeff - Weffcj * NF);
    end else begin
       Czbdsw     = OneMinusXpart* CJSWD_t * PDeff;
    end
end else begin
    Czbdsw     = OneMinusXpart* CJSWD_t * PDeff;
end
```

```
Qbdj = Qbdj1 + Qbdj2 + Qbdj3;
if (XPART > 0 && RBODYHVMOD == 1) begin
    Czbd       = XPART * CJD_t * ADeff;
    if (PDeff > Weffcj * NF) begin
        Czbdsw     = CJSWD_t * (XPART * (PDeff - Weffcj * NF) + Weffcj * NF);
    end else begin
        Czbdsw     = XPART * CJSWD_t * PDeff;
    end
    Czbdswg    = 0.0; // no gate-edge contribution to Qbdj_ext
    `JunCap(Czbd, Vbd_ext, PBD_t, MJD, czbd_p1, czbd_p2, Qbdj1_ext)
    `JunCap(Czbdsw, Vbd_ext, PBSWD_t, MJSWD, czbdsw_p1, czbdsw_p2, Qbdj2_ext)
    Qbdj_ext = Qbdj1_ext + Qbdj2_ext;
end else begin
    Qbdj_ext   = 0.0;
end
```

32.2021bug14: Protection for junction capacitance grading coefficient (MJX) against 0/0 form.

(a). <u>In first instance:</u>

- In BSIM-BULK107.1.0. Beta2, when MJX =1 then Qbxj takes the form of 0/0.
- For MJX =1, we have applied L'Hospital in Qbxj.

BSIM-BULK107.0.0

```
`define JunCap(Czbx, Vbx_jct, PBX_t, MJX, czbx_p1, czbx_p2, Qbxj) \
    if (Czbx > 0.0) begin \
        T1 = Vbx_jct / PBX_t; \
        if (T1 < 0.9) begin \
            arg = 1.0 - T1; \
            if (MJX == 0.5) begin \
                sarg = 1.0 / sqrt(arg); \
            end else begin \
                sarg = lexp(-MJX * lln(arg)); \
            end \
            Qbxj = PBX_t * Czbx * (1.0 - arg * sarg) / (1.0 - MJX); \
        end else begin \
            T2  = czbx_p1 * (T1 - 1.0) * (5.0 * MJX * (T1 - 1.0) + (1.0 + MJX)); \
            Qbxj = PBX_t * Czbx * (T2 + czbx_p2); \
        end \
    end else begin \
        Qbxj = 0.0; \
    end \
```

BSIMBULK107.1.0

```
`define JunCap(Czbx, Vbx_jct, PBX_t, MJX, czbx_p1, czbx_p2, Qbxj) \
    if (Czbx > 0.0) begin \
        T1 = Vbx_jct / PBX_t; \
        if (T1 < 0.9) begin \
            arg = 1.0 - T1; \
            if (MJX != 1) begin \
                if (MJX == 0.5) begin \
                    sarg = 1.0 / sqrt(arg); \
                end else begin \
                    sarg = lexp(-MJX * lln(arg)); \
                end \
                Qbxj = PBX_t * Czbx * (1.0 - arg * sarg) / (1.0 - MJX); \
            end else begin \
                Qbxj = -lln(arg); \
            end \
        end else begin \
            T2  = czbx_p1 * (T1 - 1.0) * (5.0 * MJX * (T1 - 1.0) + (1.0 + MJX)); \
            Qbxj = PBX_t * Czbx * (T2 + czbx_p2); \
        end \
    end else begin \
        Qbxj = 0.0; \
    end \
```

(b). <u>In second instance:</u>

- In BSIM-BULK107.1.0. Beta2, when MJS/MJSWS/MJSWGS =1 then czbs_p2/czbssw_p2/czpsswg_p2 takes the form of 0/0 respectively.
- For MJS/MJSWS/MJSWGS = 1, we have applied L'Hospital in czbs_p2/czbssw_p2/czpsswg_p2.

BSIM-BULK107.0.0

```
Czbs       = CJS_t * ASeff;
Czbssw     = CJSWS_t * PSeff;
Czbsswg    = CJSWGS_t * Weffcj * NF;
czbs_p1    = pow(0.1, -MJS);
czbs_p2    = 1.0 / (1.0 - MJS) * (1.0 - 0.05 * MJS * (1.0 + MJS) * czbs_p1);
czbssw_p1  = pow(0.1, -MJSWS);
czbssw_p2  = 1.0 / (1.0 - MJSWS) * (1.0 - 0.05 * MJSWS * (1.0 + MJSWS) * czbssw_p1);
czbsswg_p1 = pow(0.1, -MJSWGS);
czbsswg_p2 = 1.0 / (1.0 - MJSWGS) * (1.0 - 0.05 * MJSWGS * (1.0 + MJSWGS) * czbsswg_p1);
`JunCap(Czbs, Vbs_jct, PBS_t, MJS, czbs_p1, czbs_p2, Qbsj1)
`JunCap(Czbssw, Vbs_jct, PBSWS_t, MJSWS, czbssw_p1, czbssw_p2, Qbsj2)
`JunCap(Czbsswg, Vbs_jct, PBSWGS_t, MJSWGS, czbsswg_p1, czbsswg_p2, Qbsj3)
Qbsj = Qbsj1 + Qbsj2 + Qbsj3;
```

BSIMBULK107.1.0

```
Czbs       = CJS_t * ASeff;
Czbssw     = CJSWS_t * PSeff;
Czbsswg    = CJSWGS_t * Weffcj * NF;
czbs_p1    = pow(0.1, -MJS);
if (MJS == 1) begin
    czbs_p2    = 1.5 - lln(0.1);
end else begin
    czbs_p2    = 1.0 / (1.0 - MJS) * (1.0 - 0.05 * MJS * (1.0 + MJS) * czbs_p1);
end
czbssw_p1  = pow(0.1, -MJSWS);
if (MJSWS == 1) begin
    czbssw_p2    = 1.5 - lln(0.1);
end else begin
    czbssw_p2  = 1.0 / (1.0 - MJSWS) * (1.0 - 0.05 * MJSWS * (1.0 + MJSWS) * czbssw_p1);
end
czbsswg_p1 = pow(0.1, -MJSWGS);
if (MJSWGS == 1) begin
    czbsswg_p2    = 1.5 - lln(0.1);
end else begin
    czbsswg_p2 = 1.0 / (1.0 - MJSWGS) * (1.0 - 0.05 * MJSWGS * (1.0 + MJSWGS) * czbsswg_p1);
end
`JunCap(Czbs, Vbs_jct, PBS_t, MJS, czbs_p1, czbs_p2, Qbsj1)
`JunCap(Czbssw, Vbs_jct, PBSWS_t, MJSWS, czbssw_p1, czbssw_p2, Qbsj2)
`JunCap(Czbsswg, Vbs_jct, PBSWGS_t, MJSWGS, czbsswg_p1, czbsswg_p2, Qbsj3)
Qbsj = Qbsj1 + Qbsj2 + Qbsj3;
```

33.2021bug15: Smoothing function for psiph.

- In BSIM-BULK107.1.0 Beta2, psiph can go negative and cause convergence issue in square root term.
- In BSIM-BULK107.1.0, we have applied smoothing function to make psiph a positive value before going into square root term.

BSIM-BULK107.0.0

```
// Pinch-Off potential for halo region
`PO_psip(vgfbh, gam_h, 0.0, phib_h, psiph)

// Normalized inversion charge at source end of halo MOSFET
`BSIM_q(psiph, phib_h, vs, gam_h, qsh)
nq_h = 1.0 + gam_h / (2.0 * sqrt(psiph));
```

BSIMBULK107.1.0

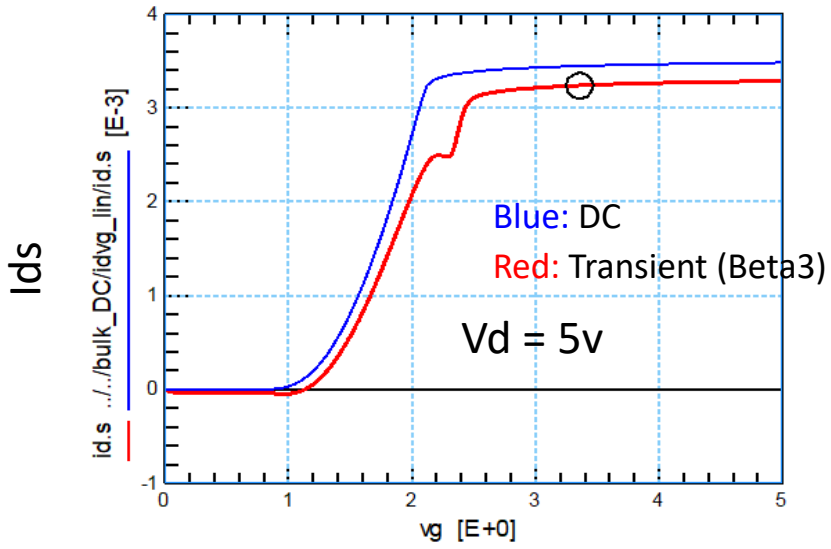
```
// Pinch-Off potential for halo region
`PO_psip(vgfbh, gam_h, 0.0, phib_h, psiph)

// Normalized inversion charge at source end of halo MOSFET
`BSIM_q(psiph, phib_h, vs, gam_h, qsh)
`Smooth(psiph, 1.0, 2.0, psiphclamp)
nq_h = 1.0 + gam_h / (2.0 * sqrt(psiphclamp));
```

34. 2021bug16: Id-Vg shows spikes in transient simulation.

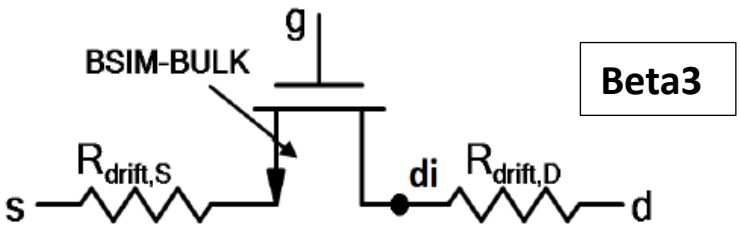
- In BSIM-BULK107.1.0 Beta3, during transient analysis, Id flowing through drift resistance goes above Idrift_sat_d.
- To decrease Id, Rdrift increases due to which Id-Vg plot shows dip during transient simulation.

Drift resistance model in Beta3 at drain side:

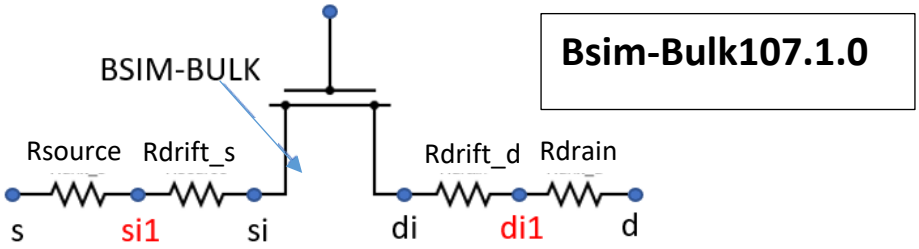
$$R_{drift,D} = \frac{RDLCW \cdot \frac{1}{NF \cdot Weff^{WR}}}{\left[1 - \left\{\frac{I_d}{I_{drft,satD}}\right\}^{MDRIFT}\right]^{\frac{1}{MDRIFT}}}$$



- In Beta3, the resistance connected between node di and d is Rdrain + rdrift for high voltage model.



- Id in the drift resistance expression of beta3 is the current flowing through rdrift.
- Now, in between node di and d (si and s) another node di1 (si1) is added and Id (Drain)= V(di1,di)/Rdrift_D and Id (Source)= V(si,si1)/Rdrift_S



- After rearranging we get rdrift expression in term of voltage rather than current.

Drift resistance model at drain side:

$$R_{drift\_D} = R_0 \left[1 + \left(\frac{\delta^{\frac{1}{MDRIFT}} * V(di1,di)}{I_{drift\_sat\_D} * R_0} \right)^{MDRIFT} \right]^{\frac{1}{MDRIFT}}$$

Ro = rdstemphv * RDLCW * WeffWRFactor * (1.0 + PDRWB * Vbsx)

$$\delta = \left(\frac{ids^{4-MDRIFT}}{ids^{4-MDRIFT} + HVFACTOR * idrift\_sat\_D^{4-MDRIFT}} \right)$$

At the source side:

$$\mathbf{R_{drift\_S} = R_0 \left[1 + \left(\frac{\frac{1}{\delta^{MDRIFT}} * V(si,si1)}{I_{drift\_sat\_S} * R_0} \right)^{MDRIFT} \right]^{\frac{1}{MDRIFT}}}$$

Ro = rdstemphv * RSLCW * WeffWRFactor * (1.0 + PDRWB * Vbs)

$$\delta = \left(\frac{ids^{4-MDRIFT}}{ids^{4-MDRIFT} + HVFACTOR * idrift\_sat\_S^{4-MDRIFT}} \right)$$

BSIM-BULK107.1.0 VA code implementation:

Rdrain calculation

```
// Rs (Source side resistance for all fingers)
Vs1          = devsign * V(si1, bi);
Vgs1_noswap  = Vg - Vs1;
T2           = Vgs1_noswap - Vfbsdr;
T3       = sqrt(T2 * T2 + 0.01);
Vgs_eff  = 0.5 * (T2 + T3);
T5       = 1.0 + PRWG_i * Vgs_eff;
Vsb1_noswap = Vs1;
T6       = (1.0 / T5) + PRWB_i * Vsb1_noswap;
T4       = 0.5 * (T6 + sqrt(T6 * T6 + 0.01));
Rsource = rdstemp * (RSourceGeo + (RSWMIN_i + RSW_i * T4) * WeffWRFactor);
// Rd (Drain side resistance for all fingers)
Vd1          = devsign * V(di1, bi);
Vgd1_noswap  = Vg - Vd1;
T2           = Vgd1_noswap - Vfbsdr;
T3       = sqrt(T2 * T2 + 0.01);
Vgd_eff  = 0.5 * (T2 + T3);
T5       = 1.0 + PRWG_i * Vgd_eff;
Vdb1_noswap = Vd1;
T6       = (1.0 / T5) + PRWB_i * Vdb1_noswap;
T4       = 0.5 * (T6 + sqrt(T6 * T6 + 0.01));
Rdrain  = rdstemp * (RDrainGeo + (RDWMIN_i + RDW_i * T4) * WeffWRFactor);
```

Rdrift_d calculation

```
if (RDSMOD == 1 && HVMOD == 1) begin
    vbi_drift = Vt * ln( NDEP_i * NDR  / pow(ni,2.0));
    if ((SHMOD != 0.0) && (RTH0 > 0.0) && (Weff_SH > 0.0)) begin
        vbi_drift = Vt * sqrt(vbi_drift * vbi_drift + 1.0e-6);
    end
    T4  = 1.0 - PDRWB * Vsb_noswap;
    `Smooth(T4, 0.0, 1.0e-3, T4)
    T1  =  qs - PTWGHV1 ;
    `Smooth(T1, 0.1, 2.0, T1)
    T2  = 10.0 * PSATXHV * T1 / (10.0 * PSATXHV + T1);
    vdrift_eff = VDRIFT_t * (1.0 + PTWGHV * T2);
    T11 = NF * Weff * `q  * vdrift_eff ;
    if (RDLCW != 0) begin
        if (GADRIFT == 0) begin
            T9 = 1.0;
        end else begin
            `Smooth(abs(V(di1, di)) - RDVDS, 0, 0.5, T10)
            T9 = 1.0 + T10 * GADRIFT;
        end
        idrift_sat_d = T11 * NDRIFTD * T9;
        T2 = 1.0 + Vsb_noswap / vbi_drift;
        `Smooth(T2, 0.0, 0.05, T2)
        T6 = (1.0 - DRB1 * ( sqrt(T2) - 1.0) - DRB2 * Vsb_noswap);
        `Smooth(T6, 0.0, 0.05, T6)
        idrift_sat_d = T6 * idrift_sat_d;
        delta_hv = pow(ids, 4 - MDRIFT) / (pow(ids, 4 - MDRIFT) + HVFACTOR * pow(idrift_sat_d, 4 - MDRIFT));
        T5 = pow(delta_hv, 1.0 / MDRIFT);
        T7 = rdstemphv * RDLCW * WeffWRFactor * T4;
        T8 = T5 * abs(V(di1, di)) / (idrift_sat_d * T7);
        `Smooth(T8, 0.0, 1.0e-3, T8)
        rdrift_d = T7 * pow(1.0 + pow(T8, MDRIFT), 1.0 / MDRIFT);
        IDRIFTSATD = idrift_sat_d;
    end
```

Rdrift_s calculation

```
if (RSLCW != 0) begin
    idrift_sat_s = T11 * NDRIFTS ;
    delta_hv = pow(ids,4-MDRIFT) / (pow(ids, 4-MDRIFT) + HVFACTOR * pow(idrift_sat_s, 4-MDRIFT));
    T5 = pow(delta_hv, 1.0 / MDRIFT);
    T7 = rdstemphv * RSLCW * WeffWRFactor * T4;
    T8 = T5 * abs(V(si, si1)) / (idrift_sat_s * T7);
    `Smooth(T8, 0.0, 1.0e-3, T8)
    rdrift_s = T7 * pow(1.0 + pow(T8, MDRIFT), 1.0 / MDRIFT);
end
```

```
if (RDSMOD != 2 && RDrainGeo > 0) begin
    gdpr = 1.0 / Rdrain;    // Note: gdpr considers all fingers
    I(d, di1) <+ V(d, di1) * gdpr;
    I(d, di1) <+ white_noise(Nt * gdpr, "rd");
    if (RDSMOD == 1 && HVMOD == 1 && RDLCW > 0) begin
        gdrift_d = 1.0 / rdrift_d;
        I(di1, di) <+ V(di1, di) * gdrift_d;
        I(di1, di) <+ white_noise(Nt * gdrift_d, "rdrift_d");
        I(di1, di) <+ flicker_noise(KFND * W * pow((ids / W), AFND), BFND ,"flicker");
    end else begin
        V(di, di1) <+ 0.0;
    end
end else begin
    V(d, di1) <+ 0.0;
    V(di1, di)  <+ 0.0;
end
```

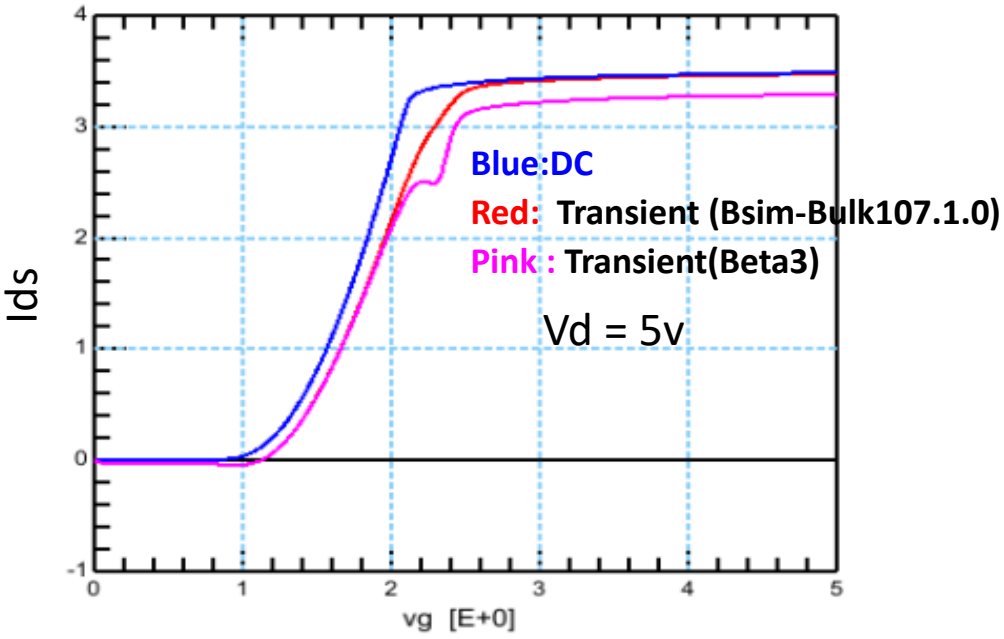
Current assign at drain side.

```
if (RDSMOD != 2 && RSourceGeo > 0) begin
    gspr = 1.0 / Rsource;    // Note: gspr considers all fingers
    I(s, si1) <+ V(s, si1) * gdpr;
    I(s, si1) <+ white_noise(Nt * gdpr, "rd");
    if (RDSMOD == 1 && HVMOD == 1 && RSLCW > 0) begin
        gdrift_s = 1.0 / rdrift_s;
        I(si1, si) <+ V(si1, si) * gdrift_s;
        I(si1, si) <+ white_noise(Nt * gdrift_s, "rdrift_s");
        I(si1, si) <+ flicker_noise(KFNS * W * pow((ids / W), AFNS), BFNS ,"flicker");
    end else begin
        V(si, si1) <+ 0.0;
    end
end else begin
    V(s, si1) <+ 0.0;
    V(si1, si) <+ 0.0;
end
```

Current assign at source side.

```
if ((SHMOD != 0) && (RTH0 > 0.0)) begin
    Pdiss = devsign * sigvds * ids * V(di, si);
    if (RDSMOD != 2 && RDrainGeo > 0) begin
        if (RDSMOD == 1 && HVMOD == 1 && RDLCW > 0) begin
            Pdiss = Pdiss + V(d, di1) * V(d, di1) * gdpr + V(di1, di) * V(di1, di) * gdrift_d;
        end else begin
            Pdiss = Pdiss + V(d, di1) * V(d, di1) * gdpr;
        end
    end
    if (RDSMOD != 2 && RSourceGeo > 0) begin
        if (RDSMOD == 1 && HVMOD == 1 && RSLCW > 0) begin
            Pdiss = Pdiss + V(s, si1) * V(s, si1) * gspr + V(si1, si) * V(si1, si) * gdrift_s;
        end else begin
            Pdiss = Pdiss + V(s, si1) * V(s, si1) * gspr;
        end
    end
    Pwr(t) <+ delTemp1 * gth + ddt(delTemp1 * cth) - Pdiss;
end else begin
    Temp(t) <+ 0.0;
end
```

Updated Self heating model



- After the updated implementation, no spike is observed in drain current during transient simulation.

35.2021bug17: Default values of body-bias dependent parameters in HVMOD

- For backward compatibility default values of parameter DRB1 and DRB2 are changed to 0.0 in BSIM-BULK107.1.0

BSIMBULK107.1.0

```
`MPRnb( DRB1      ,0.0      ,""      ,"body-bias dependence in HVMOD" )
`MPRnb( DRB2      ,0.0      ,""      ,"body-bias dependence in HVMOD" )
```

36.2021bug18: Unit correction of source and drain resistances.

- In BSIM-BULK107.1.0 Beta2, source and drain resistance units are in ohm*m^WR, but the code uses WeffWRFactor = 1/Weff *1e6.

```
rdrift_d = rdstemphv * RDLCW * WeffWRFactor * T2D * T4;
```

- In BSIM-BULK107.1.0, units of resistances RDLCW, RSLCW, RDW, LRDW, WRDW, PRDW, RSW, LRDW, WRDW, PRDW, RSWMIN, LRSWMIN, WRSWMIN, PRSWMIN, RDWMIN, LRDWMIN, WRDWMIN, PRDWMIN, RDSWMIN, LRDSWMIN, WRDSWMIN, PRDSWMIN, LRDSW, WRDSW, PRDSW
  are changed to ohm*um^WR.

  Example:

BSIM-BULK107.0.0

```
`MPRco( RDLCW      ,100.0      ,"Ohm*m^WR"    ,0      ,inf      ,"'R'esistance of the 'D'rain side drift region at 'L'ow 'C'urrent" )
`MPRco( RSLCW      ,0          ,"Ohm*m^WR"    ,0      ,inf      ,"'R'esistance of the 'S'ource side drift region at 'L'ow 'C'urrent" )
```

BSIMBULK107.1.0

```
`MPRco( RDLCW      ,100.0      ,"Ohm*um^WR"   ,0      ,inf      ,"'R'esistance of the 'D'rain side drift region at 'L'ow 'C'urrent" )
`MPRco( RSLCW      ,0          ,"Ohm*um^WR"   ,0      ,inf      ,"'R'esistance of the 'S'ource side drift region at 'L'ow 'C'urrent" )
```

37.2021bug19: Operating point should include charge Qbdj_ext from external diode.

- In BSIM-BULK107.1.0 Beta3, charge due to external diode is not included in the operating point information.
- Now in BSIM-BULK107.1.0 we have updated the operating points information.

```
<   QB = devsign * (QBi + Qovb + Qbsj + Qbdj);
---
>   QB = devsign * (QBi + Qovb + Qbsj + Qbdj + Qbdj_ext);
```

Bsim-Bulk107.0.0

Bsim-Bulk107.1.0 Implementation

```
<   CAPBD  = -devsign * ddx(Qbdj, V(di));
---
>   CAPBD  = -devsign * (ddx(Qbdj, V(di)) + ddx(Qbdj_ext, V(ddbulk)));
```

Bsim-Bulk107.0.0

Bsim-Bulk107.1.0 Implementation

```
<       QD  = devsign * (QDi + Qovd - Qbdj);
---
>       QD  = devsign * (QDi + Qovd - Qbdj - Qbdj_ext);
```

Bsim-Bulk107.0.0

Bsim-Bulk107.1.0 Implementation

38.2021bug20: Changing lln to ln for junction capacitance calculation in VA code.

BSIM-BULK107.1.0 Beta3

```
Czbs       = CJS_t * ASeff;
Czbssw     = CJSWS_t * PSeff;
Czbsswg    = CJSWGS_t * Weffcj * NF;
czbs_p1    = pow(0.1, -MJS);
if (MJS == 1) begin
    czbs_p2    = 1.5 - lln(0.1);
end else begin
    czbs_p2    = 1.0 / (1.0 - MJS) * (1.0 - 0.05 * MJS * (1.0 + MJS) * czbs_p1);
end
czbssw_p1  = pow(0.1, -MJSWS);
if (MJSWS == 1) begin
    czbssw_p2   = 1.5 - lln(0.1);
end else begin
    czbssw_p2  = 1.0 / (1.0 - MJSWS) * (1.0 - 0.05 * MJSWS * (1.0 + MJSWS) * czbssw_p1);
end
czbsswg_p1 = pow(0.1, -MJSWGS);
if (MJSWGS == 1) begin
    czbsswg_p2   = 1.5 - lln(0.1);
end else begin
    czbsswg_p2 = 1.0 / (1.0 - MJSWGS) * (1.0 - 0.05 * MJSWGS * (1.0 + MJSWGS) * czbsswg_p1);
end
```

BSIMBULK107.1.0

```
Czbs       = CJS_t * ASeff;
Czbssw     = CJSWS_t * PSeff;
Czbsswg    = CJSWGS_t * Weffcj * NF;
czbs_p1    = pow(0.1, -MJS);
if (MJS == 1) begin
    czbs_p2    = 1.5 - ln(0.1);
end else begin
    czbs_p2    = 1.0 / (1.0 - MJS) * (1.0 - 0.05 * MJS * (1.0 + MJS) * czbs_p1);
end
czbssw_p1  = pow(0.1, -MJSWS);
if (MJSWS == 1) begin
    czbssw_p2   = 1.5 - ln(0.1);
end else begin
    czbssw_p2  = 1.0 / (1.0 - MJSWS) * (1.0 - 0.05 * MJSWS * (1.0 + MJSWS) * czbssw_p1);
end
czbsswg_p1 = pow(0.1, -MJSWGS);
if (MJSWGS == 1) begin
    czbsswg_p2   = 1.5 - ln(0.1);
end else begin
    czbsswg_p2 = 1.0 / (1.0 - MJSWGS) * (1.0 - 0.05 * MJSWGS * (1.0 + MJSWGS) * czbsswg_p1);
end
`JunCap(Czbs, Vbs_jct, PBS_t, MJS, czbs_p1, czbs_p2, Qbsj1)
`JunCap(Czbssw, Vbs_jct, PBSWS_t, MJSWS, czbssw_p1, czbssw_p2, Qbsj2)
`JunCap(Czbsswg, Vbs_jct, PBSWGS_t, MJSWGS, czbsswg_p1, czbsswg_p2, Qbsj3)
Qbsj = Qbsj1 + Qbsj2 + Qbsj3;
```

39.2021bug21: Clamping Nsat for non-saturation effect.

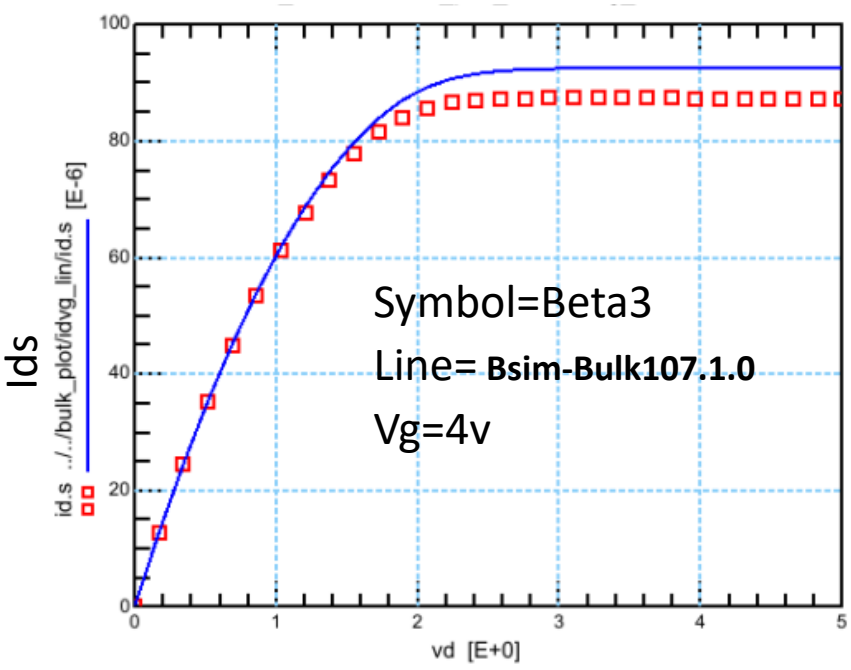
- In BSIM-BULK 107.1.0 Beta3, Nsat factor goes larger than 1, which causes Gds to be negative

```
// Non-saturation effect
 T0   = A1_t + A2_t / (qia + 2.0 * n * Vtm);
 DQSD = qs - qdeff;
 T1   = T0 * DQSD * DQSD;
 T2   = T1 + 1.0 - 0.001;
 T3   = -1.0 + 0.5 * (T2 + sqrt(T2 * T2 + 0.004));
 Nsat = 0.5 * (1.0 + sqrt(1.0 + T3));
```

- In BSIM-BULK 107.1.0, this issue is resolved by applying minimum smoothing function to Nsat, where max value of Nsat is 1

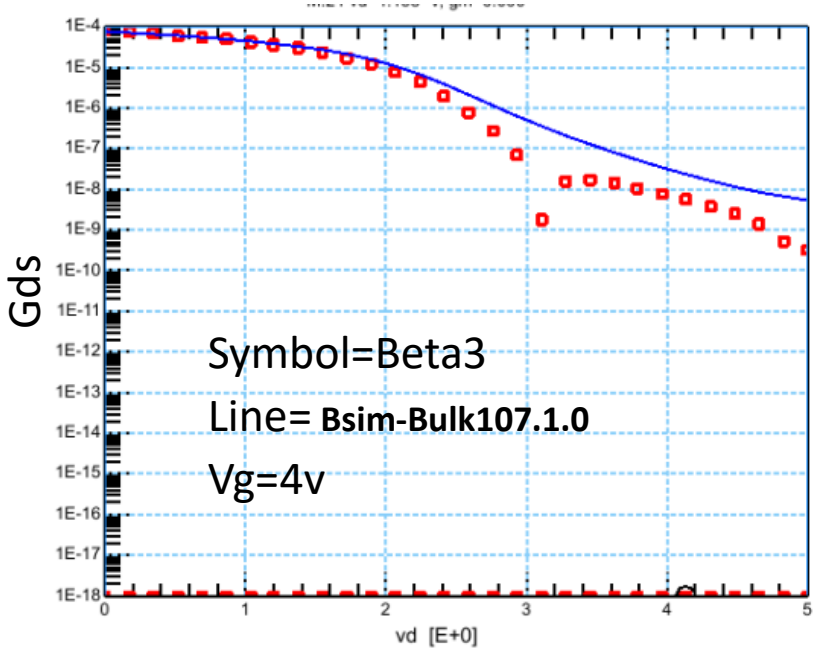
BSIMBULK107.1.0

```
// Non-saturation effect
 T0   = A1_t + A2_t / (qia + 2.0 * n * Vtm);
 DQSD = qs - qdeff;
 T1   = T0 * DQSD * DQSD;
 T2   = T1 + 1.0 - 0.001;
 T3   = -1.0 + 0.5 * (T2 + sqrt(T2 * T2 + 0.004));
 Nsat = 0.5 * (1.0 + sqrt(1.0 + T3));
 `Smooth2(Nsat, 1.0, 0.01, Nsat)
```



A1=150u

A2=250u



40. 2021bug22: Correction in JunCap macro

- In BSIM-BULK107.1.0 Beta4, Junction capacitance macro uses ‘lln’ function when arg > 0.1 and there is a missing term PBX_t * Czbx in Qbxj when MJX = 1.

- Now in BSIM-BULK107.1.0, ‘ln’ function is used instead of ‘lln’ function and missing term PBX_t * Czbx is also included in Qbxj.

BSIM-BULK107.1.0 Beta4:

```
if (MJX != 1.0) begin \
    if (MJX == 0.5) begin \
        sarg = 1.0 / sqrt(arg); \
    end else begin \
        sarg = lexp(-MJX * lln(arg)); \
    end \
    Qbxj = PBX_t * Czbx * (1.0 - arg * sarg) / (1.0 - MJX); \
end else begin \
    Qbxj = -lln(arg); \
end \
```

BSIMBULK107.1.0

```
if (MJX != 1.0) begin \
    if (MJX == 0.5) begin \
        sarg = 1.0 / sqrt(arg); \
    end else begin \
        sarg = lexp(-MJX * ln(arg)); \
    end \
    Qbxj = PBX_t * Czbx * (1.0 - arg * sarg) / (1.0 - MJX); \
end else begin \
    Qbxj = PBX_t * Czbx *(-ln(arg)); \
end \
```

41.2021bug23: VAMPyRE error in operating-point variable 'IDRIFTSATD'

- During the VAMPyRE run, BSIM-BULK107.0.0 shows error: Operating-point variable 'IDRIFTSATD' is only conditionally assigned a value.
- Now in BSIM-BULK107.1.0, the error is removed by initialising variable idrift_sat_d =0.0 and assigning operating point value outside HVMOD.

BSIM-BULK107.0.0:

```
    if (RDLCW != 0) begin
        Vdi1_abs = abs(V(di1, di));
        if (GADRIFT == 0) begin
            T9 = 1.0;
        end else begin
            `Smooth(Vdi1_abs - RDVDS, 0, 0.5, T10)
            T9 = 1.0 + T10 * GADRIFT;
        end
        idrift_sat_d = T11 * NDRIFTD * T9;
        T2 = 1.0 + Vsb_noswap / vbi_drift;
        `Smooth(T2, 0.0, 0.05, T2)
        T6 = (1.0 - DRB1 * ( sqrt(T2) - 1.0) - DRB2 * Vsb_noswap);
        `Smooth(T6, 0.0, 0.05, T6)
        idrift_sat_d = T6 * idrift_sat_d;
        T7 = rdstemphv * RDLCW * WeffWRFactor * T4;
        Vdrift_sat_d = idrift_sat_d * T7;
        delta_hv = pow(Vdi1_abs, 4 - MDRIFT) / (pow(Vdi1_abs, 4 - MDRIFT) + HVFACTOR * pow(Vdrift_sat_d, 4 - MDRIFT));
        T5 = pow(delta_hv, 1.0 / MDRIFT);
        T8 = T5 * Vdi1_abs / Vdrift_sat_d;
        `Smooth(T8, 0.0, 1.0e-3, T8)
        rdrift_d = T7 * pow(1.0 + pow(T8, MDRIFT), 1.0 / MDRIFT);
        IDRIFTSATD = idrift_sat_d;
    end
    if (RSLCW != 0) begin
        Vsi1_abs = abs(V(si, si1));
        idrift_sat_s = T11 * NDRIFTS;
        T7 = rdstemphv * RSLCW * WeffWRFactor * T4;
        Vdrift_sat_s = idrift_sat_s * T7;
        delta_hv = pow(Vsi1_abs,4-MDRIFT) / (pow(Vsi1_abs, 4-MDRIFT) + HVFACTOR * pow(Vdrift_sat_s, 4-MDRIFT));
        T5 = pow(delta_hv, 1.0 / MDRIFT);
        T8 = T5 * Vsi1_abs / Vdrift_sat_s;
        `Smooth(T8, 0.0, 1.0e-3, T8)
        rdrift_s = T7 * pow(1.0 + pow(T8, MDRIFT), 1.0 / MDRIFT);
    end
end
```

BSIMBULK107.1.0

```
    idrift_sat_d      = 0.0;

        Vdi1_abs = abs(V(di1, di));
        if (GADRIFT == 0) begin
            T9 = 1.0;
        end else begin
            `Smooth(Vdi1_abs - RDVDS, 0, 0.5, T10)
            T9 = 1.0 + T10 * GADRIFT;
        end
        idrift_sat_d = T11 * NDRIFTD * T9;
        T2 = 1.0 + Vsb_noswap / vbi_drift;
        `Smooth(T2, 0.0, 0.05, T2)
        T6 = (1.0 - DRB1 * ( sqrt(T2) - 1.0) - DRB2 * Vsb_noswap);
        `Smooth(T6, 0.0, 0.05, T6)
        idrift_sat_d = T6 * idrift_sat_d;
        T7 = rdstemphv * RDLCW * WeffWRFactor * T4;
        Vdrift_sat_d = idrift_sat_d * T7;
        delta_hv = pow(Vdi1_abs, 4 - MDRIFT) / (pow(Vdi1_abs, 4 - MDRIFT) + HVFACTOR * pow(Vdrift_sat_d, 4 - MDRIFT));
        T5 = pow(delta_hv, 1.0 / MDRIFT);
        T8 = T5 * Vdi1_abs / Vdrift_sat_d;
        `Smooth(T8, 0.0, 1.0e-3, T8)
        rdrift_d = T7 * pow(1.0 + pow(T8, MDRIFT), 1.0 / MDRIFT);
    end
    if (RSLCW != 0) begin
        Vsi1_abs = abs(V(si, si1));
        idrift_sat_s = T11 * NDRIFTS;
        T7 = rdstemphv * RSLCW * WeffWRFactor * T4;
        Vdrift_sat_s = idrift_sat_s * T7;
        delta_hv = pow(Vsi1_abs,4-MDRIFT) / (pow(Vsi1_abs, 4-MDRIFT) + HVFACTOR * pow(Vdrift_sat_s, 4-MDRIFT));
        T5 = pow(delta_hv, 1.0 / MDRIFT);
        T8 = T5 * Vsi1_abs / Vdrift_sat_s;
        `Smooth(T8, 0.0, 1.0e-3, T8)
        rdrift_s = T7 * pow(1.0 + pow(T8, MDRIFT), 1.0 / MDRIFT);
    end
end
IDRIFTSATD = idrift_sat_d;
```

42.2021bug24: Limit model parameter MDRIFT

BSIM-BULK107.0.0:

```
`MPRoo( MDRIFT     ,1.0      ,""   ,0     ,inf   ,"Non-linear resistance parameter")
```

BSIMBULK107.1.0

```
`MPRoo( MDRIFT     ,1.0      ,""   ,0     ,4   ,"Non-linear resistance parameter")
```

43. 2022bug1: Allow minr = 0

- In BSIM-BULK107.1.0 Beta5, model parameter "minr" uses MPRoz() macro which excludes zero.

BSIM-BULK107.0.0:

```
`MPRoz( minr   ),$simparam("minr", 1m) ,"Ohm"    ,"minr is the value below which the simulator expects elimination of source/drain
                                                   resistance and it will improve simulation efficiency without significantly altering the results.")
```

- Now in BSIM-BULK107.1.0, MPRcz() macro is used that includes zero.

BSIMBULK107.1.0

```
`MPRcz( minr   ),$simparam("minr", 1m) ,"Ohm"    ,"minr is the value below which the simulator expects elimination of source/drain
                                                   resistance and it will improve simulation efficiency without significantly altering the results.")
```

44. 2022bug2: Model parameter ABULK added for backward compatibility.

- In BSIM-BULK107.1.0 Beta1, ABULK parameter was removed, which makes the model backward incompatible with BSIM-BULK 107.0.0, that has been reported in Beta5 version.

BSIM-BULK107.1.0 Beta5:

For I-V

```
T1    = Leff / (Leff + sqrt(XJ_i * Xdep));
Abulk = 1.0 + (A0 * T1 - AGS * T1 * pow(qs,AGS1) * nVt) / (1.0 + KETA * Vbsx);
`Smooth(Abulk, 0.1, 0.0005, Abulk)
`Smooth(Vdsat - Vs, 0.0, 1.0e-3, Vdssat)
Vdssat    = Vdssat / Abulk;
```

For C-V

```
T1    = Leff / (Leff + sqrt(XJ_i * Xdep));
AbulkCV = 1.0 + (A0CV * T1 - AGSCV * T1 * qs * nVt) / (1.0 + KETACV * Vbsx);
`Smooth(AbulkCV, 0.1, 0.0005, AbulkCV)
VdssatCV     = VdssatCV / AbulkCV;
```

- ABULK parameter is added back, to make the model backward compatible.

BSIMBULK107.1.0

For I-V

```
if (A0 == 0 && AGS == 0)begin
    AbulkIV = 1.0;
end else begin
    T1  = Leff / (Leff + sqrt(XJ_i * Xdep));
    AbulkIV = 1.0 + (A0 * T1 - AGS * T1 * pow(qs,AGS1) * nVt) / (1.0 + KETA * Vbsx);
    `Smooth(AbulkIV, 0.1, 0.0005, AbulkIV)
end
`Smooth(Vdsat - Vs, 0.0, 1.0e-3, Vdssat)
Vdssat    = Vdssat / AbulkIV;
```

For C-V

```
if (A0CV == 0 && AGSCV == 0) begin
    AbulkCV = ABULK;
end else begin
    T1    = Leff / (Leff + sqrt(XJ_i * Xdep));
    AbulkCV = 1.0 + (A0CV * T1 - AGSCV * T1 * qs * nVt) / (1.0 + KETACV * Vbsx);
    `Smooth(AbulkCV, 0.1, 0.0005, AbulkCV)
end
VdssatCV     = VdssatCV / AbulkCV;
```

45. 2022bug3: KeyLetter x is added for Smartspice/ELDO simulator in QA perl script.

- In the latest QA package(v3.1.0), KeyLetter x in qa spec file for Smartspice/ELDO simulator is not required.

46. 2022bug4: Encountered divide by zero error with node (g,b)

- In Beta5, derivatives of T7 will create divide by zero error.

BSIM-BULK107.0.0:

```
if (A0 == 0 && AGS == 0)begin
    AbulkIV = 1.0;
 end else begin
    T1  = Leff / (Leff + sqrt(XJ_i * Xdep));
    AbulkIV = 1.0 + (A0 * T1 - AGS * T1 * pow(qs,AGS1) * nVt) / (1.0 + KETA * Vbsx);
    `Smooth(AbulkIV, 0.1, 0.0005, AbulkIV)
 end
 `Smooth(Vdsat - Vs, 0.0, 1.0e-3, Vdssat)
 Vdssat    = Vdssat / AbulkIV;
 T7        = pow((Vds / Vdssat), 1.0 / DELTA_t);
 T8        = pow(1.0 + T7, -DELTA_t);
 Vdseff    = Vds * T8;
 I(temp1) <+ Vdseff;
 vdeff     = (Vdseff + Vs) * inv_nVt;
 `BSIM_q(psip, phib_n, vdeff, gam, qdeff)
```

- In Bsim-Bulk107.1.0, 1e-6 is added in T7 equation of both IV and CV model to prevent divide by zero problem.

BSIMBULK107.1.0

```
if (A0 == 0 && AGS == 0)begin
    AbulkIV = 1.0;
end else begin
    T1  = Leff / (Leff + sqrt(XJ_i * Xdep));
    AbulkIV = 1.0 + (A0 * T1 - AGS * T1 * pow(qs,AGS1) * nVt) / (1.0 + KETA * Vbsx);
    `Smooth(AbulkIV, 0.1, 0.0005, AbulkIV)
end
`Smooth(Vdsat - Vs, 0.0, 1.0e-3, Vdssat)
Vdssat    = Vdssat / AbulkIV;
T7        = pow((Vds / Vdssat) + 1e-6 , 1.0 / DELTA_t);
T8        = pow(1.0 + T7, -DELTA_t);
Vdseff    = Vds * T8;
I(temp1) <+ Vdseff;
vdeff     = (Vdseff + Vs) * inv_nVt;
`BSIM_q(psip, phib_n, vdeff, gam, qdeff)
```

47.2022bug5: Typo Correction in gspr

- In Beta5, there was typo in source side conductance

BSIM-BULK107.1.0 Beta5:

```
if (RDSMOD != 2 && RSourceGeo > 0) begin
    gspr = 1.0 / Rsource;   // Note: gspr considers all fingers
    I(s, si1) <+ V(s, si1) * gdpr;
    I(s, si1) <+ white_noise(Nt * gdpr, "rd");
    if (RDSMOD == 1 && HVMOD == 1 && RSLCW > 0) begin
        gdrift_s = 1.0 / rdrift_s;
        I(si1, si) <+ V(si1, si) * gdrift_s;
        I(si1, si) <+ white_noise(Nt * gdrift_s, "rdrift_s");
        I(si1, si) <+ flicker_noise(KFNS * W * pow((ids / W), AFNS), BFNS ,"flicker");
```

- This typo has been corrected

BSIMBULK107.1.0

```
if (RDSMOD != 2 && RSourceGeo > 0) begin
    gspr = 1.0 / Rsource;   // Note: gspr considers all fingers
    I(s, si1) <+ V(s, si1) * gspr;
    I(s, si1) <+ white_noise(Nt * gspr, "rs");
    if (RDSMOD == 1 && HVMOD == 1 && RSLCW > 0) begin
        gdrift_s = 1.0 / rdrift_s;
        I(si1, si) <+ V(si1, si) * gdrift_s;
        I(si1, si) <+ white_noise(Nt * gdrift_s, "rdrift_s");
        I(si1, si) <+ flicker_noise(KFNS * W * pow((ids / W), AFNS), BFNS ,"flicker");
```

48.2022bug6: Divide by zero error when parameters DMCG / (DMCG+DMCI) is zero

- In Beta5, if DMCG / (DMCG+DMCI) = 0, then it will create divide by zero error in Rend equation.

BSIM-BULK107.0.0:

```
if ((DMCG + DMCI) == 0.0) begin \
    `STROBE("(DMCG + DMCI) can not be equal to zero"); \
end \
if (nuEnd == 0.0) begin \
    Rend = 0.0; \
end \
else begin \
    Rend = Rsh * Weffcj / (3.0 * nuEnd * (DMCG + DMCI)); \
end \
```

```
if (DMCG == 0.0) begin \
    `STROBE("DMCG can not be equal to zero"); \
end \
if (nuEnd == 0.0) begin \
    Rend = 0.0; \
end \
else begin \
    Rend = Rsh * Weffcj / (6.0 * nuEnd * DMCG); \
end \
```

- In Bsim-Bulk107.1.0, we have corrected this issue.

BSIMBULK107.1.0

```
if ((DMCG + DMCI) == 0.0) begin \
    `STROBE("(DMCG + DMCI) can not be equal to zero"); \
end \
if (nuEnd == 0.0 || (DMCG + DMCI) == 0.0) begin \
    Rend = 0.0; \
end else begin \
    Rend = Rsh * Weffcj / (3.0 * nuEnd * (DMCG + DMCI)); \
end \
```

```
if (DMCG == 0.0) begin \
    `STROBE("DMCG can not be equal to zero"); \
end \
if (nuEnd == 0.0 || DMCG == 0.0) begin \
    Rend = 0.0; \
end \
else begin \
    Rend = Rsh * Weffcj / (6.0 * nuEnd * DMCG); \
end \
```

49.2022bug7: Limit the layout-dependent parasitic model parameters

- In Beta5, layout dependent parameters are not bounded, they should be restricted to non-negative value

BSIM-BULK107.1.0 Beta5:

```
`MPRnb( DMCG      ,0.0     ,"m"     ,"Distance of mid-contact to gate edge" )
`MPRnb( DMCI      ,DMCG    ,"m"     ,"Distance of mid-contact to isolation" )
`MPRnb( DMDG      ,0.0     ,"m"     ,"Distance of mid-iffusion to gate edge" )
`MPRnb( DMCGT     ,0.0     ,"m      ,"Distance of id-contact to gate edge in test" )
```

- We have restricted layout dependent parameters greater than zero.

BSIMBULK107.1.0

```
`MPRcz( DMCG      ,0.0     ,"m"     ,"Distance of mid-contact to gate edge" )
`MPRcz( DMCI      ,DMCG    ,"m"     ,"Distance of mid-contact to isolation" )
`MPRcz( DMDG      ,0.0     ,"m"     ,"Distance of mid-diffusion to gate edge" )
`MPRcz( DMCGT     ,0.0     ,"m      ,"Distance of mid-contact to gate edge in test" )
```

50. 2022bug8: Code cleaning

- In Beta5, following typos were present (in red).

BSIM-BULK107.1.0 Beta5:

`MPRnb(A0 ,0.0 ,"" ,"Cofficient of depeletion width dependence in Abulk for Id-Vd flexibility")

`MPRnb(AGS ,0.0 ,"" ,"Cofficient of Source Charge dependence in Abulk for Id-Vd flexibility")

`MPRcz(KETA ,0.0 ,"" ,"Cofficient of back-bias dependence in Abulk for Id-Vd flexibility")

`MPRnb(A0CV ,A0 ,"" ,"Cofficient of depeletion width dependence in AbulkCV for Capacitance flexibility")

`MPRnb(AGSCV ,AGS ,"" ,"Cofficient of Source Charge dependence in AbulkCV for Capacitance flexibility")

`MPRnb(DMDG ,0.0 ,"m" ,"Distance of mid-ciffusion to gate edge")

`MPRoz(NEDGE ,1 ,"" ,"Flicker noise parameter for edge fet transitor")

`MPRcz(NOIA1_EDGE ,0.0 ,"" ,"Flicker noise fitting parameter in strong inversionfor edge fet transitor")

`MPRoz(NOIAX_EDGE ,1.0 ,"" ,"Flicker noise fitting parameter in strong inversionfor edge fet transitor")

`MPRcz(minr ,$simparam("minr", 1m) ,"Ohm" ,"minr is the value below which the simulator expects elimination of source/drain resitance and it will improve simulation efficiency without significantly altering the results.")

`MPRcz(JTWEFF ,0.0 ,"" ,"Trap assisted tunnelling current width dependence")

`MPRnb(DMCGT ,0.0 ,"m" ,"Distance if id-contact to gate edge in test")

// To enhance the fitting flexiblity for the gm/Id

// MNUD1 to enhance the fitting flexiblity for the gm/Id - similar approach used in BSIM-CMG

// Gate dielectric tunnelling current model parameters

- All the typos are corrected (in blue)

BSIMBULK107.1.0

```
`MPRnb( A0                  ,0.0       ,""          ,"Coefficient of depletion width dependence in Abulk for Id-Vd flexibility" )
`MPRnb( AGS                 ,0.0       ,""          ,"Coefficient of Source Charge dependence in Abulk for Id-Vd flexibility" )
`MPRcz( KETA                ,0.0       ,""          ,"Coefficient of back-bias dependence in Abulk for Id-Vd flexibility" )
`MPRnb( A0CV                ,A0        ,""          "Coefficient of depletion width dependence in AbulkCV for Capacitance flexibility" )
`MPRnb( AGSCV               ,AGS       ,""          ,"Coefficient of Source Charge dependence in AbulkCV for Capacitance flexibility" )

`MPRcz( KETACV              ,KETA      ,""          ,"Coefficient of back-bias dependence in AbulkCV for Capacitance flexibility" )
 `MPRnb( DMDG               ,0.0       ,"m"         ,"Distance of mid-diffusion to gate edge" )
`MPRoz( NEDGE               ,1         ,""          ,"Flicker noise parameter for edge fet transistor" )
`MPRcz( NOIA1_EDGE          ,0.0       ,""          ,"Flicker noise fitting parameter in strong inversion for edge fet transistor" )
`MPRoz( NOIAX_EDGE          ,1.0       ,""          ,"Flicker noise fitting parameter in strong inversion for edge fet transistor" )
`MPRcz( minr                ,$simparam("minr", 1m) ,"Ohm"    ,"minr is the value below which the simulator expects elimination of source/drain resistance and it will improve simulation efficiency without significantly altering the results.")
`MPRcz( JTWEFF              ,0.0       ,""          ,"Trap assisted tunneling current width dependence" )
`MPRnb( DMCGT               ,0.0       ,"m"         ,"Distance if mid-contact to gate edge in test" )
// To enhance the fitting flexibility for the gm/Id
// MNUD1 to enhance the fitting flexibility for the gm/Id - similar approach used in BSIM-CMG
// Gate dielectric tunneling current model parameters
```

- In BSIM-BULK107.1.0 Beta4, functions Min1 and Smooth1 are defined in the VA code, but they are never used elsewhere in the code.
- For the cleanliness of code, these functions are removed from the VA code of BSIM-BULK107.1.0.