CMOS BASELINE PROCESS IN THE UC BERKELEY MICROFABRICATION LABORATORY REPORT II

by

Laszlo Voros

Memorandum No. UCB/ERL M00/61

7 December 2000

CMOS BASELINE PROCESS IN THE UC BERKELEY MICROFABRICATION LABORATORY REPORT II

by

Laszlo Voros

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

CMOS Baseline Process
in the
UC Berkeley Microfabrication Laboratory
Report II.

Laszlo Voros

Electronics Research Laboratory
University of California, Berkeley

December, 2000

Abstract

This is the second report describing the baseline CMOS process supported by the Berkeley Microlab. The baseline process defines "standard" process modules for a twin-well, 1 μm CMOS technology with double poly-Si, double metal options. Process details are presented along with device characterization methodology and measurements. Process and device parameters are listed.

Table of Contents

    A   Detailed Process Flow

    B   Autoprober Computer Program

List of Figures and Tables

1. Introduction

The Microfabrication Laboratory at the University of California, Berkeley has been supporting silicon MOS technology from the time the present VLSI facility was opened in 1983. [1,2] In 1992 a CMOS baseline was formally established, which has been running continuously since then. The baseline specifies standard process modules for VLSI operations, and provides test circuits and starting point for various research groups such as the Berkeley Sensor and Actuator Center (BSAC) and the Berkeley Computer Aided Manufacturing group. [3,4,5] The baseline CMOS process also supports data for VLSI process control, which is ensured by monitoring 21 equipment and maintaining performance at baseline specifications.

The first baseline report [6] describes a 2 μm, n-well, double poly-Si, double metal CMOS process, which was subsequently developed into a twin-well, 1.3 μm, double poly-Si, double metal process. This report describes the latter, which shows good performance even for L=1 μm transistors. Included are the actual process and device parameters, simulation data, electrical measurements, and analytical test results. Process flow can be found in Appendix A and the source code for parameter extraction is listed in Appendix B.

2. Process Design for CMOS58-60 Run

The Microlab's 1.3 μm, twin-well, double poly-Si and double metal CMOS process was first developed in 1995. The process has 8 implantation steps and 14 lithography steps (Table 1. and Table 2.); however, the number of the masks applied is 12. The starting material is 24-36 Ω-cm p-type, <100> wafer, on which 0.8 μm (the smallest) N- and P-channel MOSFETs can be fabricated with punch-through implants. The process also contains N-field and P-field implants. The baseline process and device parameter targets are shown in Table 3.

2.1 Process Flow and Cross Sections

A brief processes flow with device cross sections are shown in Figure 1a-f. All process steps are accomplished in the Microlab except ion implantation, which is carried out at Ion Implant Services (Sunnyvale, CA). Detailed information can be found in the process outline in Appendix A, which includes the equipment used in the Microlab for CMOS processing.

2.2 Mask layout

The baseline utilizes a standard set of test structures for measurements, laid out in the scribe lane. [5] Participating groups are requested to include the scribe lane structures in their mask set, utilizing the drop-in area for their own devices, as shown in Figure 2. Runs CMOS58-60 used the full layout for BCAM application, as described in Ref.5.

Table 1. Ion Implantations

Process Step	Species	Energy (KeV)	Dose (cm^{-2})
N-Well Implant	Phosphorus	80	4×10^{12}
P-Well Implant	B11	80	3×10^{12}
P-Well Field Implant	B11	70	1.5×10^{13}
N-Well Field Implant	Phosphorus	40	3×10^{12}
N-Channel Punchthrough and Threshold Adjustment Implant	B11 B11	120 30	8×10^{11} 1.9×10^{12}
P-Channel Punchthrough and Threshold Adjustment Implant	Phosphorus B11	190 20	1×10^{12} 2.4×10^{12}
N+ S/D Implant	Arsenic	100	5×10^{15}
P+ S/D Implant	B11	20	5×10^{15}

Table 2. Lithography Steps

Lithography Step	Mask Name	Type	Field	Align. Step
N-well Formation	NWELL	Chrome	dark	
Active Area Definition	ACTIVE	Emulsion	clear	NWELL
P-Well Field Implant Photo	PFIELD (inv. of NWELL)	Emulsion	clear	ACTIVE
N-Well Field Implant Photo	NWELL	Chrome	dark	ACTIVE
N-Channel Punch-through and Threshold Adjustment Photo	PFIELD (inv. of NWELL)	Emulsion	clear	ACTIVE
P-Channel Punch-through and Threshold Adjustment Photo	PVT	Chrome	dark	ACTIVE
Gate Definition	POLY	Emulsion	clear	ACTIVE
Capacitor Formation	2nd POLY	Emulsion	clear	POLY
N+ S/D Photo	N+ S/D	Chrome	dark	POLY
P+ S/D Photo	P+ S/D	Emulsion	clear	POLY
Contact Photo	CONT	Chrome	dark	POLY
Metal Photo	METAL1	Emulsion	clear	CONT
VIA Photo	VIA	Chrome	dark	VIA
Metal2 Photo	METAL2	Emulsion	clear	METAL1

Fig.1-a. N-Well Formation

Phosphorus

Photoresist

Nitride

Oxide

N-well

p-type Si <100> 24-36 ohm-cm
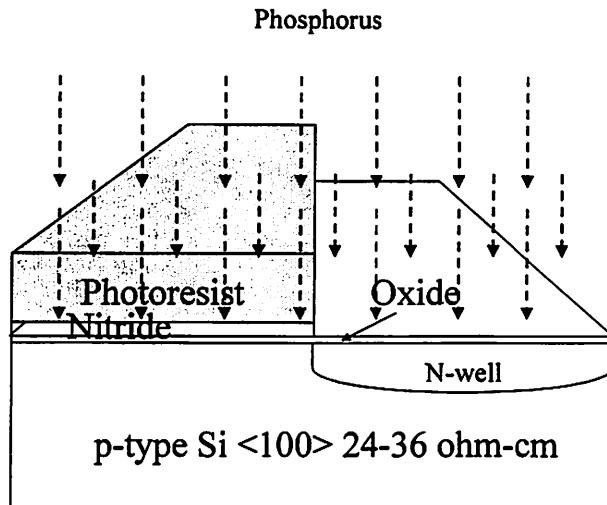
0. Bare Silicon Wafer
1. Initial Oxidation: 30nm
2. Nitride Deposition: 100nm
3. N-Well Photo
4. Nitride Etch
5. N-Well Ion Implantation: Phosphorus, 80 KeV, $4 \times 10^{12}/cm^2$

Fig. 1-b. P-Well and Active Area Formation

Boron

Photoresist

P+   P+   P+

P-well   N-well
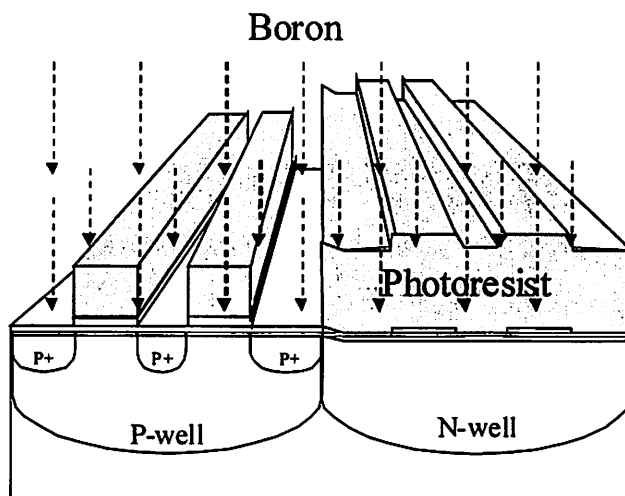
6. N-Well Cover Oxidation
7. Nitride Removal
8. P-Well Ion Implant
9. Well Drive In
10. Pad Oxidation/Nitride Deposition: 30nm/100nm
11. Active Area Photo
12. Nitride Etch
13. P-Well Field Implant Photo
14. P-Well Field Ion Implant: B11, 70 KeV, $1.5 \times 10^{13}/cm^2$

Fig. 1-c. LOCOS and Channel Implants

1) Phosphorus
2) B11

Photoresist

FOX   FOX   FOX   FOX   FOX

P-well   N-well

P-type Si
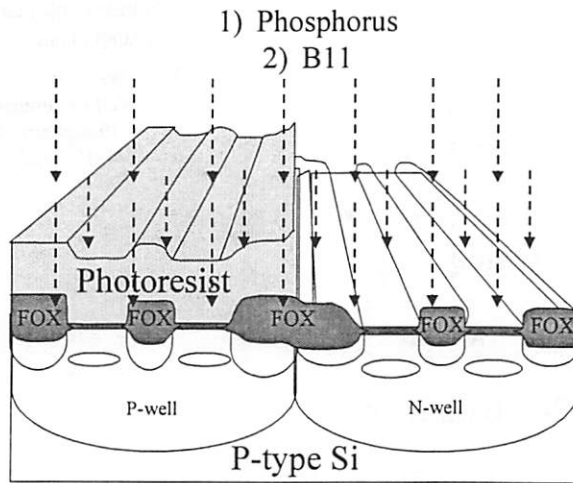
15. N-Well Field Implant Photo
16. N-Well Field Ion Implant:
    Phosphorus, 40 KeV,
    $3\times10^{12}/cm^2$
17. Locos Oxidation: 650nm
18. Nitride Removal and
    (Thin Oxide Removal)
19. Sacrificial Oxide: 20nm
20. P-Field Photo
21. N-Channel Punch-through:
    B11, 120KeV, $8\times10^{11}/cm^2$
    and Threshold Adjustment
    Implant: B11, 30 KeV,
    $1.9\times10^{12}/cm^2$
22. N-Field Photo
23. P-Channel Punch-through:
    Phosphorus, 190 KeV,
    $1\times10^{12}/cm^2$ and Threshold
    Adjustment Implant: B11,
    20 KeV, $2.4\times10^{12}/cm^2$

Fig. 1-d. Gate Formation

B11

Photoresist

FOX   FOX   FOX   FOX   FOX
     N+ N+ N+ N+   P+ P+   P+ P+

P-well   N-well
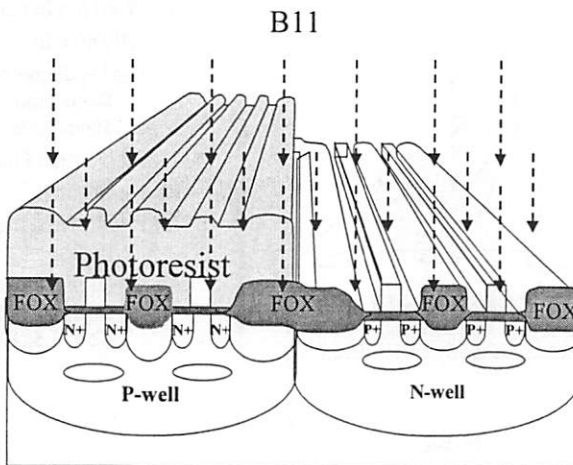
24. Gate Oxidation and
    Poly-Si Deposition
25. Gate Poly Photo
26. Gate Poly Etch
27. Capacitor Oxidation
28. Capacitor Poly Deposition
29. Capacitor Photo/ Etch
30. N+ S/D Photo
31. N+ S/D Implant
32. N+ S/D Anneal
33. P+ S/D Photo
34. P+ S/D Implant

Fig. 1-e. Contact and Metallization
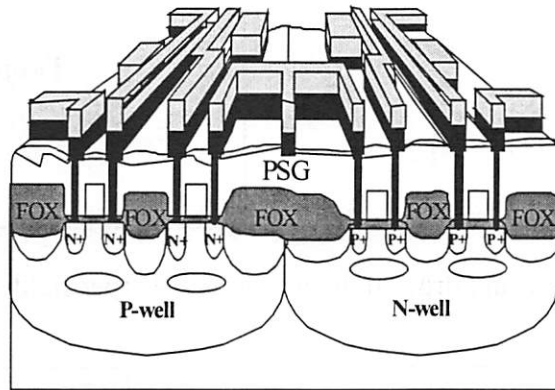
35. PSG Deposition &
    Post Densification
36. Contact Mask
37. Contact Etch
37. Metallization
38. Back Side Etch
39. Metallization (6000A)
40. Metal Mask
41. Al Etch
42. Sintering
43. Testing

Fig. 1-f. Metal2 and Sinter
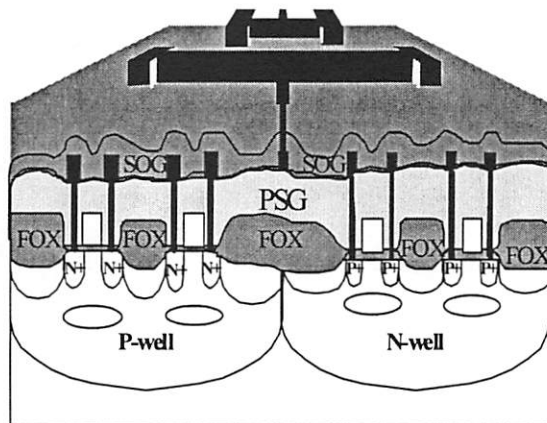
44. Planerization and
    Dielectric Film
    Deposition
45. VIA Photo
46. Etch VIA
47. Metal2 Metallization
48. Metal2 Photo
49. Etch Al
50. Sintering
51. Testing
 Final CMOS Device

Drop-in
8.12 mm
7.2 mm

Scribe Lane

Drop-in

Scribe Lane

Drop-in

Scribe Lane

Scribe Lane

Scribe Lane

Scribe Lane

Scribe Lane

Scribe Lane

Scribe Lane

8.0 mm

7.04 mm
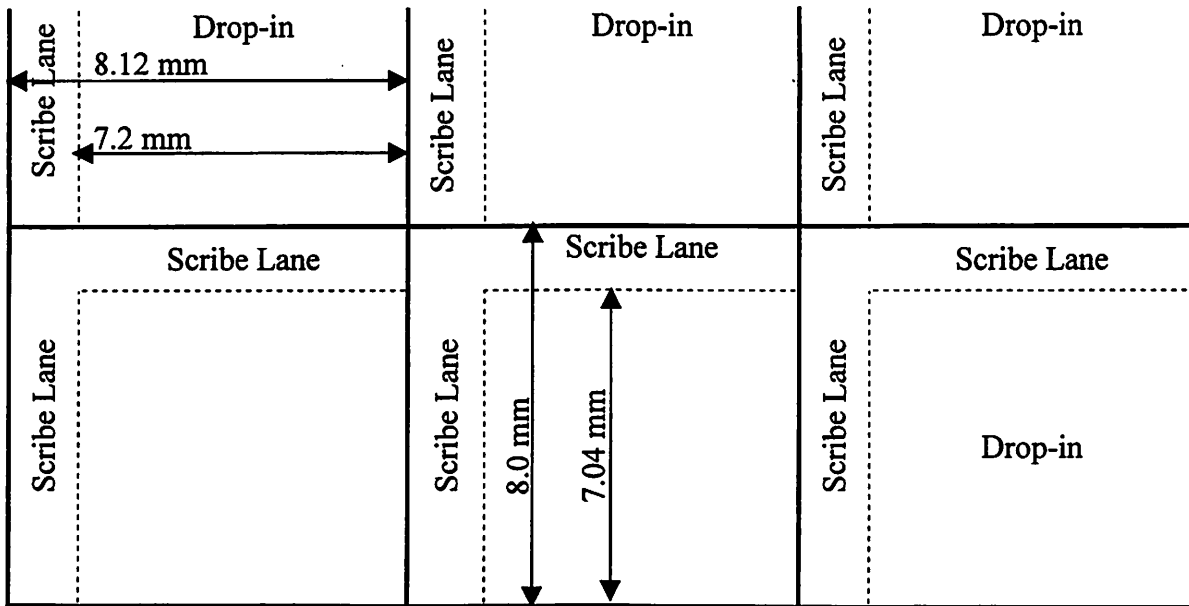
Drop-in

Drop-in

Configuration of scribe lane and drop-in die within the stepper field.

contact resistors		a.bridge	serp/cmb			large xtrs	elbows
	contact	chains	4x4 MOSFETs	serp/cmb	caps		
				align. marks	verniers		

split-cross bridge resistors

Individual MOSFETs

Drop-in
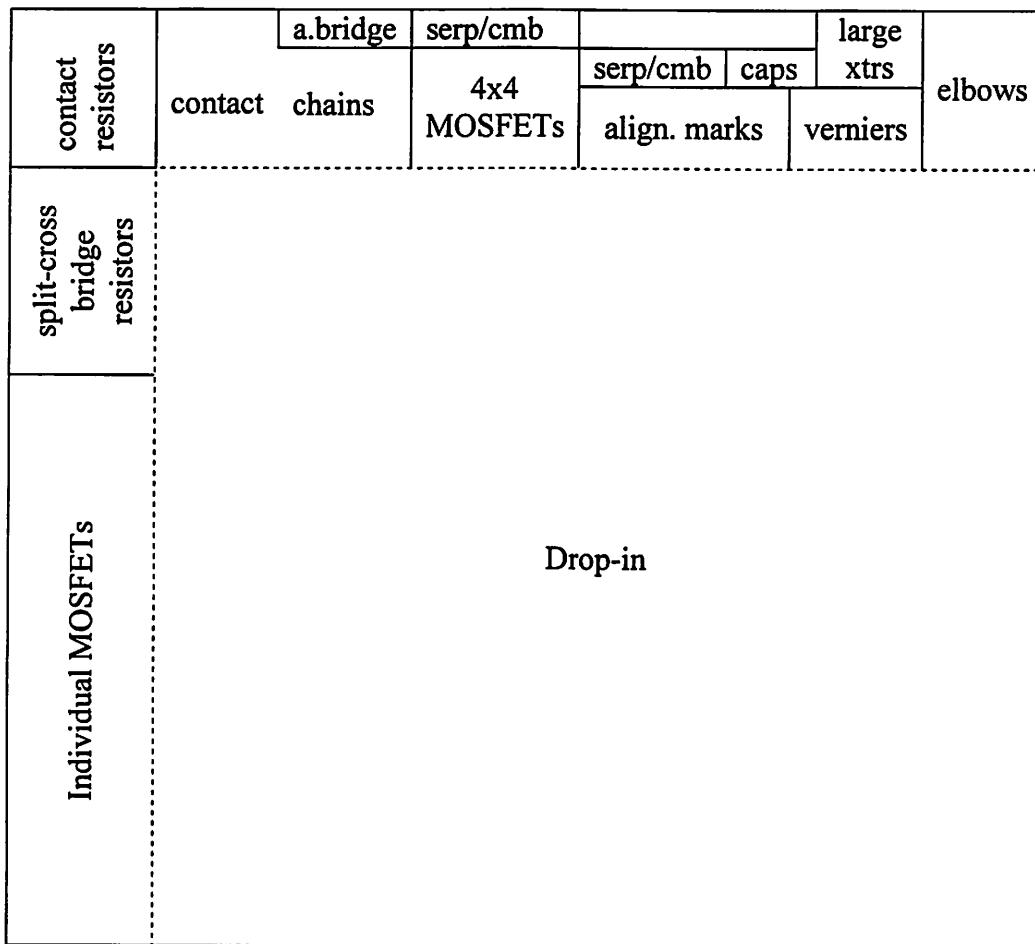
Fig. 2. Arrangement of test structures within the scribe lane Ref. 5.

7

2.3 Run Schedules

CMOS59 lot initiated and made it through step 23 (gate definition). At this point, because of staff changes, the lot was split into CMOS59a and CMOS59b for the rest of the steps, to optimize the new student and engineer's learning curve. CMOS59a, CMOS59b and CMOS60 baseline lots were completed in the October 1999 and July 2000 time frame.

Process timeline for these runs are shown in the Figure 3. CMOS59a process was delayed by equipment, the Al plasma etcher, which was down for about 3 weeks. CMOS60 run was delayed by 2 weeks due to a wafer stepper problem. However, the bottleneck usually is the implantation step, which takes about 1-2 weeks to complete by an outside service company.
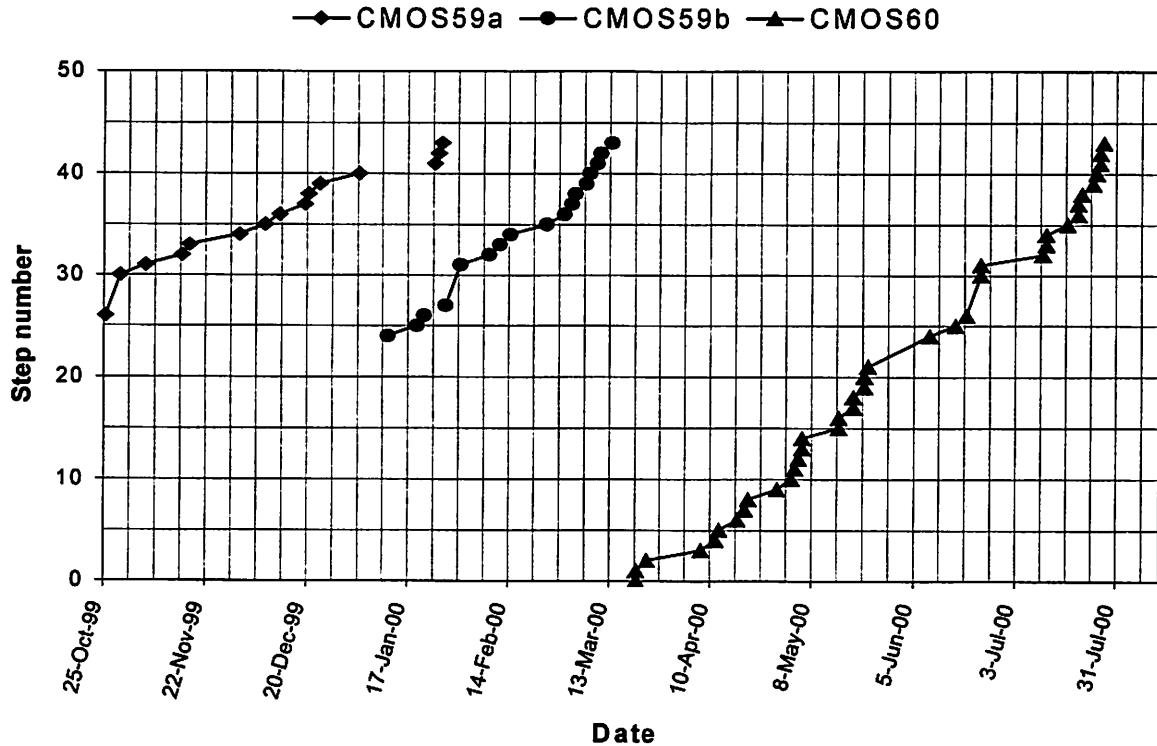
Figure 3. CMOS baseline schedule.

3. Test Results for CMOS58-CMOS60 Runs

The results of SUPREM3, Spreading Resistance Analysis (SRA), HP4145 Semiconductor Parameter Analyzer and the automatic probe testing are presented in this section.

3.1 SUPREM3 Simulations and Spreading Resistance Analysis

Spreading Resistance Analysis was requested for doping profiles of the N-channel, P-channel, N+ source-drain and P+ source-drain regions of the N and P-channel transistors. The SRA was performed by Solecon Laboratories (San Jose, CA). The results come from the lot CMOS59b. The scribe lane contains the appropriate structures for the measurements. Data can be seen in the Process and Device Parameters Table 3.

N- and P-channel Area

The simulated (SUPREM3) and measured doping profiles under the gate oxide for both types of transistors are shown in Figure 4. and 5. The simulated N-well junction depth is 3.6 μm (Figure 4.a), while the SRA junction depth (Figure 5.a) is shallower, 3 μm. For P-well junction depth, the simulated (Figure 4.b) and measured (Figure 5.b) results are matched. The simulated surface concentration and the concentration close to the surface region show a cumulative effect of the threshold, punch through and the well implantations (Figure 4.a and 4.b). On the SRA graphs only the net concentration can be seen (Fig 5. and 7.).

N+ and P+ Source-Drain

The simulated source-drain doping profiles are in Fig. 6. The experimental results are in Fig. 7. The SRA junction depth in both cases (N+ source-drain and P+ source-drain) are shallower by 0.3 μm than the simulated value. The SRA junction depth is 0.4 μm for P+ source-drain area and 0.12 μm for N+ source-drain area.

(a)   (b)
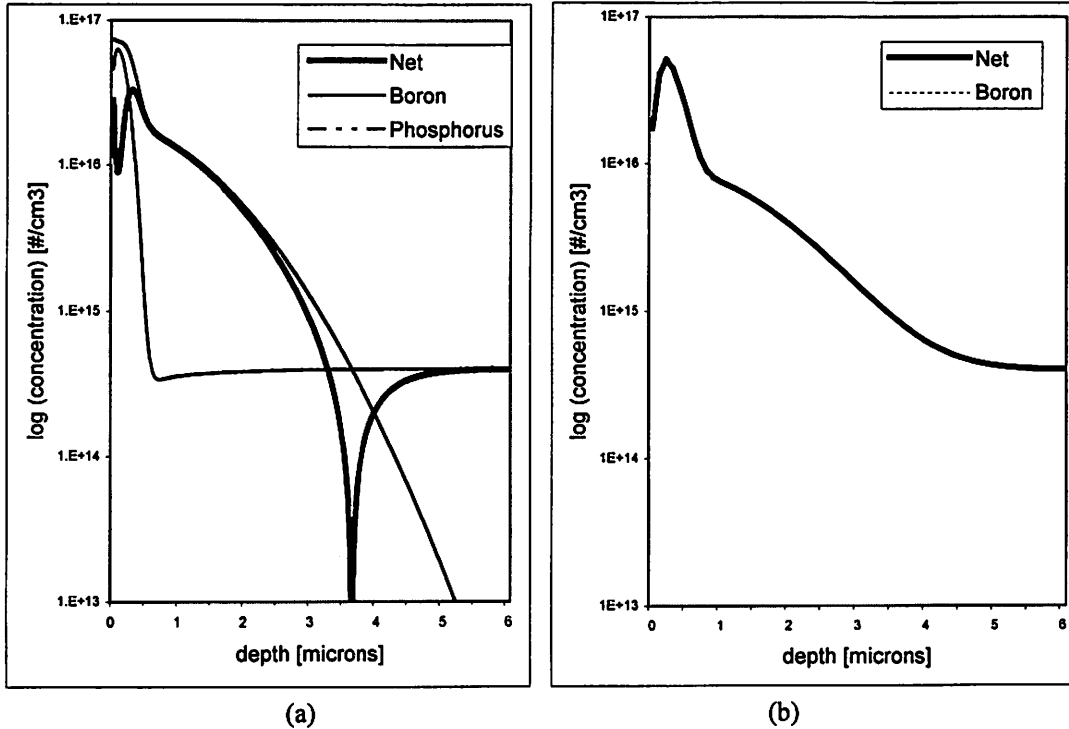
Fig.4 Suprem-3 simulation results for areas under the gate oxide of CMOS transistors.
(a) p-type transistor, (b) n-type transistor.

(a)   (b)
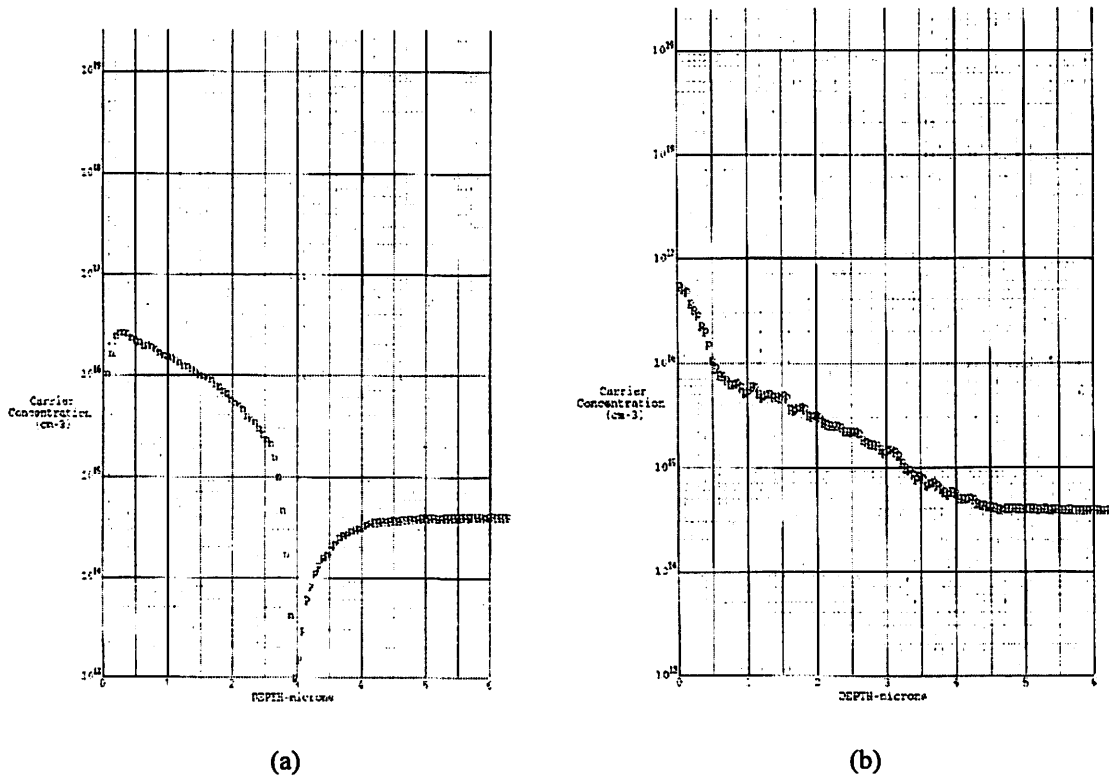
Fig.5 Experimental results for areas under the gate oxide of CMOS transistors
obtained from SRA. (a) p-type transistor, (b) n-type transistor.

10
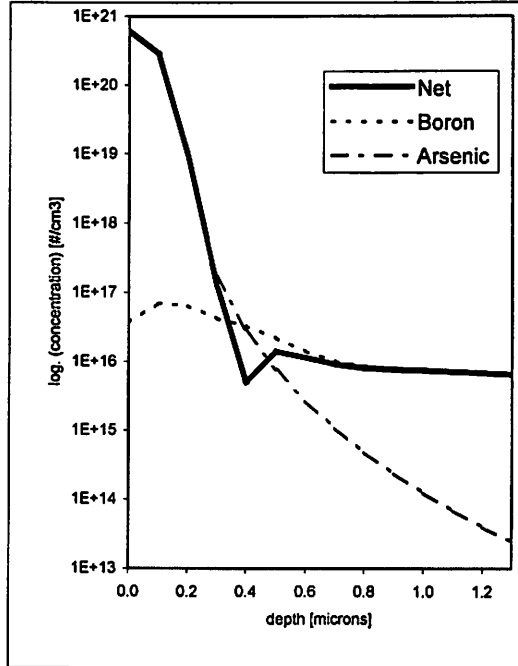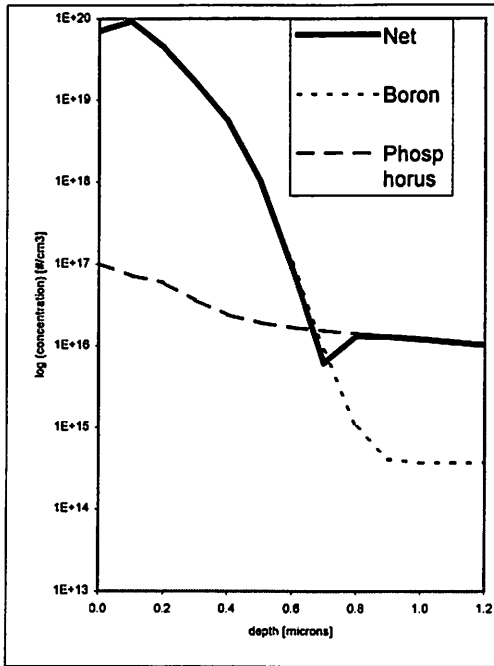
(a)   (b)

Fig 6 Suprem-3 simulation results for source-drain areas.
(a) P+ source-drain, (b) N+ source-drain.
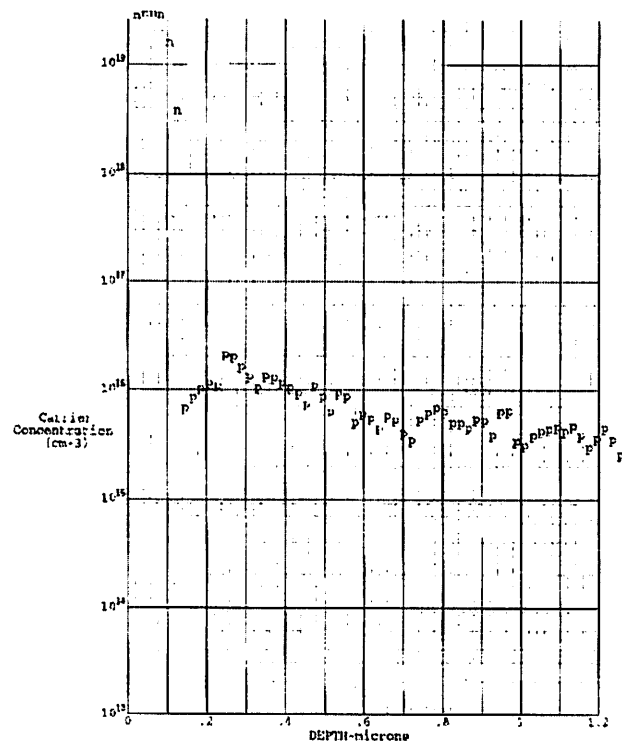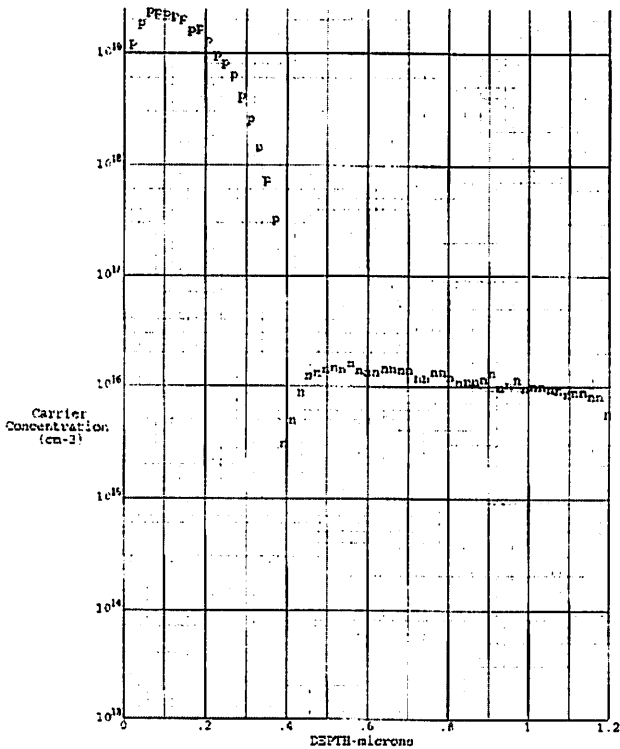
(a)   (b)

Fig.7 Experimental results for source-drain areas obtained from SRA.
(a) P+ source-drain, (b) N+ source-drain.

11

3.2 HP4145B Measurement Results

Manual measurements were taken using an HP4145 Semiconductor Parameter Analyzer, to display the I-V curves. NMOS and PMOS characteristics were obtained from CMOS59b process wafer#3. Although process was designed for 1.3 μm technology it can be seen that it works quite well for 1μm transistors (Figures 8 through 15). The features of the measured devices are W/L= 10/1 in Figures 8-15.
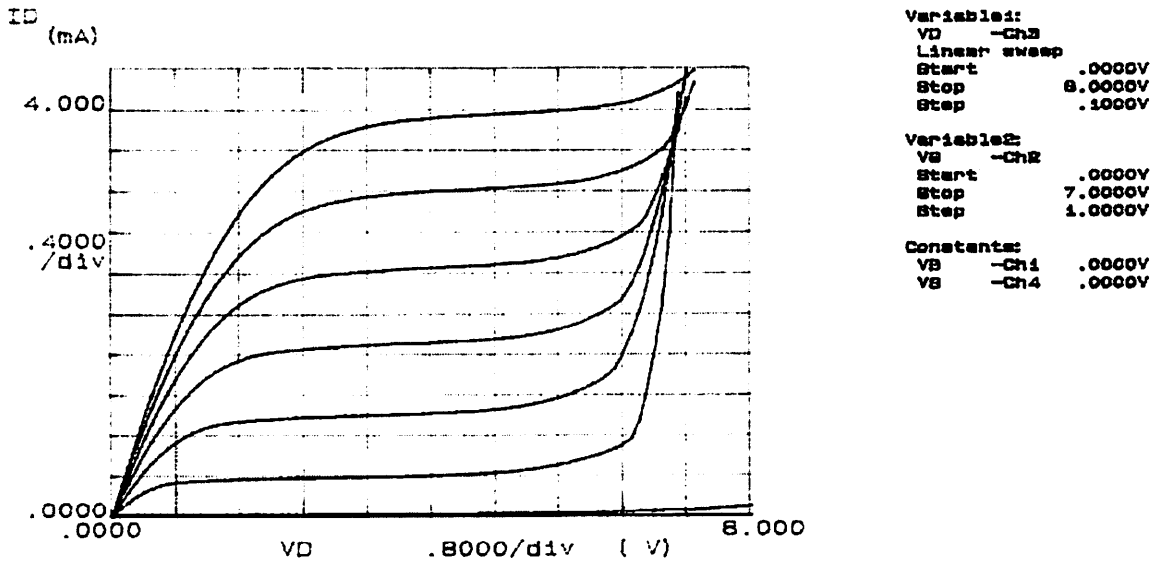
Fig. 8. NMOS (W/L=10/1) drain current vs. drain voltage characteristics.
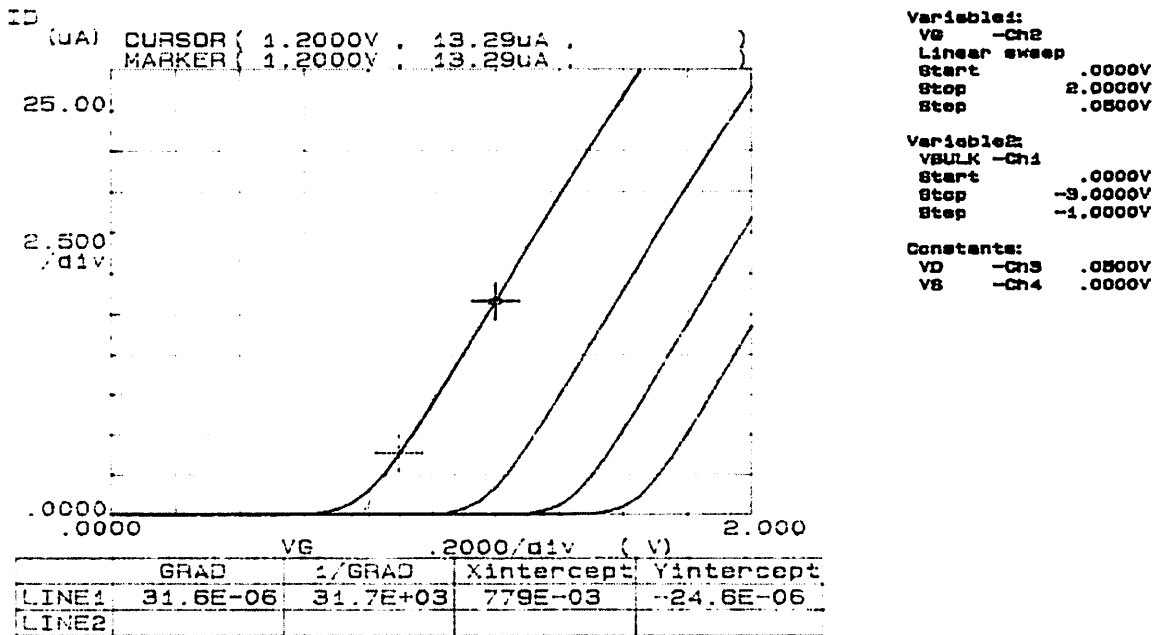
Fig. 9 NMOS (W/L=10/1) drain current vs. gate voltage at varying substrate bias.
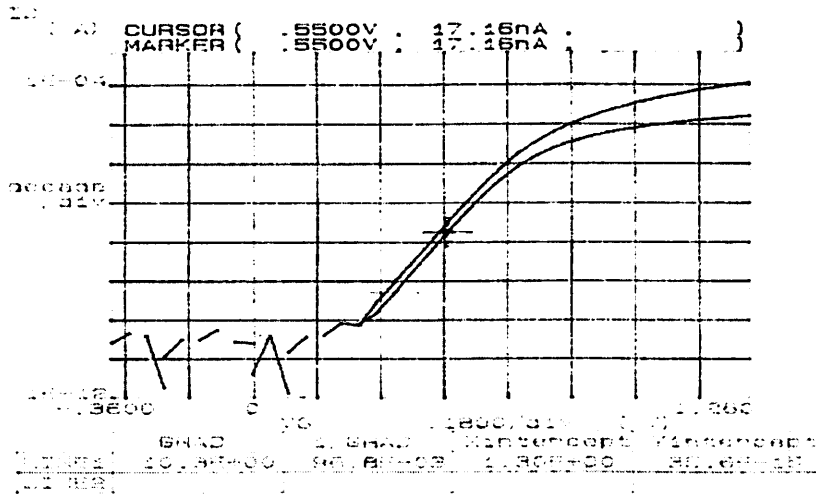
Fig 10. NMOS (W/L=10/1) subthreshold characteristics.
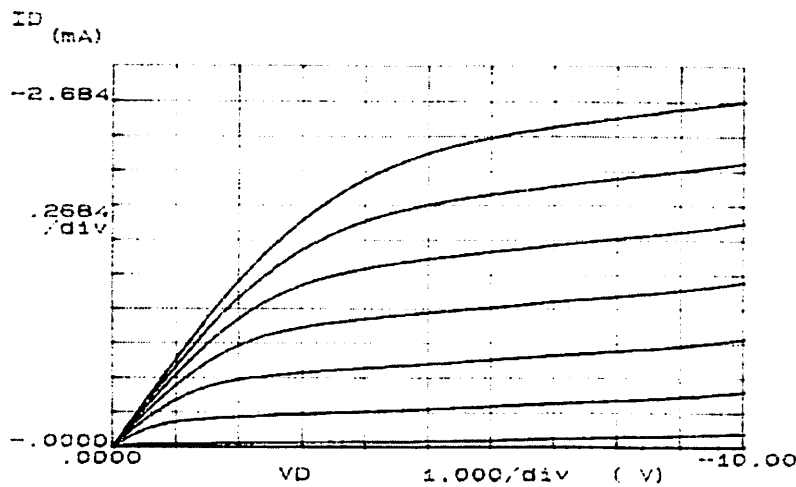
Fig 11. NMOS (W/L=10/1) drain current vs. gate voltage in saturation mode.

Fig. 12. PMOS (W/L=10/1) drain current vs. drain voltage characteristics.

ID
(uA)   CURSOR (-1.1500V , -7.372uA ,   )
       MARKER (-1.1500V , -7.372uA ,   )
-18.64

1.864
/div

.0000
.0000   VG   .2000/div (V)   -2.000

	GRAD	1/GRAD	Xintercept	Yintercept
LINE1	15.4E-06	64.8E+03	-672E-03	10.4E-06
LINE2				

Fig. 13 PMOS (W/L=10/1) drain current vs. gate voltage at varying substrate bias.
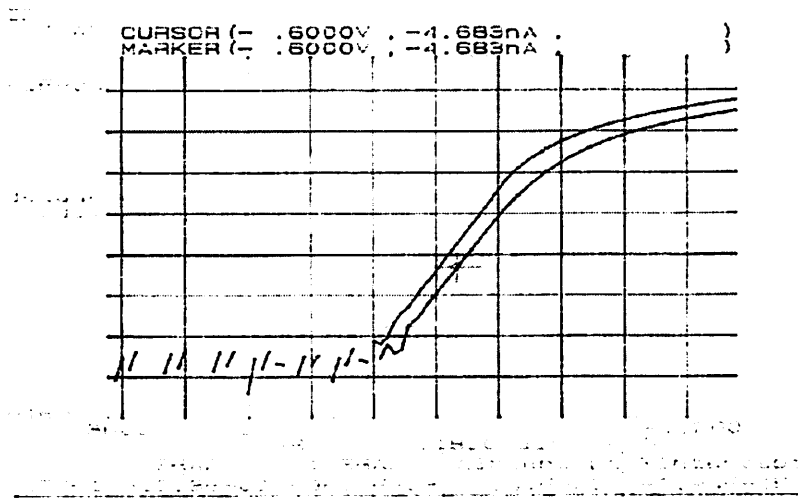
CURSOR (- .6000V , -4.683nA ,   )
MARKER (- .6000V , -4.683nA ,   )

Fig 14. PMOS (W/L=10/1) subthreshold characteristics.
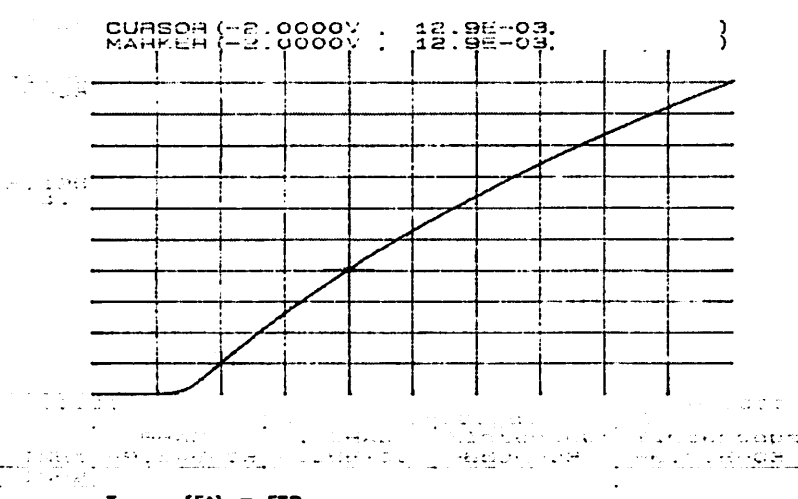
CURSOR (-2.0000V , 12.9E-03,   )
MARKER (-2.0000V , 12.9E-03,   )

I   (rA) - FID

Fig 15. PMOS (W/L=10/1) drain current vs. gate voltage in saturation mode.

3.3 Autoprober Measurement Results

An automatic probe station (autoprober) was utilized to make electrical measurements on each lot. The autoprober consists of an Electroglas probe station Model 2001X, an HP 4085A Switching Matrix, an HP4084 Switching Matrix Controller, an HP 4141 Source/Monitor (later HP 4142) and a UNIX workstation. The test structures are laid out such that the contact pads allowed for a 2x5 probe-card testing.

The autoprober enables the collection of large amounts of data for monitoring the process and for extracting device parameters. The source files for the SUNBASE control software can be found in: ~eglas/src/sunbase/ on the Microlab's main file server (silicon) and also are listed in Appendix B. The program extracts the threshold voltage, effective channel length/width, body factor, surface dopant concentration, drain induced barrier lowering, with the right setting of the prober.text and die.map files. Eighteen dies were tested on each wafer for threshold voltage. These measured dies were uniformly distributed across the wafer. Figures 16 and 17 show threshold voltage distributions of lots CMOS58-60.
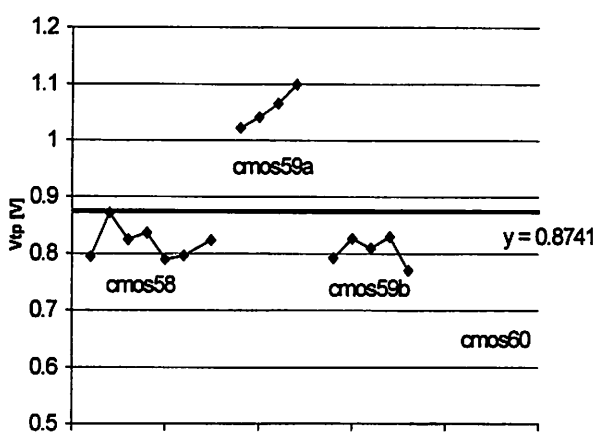
(a)
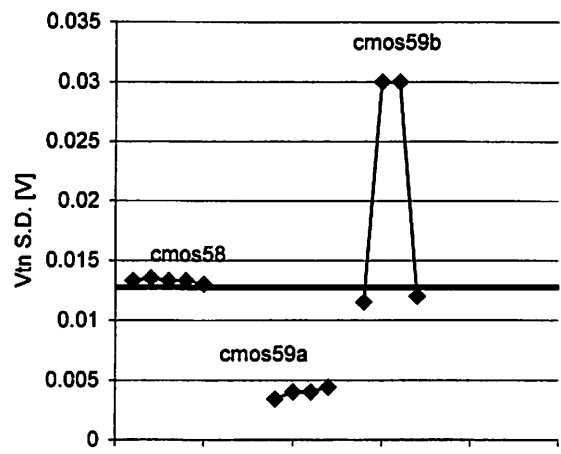
(b)

Fig. 16. (a) NMOS threshold voltage distribution, (b) standard deviation corresponding to data points in (a).
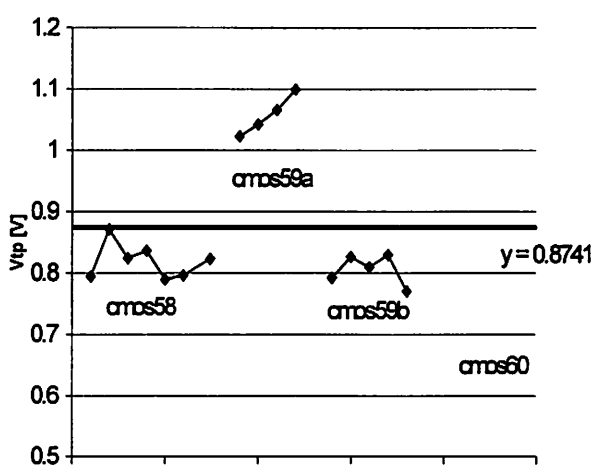
(a)
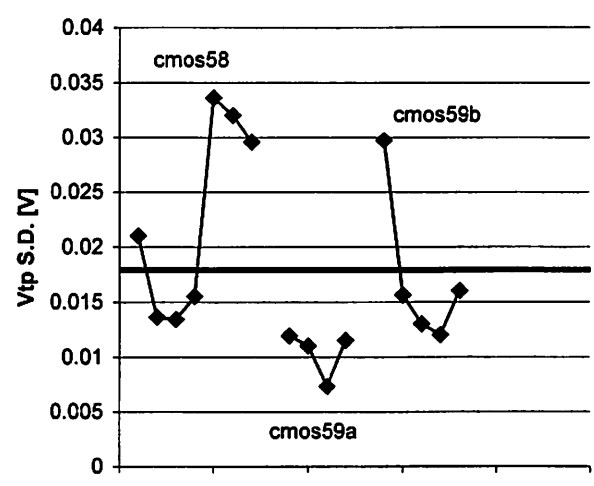
(b)

Fig.17. PMOS threshold voltage distribution, (b) standard deviation corresponding to data points in (a).

16

3.4 Design Parameters

Table 3. below is the summary of the various measurements and testing results from the process designed to produce L=1.3 μm devices. However, also L=1 μm devices also worked well. Numbers shown were extracted from measurements on L=1 μm devices.

Table 3. Process and Device Parameters Targets (from 10/1 device)

No.	Parameters	Units	NMOS	PMOS
1.	$V_{threshold}$	V	0.7	0.8
2.	Subthreshold Slope	mV/decade	99	125
3.	K'(μC_{ox}/2)	$\mu A/V^2$	18	9
4.	L_{eff}	μm	0.051	0.54
5.	W_{eff}	μm	0.522	0.047
6.	γ_1 (low V_{SB})	$V^{1/2}$	0.795	-0.446
7.	γ_2 (high V_{SB})	$V^{1/2}$	0.591	-0.371
8.	Surface dopant concentration	Atom/cm^3	5.68E16	1.79E16
9.	Substrate dopant concentration	Atom/cm^3	3.14E16	1.25E16
10.	T_{ox}	Angstrom	200 +/-20	200 +/-20
11.	X_j (S-D)	μm	0.14	0.4
12.	X_w (Well depth)	μm	4.7	3
13.	R_{diff} (sheet resistance)	Ω/square	190	171
14.	R_{poly} (sheet resistance)	Ω/square	33	33
15.	R_{well} (sheet resistance)	KΩ/square	0.5	1.5
16.	$R_{c\ M1-diff}$ (2μmx2μm)	Ω	46	47
17.	$R_{c\ M1-poly}$ (2μmx2μm)	Ω	8.8	8.8
18.	$|V_{th-field}|$	V	10	10
19.	$|$S-D Breakdown$|$	V	>7	>10

3.5 Method and measurement conditions

1. The threshold voltage was measured by the linear extrapolation method. The program used is attached in Appendix B.

2. Subthreshold slope numbers come HP 4145 measurements.

3. K' were extracted from measurements while the devices were at saturation (HP 4145B Semiconductor Parameter Analyzer).

4-5. Effective channel length and width were measured by autoprober, based on the resistance and conductance methods [7].

6-7. Gamma1, gamma2 were calculated by the autoprober program. The source files are in Appendix B.

8-9. Surface dopant concentration numbers come from autoprober measurements, which matched with SRA results. Since the dopant concentration is not vertically uniform in the ion-implanted channel region, gamma1 and gamma2 were extracted at low and high substrate bias [8]. Based on these results, dopant concentrations at the surface and substrate were obtained.

10. Gate oxide thickness was measured by Nanospec during the process.

11-12. The well depth and the source-drain depth data arise from the SRA graphs.

13-17. Parameters were measured on the automatic probe station using the electrical test structures described in Ref [5].

18-19. Measurements taken by the Semiconductor Parameter Analyzer.

4. Future work

We are in the process of implementing a deep UV lithography stepper (PAS 5500/90, ASML donation). This new stepper can resolve minimum feature sizes down to 0.3 μm. The new and more advance lithography in conjunction with other process modules such as: lightly doped drain (LDD), retrograde well and possibly salicide will enable to us develop new technology for the Microlab's baseline process. The baseline CMOS will be the test vehicle for qualifying the 6" process line upgrade in the Microlab.

5. References

[1] Katalin Voros and Ping K. Ko, *MOS Processes in the Microfabrication Laboratory*, Memorandum No. UCB/ERL M87/12, Electronics Research Laboratory, University of California, Berkeley (10 March 1987)

[2] Katalin Voros and Ping K.Ko, *Evolution of the Microfabrication Facility at Berkeley*, Memorandum No. UCB/ERL M89/109, Electronics Research Laboratory, University of California, Berkeley (22 September 1989)

[3] Andrea E. Franke, *Polycrystalline Silicon-Germanium Films for Integrated Microsystems*, PhD dissertation, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, (December 2000)

[4] Paul M. Krueger, *Tuning a Statistical Process Simulator to a Berkeley CMOS Process*, Memorandum No. UCB/ERL M88/82, Electronics Research Laboratory, University of California (15 December 1988)

[5] David Rodriguez, *Electrical Testing of a CMOS Baseline Process*, Memorandum No. UCB/ERL M94/63, Electronics Research Laboratory, University of California, Berkeley (30 August 1994)

[6] Shenqing Fang, *CMOS Baseline process in the UC Berkeley Microfabrication Laboratory*, Memorandum No. UCB/ERL M95/98, Electronics Research Laboratory, University of California, Berkeley (20 December 1995)

[7] D.K. Schroder, *Semiconductor Material and Device Characterization*, New York: John Wiley & Sons, Inc., 1990.

[8] Chenming Hu and Yuhua Cheng, *MOSFET modeling & BSIM3 User's Guide*, Kluwer Academic Publishers, pp. 80-81, 1999

Acknowledgments

Biography

Laszlo Voros received his Master of Science degree in engineering physics in 1999 from the Technical University of Budapest, in Hungary. He spent 1999 and 2000 at UCB as an associate specialist in the Microfabrication Laboratory. His primary assignments were in baseline process support and development, device testing.

Appendix A

Microlab CMOS Process
Version 5.0 (Nov. 1997)
1.3 um, twin-well, double poly-Si, double metal

0.0 Starting Wafers: 24-36 ohm-cm, p-type, <100>
        Control wafers: PCH, NCH.
        Scribe lot and wafer number on each wafer, including controls.
        Piranha clean and dip in sink8.
        Measure bulk resistivity (ohms-cm) of each on sonogage.
        R =

1.0 Initial Oxidation: target = 30 (+/- 5%) nm

   1.1 TCA clean furnace tube (tylan5), reserve tylan9.

   1.2 Standard clean wafers in sink6:
      Include PCH and NCH.
        piranha 10 minutes, 10/1 HF dip, spin-dry.

   1.3 Dry oxidation at 950 C (SGATEOX):
        60 min. dry O2 (Check the previous run result)
        20 min. dry N2
        Ox. time=
        measure oxide thickness on PCH, Tox=

2.0 Nitride Deposition (SNITC):
        Transfer wafers to tylan9 right after 1.3 and deposit
        Only include NCH.
        100 nm nitride.   Dep. time=
        measure nitride thickness on NCH, Tnit=

3.0 Well Photo: Mask NWELL (CWN chrome-df)
        (Control wafers are not included in any photoresist step)
        Standard I-line process:
        HMDS, spin (and soft bake), expose, post exposure bake,
        develop, inspect, descum and hard bake.

4.0 Etch: Plasma etch nitride in lam1.
        Recipe: Power:
        Actual Etch Time: Overetch:
        Check the oxide thickness on each work wafer:

5.0 N-Well Implant: phosphorus, 4E12/cm2, 80 KeV. Include PCH.

6.0 N-Well Cover Oxidation:

   6.1 TCA clean furnace tube (tylan2).

1

6.2 Remove PR in O2 plasma and clean wafers in sink8.

6.3 Standard clean wafers in sink6, include PCH and NCH.

6.4 Well cover oxidation at 950 (NWELLCVR):
      30 min. dry O2
    175 min. wet O2
      30 min. dry O2
      20 min. N2

7.0 Nitride Removal, include NCH

7.1 Dip in 10:1 BHF for 40 sec to remove thin oxide on top of Si3N4.

7.2 Etch nitride off in boiling phosphoric acid (sink7).
    Measure Tox in n-well on work wafers.

8.0 P-Well Implant: B11, 3E12/cm2, 80 KeV. Include NCH

9.0 Well Drive-In:

9.1 TCA clean furnace tube (tylan2).

9.2 Standard clean wafers in sink8 and 6. Include PCH and NCH.

9.3 Well drive at 1120 C (WELLDR):
      60 min. temperature ramp from 750 C to 1120 C
    240 min. dry O2
    300 min. N2

Measure oxide thickness on two control wafers.
    tox (PCH)=   tox (NCH) =

9.4 Strip oxide in 5:1 BHF.
    Measure Rs on PCH and NCH.
      Rs (PCH) =   Rs(NCH) =

10.0 Pad Oxidation/Nitride Deposition:
    target = 30 (+6) nm SiO2 + 100 (+10) nm Si3N4

10.1 TCA clean furnace tube (tylan5). Reserve tylan9.

10.2 Standard clean wafers. Include PCH, NCH.

10.3 Dry oxidation at 950 C (SGATEOX):

```
           ~1 hr. dry O2
           30 minutes dry N2 anneal.
           Measure the oxide thickness on NCH
           Tox=
```

10.4 Deposit 100 (+10) nm of Si3N4 immediately (SNITC):
      Only include PCH.
      approx.time = 22 min., temp.= 800 C.
      Measure nitride thickness on PCH.
              Tnit =

11.0 Active Area Photo: Mask ACTV (ACTV emulsion-cf)
     Standard I-line process.

12.0 Nitride Etch:
         Plasma etch nitride in lam1.   Recipe:
         Power: Time: Overetch:
     Measure Tox on each work wafer. (2 pnts measurement).
     Do not remove PR. Inspect.
     Measure PR thickness covering active area.   tpr=
     PR must be >800 nm. Hard bake again for >2hrs at 120 C.

13.0 P-Well Field Implant Photo: Mask PFIELD (CWNI emulsion-cf)
     (Reversed NWELL mask)
     Standard I-line process. (Second photo)
     N-Well area is covered with PR.

14.0 P-Well Field Ion Implant: B11, 70 KeV, 1.5E13/cm2.

15.0 N-Well Field Implant Photo: Mask NWELL (CWN chrome-df)

15.1 Remove PR in plasma O2. Clean wafers in sink8.

15.2 Standard I-line process.

16.0 N-Well Field Ion Implant: phosphorus, 40 KeV, 3E12.

17.0 Locos Oxidation: target = 650 nm

17.1 TCA clean furnace tube (tylan2).

17.2 Remove PR in O2 plasma and piranha clean wafers.
      Standard clean wafers; dip in BHF 25:1 for 5-10 sec.
      Include PCH, NCH.

17.3 Wet oxidation at 950 C (SWETOXB):
         5 min. dry O2
         4 hrs. 40 min. wet O2

```
                    5 min. dry O2
                    20 min. N2 anneal
          Measured tox on 3 work wafers.  Tox=
```

18.0 Nitride Removal, include PCH.

   18.1 Dip in 10:1 BHF for 60 sec to remove thin oxide on top of Si3N4.

   18.2 Etch nitride off in phosphoric acid at 145 C (sink7).

19.0 Sacrificial Oxide: target = 20 (+/- 2) nm

   19.1 TCA clean furnace tube (tylan5).

   19.2 Standard clean wafers, include NCH and PCH.
         Dip in 10:1 BHF until PCH and NCH dewet.

   19.3 Dry oxidation at 950 C (SGATEOX):
              30 minutes dry O2
              30 minutes N2 anneal
         Measure Tox on PCH and NCH.   Tox=

20.0 N-Channel Punchthrough and Threshold Adjustment Photo: Mask PFIELD
     (CWNI emulsion-cf).

     Standard I-line process.

21.0 N-Channel Punchthrough and Threshold Adjustment Implant. Include NCH.
        1) B11, 120 KeV, 8E11/cm2.
        2) B11, 30 KeV, 1.9E12/cm2.

22.0 P-Channel Punchthrough and Threshold Adjustment Photo: Mask PVT
     (PVT chrome-df).

     Remove PR in plasma O2 and clean wafers in sink8.
     Standard I-line process.

23.0 P-Channel Punchthrough and Threshold Adjustment Implant. Include PCH.
        1) Phosphorus, 190 KeV, 1E12,
        2) B11, 20 KeV, 2.4E12.

24.0 Gate Oxidation/Poly-Si Deposition:
        target = 20 (+/- 2.0) nm SiO2 + 450 (+/- 40) nm poly-Si

   24.1 TCA clean furnace tube (tylan5).
         Reserve poly-Si deposition tube (tylan11).

24.2   Standard clean wafers, include PCH, NCH,
       Tox (prime P<100>), and one Tpoly1 monitoring wafers.

24.3   Dip off sacrificial oxide in 10:1 HF
       until NCH and PCH dewet (approx. 1 min).

24.4   Dry oxidation at 950 C (SGATEOX):
           30 min dry O2 (Check previous run result)
           30 min N2 anneal.

24.5   Immediately after oxidation deposit 450 nm of phos.doped
       poly-Si (SDOPOLYI).
           only include Tpoly1.
             approx.time = 2 hr. 20 min., temp.= 610 C
           (Check previous run result)

24.6   Measurements
           a) Measure oxide thickness on Tox, PCH and NCH.
           b) Measure Dit and Qox on Tox.
           c) Strip oxide from PCH and NCH, and measure the sheet
resistivity.
           d) Measure poly thickness on Tpoly1.
       PCH and NCH proceed to step 27.2.
       Tpoly1 proceeds to step 32.3.

25.0 Gate Definition: Mask POLY (emulsion-cf)
       Standard I-Line process.

26.0 Plasma etch poly-Si

26.1 Etch poly in Lam4 (Recipe: 400):
       Pwr: Ave. etch time: Overetch:

26.2 Measure Tox in S/D area of each work wafer (2 pnts measurement).

26.3 Measure channel length using 1.0um gate.
           CD =

27.0 Reoxidation and Capacitor Formation:
       (If no capacitor is requested, skip step 27 through 29.2.)

27.1   TCA clean furnace tube (tylan2). Reserve tylan12 and tylan11.

27.2   Standard clean wafers, including PCH, NCH, and
       two monitoring wafers, one for dry oxidation (Tpoly2) and
       one for LTO.
       From here on: only 10 sec dip in 25/1 H2O/HF after piranha.

27.3 Dry oxidation at 900 C (SDRYOXB):
        30 min dry O2
        20 min N2 anneal.
    Measure oxide thickness on Tpoly2:
        Tpoly2 proceeds to Step 27.5.
    PCH proceeds to Step 34 and NCH proceeds to Step 31.

27.4 1) Run a coating and monitoring LTO in tylan12 to get
        dep rate. Use recipe VDOLTOC and set 0 doping.
      2) Deposit LTO for the desired oxide thickness.
       3) Measure LTO thickness on monitoring wafer:

27.5 Second poly-Si deposition: immediately after oxidation
        deposit 450 nm of phos.doped poly-Si (SDOPOLYH):
        only include Tpoly2.
           approx.time = 2 hr. 18 min, temp.= 610 C.
      Measure second poly thickness on Tpoly2:
        Tpoly2 proceeds to step 32.3.

28.0 Capacitor Photo: Mask CAP-CE (CAP emulsion-cf)
        Standard I-Line process.

29.0 Plasma etch poly-Si:

29.1 Etch 2nd poly in Lam4 (Recipe: 400):
      Power: Actual etch time: Overetch:

29.2 Measure Tox in S/D area on each work wafer.
      Remove PR in O2 plasma.
      Piranha clean wfrs in sink8.
          Dehydrate wfrs in oven for > 30 min. at 120 C.

30.0 N+ S/D Photo: Mask N+S/D (NSD chrome-df)
        Standard I-line process.

31.0 N+ S/D Implant: Arsenic, 100 keV, 5E15/cm2,   include NCH.

32.0 N+ S/D Anneal

32.1 TCA clean furnace tube (tylan7).

32.2 Remove PR in O2 plasma and piranha clean wafers
        in sink8 (no dip here).

32.3 Standard clean wafers in sink6, incl. PCH, NCH, Tpoly1,
        and Tpoly2.

32.4 Anneal in N2 at 900 C for 30 min (N2ANNEAL).

32.5 Strip oxide from NCH, Tpoly1, and Tpoly2.
Measure Rs of N+ S/D implant: Rs(NCH)=
Measure Rs of poly1 on Tpoly1: Rs(Tpoly1)=
Measure Rs of poly2 on Tpoly2: Rs(Tpoly2)=

33.0 P+ S/D Photo: Mask P+S/D (PSD emulsion-cf)
Standard I-line process.

34.0 P+ S/D Implant: B11, 20 keV, 5E15/cm2, include PCH.

35.0 PSG Deposition and Densification: target = 700 nm

35.1 Remove PR in O2 plasma and clean wafers in sink8 (no dip).

35.2 Standard clean wafers in sink6 (10 sec dip).
Include one PSG monitoring wafer.

35.3 Deposit 700 nm PSG, PH3 flow at 10.3 sccm (SDOLTOD).
approx.time = 22 min. (check current dep. rate)
temp. = 450 C

35.4   Densify glass in tylan2 at 900 C, immediately after
PSG deposition (PSGDENS). Include PSG control.
5 min dry O2
20 min wet O2
5 min dry O2

Measure tPSG (using PSG control and working wafers):
N+ region Tox =
P+ region Tox =
Etch oxide on PCH.
Measure Rs of P+ S/D implant: Rs(PCH)=

35.5 Do wet oxidation dummy run afterwards to clean tube:
1 hr wet oxidation at 950 C (SWETOXB).

36.0 Contact Photo: Mask CONT (CONT chrome-df)
Standard I-Line process.

37.0 Contact Plasma Etch in lam2:
Recipe: Power: Etch time: Overetch:

38.0 Back side etch:

38.1 Remove PR in O2 plasma, piranha clean wafers in sink8 (no dip).
Dehydrate wafers in oven at 120 C for >30 min.

38.2 Etch backside:
  (PCH and NCH can be included in b), c) and d).
  a) Spin PR on front side, hard bake.
  b) Dip off oxide (PSG) in 5:1 BHF.
  c) Etch poly-Si (poly2 thickness) in lam4.
  d) Etch oxide off in 5:1 BHF (cap. ox. thickness).
  e) Etch poly-Si (poly1 thickness) in lam1.
  f) Final dip in BHF until back dewets.
  g) Remove PR in PRS2000, piranha clean wfrs in sink8
     (no dip).

39.0 Metallization: target = 600 nm
          Stnd clean wfrs and do a 30 sec. 25/1 H2O/HF dip just
          before metallization.
          Sputter Al/2% Si on all wafers in CPA.

40.0 Metal Photo: Mask METAL1-CM (M1 emulsion-cf)
      Standard I-line process.

41.0 Plasma etch Al in Lam3.
      Remove PR in PRS2000 or technics-c.   tAl=
      Probe test devices.

42.0 Sintering: 400 C for 20min in forming gas (tylan13).
          No ramping, use SINT400 program.

43.0 Testing:
          1.0 um N- and P-channel devices, capacitors and inverter
          Measure the sheet resistivities of PCH and
          NCH on prometrix.

44.0 Planerization and Dielectric Film Deposition:

44.1 PECVD thin oxide (50 nm) in technics-B:
     N2O: 54.0, Silane: 14.0, Pwr: 15 W, Pressure: 360-420 mT.
     ~5 min. Measure Tox on dummy wafers.

44.2 SOG coating on the Headway spinner at 3000 rpm.

44.3 SOG cure:
     a) Oven in Y2, 120 C, 30 min.
     b) Oven in R1, 200 C, 30 min.
     c) Tylan14 (SVANNEAL): 400oC, 30 min.
     d) Measure Tox and refractive index on dummy wafers.

44.4 ECR thick oxide (900 nm) in pqecr:
          Check the deposition rate of previous run.
     Measure Tox and refractive index on dummy wafers.

8

45.0 VIA Photo: Mask VIA (VIA chrome-df)
        Standard I-line process.

___

46.0 Etch VIA in lam2.
        Recipe: Etch time: Overetch:
        Need overetch.

___

47.0 Metal2 Metallization.   target = 800-900 nm
        Remove PR in PRS2000 or technics-c. Rinse the wafers in
     sink7 and spin dry.
        Sputter Al/2% Si CPA.

___

48.0 Metal Photo: Mask METAL2-CM (M2 emulsion-cf)
        Standard I-line process.
        Hard bake for >2 hrs.

___

49.0 Plasma etch Al in Lam3.
        Remove PR in PRS2000.

___

50.0 Sintering: 400 C for 20min in forming gas (tylan13).
        No ramping, use SINT400 program.

___

51.0 Testing:
        Measure Metal1 and Metal2 contact chain.

___

<div align="center">End of Process</div>

___

Appendix: CMOS43 Alignment Offsets, GCA 6200 Wafer Stepper

```
box,center,expand;
c  c  c  c
c  c  c  c
c  c  c  c
c  c  c  c
c  c  c  c
c  c  c  c
c  c  c  c
c  c  c  c
```

```
        XSHIFT        YSHIFT        Label on Mark
CWN (nwell) 0      -3.250        PCH CWN
CAA (active)      .200  -3.250        ACTIVE CAA
CPG (poly gate)   .400  -3.250        GATE CPG
CCA (contact)     .600  -3.250        CONTACT CCA
M1 (metal1) .800  -3.250        METAL1 CMF
VIA            1.20 -3.250        CONTACT CVA
M2 (metal2)       1.00      -3.250        METAL2 CM2
```

```
#define MODULE "VTWDLD"

#include "modtools.h"
#include "vt.h"
#include "ld.h"
#include "wd.h"
#include "dibl.h"
module_function VTWDLD;

typedef struct {
  float VGSstart;
  float VGSstop;
  float VGSstep;
  float VBSstart;
  float VBSstop;
  float VBSstep;
  float VDS;
  float NoiseLevel;
  int   RegPoints;
  int   MaxTries;
  float SlopeTol;
  float tox;    /*in A, 1e-10m, or 1e-8 cm, need to
                   convert to cm when using*/
  float Npeak;
} BODYE_param;

/*VDS changed to 25mV now, 50mV originally*/
#define BODYE_N_PAR  {\
/* VGSstart    */ 0.0,\
/* VGSstop     */ 6.0,\
/* VGSstep     */ 0.1,\
/* VBSstart    */ 0,\
/* VBSstop     */ -5,\
/* VBSstep     */ -.2,\
/* VDS         */   25e-3,\
/* NoiseLevel */1e-8,\
/* RegPoints   */ 5,\
/* MaxTries    */ 5,\
/* SlopeTol    */ 0.4,\
/* tox(A)      */ 300.0,\
/* Npeak       */ 4.5e16\
};

/*VDS changed to 25mV now, 50mV originally*/
#define BODYE_P_PAR  {\
/* VGSstart    */ 0.0,\
/* VGSstop     */ -6.0,\
/* VGSstep     */ -0.1,\
/* VBSstart    */ 0,\
/* VBSstop     */ 5,\
/* VBSstep     */ .2,\
/* VDS         */ -25e-3,\
/* NoiseLevel */ 1e-8,\
/* RegPoints   */ 5,\
/* MaxTries    */ 5,\
/* SlopeTol    */ 0.4,\
/* tox(A)       */ 300.0,\
/* Npeak       */ 3.5e16\
};
```

```c
#include "vtldwd.h"
#include "cf.h"

extern VT_param VT_N_par;
extern VT_param VT_P_par;
extern LD_param LD_N_par;
extern LD_param LD_P_par;
extern WD_param WD_N_par;
extern WD_param WD_P_par;
extern DIBL_param DIBL_N_par;
extern DIBL_param DIBL_P_par;
extern BODYE_param BODYE_N_par;
extern BODYE_param BODYE_P_par;
BODYE_param BODYE_N_par = BODYE_N_PAR;
BODYE_param BODYE_P_par = BODYE_P_PAR;


typedef struct {
  char text[20];
  int ld, vt, dibl, wd, bodye;
  int Y;
  float vt_VBS[30];
  float vt_sum[30];
  float vt_sum2[30];
  int vt_count[30];
  float Gamma_sum;
  float Gamma_sum2;
  int Gamma_count;
  float Gamma1_sum;
  float Gamma1_sum2;
  int Gamma1_count;
  float Gamma2_sum;
  float Gamma2_sum2;
  int Gamma2_count;
  float N_sur_sum;
  float N_sur_sum2;
  int N_sur_count;
  float N_sub_sum;
  float N_sub_sum2;
  int N_sub_count;
  int cferror[8];
} TrnType;



static void BODYE_regression(double datarray[30][2], int index1,
                             int index2, double *a,double *b);
static void BODYE_readparams(BODYE_param *P, char ***paramptr);

static void measres(char type, FetType *dut, LD_param *P, FILE *flderr,
                float data[MNOTRN][1+MNOLINES],
                char baddev[MNOTRN][20], int *trn, int *errors, int *numpoints);
static double measvt(VT_param *P, char type, FetType *dut,
                FILE *fvt, FILE *fvterr,
                TrnType **trnlist, int *vt_VBS_ct);
static void measbodye(BODYE_param *P, VT_param *vtP, char type, FetType *dut,
                FILE *fbodye, TrnType **trnlist);
static void make_list(char **paramlist, TrnType ***trnlstptr,int *bvt,
                int *bld, int *bwd, int *bdibl, int *bbodye,
                VT_param **vtP, WD_param **wdP, LD_param **ldP,
                DIBL_param **diblP,BODYE_param **bodyeP, char *type, int *wdCount,
                int *diblCount);

static void statsLd (TrnType **trnlist, float ld_sum[3], float ld_sum2[3],
                int ld_count[3], int ld, int vt_VBS_ct);

static void statsVt (TrnType **trnlist, int vt_VBS_ct);
static void statsGamma (TrnType **trnlist);

/*Given resistance vs length, calculate delta lengths*/
static void ldCal(TrnType **trnlst, int w5, int w10, int w50,
                float data[3][MNOTRN][1+MNOLINES],
                int trn[3], int errors[3],    char baddev[3][MNOTRN][20],
```

```c
                  int numpoints,  int ld, int last, int vt_VBS_ct)
{
  float w;
  float a[MNOLINES], b[MNOLINES], x[MNOPTS], r[MNOPTS], res, dl;
  int wi, i, j, k,m;
  static float ld_sum[3]={0,0,0}, ld_sum2[3]={0,0,0};
  static int FirstTime=1,  ld_count[3]={0,0,0};
  static FILE* fld;
  typedef enum {GOOD, BAD} StatusType;
  StatusType deviceStatus=GOOD;

  if (FirstTime) {
    fld=fopen("out.ld", "w");
    fprintf(fld, "dieX\tdieY\twidth\tLd\tRes\tTransistors Discarded\n");
    FirstTime=0;
  }
  for (wi=0;wi<=2;wi++){
    if (wi==0)
      if (w5==0)  continue;
      else w=5;
    if (wi==1)
      if (w10==0) continue;
      else w=10;
    if (wi==2)
      if (w50==0) continue;
      else w=50;
    if (trn[wi]>1) {
      LD_regression(data[wi], trn[wi], numpoints, a, b);
      k=0;
      for (i=0; i<numpoints; i++)
        for (j=i+1; j<numpoints; j++) {
          x[k]=-(a[j]-a[i])/(b[j]-b[i]);
          r[k]=a[j]+x[k]*b[j];
          k++;
        }
      dl=0;
      for (m=0;m<k;m++) {
        dl+=x[m];
        res+=r[m];
      }
      dl/=(2*k);
      res/=k;
      if (dl<0 || dl>1)
        deviceStatus=BAD;
      else{
        deviceStatus=GOOD;
        ld_sum[wi]+=dl;
        ld_sum2[wi]+=dl*dl;
        ld_count[wi]++;
      }
      fprintf (fld, "%s\t%g\t%6.3g\t%g\t%s\t",
               Pdie(), w, dl, res,
               (deviceStatus==GOOD)? "GOOD" : "BAD");

    }
    else {
      fprintf (fld, "%s\t%g\t", Pdie(), w);
      fprintf (fld, "Less than two transistors available.\t");
    }
    if (errors[wi])
      for (i=0; i<errors[wi]; i++)
        fprintf (fld," %s", baddev[wi][i]);
    fprintf(fld, "\n");
  }

  if(last){
    fclose(fld);
    statsLd(trnlst, ld_sum, ld_sum2, ld_count, ld, vt_VBS_ct);
  }
}
```

```c
/*function to call different combined routines:   threshold
 *voltage(vt), delta width(wd),   delta length(ld), body
 *effect(bodye), drain-induced barrier lowering(dibl) and
 *its dibl's relating parameters such as subthreshold swing,
 *Source-Drain leakage, depending on the specification in
 *prober.text.*/
void *VTWDLD (char **paramlist, FILE *dummy, int last)
{
  float data[3][MNOTRN][1+MNOLINES];
  static int FirstTime=1, vt_VBS_ct;
  static VT_param *vtP;
  static LD_param *ldP;
  static WD_param *wdP;
  static DIBL_param *diblP;
  static BODYE_param *bodyeP;
  static char type;
  static int wdCount=0;
  static int diblCount=0;
  char baddev[3][MNOTRN][20];
  int numpoints, wi, cferr, w5, w10, w50,
  trn[3]={0,0,0}, errors[3]={0,0,0};
  double i;
  static int vt, wd, ld, dibl, bodye, inv;
  static TrnType **trnlst;
  TrnType **trnlist;
  FetType *dut;
  static FILE *fcf, *fvt, *fvterr, *flderr, *fbodye;
  double vtUse, bigVDS;

  D(printf (">  VTWDLD\n"););
  if (FirstTime) {
    make_list(paramlist, &trnlst, &vt, &ld, &wd, &dibl,&bodye,
              &vtP, &wdP,&ldP,&diblP,&bodyeP, &type, &wdCount, &diblCount);
    fcf=fopen("out.cf.err", "w");
    if (vt) {
      fvt=fopen("out.vt", "w");
      fvterr=fopen("out.vt.err", "w");
      fprintf(fvt, "dieX\tdieY\tname\tdevX\tdevY\tVt\tVbs\tTries\tComment\n");
    }
    if (ld) {
      flderr=fopen("out.ld.err", "w");
    }
    fprintf(fcf, "*** Catastrophic Failure Test ***\n");
    fprintf(fcf, "dieX\tdieY\tname\tdevX\tdevY\n");
    if (dibl)
      bigVDS=getVDS(type);
    if (bodye) {
      fbodye=fopen("out.bodye", "w");
      fprintf(fbodye, "dieX\tdieY\tname\tdevX\tdevY\tbodye\t\
Gamma\tGamma1\tGamma2\tN_sur\tN_sub\n");
    }
    if (type == 'P') inv=-1;
    else inv=1;
    FirstTime=0;
  }
  trnlist=trnlst;
  while (*trnlist){
    printf ("**********  %s **********\n", (*trnlist)->text);
    dut = FindDev ((*trnlist)->text);
    MoveTo (dut);
    DCSturnoff(0);
    connect (1, dut->drain);
    connect (2, dut->gate);
    connect (3, dut->source);
    connect (4, dut->bulk);
    if (type=='N')
      (*trnlist)->cferror[cferr=Ncftest(1,2,3,4,dut)]++;
    else
      (*trnlist)->cferror[cferr=Pcftest(1,2,3,4,dut)]++;
    printcferr(dut,cferr,fcf);
    if ((*trnlist)->vt) {
```

```
      if (cferr) {
        VT_prcferr(dut, cferr, fvterr);
        /*for (i=vtP->VBSstart; inv*i>= inv*vtP->VBSstop; i+=vtP->VBSstep)*/
        i=0;
        VT_printcferr(dut, i, cferr, fvt);
      }
      else
        vtUse=measvt(vtP, type, dut, fvt, fvterr,
                     trnlist, &vt_VBS_ct);
    }
    if ((*trnlist)->dibl)
      measDIBL (type, vtUse, dut,
                last, cferr, diblCount, bigVDS);
    if ((*trnlist)->bodye)
      measbodye(bodyeP,vtP, type, dut, fbodye, trnlist);

    if ((*trnlist)->ld){
      switch ((int)dut->W) {
      case 5:
        wi=0;
        w5=1;
        break;
      case 10:
        wi=1;
        w10=1;
        break;
      case 50:
        wi=2;
        w50=1;
        break;
      }          `
      LD_printcferr(dut, cferr, flderr);
      if (cferr)
        strcpy(baddev[wi][errors[wi]++], (*trnlist)->text);
      else
        measres(type, dut, ldP, flderr, data[wi], baddev[wi], &(trn[wi]),
                &(errors[wi]), &numpoints);
    }
    if ((*trnlist)->wd)
      measWD (type, dut, wdP, last, wdCount);
    trnlist++;
  }
  DCSturnoff(0);
  /*This will add a function call to the ld calculation routine.*/
  if (ld)
    ldCal( trnlst, w5, w10, w50, data, trn, errors, baddev, numpoints,
           ld, last, vt_VBS_ct);
  fflush (fcf);
  if (vt) {
    fflush (fvt);
    fflush (fvterr);
  }
  if (ld){
    fflush (flderr);
  }
/* Depending on how I use FirstTime also I should
   be freeing the memory that trnlist occupies (first every
   singe variable of type TrnType and then the list */
  if (last){
    fclose (fcf);
    if (vt) {
      fclose(fvt);
      fclose(fvterr);
      statsVt (trnlst, vt_VBS_ct);
    }
    if (bodye) {
      fclose(fbodye);
      statsGamma (trnlst);
    }
    if (ld) {
      fclose(flderr);
```

```c
      }
    }
    D(printf ("< VTWDLD\n"););
    return (void *) NULL;
  }

static void statsGamma (TrnType **trnlist)
{
    int N, N1, N2, Nr, Nb;
    float mean, stddev;
    float mean1, stddev1;
    float mean2, stddev2;
    float meanNr, stddevNr;
    float meanNb, stddevNb;
    FILE *fstat;
    fstat = fopen ("out.stat.bodye", "w");
    while (*trnlist){
      fprintf(fstat,"*********** Transistor: %s\n", (*trnlist)->text);
      N=(*trnlist)->Gamma_count;
      mean=(*trnlist)->Gamma_sum/N;
      stddev=sqrt(((*trnlist)->Gamma_sum2 - N*mean*mean)/(N-1));
      fprintf(fstat, "NoTrn: %d\tmean: %g\tstddev: %g \n",
              (*trnlist)->Gamma_count, mean, stddev);
      N1=(*trnlist)->Gamma1_count;
      mean1=(*trnlist)->Gamma1_sum/N1;
      stddev1=sqrt(((*trnlist)->Gamma1_sum2 - N1*mean1*mean1)/(N1-1));
      fprintf(fstat, "NoTrn: %d\tmean1: %g\tstddev1: %g \n",
              (*trnlist)->Gamma1_count, mean1, stddev1);
      N2=(*trnlist)->Gamma2_count;
      mean2=(*trnlist)->Gamma2_sum/N2;
      stddev2=sqrt(((*trnlist)->Gamma2_sum2 - N2*mean2*mean2)/(N2-1));
      fprintf(fstat, "NoTrn: %d\tmean2: %g\tstddev2: %g \n",
              (*trnlist)->Gamma2_count, mean2, stddev2);

      Nr=(*trnlist)->N_sur_count;
      meanNr=(*trnlist)->N_sur_sum/Nr;
      stddevNr=sqrt(((*trnlist)->N_sur_sum2 - Nr*meanNr*meanNr)/(Nr-1));
      fprintf(fstat, "NoTrn: %d\tmeanN_sur: %g\tstddevN_sur: %g \n",
              (*trnlist)->N_sur_count, meanNr, stddevNr);

      Nb=(*trnlist)->N_sub_count;
      meanNb=(*trnlist)->N_sub_sum/Nb;
      stddevNb=sqrt(((*trnlist)->N_sub_sum2 - Nb*meanNb*meanNb)/(Nb-1));
      fprintf(fstat, "NoTrn: %d\tmeanN_sub: %g\tstddevN_sub: %g \n",
              (*trnlist)->N_sub_count, meanNb, stddevNb);
      trnlist++;
    }
  }
static void statsVt (TrnType **trnlist, int vt_VBS_ct)
{
    int i, n=0, k, N;
    float mean, stddev;
    FILE *fstat;

    fstat = fopen ("out.stat.vt", "w");
    while (*trnlist){
      fprintf(fstat,"**********************************************************\n");
      fprintf(fstat,"*********** Transistor: %s\n", (*trnlist)->text);
      fprintf(fstat,"***** Catastrophic Failure Test:\n");
      n=0;
      for (i=0;i<8;i++) n+=(*trnlist)->cferror[i];
      fprintf(fstat, "Transistors Tested: %d,\tFailed:%d\n", n, n-(*trnlist)->cferror[0]);
      if ((i=(*trnlist)->cferror[0]))
        fprintf(fstat, "No catastrophic failure          : %d\n", i);
      if ((i=(*trnlist)->cferror[1]))
        fprintf(fstat, "Short on Gate                    : %d\n", i);
      if ((i=(*trnlist)->cferror[2]))
        fprintf(fstat, "Short on Body                    : %d\n", i);
      if ((i=(*trnlist)->cferror[3]))
        fprintf(fstat, "Short between Drain & Source     : %d\n", i);
      if ((i=(*trnlist)->cferror[4]))
```

```
      fprintf(fstat, "Open on Source or Drain        : %d\n", i);
    if ((i=(*trnlist)->cferror[5]))
      fprintf(fstat, "Open on Gate                   : %d\n", i);
    if ((i=(*trnlist)->cferror[6]))
      fprintf(fstat, "Open on Body                   : %d\n", i);
    if ((i=(*trnlist)->cferror[7]))
      fprintf(fstat, "Small signal Source-Drain Short : %d\n", i);
    fprintf(fstat, "\n");
    if ((*trnlist)->vt){
      fprintf(fstat, "***** Threshold Voltage:\n");
      for(k=0; k<vt_VBS_ct; k++){
        N=(*trnlist)->vt_count[k];
        mean=(*trnlist)->vt_sum[k]/N;
        stddev=sqrt(((*trnlist)->vt_sum2[k] - N*mean*mean)/(N-1));
        fprintf(fstat, "VBS: %g\tNoTrn: %d\tmean: %g\tstddev: %g \n",
                (*trnlist)->vt_VBS[k], (*trnlist)->vt_count[k], mean, stddev);
      }
    }
    fprintf(fstat, "\n\n");
    trnlist++;
  }
  fclose(fstat);
}


void statsLd (TrnType **trnlist, float ld_sum[3], float ld_sum2[3],
           int ld_count[3], int ld, int vt_VBS_ct)
{
  int wi, N;
  float w, mean, stddev;
  FILE *fstat;

  fstat = fopen ("out.stat.ld", "w");
  D(printf("> stats\n"););
  if (ld){
    fprintf(fstat, "\n\n");
    fprintf(fstat, "*****************************************************************\n");
    fprintf(fstat, "***** Delta L:\n");
    for(wi=0; wi<3; wi++){
      if (wi==0)
        if (ld_count[wi]==0) continue;
        else w=5;
      if (wi==1)
        if (ld_count[wi]==0) continue;
        else w=10;
      if (wi==2)
        if (ld_count[wi]==0) continue;
        else w=50;
      N=ld_count[wi];
      mean=ld_sum[wi]/N;
      stddev=sqrt((ld_sum2[wi] - N*mean*mean)/(N-1));
      fprintf(fstat,"width: %g\tNoDies: %d\tmean: %g\tstddev: %g\n",
              w, ld_count[wi], mean, stddev);
    }
  }
  D(printf("< stats\n"););
  fclose (fstat);
}

void measres(char type,FetType *dut,LD_param *P, FILE *flderr,
             float data[MNOTRN][1+MNOLINES],
             char baddev[MNOTRN][20],int *trn,int *errors,int *numpoints)
{
  char *result;
  DatArrType datarray;
  int i, error, count;

  D(printf("> measres\n"););
  count=0;
  DCShold (4,'V',P->VBS,.01);
  DCShold (3,'V',0,.1);
```

```
    DCShold (1,'V',P->VDS,.1);
    do{
      DCSsweep (2, VOLTAGE, LINEAR, P->VGSstart, P->VGSstop, P->VGSstep, .01);
      result = DCStrack("1");
      *numpoints=DatFormat (&datarray, result, 1);
      free(result);
      if (type=='N')
        error=LD_Ncheckdata(P, datarray, dut, *numpoints);
      else
        error=LD_Pcheckdata(P, datarray, dut, *numpoints);
    } while ( (++count < P->MaxTries) && (error==1) );
    LD_printcrverr(dut, *numpoints, count, error, datarray, flderr);
    if (error==0) {
      for (i=0; i<*numpoints; i++)
        data[*trn][i+1]=P->VDS/datarray[i][0]->value;
        data[*trn][0]=dut->L;
        (*trn)++;
    }
    else strcpy(baddev[(*errors)++], dut->Name);
    free_dat (datarray,*numpoints,1);
    D(printf("< measres\n"););
}
double measvt(VT_param *P, char type, FetType *dut, FILE *fvt,
              FILE *fvterr, TrnType **trnlist, int *vt_VBS_ct)
{
  int numpoints, j, error, count, inv, k=0;
  char *result;
  DatArrType datarray;
  float a, b, bmax, abmax, i, vt;
  double vtReturn;/*the return value when VBS=0*/
  vtReturn=0.0;

  D(printf("> measvt\n"););
  if (type == 'P') inv=-1;
  else inv = 1;
  DCShold (1, 'V', P->VDS, .1);
  DCShold (3, 'V', 0, .1);
  for (i=P->VBSstart; inv*i>= inv*P->VBSstop; i+=P->VBSstep){
    DCShold (4, 'V', i, .01);
    count=0;
    error=0;
    do {
      DCSsweep (2,VOLTAGE,LINEAR,P->VGSstart,P->VGSstop,P->VGSstep,.01);
      result = DCStrack ("1");
      numpoints=DatFormat (&datarray,result,1);
      free (result);
      if (type=='N') error=VT_Ncheckdata(P,datarray,numpoints);
      else error=VT_Pcheckdata(P,datarray,numpoints);
    } while ( (++count < P->MaxTries) && (error==1) );
    bmax=0;
    for (j=0;j<numpoints-P->RegPoints;j++){
      VT_regression(P,datarray,j,&a,&b);
      if (b > bmax) {
        bmax=b;
        abmax=a;
      }
    }
    vt=-abmax/bmax;
    if(fabs(i)<1e-6)
      vtReturn=vt;
    VT_printresult(dut,datarray,numpoints,vt,i,count,error,fvt,fvterr);
    free_dat (datarray,numpoints,1);
    if (!error){
      (*trnlist)->vt_VBS[k]=i;
      (*trnlist)->vt_sum[k]+=vt;
      (*trnlist)->vt_sum2[k]+=vt*vt;
      (*trnlist)->vt_count[k]++;
    }
    k++;
  }
  *vt_VBS_ct=k;
```

```c
  D(printf("< measvt\n"););
  if (vtReturn < 1e-6)
    return vt;
  else
    return vtReturn;
}


/*measure body effect (ie, gamma)*/
void measbodye(BODYE_param *P,VT_param *vtP, char type, FetType *dut,
               FILE *fbodye, TrnType **trnlist)
{
  int numpoints, j, error, count, inv, k=0;
  char *result;
  DatArrType datarray;
  int GammaErrorFlag;
  int GammaGood=1;
  double Gamma;
  double Gamma1;
  double Gamma2;
  double datarrayGamma[30][2];
  double aGamma, bGamma;
  double phi;
  double const Ni=1.45e10;
  /*const double phiMn=0.7474;*/ /*To be changed to user input*/
  /*const double phiMp=0.7348;*/ /*To be changed to user input*/
  double N_sur, N_sub;
  float a, b, bmax, abmax, i, vt;

  D(printf("> measbodye\n"););
  phi=2*0.025*log(P->Npeak/Ni);

  if (LogFile) {
    fprintf(LogFile, "P->Npeak=%f", P->Npeak);
    fprintf(LogFile, "P->tox=%f", P->tox);
    fprintf(LogFile, "phi=%f", phi);
  }
  if (type == 'P') {
    inv=-1;
    /*phi=phiMp;*/
  }
  else{
    inv = 1;
    /*phi=phiMn;*/
  }

  DCShold (1, 'V', P->VDS, .1);
  DCShold (3, 'V', 0, .1);
  for (i=P->VBSstart; inv*i >= inv*P->VBSstop; i+=P->VBSstep){
    DCShold (4, 'V', i, .01);
    count=0;
    error=0;
    GammaErrorFlag=0;

    do {
      DCSsweep (2,VOLTAGE,LINEAR,P->VGSstart,P->VGSstop,P->VGSstep,.01);
      result = DCStrack ("1");
      numpoints=DatFormat (&datarray,result,1);
      free (result);
      if (type=='N') error=VT_Ncheckdata(vtP,datarray,numpoints);
      else error=VT_Pcheckdata(vtP,datarray,numpoints);
    } while ( (++count < P->MaxTries) && (error==1) );
    bmax=0;
    for (j=0;j<numpoints-P->RegPoints;j++){
      VT_regression(vtP,datarray,j,&a,&b);
      if (b > bmax) {
        bmax=b;
        abmax=a;
      }
    }
    vt=-abmax/bmax;
```

```c
      free_dat (datarray,numpoints,1);

      if(error && !GammaErrorFlag)
        GammaErrorFlag=1;

      datarrayGamma[k][1]=sqrt(fabs(i)+fabs(phi));
      datarrayGamma[k][0]=vt;
      k++;
    }
    /*plotting vt vs sqrt(Vbs+phi surface), the slope is the
     *body effect, gamma, then calcaute surface doping
     *concentration(N_sur) and substrate doping concentration
     *(N_sub) */
  if((*trnlist)->bodye){
    BODYE_regression(datarrayGamma, 5, k-1, &aGamma, &bGamma);
    Gamma=bGamma;
    BODYE_regression(datarrayGamma, 5, 11, &aGamma, &bGamma);
    Gamma1=bGamma;
    /*Si relative permitivity=3.9,  SiO2 relative permitivity=11.7,
     *vacuum permitivity=8.85418e-14, tox=300A=3e-6cm, basic
     *eletron charge=1.602e-19*/
    N_sur=(Gamma1*3.9*8.85418e-14/(P->tox/1e8))*
      (Gamma1*3.9*8.85418e-14/(P->tox/1e8))
      /(2*11.7*8.85418e-14*1.602e-19);
    BODYE_regression(datarrayGamma, 20, k-1, &aGamma, &bGamma);
    Gamma2=bGamma;
    N_sub=(Gamma2*3.9*8.85418e-14/(P->tox/1e8))*
      (Gamma2*3.9*8.85418e-14/(P->tox/1e8))
      /(2*11.7*8.85418e-14*1.602e-19);

    /*check to see if the device is good based on body effect
     *criterion*/
    if(fabs(Gamma) > 1 || GammaErrorFlag)
      GammaGood=0;
    else
      GammaGood=1;

    fprintf (fbodye,"%s\t%s\t%d\t%d\t%f\t%f\t%f\t",
            Pdie(), dut->Name, dut->X, dut->Y, Gamma, Gamma1, Gamma2);

    fprintf (fbodye,"%g\t%g\t%s\n", N_sur, N_sub, (GammaGood)? "GOOD":"BAD");
    fflush(fbodye);
    if(GammaGood){
      (*trnlist)->Gamma_sum+=Gamma;
      (*trnlist)->Gamma_sum2+=Gamma*Gamma;
      (*trnlist)->Gamma_count++;
      (*trnlist)->Gamma1_sum+=Gamma1;
      (*trnlist)->Gamma1_sum2+=Gamma1*Gamma1;
      (*trnlist)->Gamma1_count++;
      (*trnlist)->Gamma2_sum+=Gamma2;
      (*trnlist)->Gamma2_sum2+=Gamma2*Gamma2;
      (*trnlist)->Gamma2_count++;
      (*trnlist)->N_sur_sum+=N_sur;
      (*trnlist)->N_sur_sum2+=N_sur*N_sur;
      (*trnlist)->N_sur_count++;
      (*trnlist)->N_sub_sum+=N_sub;
      (*trnlist)->N_sub_sum2+=N_sub*N_sub;
      (*trnlist)->N_sub_count++;
    }
  }
  D(printf("< measbodye\n"););
}


void BODYE_regression(double datarray[30][2], int index1, int index2,
                      double *a,double *b)
{
  double mx=0, my=0, cxy=0, cxx=0;
  int i, begin, end;
/*
 * The next two decisions are  to prevent the program from halting
 * the machine.  Usually index1 is less than index2, the decisions
```

```
 * are only needed mostly when the device in question is bad.
 */
  if(index1<index2){
    begin=index1;
    end=index2;
  }
  else{
    begin=index2;
    end=index1;
  }

  D(printf("> BODYE_regression\n"));
  for(i=begin;i<=end;i++) {
    mx+=datarray[i][1];
    my+=datarray[i][0];
  }
  mx/=(end-begin+1);
  my/=(end-begin+1);
  for(i=begin;i<=end;i++) {
    cxy += (datarray[i][1]-mx)*(datarray[i][0]-my);
    cxx += (datarray[i][1]-mx)*(datarray[i][1]-mx);
  }
  *b=cxy/cxx;
  *a=my-(*b)*mx;
  D(printf("< BODYE_regression\n"));
}


/*Reading the devices list from prober.text, for each device,
 *specify when a parameter (vt, ld, wd, dibl, or bodye) is
 *needed to measure*/
void make_list(char **paramlist, TrnType ***trnlstptr,int *bvt,
               int *bld, int *bwd, int *bdibl, int *bbodye,
               VT_param **vtP, WD_param **wdP, LD_param **ldP,
               DIBL_param **diblP, BODYE_param **bodyeP, char *type,
               int *wdCount,int *diblCount)
{
  char **first_trn,param[20],*par;
  TrnType **trnlist, **list_start, **x, **old, **ymin,*xx;
  int vt=0,ld=0,wd=0,dibl=0, bodye=0, i,cont;
  FetType *dut;

  D(printf("> make_list\n"););
  trnlist= (TrnType **)calloc(MNOTRN, sizeof (TrnType *));
  list_start=trnlist;
  first_trn=paramlist;
  for (first_trn=paramlist+1;
       ((strncmp("VT",(*first_trn), 2)==0) ||
        (strncmp("LD",(*first_trn), 2)==0) ||
        (strncmp("WD",(*first_trn), 2)==0) ||
        (strncmp("DIBL",(*first_trn), 4)==0) ||
        (strncmp("BODYE",(*first_trn), 5)==0) ||
        ((**first_trn)=='+'));
       first_trn++);
  dut = FindDev (*first_trn);
  if (dut->Y <= 2240) *type='N';
  else *type='P';
  if (*type=='N') {
    *vtP=&VT_N_par;
    *ldP=&LD_N_par;
    *wdP=&WD_N_par;
    *diblP=&DIBL_N_par;
    *bodyeP=&BODYE_N_par;
  }
  else {
    *vtP=&VT_P_par;
    *ldP=&LD_P_par;
    *wdP=&WD_P_par;
    *diblP=&DIBL_P_par;
    *bodyeP=&BODYE_P_par;
  }
```

```c
while (*++paramlist) {
  i=0;
  cont=0;
  par=*paramlist;
  while ((*par!='\0') && (*par!='\n') && (*par!=' ')) param[i++]=*par++;
  param[i]='\0';
  if (strcmp("VT", param)==0) {
    vt=1;
    ld=0;
    wd=0;
    dibl=0;
    bodye=0;
    *bvt=1;
    VT_readparams(*vtP, &paramlist);
    cont=1;
  }
  if (strcmp("LD", param)==0) {
    ld=1;
    wd=0;
    vt=0;
    dibl=0;
    bodye=0;
    *bld=1;
    LD_readparams(*ldP, &paramlist);
    cont=1;
  }
  if (strcmp("WD", param)==0) {
    wd=1;
    ld=0;
    vt=0;
    dibl=0;
    bodye=0;
    *bwd=1;
    WD_readparams(*wdP, &paramlist);
    cont=1;
  }
  if (strcmp("DIBL", param)==0) {
    dibl=1;
    vt=0;
    ld=0;
    wd=0;
    bodye=0;
    *bdibl=1;
    DIBL_readparams(*diblP, &paramlist);
    cont=1;
  }
  if (strcmp("BODYE", param)==0) {
    bodye=1;
    vt=0;
    ld=0;
    wd=0;
    dibl=0;
    *bbodye=1;
    BODYE_readparams(*bodyeP, &paramlist);
    cont=1;
  }
  if (cont) continue;
  old=NULL;
  for (x=list_start; x<trnlist; x++)
    if (strcmp((*x)->text, param) == 0) old=x;
  if (old != NULL) x=old;
  else {
    (*trnlist) = (TrnType*) malloc (sizeof(TrnType));
    x=trnlist++;
    i=0;
    do {
      (*x)->text[i] = param[i];
    } while (param[i++] != '\0') ;
    dut = FindDev (param);
    (*x)->Y=dut->Y;
    (*x)->vt = 0;
```

```
      (*x)->ld = 0;
      (*x)->wd = 0;
      (*x)->dibl = 0;
      (*x)->bodye = 0;
      for (i=0;i<10;i++)
        (*x)->vt_sum[i] = 0;
      for (i=0;i<10;i++)
        (*x)->vt_sum2[i] = 0;
      for (i=0;i<10;i++)
        (*x)->vt_count[i] = 0;
      for (i=0;i<8;i++)
        (*x)->cferror[i] = 0;
    }
    if (vt == 1)  (*x)->vt = 1;
    if (ld == 1)  (*x)->ld = 1;
    if (wd == 1){
      (*x)->wd = 1;
      (*wdCount)++;
    }
    if (dibl == 1)  {
      (*x)->dibl = 1;
      (*diblCount)++;
    }
    if (bodye == 1)  (*x)->bodye = 1;
  }
  *trnlist=NULL;
  for (trnlist=list_start; *(trnlist+1) != NULL; trnlist++){
    ymin=trnlist+1;
    for (x=trnlist+1; *x != NULL; x++)
      if ((*x)->Y < (*ymin)->Y) ymin=x;
    if ((*ymin)->Y < (*trnlist)->Y) {
      xx=*ymin;
      *ymin=*trnlist;
      *trnlist=xx;
    }
  }
  *trnlstptr=list_start;
  D(printf("< make_list\n"););
}




void BODYE_readparams(BODYE_param *P, char ***paramptr)
{
  D(printf("> BODYE_readparams\n"));
  while (**++(*paramptr) == '+') {
    sscanf (*(*paramptr), "+ VGSstart = %g",     &P->VGSstart);
    sscanf (*(*paramptr), "+ VGSstop = %g",      &P->VGSstop);
    sscanf (*(*paramptr), "+ VGSstep = %g",      &P->VGSstep);
    sscanf (*(*paramptr), "+ VBSstart = %g",     &P->VBSstart);
    sscanf (*(*paramptr), "+ VBSstop = %g",      &P->VBSstop);
    sscanf (*(*paramptr), "+ VBSstep = %g",      &P->VBSstep);
    sscanf (*(*paramptr), "+ VDS = %g",          &P->VDS);
    sscanf (*(*paramptr), "+ RegPoints = %d",    &P->RegPoints);
    sscanf (*(*paramptr), "+ MaxTries = %d",     &P->MaxTries);
    sscanf (*(*paramptr), "+ NoiseLevel = %g",   &P->NoiseLevel);
    sscanf (*(*paramptr), "+ SlopeTol = %g",     &P->SlopeTol);
    sscanf (*(*paramptr), "+ tox = %g",          &P->tox);
    sscanf (*(*paramptr), "+ Npeak = %g",        &P->Npeak);
  }
  (*paramptr)--;
  D(printf("< BODYE_readparams\n"));
}
```

Authors: V. Gutnik, S. Fang, and D. Rodriguez