ET4ICP - IC Technology Lab
Code Documentation

Version 1.0

Prof. Sten Vollebregt

January 11, 2026

Contents

Document Revision History

Version	Date	Author	Reviewed By	Description of Changes
1.0	2025-10-20	Pratap Kygonahalli	Sten Vollebregt	Initial draft. Outlined main sections and overall document structure.

1 Introduction

This documentation provides a comprehensive technical reference for a Technology Computer-Aided Design (TCAD) project. Its purpose is to create a "virtual wafer fab" environment for simulating the complete manufacturing process and subsequent electrical testing of individual NMOS and PMOS transistors. This approach is fundamental to modern semiconductor development, as it allows engineers to predict device performance, optimize fabrication recipes, and explore the impact of process variations before committing to the immense cost and time of actual silicon fabrication.

The project is structured into three main phases:

1. **Process Simulation (SProcess):** This phase emulates the physical and chemical processes inside a fabrication facility. It simulates steps like ion implantation to introduce dopants, high-temperature annealing for diffusion and activation, chemical vapor deposition (CVD) for material growth, and lithographic patterning for defining device features. The output is a virtual, two-dimensional model of the transistor, complete with its material composition and doping profiles.
2. **Device Simulation (SDevice):** This phase takes the virtual transistor model and subjects it to electrical analysis. By solving a set of coupled, non-linear partial differential equations—namely the Poisson equation and the electron/hole continuity equations it calculates the device's electrical behavior under user-defined bias conditions. This allows for the extraction of key performance metrics and characteristic I-V curves.
3. **Visualization (SVisual):** Post-processes the data from the previous stages to generate plots of doping profiles and current-voltage (I-V) characteristics.

A key feature of this workflow is its parameterization. Critical process variables, such as the device type (`@deviceType@`) or the dose for the threshold voltage adjustment implant (`@vtAdj@`), are exposed as parameters. This enables automated, large-scale studies like Design of Experiments (DoE) to be conducted.

1.1 Workflow Overview and File Dependencies

The simulation flow is orchestrated as a sequence of dependent steps, typically managed within the Sentaurus Workbench graphical user interface. Each script forms a node in the project, processing the output of its predecessor and feeding its result to the next. This creates a robust and reproducible simulation chain from initial process definition to final graphical results. Figure 1 illustrates this flow and the dependencies between the provided files.

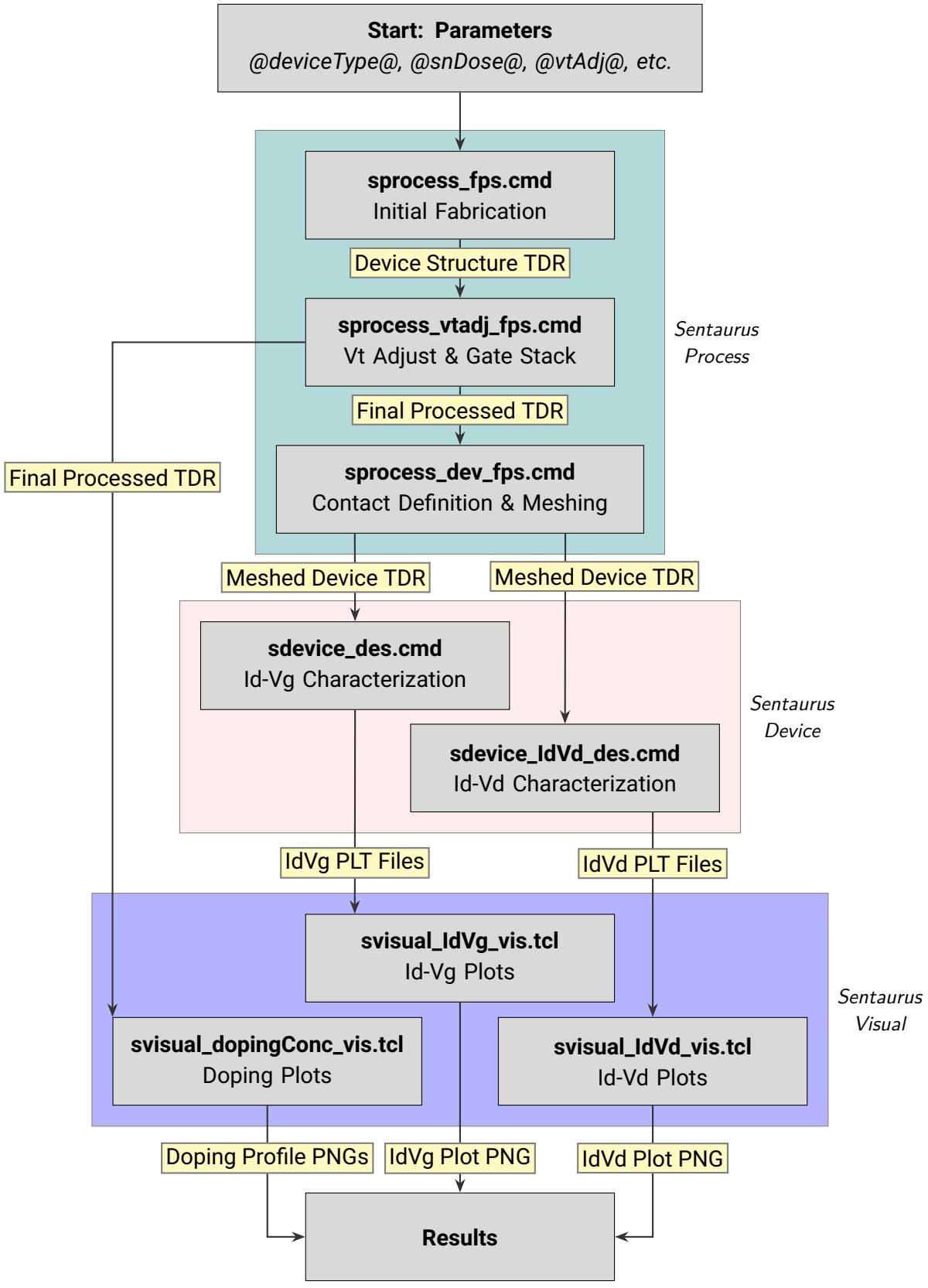

Figure 1: NMOS & PMOS Simulation Flow

2 Process Simulation (Sentaurus Process)

This chapter provides a detailed breakdown of the scripts responsible for the virtual fabrication of the semiconductor device. Sentaurus Process (SProcess) is a powerful simulator that models the complex physics and chemistry of fabrication steps, enabling the creation of accurate virtual device structures.

2.1 *sprocess_fps.cmd*

This script simulates the initial, fundamental steps of the CMOS fabrication process, including well formation and source/drain implantation.

- **Purpose:** To create the basic doped regions (N-well for PMOS, source/drain regions for both types) on a silicon substrate.
- **Key Parameters:** `@deviceType@`, `@snDose@`, `@snEnergy@`, `@spDose@`, `@spEnergy@`, `@nwdose@`, `@nwenergy@`
- **Output:** A TDR file (n@node@.tdr) containing the device structure after initial implants and diffusions.

The key command sections are described as below

- **Parameter Setup:** The script begins by configuring the physics database (PDB) for the simulation
  - `pdbSet Silicon Dopant DiffModel Fermi`: This command selects the Fermi model for dopant diffusion. This is a sophisticated model that is crucial for accuracy in this process, as it accounts for concentration-dependent diffusivity and the influence of the electric field on dopant movement, effects which are dominant in the heavily-doped source/drain regions.
  - `pdbSet Implant ResistSkip 1`: This is a computational efficiency setting. It instructs the implant simulator to treat the photoresist as a simple masking layer without calculating the ion-stopping physics within it. Since the resist is stripped away later, this approximation reduces simulation time with no loss of accuracy for the final silicon structure
- **Mask and Geometry Definition:**
  - `mask name=... segments={...} negative`: This command defines a lithographic mask. The segments define the start and end coordinates of the mask features. The negative keyword inverts the mask, meaning the implant will occur in the regions not defined by the segments.
  - `line x location=...` and `line y location=...`: These commands establish the initial rectangular grid for the 2D simulation domain. The spacing parameter controls the initial mesh density.
- **Initial Wafer and Oxide Growth**:

- `region Silicon ...`: Creates the starting silicon material with a background Boron concentration of $1 \times 10^{16} cm^{-3}$.
- `diffuse temp.ramp=dibar`: Simulates the growth of a "dirt barrier" oxide layer using a multi-step temperature ramp (dibar) with varying gas flows.

- **Device-Specific Implantation Flow**:
  - **PMOS Workflow**: If `@devicetype@` is PMOS, the script first simulates the N-well formation. It applies a photoresist (`photo thickness= 2<um> mask= nw`), performs a high-energy Phosphorus implant (`implant Phosphorus dose=$nwdose`), strips the resist, and then executes a high-temperature drive-in anneal (`diffuse temp.ramp=nwdrivein`) to form the deep N-well.
  - **NMOS Workflow**: For NMOS, this N-well step is skipped
  - **Source/Drain Implants**: Both workflows then proceed to the source/drain implants. The script applies masks for n-type (sn) and p-type (sp) regions and performs Arsenic and Boron implants, respectively.
- **Remeshing**: Before saving, refinebox and grid remesh are used to refine the simulation mesh in the active areas of the device, ensuring higher accuracy for subsequent simulation steps.
  - `refinebox min=... max=... xrefine=... yrefine=...`: This command specifies regions where the mesh needs to be denser. Finer meshing is applied in the electrically active areas near the surface to accurately capture the steep doping gradients at the junctions.
  - `grid remesh`: his command rebuilds the grid according to the specifications, ensuring the output structure is ready for the next, more detailed simulation steps.

2.2 *sprocess_vtadj_fps.cmd*

This script loads the previously created structure and performs the critical steps of threshold voltage (V_t) adjustment, gate oxide growth, and metallization.

- **Purpose**: To finalize the front-end-of-line (FEOL) process by creating the gate stack and defining the contact metallization, with a specific focus on tuning the device threshold voltage.
- **Key Inputs**: `@deviceType@`, `@vtAdj@`, `@vtAdjEnergy@`.
- **Output**: A TDR file (n@node@.tdr) of the fully fabricated device structure.

The key command sections are described as below

- **Load Structure**: The script starts with `init tdr=n@node|sprocess@`, loading the TDR file generated by *sprocess_fps.cmd*
- V_t **Adjust Implant**: The core of this step is the command `implant Boron dose=${vtadj} energy=${vtadjenergy}`. This is a shallow, low-dose implant into the channel region designed to precisely tune the device's turn-on voltage
- **Gate Oxide Formation**:

- `pdbSetBoolean ... DopantDependentReaction 1`: This enables a physically accurate model for thermal oxidation. It accounts for the fact that the oxidation rate of silicon is dependent on the type and concentration of dopants at the surface.
- `diffuse temp.ramp=gateoxide`: A carefully controlled temperature ramp is executed to grow a thin, high-quality gate oxide. The ramp includes steps with different gas mixtures (H2, O2, N2) to achieve the desired thickness and electrical properties.

- **Contact Opening and Metallization**:
  - `etch oxide mask=co ...`: An anisotropic etch is performed using the co (contact) mask to open windows down to the silicon.
  - `deposit Aluminum thickness=1<um>`: A 1-micron layer of Aluminum is deposited over the entire structure.
  - `etch Aluminum mask=al ...`: The Aluminum is then etched using the al mask to define the final electrodes and interconnects
- **Final Geometry Transform**: `transform reflect right` duplicates and mirrors the simulated half-device to create a full device structure, which is a common technique to save computation time.

2.3 *sprocess_dev_fps.cmd*

This script serves as a bridge between the physical world of process simulation and the electrical world of device simulation. It takes the final fabricated structure and prepares it by defining electrical terminals and generating a suitable mesh for the numerical solver.

- **Purpose**: To define the electrical contacts (electrodes) and generate a high-quality mesh for the SDevice solver.
- **Key Inputs**: `@deviceType@`
- **Output**: A meshed TDR file (n@node@_dev_fps.tdr) ready for device simulation.

The key command sections are described as below

- **Load Structure**: `init tdr=n@node|sprocess_vtadj@`, loads the fully processed device structure
- **Contact Definition**:
  - The core of this script is the `contact box` command, which defines the named electrodes based on geometric coordinates. Separate definitions are provided for NMOS and PMOS devices to account for their different layouts.
  - `contact name=substrate merge={substrate0 substrate1}`: This command is used to merge multiple contact regions (e.g., on the left and right sides) into a single logical electrode.
- **Meshing for SDevice**:

- `grid set.Delaunay.type=boxmethod`: This specifies the use of a Delaunay triangulation algorithm for meshing, which is well-suited for numerical device simulation as it tends to produce well-conditioned mesh elements (avoiding triangles with excessively small angles), improving solver stability and accuracy.
- `struct tdr=n@node@_dev !Gas !interfaces`: This is the final save command. It saves the structure, which now includes the mesh and electrode definitions. The !Gas and !interfaces flags are important optimizations; they strip out information related to the ambient gas and non-semiconductor interfaces from the TDR file, which are not needed by the SDevice solver, resulting in a smaller and more efficient input file.

3 Device Simulation (Sentaurus Device)

This chapter details the scripts that perform the electrical characterization. Sentaurus Device (SDevice) is a numerical simulator that solves the fundamental semiconductor physics equations on the mesh provided by SProcess. This allows for the prediction of the device's electrical behavior without fabricating physical hardware.

3.1 *sdevice_des.cmd*

This script is designed to simulate the transistor's transfer characteristics, or $I_d - V_g$ curve, which is fundamental to understanding its switching behavior and extracting parameters like threshold voltage.

- **Purpose**: To calculate the drain current (I_d) as a function of the gate voltage (V_g) while keeping the drain and substrate voltages at fixed potentials.
- **Key Inputs**: `@deviceType@`, `@vtAdj@`, `@fixedCharge@`.
- **Output**: A series of .plt data files, where each file contains the $I_d - V_g$ data for a specific substrate bias condition.

The key command sections are described as below

- **Variable Setup**: A TCL (Tool Command Language) block at the beginning of the file sets up simulation parameters dynamically.
  - `set SIGN 1.0` or `-1.0`: This variable is used to conveniently manage voltage polarities for NMOS (positive voltages) and PMOS (negative voltages) devices.
  - `set ListOfVb  0.0 1.0 2.0 `: This defines a list of substrate biases (Vb) that will be simulated. Running the simulation at multiple substrate biases allows for the characterization of the body effect
- **File and Electrode Sections**:
  - `Grid="n@node|sprocess_dev@_dev_fps.tdr"`: This line specifies the input file which is the meshed device structure created by sprocess_dev.cmd

- `Electrode ...`: This section declares the names of all the electrodes defined in the input TDR and sets their initial voltage bias to 0V.

- **Physics Section**: This section defines the physical models that the SDevice solver will use. The choice of models is crucial for simulation accuracy.
  - `Physics (MaterialInterface="Oxide/Silicon") Traps (FixedCharge Conc=@fixedCharge@)` : This model is essential for accurately predicting the threshold voltage. It introduces a fixed positive charge density (Q_f) at the silicon/silicon dioxide interface, which is a byproduct of the thermal oxidation process and has a strong influence on the device's electrostatics
  - `Mobility (Enormal (UniBo))`: This specifies the mobility model. The Enormal (UniBo) model accurately captures the phenomenon of mobility degradation due to the high vertical electric field applied by the gate. As the gate voltage increases, carriers in the channel are pulled closer to the Si/SiO_2 interface and experience increased surface scattering, which reduces their mobility and, consequently, the drain current.
  - Other models like SRH (Shockley-Read-Hall recombination), Auger (recombination in heavily doped regions), and OldSlotboom (bandgap narrowing) are included to provide a comprehensive and physically realistic simulation.
- **Solve Section**: This block defines the sequence of numerical operations.
  - The simulation starts with a simple Poisson equation solve to establish a stable initial guess for the electrostatic potential.
  - The main simulation is performed with `Coupled Poisson Electron Hole` , which instructs the solver to self-consistently solve the three fundamental equations of semiconductor device physics.
  - **Voltage Ramping Logic**: A foreach loop in TCL iterates through the substrate biases defined in ListOfVb. For each bias:
    * A `Quasistationary` command ramps the substrate voltage to the target value. The InitialStep, MinStep, and MaxStep parameters control the adaptive voltage step algorithm, ensuring the solver can converge even through numerically challenging regions.
    * `save(FilePrefix="...")`: The converged state of the device at the target substrate bias is saved to a file.
    * `Load(FilePreFix="...")`: This saved state is then reloaded as the starting point for the gate voltage sweep.
    * Another `Quasistationary` command sweeps the gate voltage from 0V to its final value (e.g., 5V). The drain current is recorded at each voltage step and saved to a .plt file

3.2 *sdevice_IdVd_des.cmd*

This script simulates the transistor's output characteristics, or $I_d - V_d$ curves. This data reveals the device's behavior in its linear and saturation regions of operation and is used to characterize its output resistance and current-driving capability.

- **Purpose**: To calculate the drain current (I_d) as a function of drain voltage (V_d) for a set of different, fixed gate voltages. potentials.
- **Key Inputs**: `@deviceType@`, `@vtAdj@`, `@fixedCharge@`.
- **Output**: A series of .plt data files, where each file contains an $I_d - V_d$ curve corresponding to a specific gate voltage.

The key command sections are described as below

- **Variable Setup**: The initial TCL block sets up the simulation parameters.
  - `set ListOfVgs  0.0 1.0 ...  7.0` : An array of gate voltages (Vgs) is defined. The simulation will generate one output curve for each value in this list.
- **File, Electrode, and Physics Sections**: These sections are configured identically to the sdevice_des.cmd script. They load the same meshed device and invoke the same set of calibrated physics models to ensure consistency between the simulations.
- **Solve Section**: The solve block uses a highly efficient two-stage, nested-loop methodology to generate the family of curves.
  - **Stage 1: Gate Voltage Ramping and Saving:**
    * A foreach loop iterates through each gate voltage ($Vgate) in `ListOfVgs`
    * For each $Vgate, a `Quasistationary` command ramps the gate to that voltage
    * Critically, the converged solution (the state of all physical quantities on the mesh) is immediately saved to a uniquely named file using `save(FilePrefix="...gateBias_$VgateInt")`. This creates a set of stable starting points.
  - **Stage 2: Loading States and Sweeping Drain Voltage:**
    * A second foreach loop also iterates through the gate voltages in `ListOfVgs`
    * Inside the loop, `Load(FilePreFix="...gateBias_$VgateInt")` loads the previously saved state for a specific gate bias. This is far more efficient than re-solving from Vg=0V each time.
    * A `Quasistationary` command is then executed to sweep the drain voltage from 0V to its maximum value. The drain current is recorded at each step and saved to a .plt file specific to that gate bias. This process is repeated for every gate voltage, generating the complete family of output curves.

4 Visualization (Sentaurus Visual)

This chapter covers the TCL scripts that use Sentaurus Visual to post-process the simulation data, automating the creation of plots and providing visual results that can be directly compared to theoretical calculations and lab measurements.

4.1 *svisual_dopingConc_vis.tcl*

- **Purpose**: To visualize the physical results of the process simulation, focusing on the device structure and the distribution of dopants.
- **Input**: The final processed TDR file, n@node|sprocess_vtadj@_fps.tdr.
- **Output**: Several PNG image files showing the 2D device cross-section and various 1D doping concentration profiles.

The key command sections are described as below

- `load_file "n@node|sprocess_vtadj@_fps.tdr"`: Loads the TDR file from the process simulation
- `create_plot -dataset $myData2D`: Creates a new 2D plot window. By default, this will show the device geometry, with different materials shaded and a color contour representing a physical quantity, such as `NetActive` doping.
- `create_cutline -plot $myPlot2D -type y -at 50.0`: It creates a 1D dataset by taking a "slice" or cutline through the 2D device structure. In this example, it creates a horizontal (-type y) cutline at the Y-coordinate of 50.0, which corresponds to the device channel. Vertical cutlines (-type x) are also used to probe junction depths.
- `create_curve -axisX X -axisY NetActive -dataset $dopingChannelCutline`: This command takes the 1D data from a cutline and plots it on a new 1D graph. Here, it plots the `NetActive` doping concentration (Y-axis) as a function of the X-coordinate (X-axis) along the channel.
- `set_axis_prop -plot $dopingChannelPlot -axis y -type log`: This customizes the plot axes. Setting the Y-axis to a logarithmic scale is essential for viewing doping concentrations, which can span many orders of magnitude (e.g., from $10^{15} cm^{-3}$ in the channel to $10^{20} cm^{-3}$ in the source/drain).
- `export_view "$deviceType_$vtAdj_$fileName" -format png -overwrite`: This command saves the current plot view to a PNG image file. The filename is constructed dynamically from script variables, ensuring that plots from different simulation runs have unique and descriptive names. The -overwrite flag allows the script to be re-run without manual file deletion.

4.2 *svisual_IdVg_vis.tcl*

- **Purpose**: To automatically plot the $I_d - V_g$ transfer characteristics generated by the sdevice_des.cmd script.

- **Input**: The .plt files generated by sdevice_des.cmd.
- **Output**: A single PNG image (IdVg.png) containing one or more $I_d - V_g$ curves on the same axes.

The key command sections are described as below

- `glob -nocomplain n@node|sdevice@_..._IdVg*.plt`: The glob command is used to find all files in the project directory that match the specified wildcard pattern. This allows the script to automatically find and plot all the output files from a given simulation, even if the number of files (e.g., number of substrate biases) changes.
- The `create_curve` command plots drain TotalCurrent (Y-axis) versus gate OuterVoltage (X-axis).

4.3 *svisual_IdVd_vis.tcl*

- **Purpose**: To automatically plot the family of $I_d - V_d$ output characteristics from the sdevice_IdVd_des.cmd simulation
- **Input**: The .plt files generated by sdevice_des_IdVd.cmd.
- **Output**: A single PNG image (IdVd.png) containing the complete family of output curves.

The key command sections are described as below

- The script's structure and commands are very similar to the svisual_IdVg_vis.tcl script, demonstrating a modular approach to visualization.
- The `create_curve` command plots drain TotalCurrent (Y-axis) versus drain OuterVoltage (X-axis).
- `export_view` saves the complete plot, containing the full family of curves, to a single PNG file.