Circuit Designer's Etiquette

Harald Pretl, Institute for Integrated Circuits (IIC), Johannes Kepler University, Linz

Release: Spring 2024

Prolog

A consistent naming and schematic drawing style, as well as VHDL/Verilog coding scheme, is a huge help in avoiding errors and increasing productivity. Even if just one person works on a design, the error rate is lowered. If multiple persons work together in a team, a consistent working style is a big help for smooth cooperation without misunderstanding each other's intentions. Consistency also helps to reuse existing blocks. In a well-done design, the documentation is included in the schematic/source code, so there is no searching for a piece of documentation somewhere else (which is often not found anyway).

Pins

- Name package pins (interfacing with the outside the IC) in **UPPERCASE**, and all internal signals in **lowercase**.
- Supply voltages like VDD/VCC and ground like VSS/GND need to start with either `VDD`, `VCC`, `VSS`, `VEE` or `GND`, plus a suitable suffix. Examples: `VDD1`, `VDD_AMP`, `vdd_ldo_out`, `VSS_ANA` (uppercase means connected to a pin, lowercase means a VDD is created on-chip by, e.g., an LDO).
- Preferred are `VDD` / `VSS` for CMOS and `VCC` / `GND` for bipolar circuits. In BiCMOS circuits `VDD` / `VSS` are preferred, as usually, the digital content is the major part.
- Digital signals in an analog schematic should start with `di_` (for digital input) or `do_` (for digital output). Example: `di_ctrl1`. In the rare case of a bi-directional digital signal `dio_` can be used.
- Name digital signals consistently: `di_pon` is active-high, `di_pon_b` is active-low (`_b` standing for the negating "bar"); as an alternative, this last signal could be named `di_disable`. `di_reset` is an active-high reset, but often a reset is active-low, so it needs to be named `di_reset_b` (an alternative is `di_resetn`).
- In mixed-voltage designs, it might be useful to append the voltage level of a signal to avoid connecting incompatible inputs and outputs. Example: `do_comp_1v2` or `di_poweron_3v3`.
- Digital buses always have the MSB to the left and LSB to the right. Example: `do_adc[7:0]`.
- Analog signals should start with a `v` for a voltage signal or `i` for a current signal. It is often useful to include a value for bias signals or make the naming meaningful. RF signals, which are often neither voltage nor current signals, start the name with `rf_`. Examples: Signal and pin names like `ibias_30u` (30uA of bias current), `vbg_1v2` (a bandgap voltage of 1.2V), `vin_p`, `v_filt_out_n`, and `rf_lna_i` speak for themselves.
- Appending analog signals with `_i` and `_o` might be useful if a clear direction is obvious in the signal flow. If a signal is bi-directional, it is better to skip `_io`.

- Consistently use pin types `input`, `output`, or `inout` to indicate signal flow. Power supply pins are of `inout` type.

Schematics

- In analog schematics, add a textual note about basic circuit performance. For example, in an amplifier, note things like suitable supply range, typical and w.c. current consumption, gain, GBW, input voltage range, PSRR, and other useful information.
- If a circuit has a quirk or is particularly clever, add a note on how it works, so others can understand the function without excessive analysis (reviewing a circuit should not be a brain teaser).
- Use provided borders or drawing templates for schematics, and fill the data in, like circuit designer name, date, change history, project name, etc.
- Use a versioning system for your data, and check in often. This avoids data loss, and going back to an earlier design stage is simple. `SVN` is often preferable to `GIT` for binary data.
- Draw uncluttered clear circuits. Ideally, the circuit function is apparent by inspection **quickly**. Everyone can obscure an inverter so that it takes 5 minutes to recognize it, but this is not a good design.
- Don't alter the standard grid setting while drawing schematics (also make sure that the pins in your drawn symbols are on the standard grid)! Off-grid schematic elements will haunt you and your colleagues forever!
- Once a schematic is finished, take the time to name component instances properly (you can use speaking names like `Rstab` or simply use `R1`, `R2`, etc.). Use iterated instances to clean up the circuit. Use wire bundles to clean up circuits where useful. A clever technique is to use bundles and iterated instances to efficiently draw large resistor ladders, for example (however, use with care).
- Avoid connection-by-name, as it makes the circuit hard to read. However, there is a fine line to not cluttering circuits. Signals with many connections (`vdd`, `vss`, `pon`, `pon_b`) are often better done with connection-by-name instead of drawing a wire.
- Some tools allow the use of colored wires, which might be used to mark signal paths, bias lines, etc. However, this should not be overdone; use it with care.
- If you add auxiliary elements like current probes, ensure they get proper treatment when creating the netlist for the LVS (some elements should be shorted, and some elements simply taken out). Ideally, only use a single schematic for simulation, LVS, etc. By using tool features this can usually be done, and avoids the need to keep multiple schematics of one block in sync.
- Use annotations in the schematics to (1) denote current levels in branches, (2) denote bias voltage levels, (3) explain the function of logic input signals, and (4) put in logic tables if not obvious.
- Add comments concerning the layout, like matching devices, certain considerations of placement, sensitive nodes, etc.
- Add simple ASCII diagrams for timing signals if useful.

- Name internal signals (signals connected to pins are anyway named like the port) in a meaningful way; this makes tracking signals in simulation or layout much easier (automatic net names like `net0032` are of not much help).
- Properly name instances, not just `I1` or `I2`; better is `amp1`, `inv2`, etc. (a descriptor in a tool output like `I1/I13/I5/net017` is not helpful; compare that to `adc1/bias/bg/vref_int`).
- On check-and-save, never ignore warnings; just fix them! They will annoy you and others forever and might flag critical design flaws.
- Name cells interpretably, ideally making the function clear already by the name. It is often useful to prefix or postfix a cell by the project name and design iteration. Example: In the project `GIGAPROJECT`, the cells which are changed in the second design step are prefixed with `g2_`, like `g2_amp_bias`. Of course, more letters as a project abbreviation are useful if a name collision is likely to happen.
- Cell names in lowercase are a good choice, as otherwise, capitalization leads to inconsistency in cell names. Use `_` to break words instead of `CamelCase`, like `amp_bias_startup`.
- When building a design, start with the hierarchy first; plan a suitable design structure, and define all interfaces. Implement simple behavioral models for every circuit block (either with controlled sources or using Verilog-A or VHDL/Verilog digital models). In this way, you can simulate the overall design early and find issues in the hierarchy or the interconnects. Then, populate the hierarchy with the detailed circuit designs in the leaf cells. At each point in the design process, you have a design that can be simulated, with some blocks as behavioral models and some blocks already designed. Try to avoid scattered circuit elements (digital or analog) in the hierarchy; it is better to push all components into the leaf cells.
- Avoid huge schematics, better break them down into smaller, maintainable, and self-contained blocks, and provide a simulation test bench for these simple blocks. In this way, later re-simulation across the hierarchy is easily possible.
- When building up the hierarchy, choose pin names and signal names as consistently as possible. Example: use the signal name `vref_int` when connecting two leaf cells with the pin names `vref_int_o` and `vref_int_i`.
- Avoid the excessive use of net breakers like small resistors, as they inhibit net tracing and can lead to simulation convergence issues. If a net breaker is needed (or a current should be probed) use a 0V dc voltage source.

Symbols

- Spend time drawing nice symbols! Ideally, the underlying circuit functionality is apparent by just looking at the symbol.
- Arrange the pins in a meaningful way.
- Group pins that belong together. An often useful arrangement is to locate the inputs on the left side, outputs on the right, digital control inputs at the bottom, and supplies at the top.
- Make the origin of a symbol in the top-left corner. In this way, symbols can be changed more easily, for example, by swapping out different versions of blocks.
- The cell name (and potentially library name) should be visible in the symbol, not only in the properties.

Design Robustness

- It is good practice to buffer incoming digital signals with a local inverter (connected to the local block supply) before connecting it to internal nodes. This improves the slew rate of the control signal and lowers the chance of unwanted cross-talk.
- Consider dummy elements for good matching, and try to make useful unit sizes of components. This will make the layout creation much smoother.
- The golden rule of good analog performance is good matching, and good matching is achieved by identical components (size, orientation, surroundings)! If the layout does not look nice (humans like symmetry), it will not perform well.
- Consider supply decoupling and bias voltage decoupling inside the cells. Often, dummy elements can be used for that. Be aware, however, of unwanted supply resonances (think bond wire `L` and decoupling `C`) and slow transients of bias nodes after disturbance.
- Always implement a proper power-down mode. Avoid floating nodes in off-mode. The better defined the on- as well as the off-mode are, the less the chance of leakage currents. Always simulate both modes (on and off), and also simulate a transient power-up of a circuit to identify issues with slow bias start or insufficient turn-off, or nasty feedback loop instabilities during transients.
- When drawing the first schematic, add parasitic capacitances to each node. If all nodes are labeled, a capacitor bank is easily put into one corner of the schematic with parasitic caps tied to the ground. Use 5fF as a starting value (and replace it later with the correct value from parasitic extraction). This accounts for some wiring parasitics in layout and helps to account for these layout impairments early in the design phase and later when simulating the schematic instead of the extracted netlist with parasitics.

Rules for Good Mixed-Signal and RF Circuits

- Separate analog and digital power supply, connect to package pins with multiple bond wires/bumps, and separate noisy and clean `vdd` / `vss` from each other!
- Prevent supply loops; keep `vdd` and `vss` lines close to each other (incl. bond wires and PCB traces)! This minimizes `L` and coupling factor `k`.
- Some prefer a massive (punched) ground plane, which is possible if you have enough metal levels. With a ground plane, the return path of a signal or supply line is just a few microns away.
- Use chip-internal decoupling capacitors, and decouple bias voltages to the **correct** potential (`vdd` or `vss`, or another node, depending on the circuit)!
- Use substrate contacts and guard rings to lower substrate crosstalk but use a quiet potential for connection; use triple-well if available! Connecting a guard-ring/substrate contact to a noisy supply is a prime noise injector (usually unwanted).
- Physically separate quiet and noisy circuits (at least by the epi thickness)!
- Reduce circuit noise generation as much as possible (avoid switching circuits if possible, use constant-current circuits instead, and use series/shunt regulators for supply isolation).
- Reduce sensitivity of circuits to interference (by using a fully differential design with high PSRR/CMRR, symmetrical layout parasitics, and good matching)!

VHDL/Verilog Coding Guide

These recommendations are specifically targeted at Verilog; however, they apply similarly to VHDL.

- Use automatic checkers (linters) to see whether your code contains errors or vulnerabilities. Commercial or open-source tools allow this, e.g., Icarus Verilog (`iverilog -g2005 -tnull FILE.v`) or Verilator (`verilator --lint-only -Wall FILE.v`).
- Write readable and maintainable code; use speaking variable names, and use a naming convention for inputs (beginning with `i_`) and outputs (beginning with `o_`). Active-low signals have an `_n` or `_b` in their name, like `i_reset_n`. Use comments to explain the intention.
- With a synchronous reset reset-related racing conditions are often avoided. If an asynchronous reset is desirable (which is often the case), ensure the reset signals are free from race conditions.
- Module-local registers and wires could append `_w` (for Verilog `wire`) or `_r` (for Verilog `reg`) to make their function clear. This is not required in SystemVerilog where the unified type `logic` should be used.
- Use an `assign` statement for logic as this often is easier to read than an `always @(*)` block. The ternary operator `COND ? TRUE : FALSE` can help with conditional assignments and is often a better choice than a (nested) `if ... else` statement.
- Declare all outputs explicitly with either `reg` or `wire`.
- Use local parameter definitions with `localparam` in a module to make the code easier to follow. Name parameters in **UPPERCASE**.
- Take care to reset all registers to a defined state (in simulation and HW).
- Use the rule of "one file per module." The filename shall match the module declaration.
- Use `` `default_nettype none `` at the beginning of a file containing a module definition. After the module you can use `` `default_nettype wire ``. This will add a safety net against typos in signal names.
- In a logic assign block, use `assign @(*) begin ... end` instead of spelling out the signals in the sensitivity list. Forgetting a signal could lead to serious mismatches between simulation and HW.
- Make your code flexible by making bit widths and other values parameterized using a `localparam` or module parameter.
- Be cautious of implicit type conversions and bit-width adaptions; better make explicit conversions and match bit widths in assignments.
- Use only blocking assignments (`=`) in `always @(*)` blocks, and only non-blocking assignments (`<=`) in clocked `always @(posedge ...)` blocks.

Further Reading

- Good information about drawing schematics, design testbenches, etc: https://circuit-artists.com
- Sutherland/Mills, *Verilog and SystemVerilog Gotchas - 101 Common Coding Erorrs and How to Avoid Them*, Springer, 2010
- B. Razavi, *The Analog Mind*, column in IEEE Solid-State Circuits Magazine