Thoughts and Advice

Carsten Wulff, carsten@wulff.no

I. Advice

This is some advice, use it, or ignore it, who cares.

Try to figure out what makes you happy, and do more of that

If you don't know how to say sorry when you do something stupid, learn.

When life sucks, run, or exercise, it's the only thing that works

Get a mac, time machine, and offsite backup. That ensures you'll never loose data.

Find a problem that you really want to solve, and learn a programming language to solve it. There is absolutely no point in saying "I want to learn programming", then sitting down with a book to read about programming, and expect that you will learn programming that way. It will not happen. The only way to learn programming is to program, a lot.

Learn to check your assumptions. You will make mistakes, and you need to get good at finding the mistakes you made.

Take your time to write a verification plan. And stick to it. Without sufficient simulation your circuit will not work.

Status	Abstraction	Design	Layout	Why
:construction:	Chip	SystemVerilog	digital	Complex connections, few analog interfaces
:construction:	Module	SystemVerilog	digital	Large amount of digital signals, few analog signals
:warning:	Block	Schematic	programmatic	Large amount of critical analog interfaces, few digital
:white_check_mark:	Cell	Netlist/JSON	compiled	Few analog interfaces, few digital interfaces
:white_check_mark:	Device	JSON	compiled	Polygon pushing
:white_check_mark:	Technology	JSON/Rules	compiled	Custom for each technology

A. What the team needs to know to design ICs

There are a multitude of tools and skills needed to design professional ICs. It's not likely that you'll find all the skills in one human, and even if you could, one human does not have sufficient bandwidth to design ICs with all it's aspects in a reasonable timeline

That is, unless we can find a way to make ICs easier.

The skills needed are

- *Project flow support*: **Confluence**, JIRA, risk management (DFMEA), failure analysis (8D)
- *Language*: **English**, **Writing English (Latex, Word, Email)**
- *Psychology*: Personalities, convincing people, presentations (Powerpoint, Deckset), **stress management (what makes your brain turn off?)**
- *DevOps*: **Linux**, bulid systems (CMake, make, ninja), continuous integration (bamboo, jenkins), **version control (git)**, containers (docker), container orchestration (swarm, kubernetes)
- *Programming*: Python, C, C++, Matlab *Since 1999 I've programmed in Python, Go, Visual BASIC, PHP, Ruby, Perl, C#, SKILL, Ocean, Verilog-A, C++, BASH, AWK, VHDL, SPICE, MATLAB, ASP, Java, C, SystemC, Verilog, Assembler, and probably a few I've forgotten.*
- *Firmware*: signal processing, algorithms, software architecture, security
- *Infrastructure*: **Power management**, **reset**, **bias**, **clocks**
- *Domains*: CPUs, peripherals, memories, bus systems
- *Sub-systems*: **Radio's**, **analog-to-digital converters**, **comparators**
- *Blocks*: **Analog Radio**, Digital radio baseband
- *Modules*: Transmitter, **receiver**, de-modulator, timing recovery, state machines
- *Designs*: **Opamps**, **amplifiers**, **current-mirrors**, adders, random access memory blocks, standard cells
- *Tools*: **schematic**, **layout**, **parasitic extraction**, synthesis, place-and-route, **simulation**, (System)Verilog, **netlist**
- *Physics*: transistor, pn junctions, quantum mechanics

B. Zen of IC design (stolen from Zen of Python)

When you learn something new, it's good to listen to someone that has done whatever it is before.

Here is some guiding principles that you'll likely forget.

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts (especially schematics).
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one **and preferably only one** obvious way to do it.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.

C. IC design mantra

To copy an old mantra I have on learning programming

> Find a problem that you really want to solve, and learn programming to solve it. There is no point in saying "I want to learn programming", then sit down with a book to read about programming, and expect

> that you will learn programming that way. It will not happen. The only way to learn programming is to do it, a lot. – Carsten Wulff

And run the perl program

```
s/programming/analog design/ig
```

D. Analog Design Process

- Define the problem, what are you trying to solve?
- Find a circuit that can solve the problem (papers, books)
- Find right transistor sizes. What transistors should be weak inversion, strong inversion, or don't care?
- Write a verification plan. Plan to simulate everything that could go wrong.
- Check operating region of transistors (.op)
- Check key parameters (.dc, .ac, .tran)
- Check function. Exercise all inputs. Check all control signals
- Check key parameters in all corners. Check mismatch (Monte-Carlo simulation)
- Do layout, and check it's error free. Run design rule checks (DRC). Check layout versus schematic (LVS)
- Extract parasitics from layout. Resistance, capacitance, and inductance if necessary.
- On extracted parasitic netlist, check key parameters in all corners and mismatch (if possible).
- If everything works, then your done.

On failure, go back as far as necessary

II. Stuff to ponder

Over a period of 10 months I was fortunate to spend some time at Electronics and Computer Engineering Department, University of Toronto. I was there as a grad student doing research on Comparator Based Switched Capacitor Circuits. Each Wednesday we had a meeting with the other Ph.D. and Master students, which was attended by Professor Ken Martin, Professor David Johns and Professor Trond Ytterdal. The quotes and tips here should not be taken as facts, but rather as "heads-up' ' statements. Take these tips as something that should be checked and thought about. I do not remember which quotes/tips came from which professor, or indeed which student. So here goes

*This is important:** Do not worry about unknowns. Make a list of unknowns and find a test to check whether the unknown is a problem. Fixing things based on guesses will cause trouble.
1) AC open, DC closed switch: In SPICE there is usually a switch or capacitor/inductor that has the behavior of being open at AC and closed at DC or visa versa. Useful for setting common mode voltages in simulation of differential operational transconductance amplifiers.
2) Measure capacitance: To measure capacitance on a node in a circuit simulation.

Bias the block Put a small dc current into the node Measure the delta V over a short time period Calculate capacitance from i = C dv/dt Always include a replica with a know capacitance value, i.e. a capacitor, to check your testbench.

3) On Analog Design: Normally source jitter will dominate

This comment was made in reference to a 10-bit 50MHz ADC. So if you're designing such a ADC you probably don't have to worry about jitter in the clock circuit. However, you should be careful about your clock input. I know some people do differential sinusoidal clocks and create a square wave clock on the inside of the chip. Using differential signaling will help with possible interference from nearby lines.
4) Distortion from ESD protection circuits: When you go to high resolutions (> 10 bit) and high speed (>50MHz) the non-linear capacitance of the ESD protection starts to matter. If you're doing an ADC above this area you should read Analysis and Measurement of Signal Distortion due to ESD Protection Circuits.
5) Add net names in layout: Always add net names to layout nets, this will help LVS to match nets. It will also save you when tracking down shorts.
6) Decouple to source node: In current mirrors, decouple to the source node. By decoupling between source and for example vss, any high frequency jumps on vss will also appear on the gate, thus the gate source voltage will stay constant and current will not change
7) Worry about current densities when routing > 20um: Metal wires on-chip have a maximum allowed dc current. This is due, among other things, to electromigration. At high current densities the aluminum atoms may migrate, and thus leave a void that might grow over time into to a discontinuity. Why exactly >20um I don't know, but it was mentioned in a meeting as a rule of thumb. Current densities are usually around 1mA/square, but varies with technology
8) Always shield signal lines above 10 bit level: If you're doing an ADC, or indeed any circuit, that requires > 10 bit accuracy you should shield your signal lines. On chip you use metal below, above and sides. The same for PCBs. Sensitive signals can be routed in in-between layers.
9) Use a current source to feed inverter based oscillators: Check non-overlapping clocks in slow, high temp and low vdd

For non overlapping clocks you should check that the two clocks just meet in slow corner, high temperature and low vdd. By meet I mean one clock should start to rise when the other is almost at zero. Supposedly this PVT corner is the worst for non-overlap, but I have not checked.
10) Analog Sampling: If possible, you should sample analog just before digital IO switches. In other words, sample during quiet time.
11) Cascode devices should be minimum size: You get less capacitance this way.
12) The unit transistor W/L should be around 10-20:
13) Place noisy digital blocks in deep N-well: By separating substrates you improve noise immunity
14) Shield analog blocks with deep N-well ring: Same thing as above
15) Always route differential signals differentially: Mismatch between parasitic capacitances/resistors in differential signal routing (differential means; two signals where one signal is phase shifted 180 degrees) can introduce errors. The error is

reduced if the parasitics are matched, since the differential system cancels some of the errors.
16) On-chip decoupling: Remember to check whether you need on-chip decoupling of references and power. In most designs you do need decoupling, especially if you run at high speeds (> 10MHz).
17) Variable delayed clock: If jitter is not important, and you want a variable delayed clock, you can use a current starved inverter Place a current source inside or on the outside of your inverter, and use a current mirror to control the maximum current.
18) On Calibration & Test: Use serial shift registers for calibration bits

It is common to include some off-line startup calibration circuits in ICs. For example to tune transconductances, resistances, capacitances, offset voltages etc. Usually this leads to some form of DAC that needs a digital input. For these digital inputs a serial shift register should be used. Indeed for any digital input that does not have to be synchronous, use serial shifting. The best is to have a commercial bus like SPI or I2S, but this might be overkill.

Two registers should be used, one long shift register and one parallel load register. First you shift in all you calibration bits, then you load them into the parallel register. The calibration DACs are connected to the parallel register. This is to avoid any funny stuff happening when you load your bits. It's good to have control over the state of your circuit at all times. A long shift register is not a problem, it does not cost much to add some more bits. In a recent ADC I made, the shift register was 272 bits long.

The calibration register should have 4 inputs: data, data clock, reset, load. The load signal is used to do the parallel load after shifting in all the bits. You should also include a data output so you can check what was loaded in.

On all inputs you should use a Schmitt trigger to improve noise immunity. Especially since the input data and clock may be feed from a computer with slow rise and fall times.
19) Design for test: Make sure that all on-chip DC voltages (bias points, power, references) are available off-chip for measurement. Either through probe pads or analog test multiplexers.

Use analog test multiplexer

Use an analog test multiplexer with T-switches to give you access to internal nodes. A T-switch has two transmission gates and one NMOS. The first transmission gate is placed close to the analog node you want to test, the second close the test output. The NMOS is placed on the output side (not analog node side) of the first transmission gate. When the T-switch is ON the two transmission gates are closed and the NMOS is open. When the T-switch is OFF the NMOS grounds the long line between the transmission gates, thus preventing leakage between different test points of the analog multiplexer.
20) Calibration currents: If you have a current source with off-line calibration, the calibration current should be +- 50% versus nominal.
21) Calibration DACs: For calibration DACs use 3 or 4 thermometer encoded bits and the LSBs binary encoded.
22) Access to gain boosters: If you're using gain boosters the boost voltage should be accessible off-chip.
23) Default calibration state: The calibration DACs should start up in a default state close to the expected state. Use inverters between the calibration register and calibration DACs to set the default state.
24) Crystal Oscillators: Crystal oscillators are used to generate a clean, low phase noise (low jitter), clock signal.
25) Careful circuit board design: If you move into the +10 bit accuracy range the circuit board (PCB) becomes important. Especially if you're running at high frequencies. If you have designed and produced an ADC, you want to test the ADC performance, not the PCB. So you might want to exclude the circuit board as an error source. To do this you can include a parallel ADC of same or higher resolution to test the PCB performance.

I know of cases where a guy did a 12 bit ADC, made a circuit board, and tested. The test showed very poor performance < 10 bit, and it turned out to be the circuit board. Most of the extra noise was because he hadn't shielded his input signal and taken care of routing of input signal. He spent in excess of 3 months to track down the problem. And the solution was to redo the circuit board and take better care of the input signal.

Short circuit input on your ADC to measure noise floor
26) Single ended to differential converter: In the range 0Hz - 2MHz you can use ADC driver, AD8138 Analog Devices For single ended to differential conversion in the range > 2MHz a transformer should be used.
27) Non-harmonic spurs: When measuring your ADC, if you have spurs that are not harmonics in your FFT you should try to change the clock frequency (sampling frequency) to see if the spurs change frequency as well. If they do, they may be aliased interference from nearby RF transmitters.
28) Low capacitance probes are available, as low as 0.1pF - 2pF:
29) Information sources: Handbook on filter synthesizing: Martin Snelgrove Phd thesis

State-Space Adaptive IIR Filters, David A. Johns

The Data Conversion Handbook, Walt Kester

Carsten Wulff received the M.Sc. and Ph.D. degrees in electrical engineering from the Department of Electronics and Telecommunication, Norwegian University of Science and Technology (NTNU), in 2002 and 2008, respectively. During his Ph.D. work at NTNU, he worked on open-loop sigma-delta modulators and analog-to-digital converters in nanoscale CMOS technologies. In 2006-2007, he was a Visiting Researcher with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. Since 2008 he's been with Nordic Semiconductor in various

roles, from analog designer, to Wireless Group Manager, to currently Principle IC Scientist. From 2014-2017 he did a part time Post.Doc focusing on compiled, ultra low power, SAR ADCs in nanoscale technologies. He's also an Adjunct Associate Professor at NTNU. His present research interests includes analog and mixed-signal CMOS design, design of high-efficiency analog-to-digital converters and low-power wireless transceivers. He is the developer of Custom IC Compiler, a general purpose integrated circuit compiler, and makes the occational video on analog integrated circuits at https://www.youtube.com/@analogicus. For full CV see https://analogicus.com/markdown-cv/.