Analog SystemVerilog

Carsten Wulff, carsten@wulff.no

Design of integrated circuits is split in two, analog design, and digital design.

Digital design is highly automated. The digital functions are coded in SystemVerilog (yes, I know there are others, but don't use those), translated into a gate level netlist, and automatically generated layout. Not everything is push-button automation, but most is.

Analog design, however, is manual work. We draw schematic, simulation with a mathematical model of the real world, draw the analog layout needed for the foundries to make the circuit, verify that we drew the schematic and layout the same, extract parasitics, simulate again, and in the end get a GDSII file.

When we mix analog and digital designs, we have two choices, analog on top, or digital on top.

In analog on top we take the digital IP, and do the top level layout by hand in analog tools.

In digital on top we include the analog IPs in the SystemVerilog, and allow the digital tools to do the layout. The digital layout is still orchestrated by people.

Which strategy is chosen depends on the complexity of the integrated circuit. For medium to low level of complexity, analog on top is fine. For high complexity ICs, then digital on top is the way to go.

Below is a description of the open source digital-on-top flow. The analog is included into GDSII at the OpenRoad stage of the flow.

The GDSII is not sufficient to integrate the analog IP. The digital needs to know how the analog works, what capacitance is on every digital input, the propagation delay for digital input to digital outputs , the relation between digital outputs and clock inputs, and the possible load on digital outputs.

The details on timing and capacitance is covered in a Liberty file. The behavior, or function of the analog circuit must be described in a SystemVerilog file.

But how do we describe an analog function in SystemVerilog? SystemVerilog is simulated in an digital simulator.

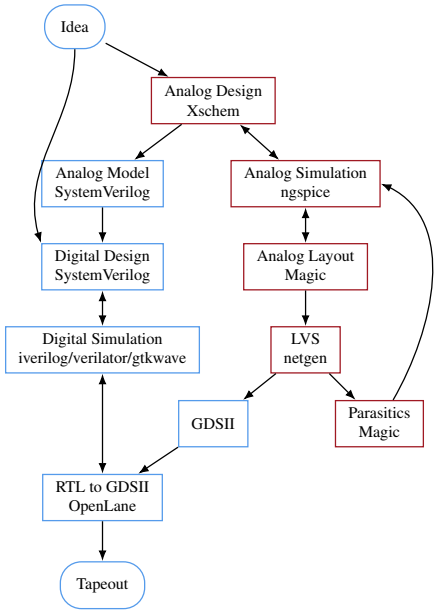

Figure 1: Mixed signal design flow, where the analog design is captured both as a schematic for ngspice and as a SystemVerilog model for the digital simulation

Digital Simulation

Conceptually, the digital simulator is easy.

- The order of execution of events at the same time-step does not matter
- The system is causal. Changes in the future do not affect signals in the past or the now

In a digital simulator there will be an event queue, see below. From start, set the current time step equals to the next time step. Check if there are any events scheduled for the time step. Assume that execution of events will add new time steps. Check if there is another time step, and repeat.

Since the digital simulator only acts when something is supposed to be done, they are inherently fast, and can handle complex systems.

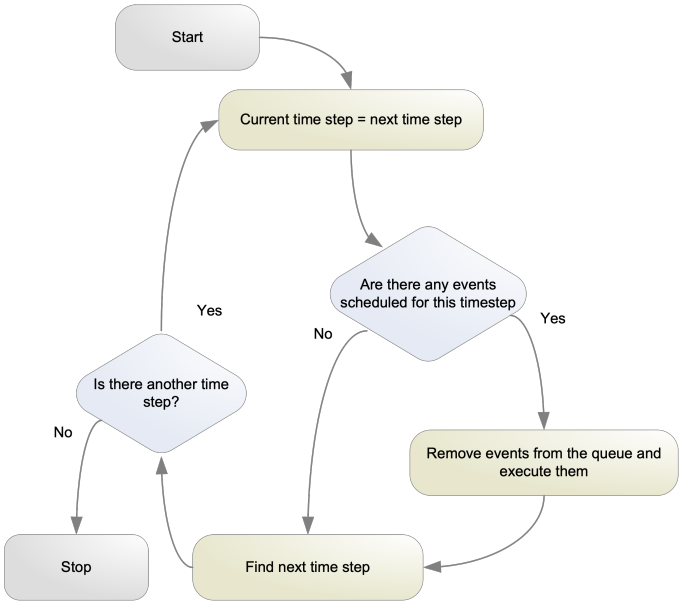

Figure 2: Flow chart of a digital simulator - advance to the next time step, execute the events scheduled there, and repeat until the queue is empty

It's a fun exercise to make a digital simulator. On my Ph.D I wanted to model ADCs, and first I had a look at SystemC, however, I disliked C++, so I made SystemDotNet

In SystemDotNet I implemented the event queue as a hash table, so it ran a bit faster. See below.

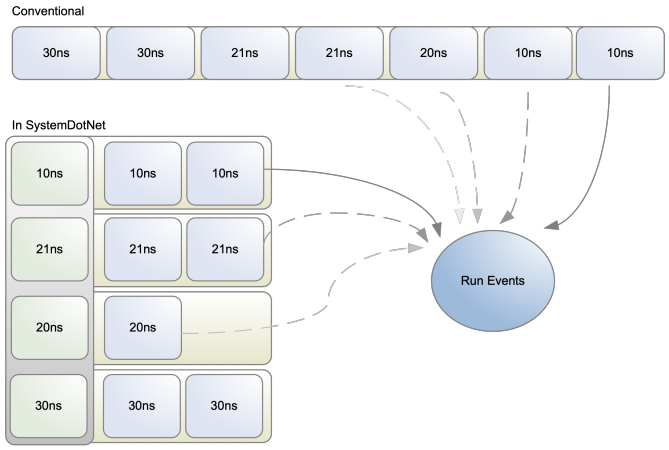

Figure 3: Event queue implemented as a hash table in SystemDotNet, where events at 10, 20, 21 and 30 ns are looked up by time step before being run. Source: the SystemDotNet project on SourceForge

Digital Simulators

There are both commercial and open source tools for digital simulation. If you've never used a digital simulator, then I'd recommend you start with iverilog. I've made some examples at dicex.

Commercial

- Cadence Excelium
- Siemens Questa
- Synopsys VCS

Open Source - iverilog/vpp - Verilator - SystemDotNet
Counter: Below is an example of a counter in SystemVerilog. The code can be found at counter_sv.

In the always_comb section we code what will become the combinatorial logic. In the always_ff section we code what will become our registers.

```
module counter(
               output logic [WIDTH-1:0] out,
               input logic              clk,
               input logic              reset
               );

   parameter WIDTH = 8;

   logic [WIDTH-1:0]                    count;
   always_comb begin
      count = out + 1;
   end

   always_ff @(posedge clk or posedge reset) begin
      if (reset)
        out <= 0;
      else
        out <= count;
   end

endmodule // counter
```

In the context of a digital simulator, we can think through how the event queue will look.

When the clk or reset changes from zero to 1, then schedule an event where if the reset is 1, then out will be zero in the next time step. If reset is 0, then out will be `count` in the next time step.

In a time-step where `out` changes, then schedule an event to set `count` to `out` plus one. As such, each positive edge of the clock at least 2 events must be scheduled in the register transfer level (RTL) simulation.

For example:

```
Assume `clk, reset, out = 0`

Assume event with `clk = 1`

0: Set `out = count` in next event (1)

1: Set `count = out + 1` using
   logic (may consume multiple events)

X: no further events
```

When we synthesis the code below into a netlist it's a bit harder to see how the events will be scheduled, but we can notice that clk and reset are still inputs, and for example the clock is connected to d-flip-flops. The image below is the synthesized netlist

It should feel intuitive that a gate-level netlist will take longer to simulate than an RTL, there are more events.

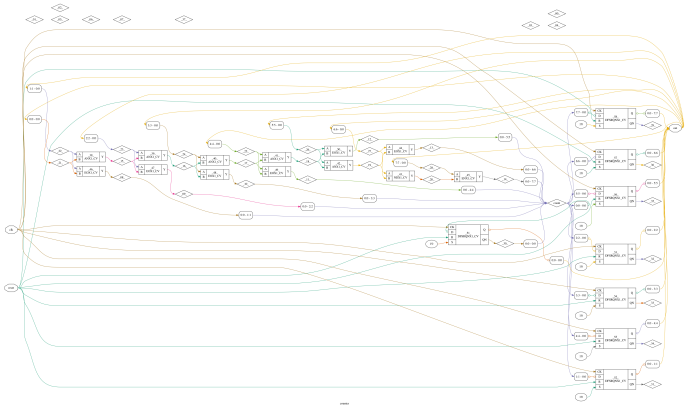

Figure 4: Gate level netlist of the counter after synthesis, with clk and reset driving eight D flip-flops and the adder logic

Transient Analog Simulation

Analog simulation is different. There is no quantized time step. How fast "things" happen in the circuit is entirely determined by the time constants, change in voltage, and change in current in the system.

It is possible to have a fixed time-step in analog simulation, for example, we say that nothing is faster than 1 fs, so we pick that as our time step. If we wanted to simulate 1 s, however, that's at least 1e15 events, and with 1 event per microsecond on a computer it's still a simulation time of 31 years. Not a viable solution for all analog circuits.

Analog circuits are also non-linear, properties of resistors, capacitors, inductors, diodes may depend on the voltage or current across, or in, the device. Solving for all the non-linear differential equations is tricky.

An analog simulation engine must parse spice netlist, and setup partial/ordinary differential equations for node matrix

The nodal matrix could look like the matrix below, i are the currents, v the voltages, and G the conductances between nodes.

$$\begin{pmatrix} G_{11} & G_{12} & \cdots & G_{1N} \\ G_{21} & G_{22} & \cdots & G_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ G_{N1} & G_{N2} & \cdots & G_{NN} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{pmatrix} = \begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_N \end{pmatrix}$$

The simulator, and devices model the non-linear current/voltage behavior between all nodes

as such, the G's may be non-linear functions, and include the v's and i's.

Transient analysis use numerical methods to compute time evolution

The time step is adjusted automatically, often by proprietary algorithms, to trade accuracy and simulation speed.

The numerical methods can be forward/backward Euler, or the others listed below.

- Euler
- Runge-Kutta
- Crank-Nicolson
- Gear [1]

If you wish to learn more, I would recommend starting with the original paper on analog transient analysis.

SPICE (Simulation Program with Integrated Circuit Emphasis) published in 1973 by Nagel and Pederson

The original paper has spawned a multitude of commercial, free and open source simulators, some are listed below.

If you have money, then buy Cadence Spectre. If you have no money, then start with ngspice.

Commercial - Cadence Spectre - Siemens Eldo - Synopsys HSPICE

Free - Aimspice - Analog Devices LTspice - xyce

Open Source - ngspice

Mixed Signal Simulation

It is possible to co-simulate both analog and digital functions. An illustration is shown below.

The system will have two simulators, one analog, with transient simulation and differential equation solver, and a digital, with event queue.

Between the two simulators there would be analog-to-digital, and digital-to-analog converters.

To orchestrate the time between simulators there must be a global event and time-step control. Most often, the digital simulator will end up waiting for the analog simulator.

The challenge with mixed-mode simulation is that if the digital circuit becomes to large, and the digital simulation must wait for analog solver, then the simulation would take too long.

Most of the time, it's stupid to try and simulate complex system-on-chip with mixed-signal , full detail, simulation.

For IPs, like an ADC, co-simulation works well, and is the best way to verify the digital and analog.

But if we can't run mixed simulation, how do we verify analog with digital?

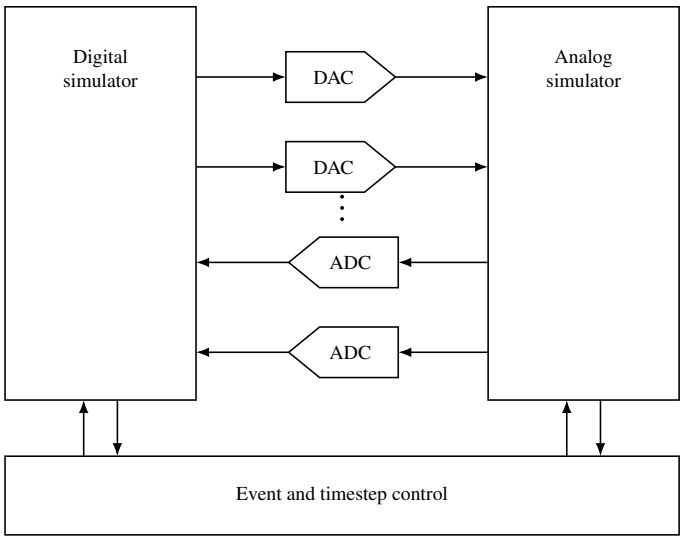

Figure 5: Mixed signal simulator, where a digital and an analog simulator exchange values through DAC and ADC connect modules under a shared event and timestep control

Analog SystemVerilog Example

TinyTapeout TT06_SAR

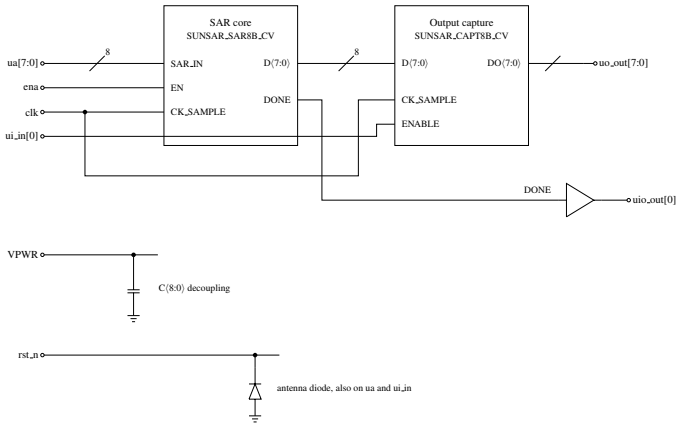

Figure 6: Top level of the TinyTapeout tt_um_TT06_SAR_wulffern 8-bit SAR ADC, redrawn from the xschem schematic: the SAR core, the output capture block, the buffered DONE on uio_out[0], the power decoupling and the antenna diodes on the inputs

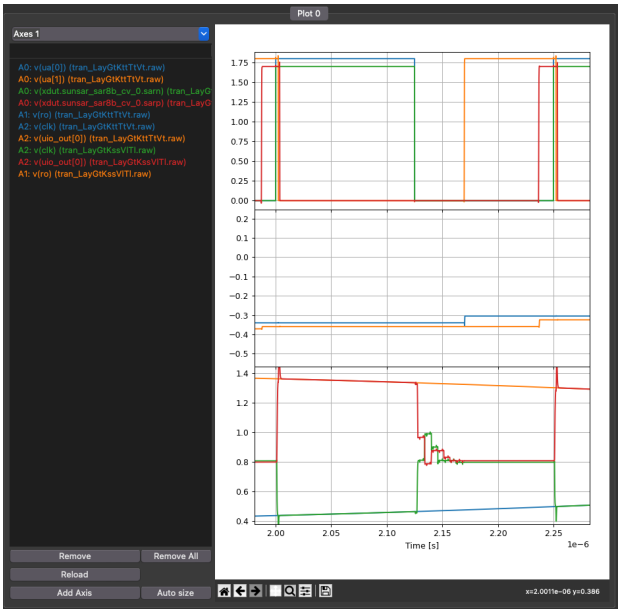

Figure 7: ngspice transient waveforms of the SAR, showing the sampled analog input, the clock and uio_out[0], and the sarp/sarn comparator inputs settling through the bit cycling

SAR operation

The key idea is to model the analog behavior to sufficient detail such that we can verify the digital code. I think it's best to have a look at a concrete example.

- Analog input is sampled when clock goes low (sarp/sarn)
- uio_out[0] goes high when bit-cycling is done
- Digital output (ro) changes when uio_out[0] goes high

```
//tt06-sar/src/project.v
module tt_um_TT06_SAR_wulffern (
                                    input wire         VGND,
                                    input wire         VPWR,
                                    input wire [7:0]   ui_in,
                                    output wire [7:0]  uo_out,
                                    input wire [7:0]   uio_in,
                                    output wire [7:0]  uio_out,
                                    output wire [7:0]  uio_oe,
`ifdef ANA_TYPE_REAL
                                    input real         ua_0,
                                    input real         ua_1,
`else
                                    // analog pins
                                    inout wire [7:0]   ua,
`endif
                                    input wire         ena,
                                    input wire         clk,
                                    input wire         rst_n
                                    );

//tt06-sar/src/tb_ana.v
`ifdef ANA_TYPE_REAL
   real         ua_0  = 0;
   real         ua_1 = 0;

`else
   tri [7:0]    ua;
   logic        uain = 0;
   assign ua = uain;
`endif

`ifdef ANA_TYPE_REAL
   always #100 begin
      ua_0 = $sin(2*3.14*1/7750*$time);
      ua_1 = -$sin(2*3.14*1/7750*$time);
   end
`endif
```

```
//tt06-sar/src/tb_ana.v
   tt_um_TT06_SAR_wulffern dut (
                                .VGND(VGND),
                                .VPWR(VPWR),
                                .ui_in(ui_in),
                                .uo_out(uo_out),
                                .uio_in(uio_in),
                                .uio_out(uio_out),
                                .uio_oe(uio_oe),
`ifdef ANA_TYPE_REAL
                                .ua_0(ua_0),
                                .ua_1(ua_1),
`else
                                .ua(ua),
`endif
                                .ena(ena),
                                .clk(clk),
                                .rst_n(rst_n)
                                );

#tt06-sar/src/Makefile
runa:
    iverilog  -g2012 -o my_design -c tb_ana.fl -DANA_TYPE_REAL
    vvp -n my_design

rund:
    iverilog  -g2012 -o my_design -c tb_ana.fl
    vvp -n my_design

   //tt06-sar/src/project.v
   //Main SAR loop
   always_ff @(posedge clk or negedge clk) begin
      if(~ui_in[0]) begin
         state <= OFF;
         tmp = 0;
         dout = 0;
      end
      else begin
         if(OFF) begin

         end
         else if(clk == 1) begin
            state = SAMPLE;
         end
         else if(clk == 0) begin
            state = CONVERT;
      `ifdef ANA_TYPE_REAL
            smpl = ua_0 - ua_1;
            tmp = smpl;

            for(int i=7;i>=0;i--) begin
               if(tmp >= 0) begin
                  tmp = tmp - lsb*2**(i-1);
                  if(i==7)
                    dout[i] <= 0;
                  else
                    dout[i] <= 1;
               end


               else begin
                  tmp = tmp + lsb*2**(i-1);
                  if(i==7)
                    dout[i] = 1;
                  else
                    dout[i] = 0;
               end
            end
`else
            if(tmp == 0) begin
               dout[7] <= 1;
               tmp <= 1;

            end
            else begin
               dout[7] <= 0;
               tmp  = 0;
            end
`endif

         end
         state = next_state;
      end // else: !if(~ui_in[0])
   end // always_ff @ (posedge clk)

   //tt06-sar/src/project.v
   always @(posedge done) begin
      state = DONE;
      sampled_dout = dout;
   end

   always @(state) begin
      if(state == OFF)
        #2 done = 0;
      else if(state == SAMPLE)
        #1.6 done = 0;
      else if(state == CONVERT)
        #115 done = 1;
   end
```

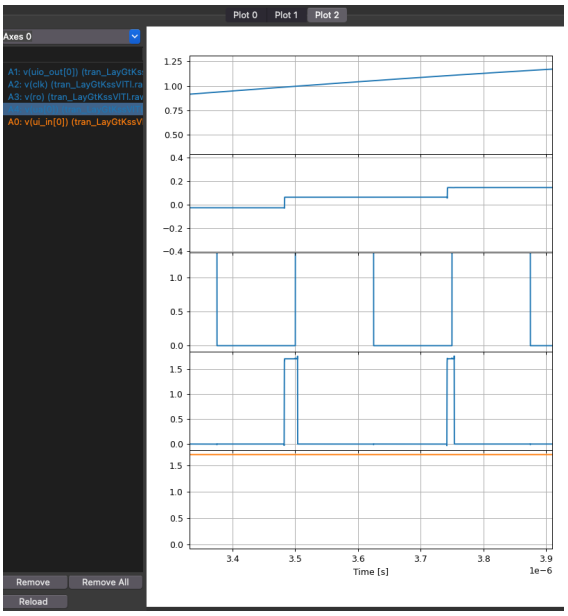

Figure 8: ngspice transient of the SAR at layout corner, with the ramping analog input, the digital output code, the clock and the uio_out[0] done pulses

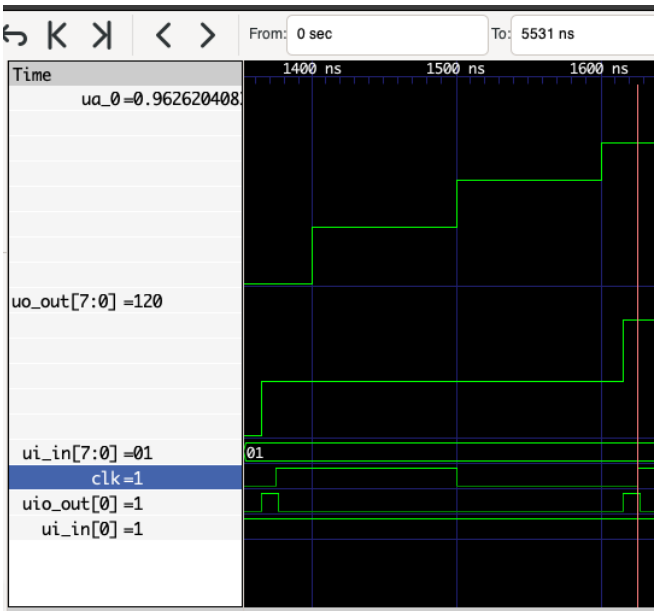

Figure 9: The same behaviour in gtkwave from the SystemVerilog real-number model, where uo_out[7:0] steps as ua_0 ramps

SUMMARY

The one-page version of this chapter:

- Digital simulators are fast because they only compute at events; analog simulators are slow because they solve the whole matrix at every timestep
- Analog SystemVerilog models the analog blocks with real-valued signals inside the digital simulator - the mixed-signal chip simulates at digital speed
- The model is a contract: it must match the schematic's behavior at the pins, and drift between them is found in co-simulation
- A SAR ADC in TinyTapeout shows the whole flow: the same testbench drives the Verilog model and the transistor netlist
- Verification from zero supply, every time - models make that cheap enough to actually do

WOULD YOU LIKE TO KNOW MORE?

For more information on real-number modeling I would recommend The Evolution of Real Number Modeling

[1] C. Gear, "Simultaneous numerical solution of differential-algebraic equations," *IEEE Transactions on Circuit Theory*, vol. 18, no. 1, pp. 89–95, 1971, doi: 10.1109/TCT.1971.1083221.

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/.