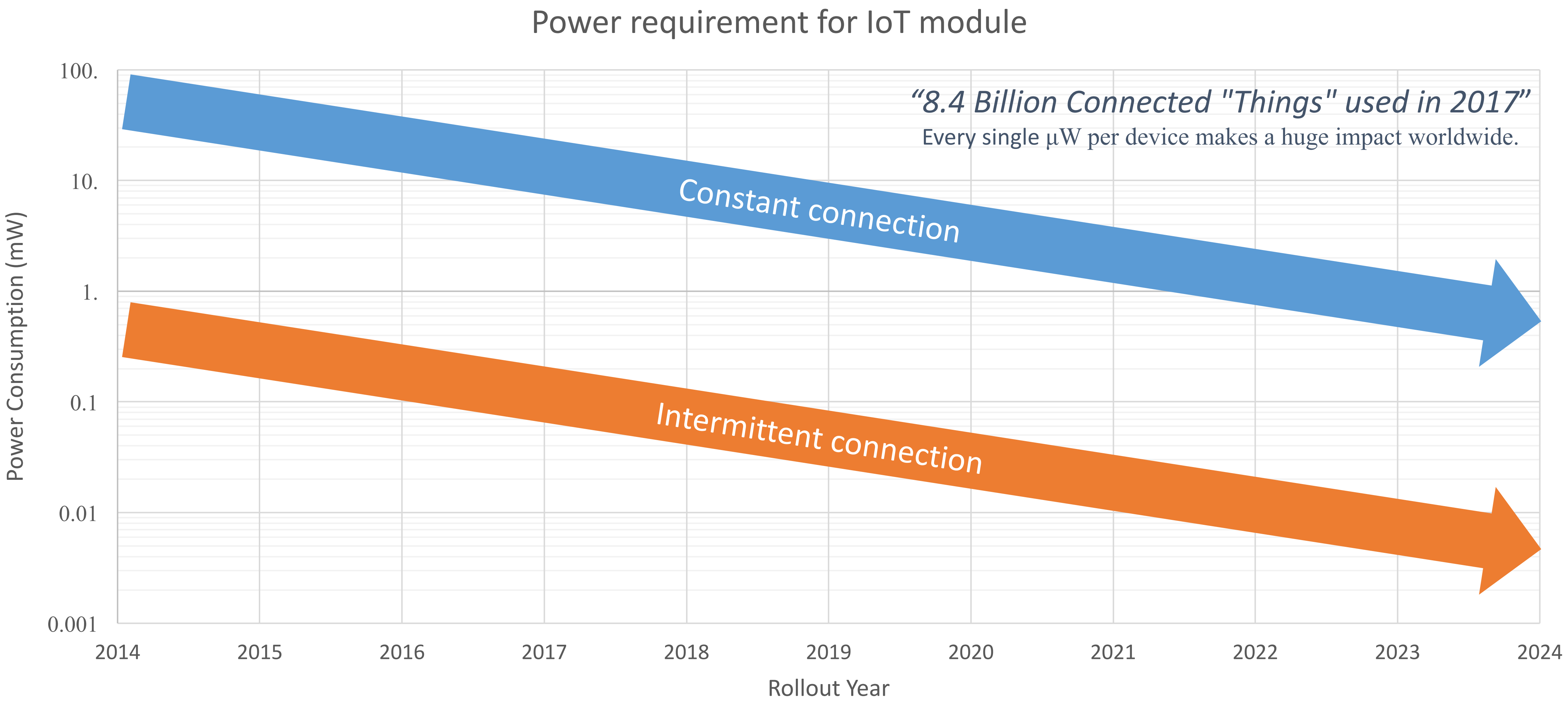
Accelerating Analog Design with SystemC-AMS

October 28, 2019

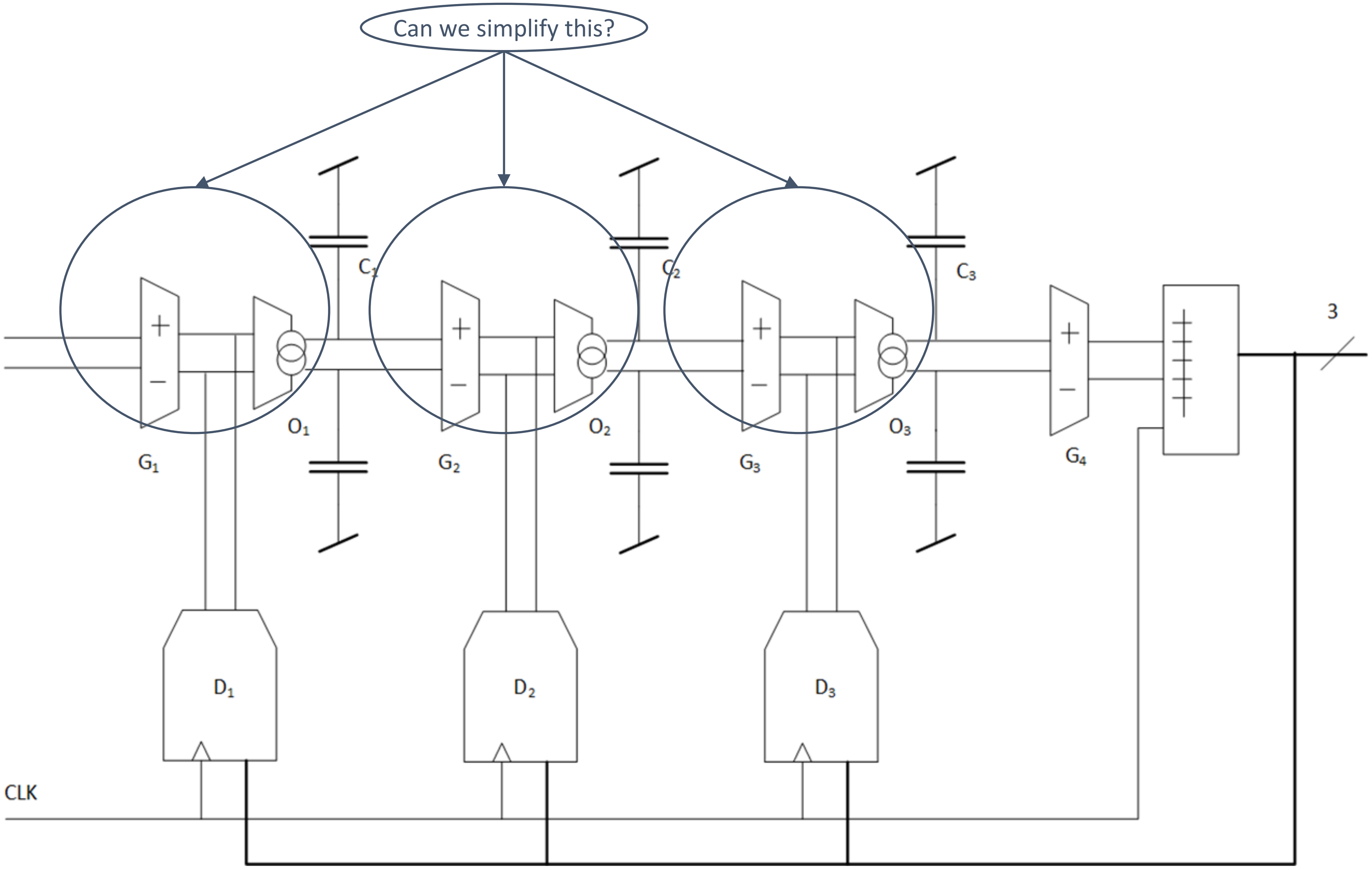
Américo Dias

Motivation and Problem Statement



For illustration purposes only. Quote: goo.gl/MQj1St

Case study: How to reduce existing SDM power consumption?

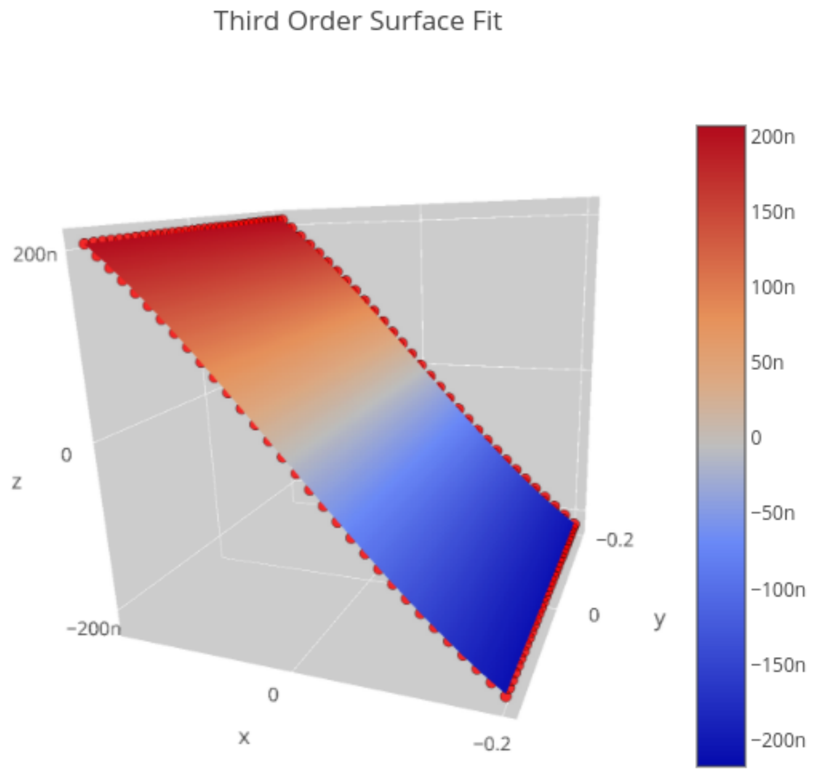


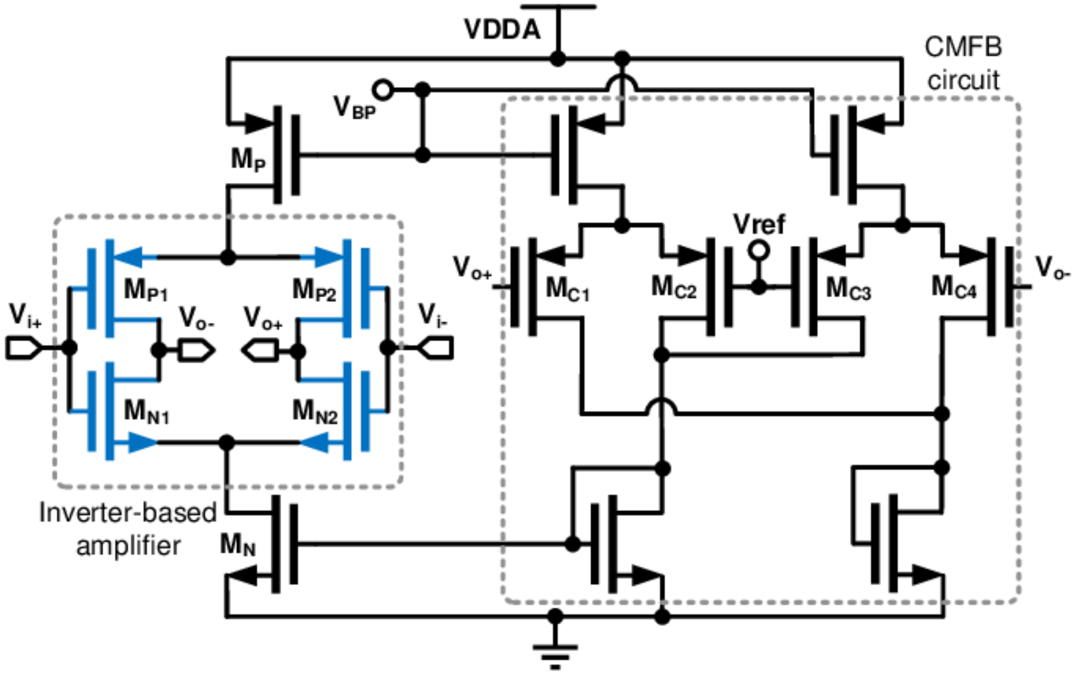
Design Strategy

1. Create a macromodel based modulator.
2. Use the ideal model to set a reference point.
3. Study alternative implementations and use polynomial fitting to capture their behavior.
4. Implement the curve fitting results in the macromodel simulation to study the degradation on the circuit performance (SNR, SNDR, etc).

Predict circuit behaviour before spending time on the practical implementation!
(i.e. Feasibility Study)

Inverter based gm cell

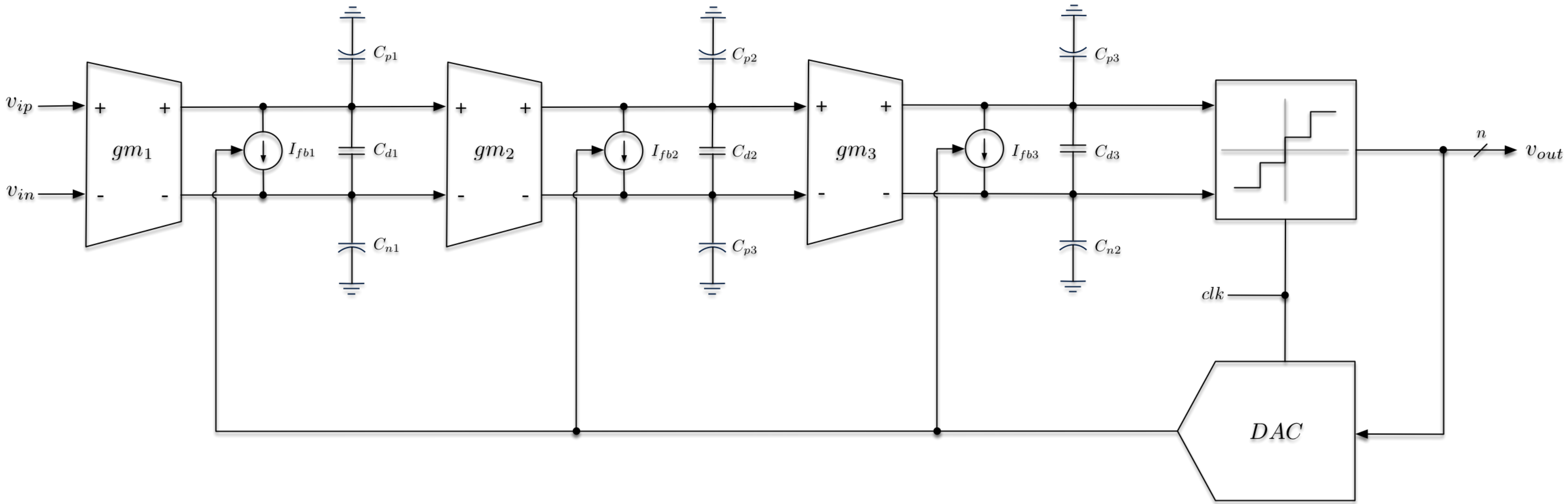




Tao, Sha & Chi, Jiazuo & Rusu, Ana. (2015). Design Considerations for Pipelined Continuous-Time Incremental Sigma-Delta ADCs. 10.1109/ISCAS.2015.7168808.

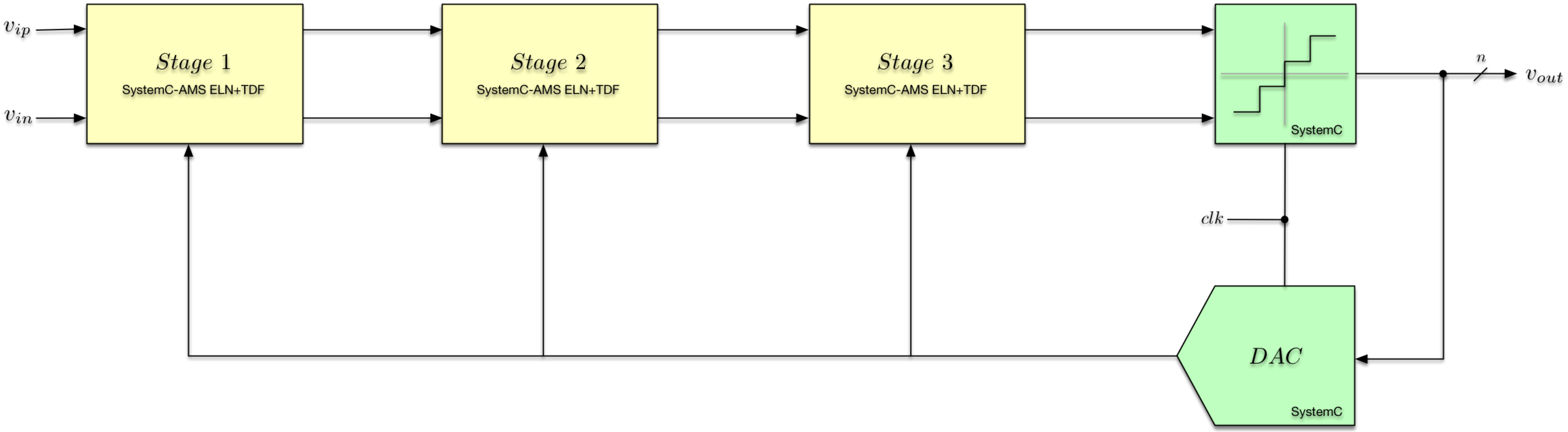
$$I_{out}(v_{in}, v_{out}) = c_0 + c_1 v_{in} + c_2 v_{out} + c_3 v_{in}^3 + c_4 v_{out}^3 + c_5 (v_{in} - v_{out})^2 (v_{in} + v_{out})$$

Spectre Macromodel Implementation



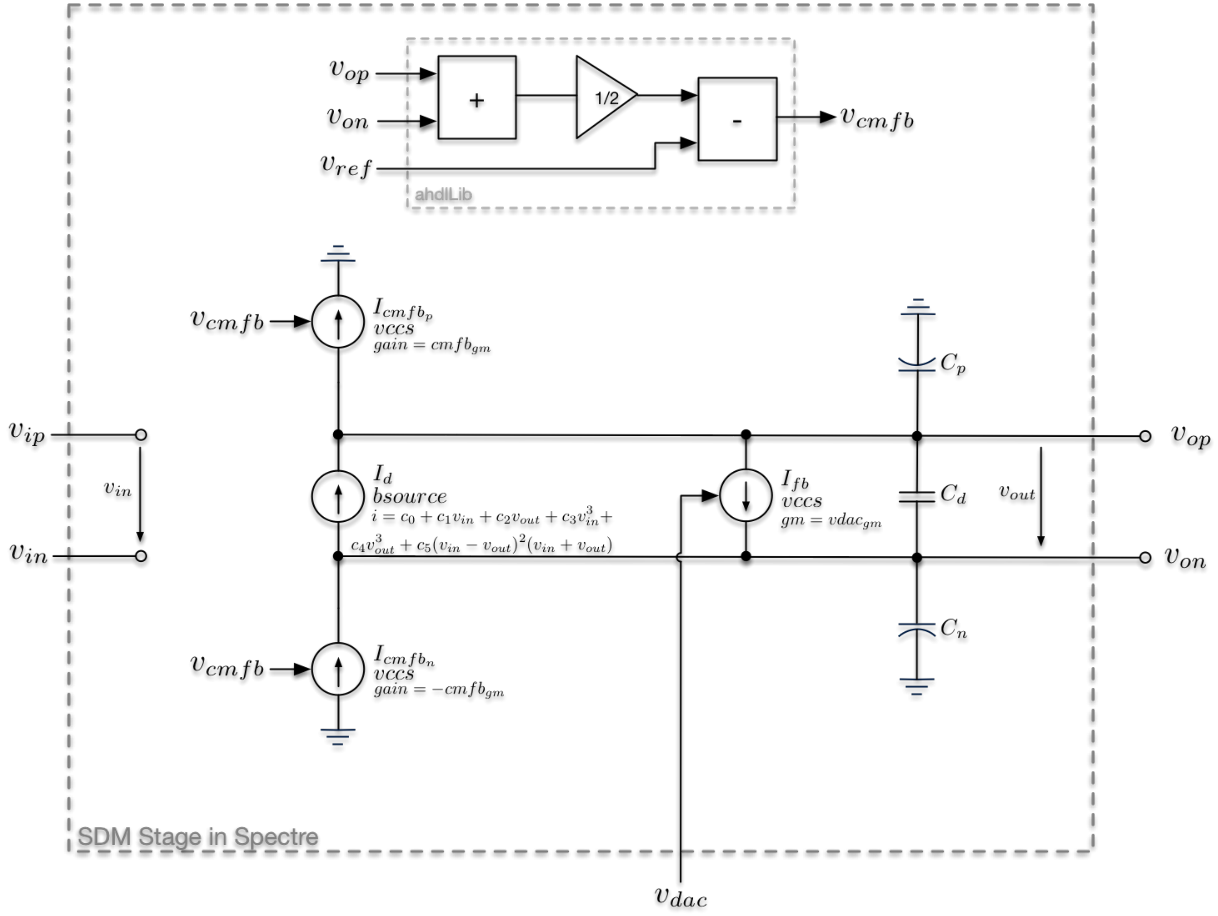
- Basic spectre/spice components like capacitors, vccs, and bsource for modeling the third order fit.
- VerilogA for the quantizer and dac, and part of the common mode feedback (not shown).

SystemC-AMS Model Implementation

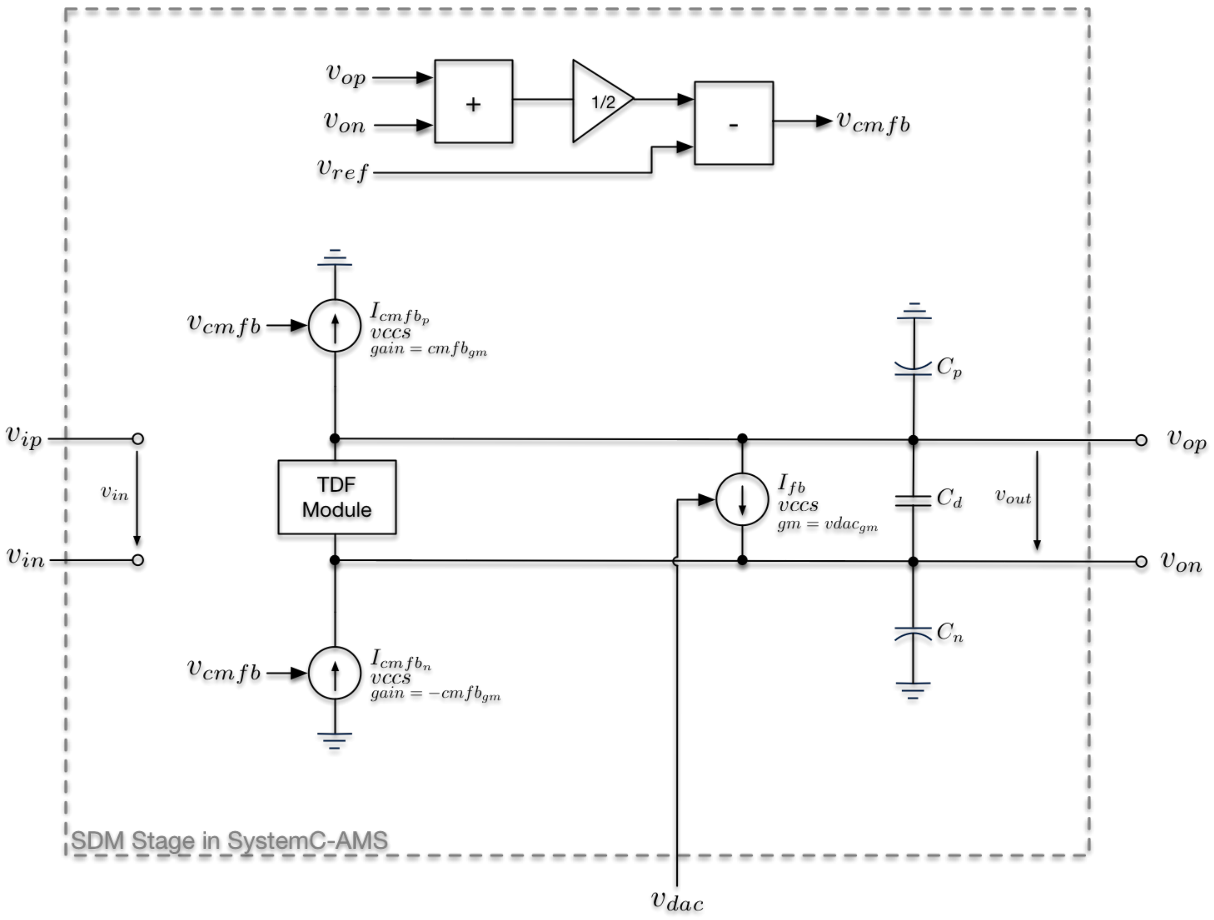


- ELN (Electrical Linear Networks): Has most of the passive components and sources like spice, however doesn't have bsource and TDF to implement the third order fit.
- TDF (Timed Data Flow): The TDF model of computation shall define the procedural behavior that processes samples, which are tagged in time.
- Quantizer and DAC can be implemented in standard SystemC.

GM Cell implementation in Spectre

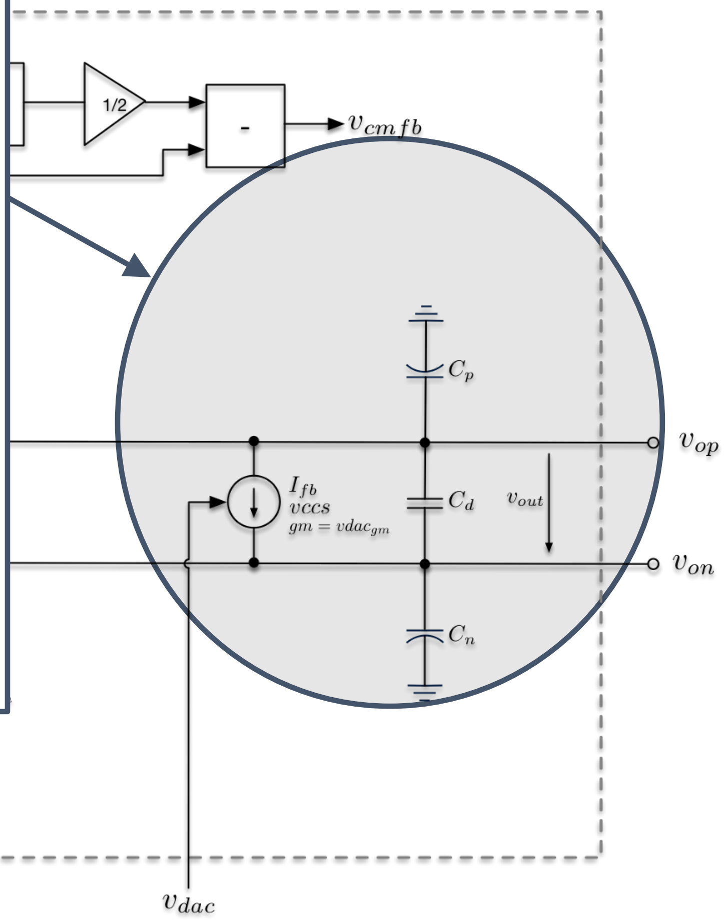


GM Cell implementation in SystemC-AMS



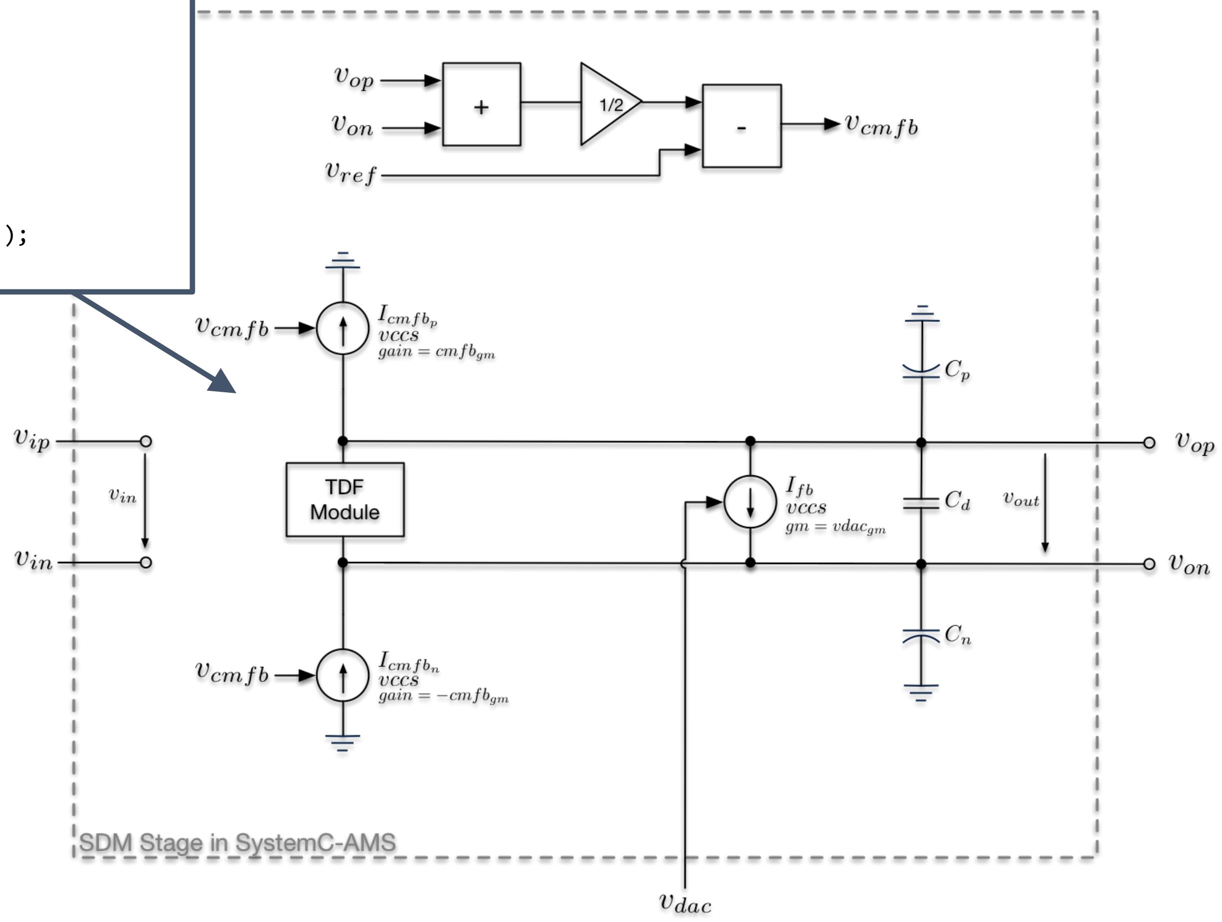
GM Cell implementation in SystemC-AMS

```
// Capacitors

cdiff = new sca_eln::sca_c("cdiff", cdiff_value);
cdiff->p(vop);
cdiff->n(von);

cp = new sca_eln::sca_c("cp", cp_value);
cp->p(vop);
cp->n(vop_cap);

cn = new sca_eln::sca_c("cn", cn_value);
cn->p(von);
cn->n(von_cap);

// ESR for better convergence

ifb_r_p = new sca_eln::sca_r("ifb_r_p", 1e-9);
ifb_r_p->p(vop_cap);
ifb_r_p->n(gnd);

ifb_r_n = new sca_eln::sca_r("ifb_r_n", 1e-9);
ifb_r_n->p(von_cap);
ifb_r_n->n(gnd);

// DAC feedback
ifb_p = new sca_eln::sca_de::sca_isource("ifb_p", vdac_gm);
ifb_p->p(vop);
ifb_p->n(von);
ifb_p->inp(sc_in_vdac);
```



GM Cell implementation in SystemC-AMS

```
/**
* Processing thread
*/
void sca_tdf_sdm_idiff_calc::processing(void) {
double vin = sca_tdf_in_vin.read();
double vout = sca_tdf_in_vout.read();

sca_tdf_out_iout.write(mult*(coef_0 +
   coef_1 * vin +
   coef_2 * vout +
   coef_3 * pow(vin,3) +
   coef_4 * pow(vout, 3) +
   coef_5 * pow(vin-vout,2)*(vin+vout)));
}
```



Quantizer in VerilogA and SystemC-AMS

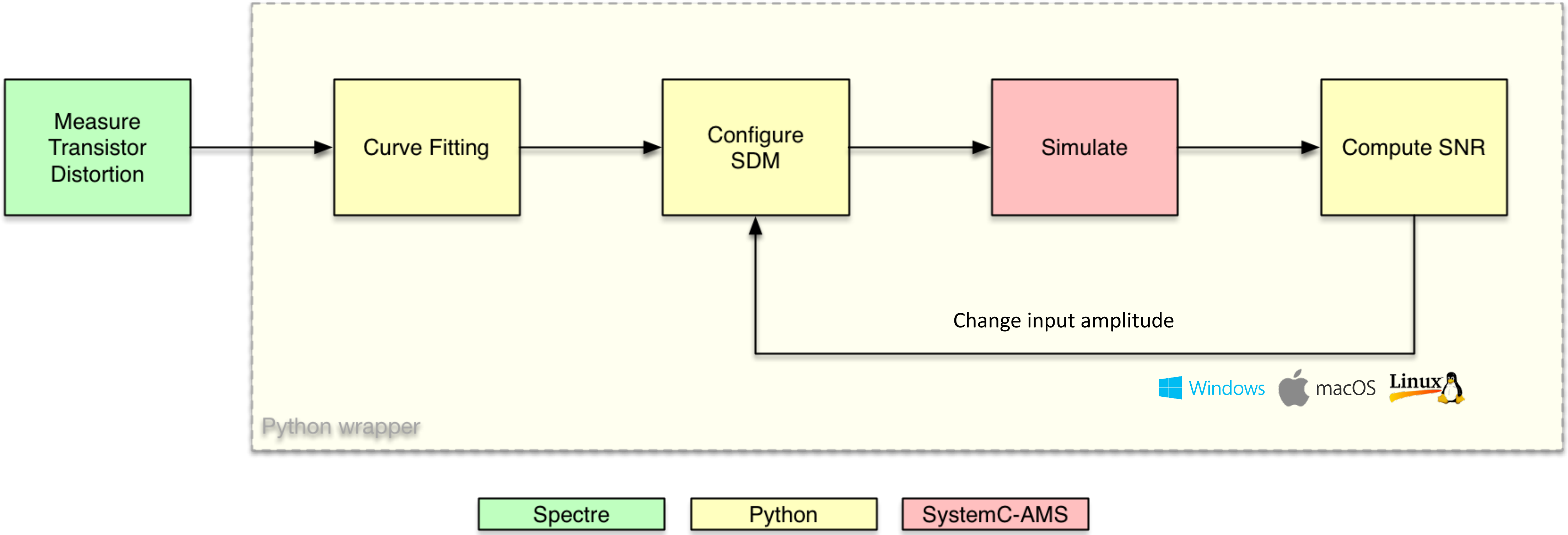
VerilogA

```
@ (cross(V(vclk) - vtrans_clk, 1)) begin
vdiff = gain*(V(vinp)-V(vinn));

if(vdiff <= level[0]) begin
vo = -2;
end else if (vdiff > level[0] && vdiff <= level[1]) begin
vo = 1;
end else if (vdiff > level[1] && vdiff <= level[2]) begin
vo = -0;
end else if (vdiff > level[2] && vdiff <= level[3]) begin
vo = 1;
end else if (vdiff > level[3]) begin
vo = 2;
end

end

//
// assign the outputs
//
V(vout) <+ transition( vo, tdel, trise, tfall );
```

SystemC-AMS

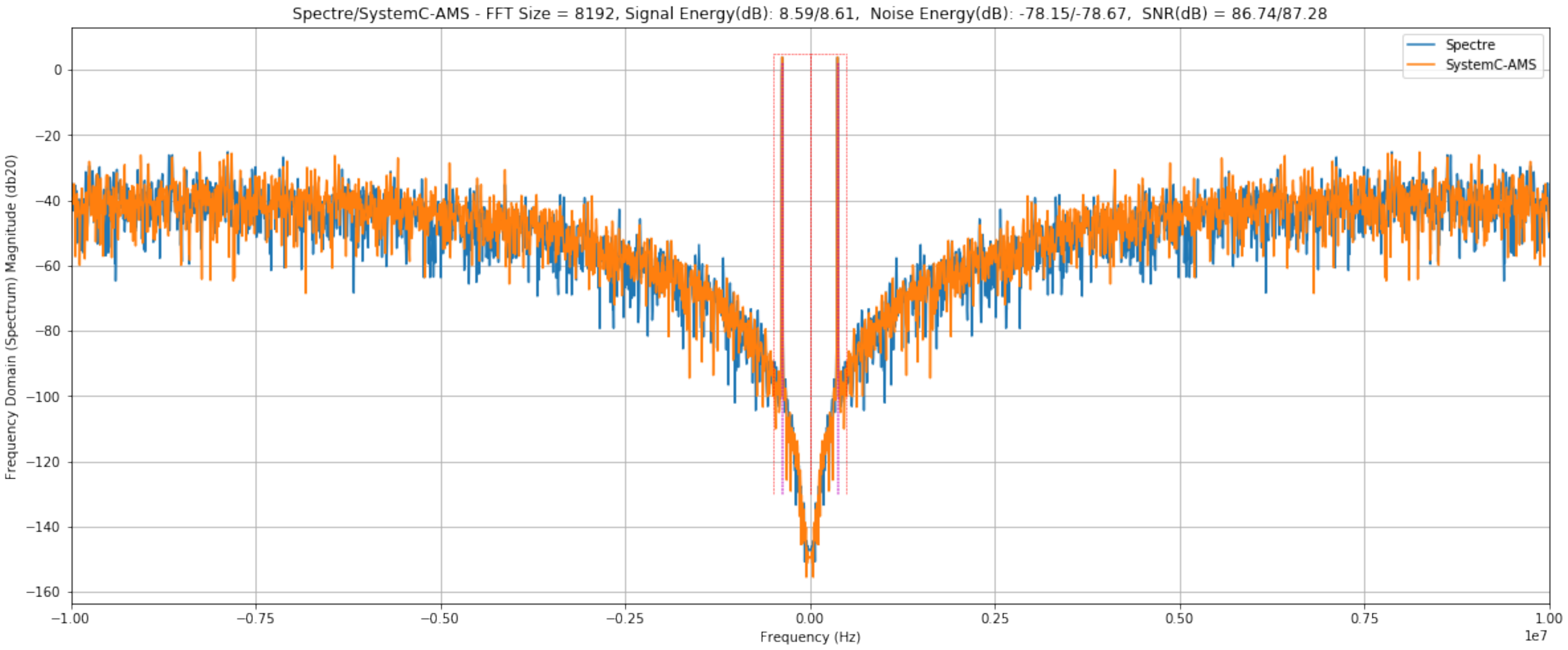
```
/**
* Processing thread
*/
void sc_sdm_quantizer::sig_proc(void) {

   double vdiff = input_gain * (sc_in_vinp.read() - sc_in_vinn.read());

   sc_out_sync.write(!sc_out_sync.read());

    if (vdiff <= threshold[0])
        sc_out_vout.write(-2);
    else if (vdiff > threshold[0] && vdiff <= threshold[1])
        sc_out_vout.write(-1);
    else if (vdiff > threshold[1] && vdiff <= threshold[2])
        sc_out_vout.write(0);
    else if (vdiff > threshold[2] && vdiff <= threshold[3])
        sc_out_vout.write(1);
    else if (vdiff > threshold[3])
        sc_out_vout.write(2);
}
```

Design flow

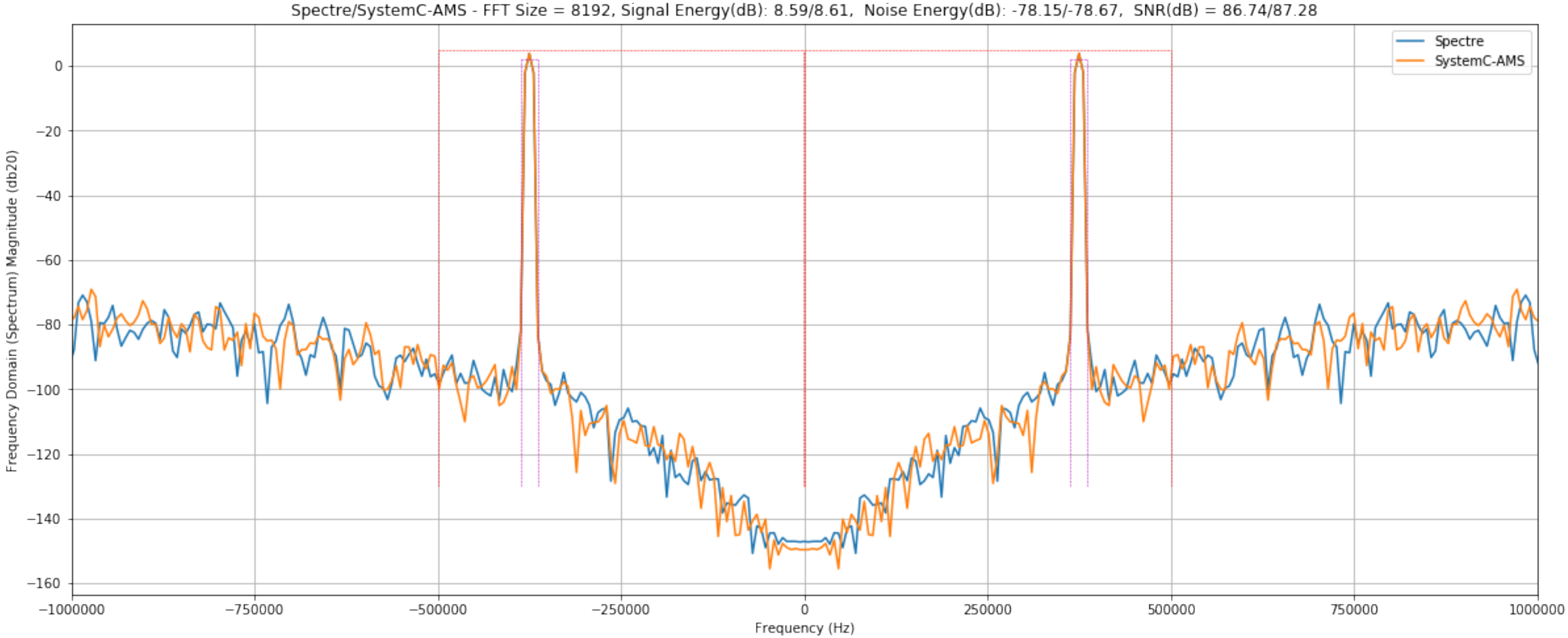


- **Specte** (or other spice simulator) is always required to measure the transistor distortion for specific operating point. <u>This is done only once.</u>
- **Python** can be used to automatically run the rest of the flow, configuring and launching the SystemC-AMS executable.
- **Python** can replace **Matlab** using the right libraries like numpy.
- **Python + SystemC-AMS** solution is virtually free and run in all major operating systems.
- Each **SystemC-AMS** simulation takes less than 1 second. **Spectre** model requires more than 2 minutes.

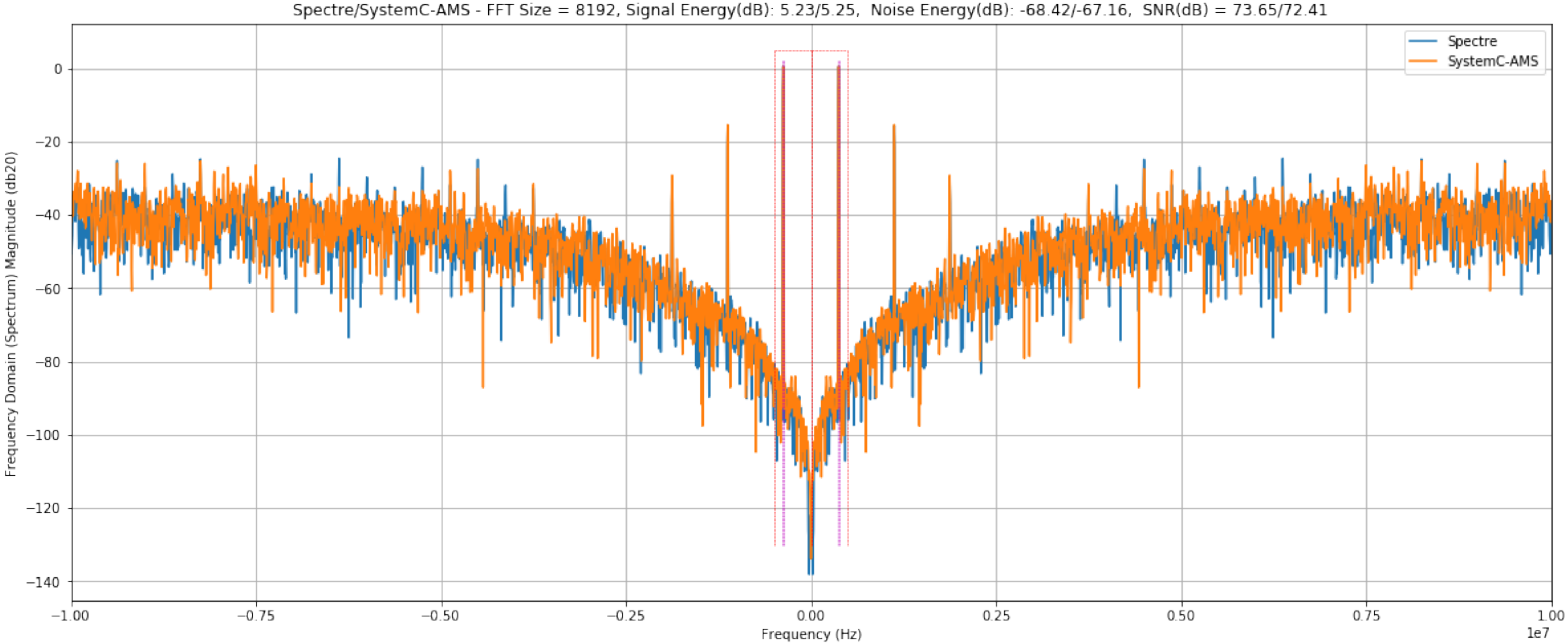
Maximum SNR with ideal model



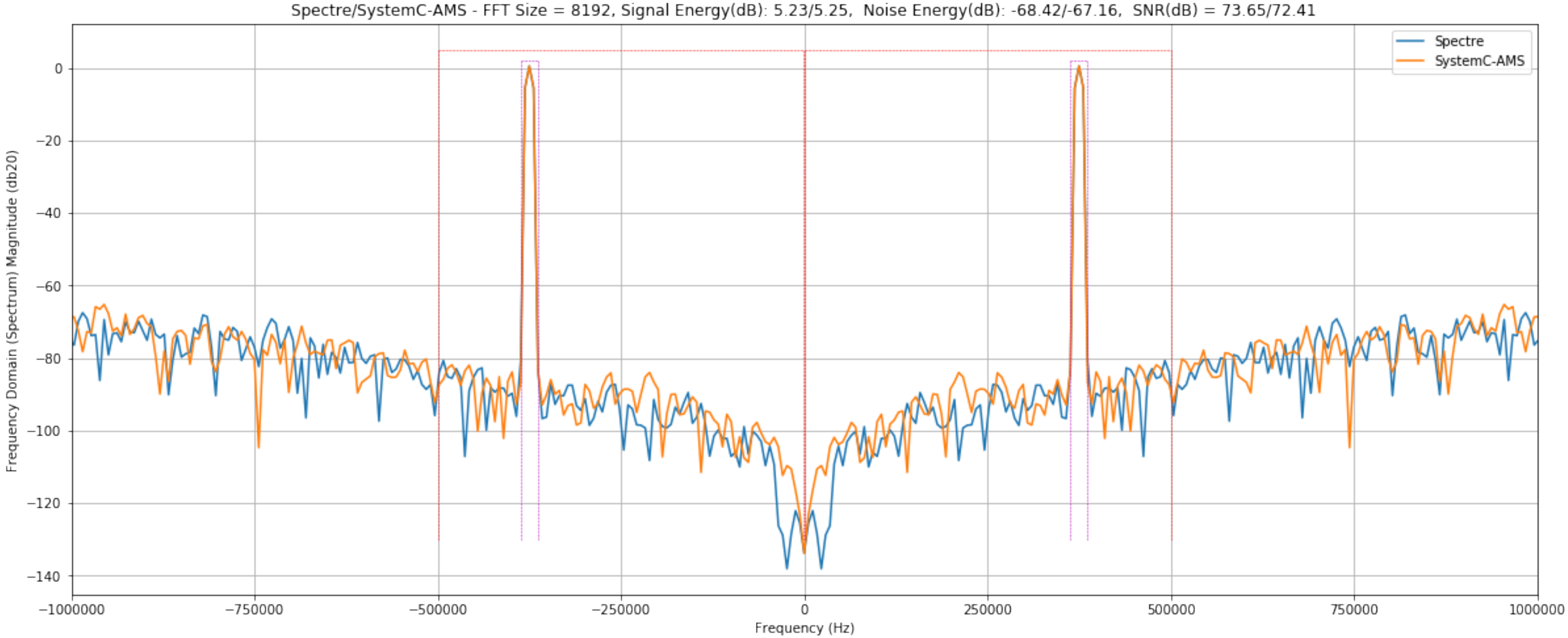
Maximum SNR with ideal model (zoom)



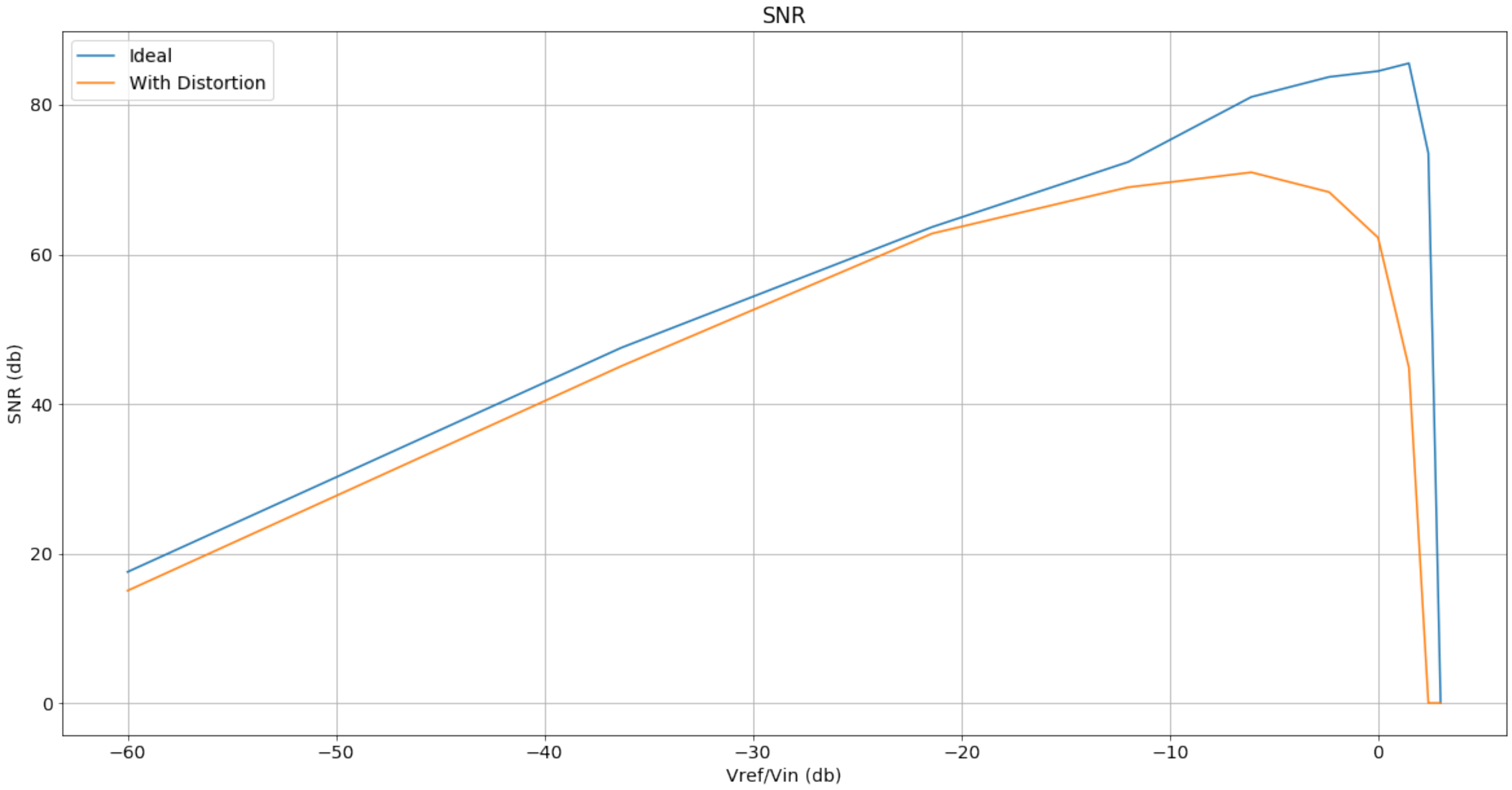
SNR with polynomial fit



SNR with polynomial fit (zoom)



Dynamic range



Conclusion

Pros:

- SystemC-AMS is a good tool for early phase studies.
- Much faster than spice (120x in this example!)
- By developing his own simulator, the designer gains a better understanding of the circuit.
- The simulation can be customized and tuned for the designer needs.
- The simulator runs on local machine (independently of the OS).
- It is free and can be easily controlled by Python (or other script languages).

Cons:

- It's more difficult and time consuming to develop a systemc-ams model than spice (learning curve).
- The designer need to ensure its correctness and accuracy (compare with spice for peace of mind).

Other resources

- IEEE 1666.1-2016 Standard: https://standards.ieee.org/findstds/standard/1666.1-2016.html
- Wikipedia Page: https://en.wikipedia.org/wiki/SystemC_AMS
- How to compile SystemC/SystemC-AMS libraries:
  - Windows: https://goo.gl/NjQBtD
  - Linux: https://goo.gl/rC9ZLv
  - Mac: https://goo.gl/69nfz8
- PLL design example (personal project): https://goo.gl/ZaFExo

Thank you