An example of differential Voltage Controlled Oscillator (VCO) and Differential to Single Ended Converter (D2S)

- DELAY CELL BIASING
- D2S CONVERTER
- VCO+D2S BEHAVIOUR
- COMPLETE VerilogHDL MODEL

Özgür Çobanoğlu
Univ. and INFN of Turin
http://cern.ch/Ozgur.Cobanoglu
http://www.ph.unito.it/~cobanogl
• Highest gain (and the centroids for Gaussians of ANY variation) should be @ trip point ($V_{\text{in}}=V_{\text{out}}=V_{\text{dd}}/2=600\text{mV}$)

  ➢ Size the transistors to set the trip point to 600mV
  ➢ Maximize gain (and centralize ALL kinds of variation) @ trip point
  ➢ Consider process corners
TEST CIRCUIT

Size, Finger, Mult., Model

12u/120n, f=3, m=6, rf
34u/120n, f=17, m=1, lvt_rf
64u/1u, f=1, m=1, rf

Bias

Delay Cell

This is the circuit for the next page.

Ö.Ç.
Biasing point of the delay cell for typical mean and corners.

First plot is the AC gain as a function of frequency and the second one is the differential output as a function of differential input.

Variation of AC gain of the delay cell as a function of Vctrl and Common Mode of differential input (from AC analysis).
Considering CM variation, proceed as for delay cell:

- Highest gain (and the centroids for Gaussians of ANY variation) should be @ trip point (Vin=Vout=Vdd/2=600mV)
  - Size the transistors to set the trip point to 600mV
  - Maximize gain (and centralize ALL kinds of variation) @ trip point
  - Consider process corners
First trial was failure:
Duty cycle != 50%, so I tried different architectures before coming back to v1.

Phase_0
Phase_90
XOR out
%2 out

High resolution pictures; zoom in.
Adjust the current of the differential pair to keep the CM of the D2S converter output at the switching point of an inverter.
Duty cycle was better but rise time was not.

Both the approaches could exist in one implementation(?)

I stayed with v1, anyway.
AC Gain of the whole D2S as a function of Input Common Mode and Vctrl. (Note that differential circuits have a dependency on the input common mode, i.e. vertical change in color)

Proper biasing region
Vctrl = 0.4 V – 1.2 V
VCO+D2S BEHAVIOUR

Putting things all together

Ö.Ç.
We will use this circuit to extract behavioral parameters to be imported by verilogHDL model.

The VCOs are intended to be the “plant” of a control loop (e.g. Charge Pump Phase Locked Loop CP-PLL).
Wave forms at 4.32GHz for the two “extremes”

See the next page for all the other waveforms @ 4.32GHz (marked with gray circles)
These **wave forms** correspond to the points marked with **gray circles** in the previous page and they are in order starting from the left to the right representing “all” the corners specified as stated in the title @ 4.32GHz.

Ö.Ç.

*High resolution pictures; zoom in.*
Transient simulations have been performed for **2ns** & with steps of \( V_{ctrl} = \pm 50\text{mV} \)

- Frequencies lower than \( 1/(2\text{ns}) \) can not be seen; this is the reason for the two un-expected points (labeled as outScale on the legend).
- See the wave form for a transient simulation of **20ns**

- **475mV < CM < 700mV** within the tuning range
VCO+D2S “ALL” THE CORNERS  As specified at the GBT meeting on 14-03-2007 @ CERN
Intentionally left blank.

Ö.Ç.
Intentionally left blank.

Ö.Ç.
Simulated results are approximated by 9th order fit curves and mathematical phrases were obtained to be used within the VerilogHDL model.
All the corners, random jitter and duty cycle errors for all the corners @ 4.32GHz are included in the model.
The output

Range checks @ every new Corner and/or Vctrl

Behavior parameters for different corners @ every change in Corner and/or Vctrl

```verilog
counter = counter + 1;
if (counter == 8) begin
    // Behavior parameters for a specific operation
    // Example: Duty cycle error calculation
    dutyCycleError = dutyCycleError0;
end
```
Frequency calculation
that is, fit polynomial
A probe for each
module to collect
statistics

```verilog
a9 = 1.0;
dutyCycleError = dutyCycleError13;

if (Corner == 14) begin
  if (vctrl < range4corner14_first || vctrl > range4corner14_last) begin
    // Parameter extraction has been performed for a range but not between 0 to Vdd
    // This is the proper operation range of the VCO
    $display("\nERROR : Parameter covers the following range : %0f mV < Vctrl < %0f mV",
      range4corner14_first, range4corner14_last);
    $display(" : for Corner=%0d, but it is %0d\n", Corner, Vctrl);
  
  a0 = -3.39792e+12;
a1 = 3.18654e+13;
a2 = -1.1307e+14;
a3 = 1.65853e+14;
a4 = 1.0;
a5 = -2.99961e+14;
a6 = 3.42665e+14;
a7 = -1.24852e+14;
a8 = 1.0;
a9 = 1.0;
dutyCycleError = dutyCycleError14;
  end

  // Fit polynomial for the control curve that produces the frequency
  // at which the VCO oscillates; frequency error is almost zero.

  freq = (a0 +
    a1 * vctrl +
a2 * vctrl*vctrl +
a3 * vctrl*vctrl*vctrl +
a4 * vctrl*vctrl*vctrl*vctrl +
a5 * vctrl*vctrl*vctrl*vctrl*vctrl +
a6 * vctrl*vctrl*vctrl*vctrl*vctrl*vctrl +
a7 * vctrl*vctrl*vctrl*vctrl*vctrl*vctrl*vctrl +
a8 * vctrl*vctrl*vctrl*vctrl*vctrl*vctrl*vctrl +
a9 * vctrl*vctrl*vctrl*vctrl*vctrl*vctrl*vctrl*vctrl) / 1.0e9;
end

// This is the probe module collecting statistics related to VCO functioning,
// The output is always produced and visualized by a ROOT script having a similar
// name with the data file in the same directory

probe_vco probe(.fileHandle(data_probe_vco),
  .reference(ClkRef),
  .signal(VCOout),
  .phaseErrorLimit(phaseErrorLimit));
endmodule
```
Model is self-consistent as expected: it is exactly the same as what we set, so the module correctly represents the transistor level implementation of the VCO.
Duty cycle error is included in VerilogHDL model compiled by two different compilers: **VerilogXL** and **iVerilog**.
• Each module has a probe which creates data files with interesting statistics
• Each probe data file is read by a ROOT script
• Because examining waveforms to debug a model is not a secure practice

Figure shows an almost full scan for VCO: all corners and almost all possible control voltages. Values correctly correspond to the ones set within the model.
• Figure shows an almost full scan for VCO: same as the previous page but a *jitter of (+) or (-) 5ps* is introduced.
OPEN TOOLS – Mixed Simulator QUCS, iVerilogHDL, FreeVHDL

- All-in-one tool
- Ability to synthesize HDL code
- Both VHDL and VerilogHDL support for digital simulations
- Very fast, simple models, multiple languages etc.

Verilog code generator C++ code

Verilog code generated

High resolution pictures; zoom in.
PROBLEM – Vctrl Spikes

- So far, the VCO model was similar to a mathematical function like $F(V_{ctrl})$, that is, the output frequency is a function of input control voltage, that is, for every value change of Vctrl we would have a new value for the output frequency.

- In real life, this definition is not enough to define a VCO practically. One must define “how” the input control voltage will be translated to output wave, that is, not the output frequency but the output phase.

- To arrive at the problem statement, consider the following VCO output:

- A very narrow (like 2fs, which is almost impossible in real life) spike of half a volt is applied to the Vctrl, thus the VCO “updates” the period variable within the module two times which are extremely close (2fs) to each other. Period becomes very different from the initial value for a very small amount of time then it returns back to its initial value. So the average control voltage is almost un-affected. Even though, this looks proper, it is not; because the next transition should “wait” this period long, thus an effective phase jump occurs. In real life however a 2fs spike can not change the output phase of a VCO this much.

- If the VCO is the “plant” of a control system like a PLL, then the verilog implementation given so far can not be used in case of a high feedback divide ratio. So an updated version that can handle this problem must be developed.

- One can add too many delay cells sequentially and apply the control voltage to each of the delay cells and divide the output frequency by the number of delay cells used. This technique can not really “solve” the problem but it significantly decreases the size of “phase jump”. Another technique could be calculating the output phase instead of output frequency and/or using “analog” clock signals instead of squares.