Read Me Document :

Current Starved Voltage Controlled Ring Oscillator:

Requirements :

1. User is required to download and install LTspice XVII. Download from https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html

2. User is required to download and install GNU Octave. Download here https://www.gnu.org/software/octave/download.html. Once Octave has been installed, the following commands must be run only once in order to download and install the spreadsheet I/O and parallelization functionality:

   ```
   pkg install -forge io; % Excel spreadsheet I/O functions
   pkg install -forge parallel; %parallelization functionality
   ```

   It is important to note that the parallel package cannot be installed on Windows OS. The optimization code will still be able to run on the Windows version of Octave, but it will not be able to take advantage of the parallelization of LTspice in the **metric_extract.m** subroutine.

3. User needs PTM_65nm technology file. This is provided in the package but user can download from : http://ptm.asu.edu/latest.html.
   It is already provided in the Circuits folder named: 65nmptmcmos_models.txt
   Caution: The optimization is not designed to work with process files different from PTM65nm at this time.

4. The IP folder "VCO9" containing the LTspice files must be placed in the directory "Nesterov_v2/Circuits/" before running the main optimization script "LITE_MAIN.m".

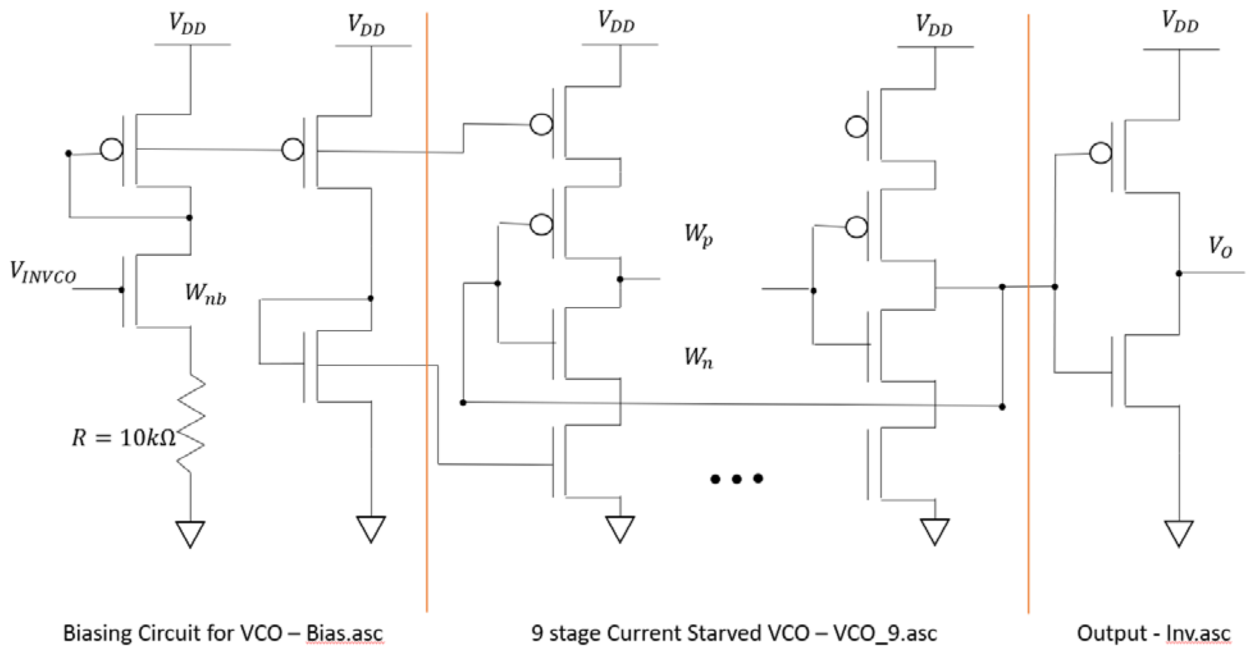
The Current Starved Voltage Controlled Ring Oscillator (CSVCRO) is a single ended CMOS VCO designed using PTM65nm process file on LTspice.

The CSVCRO under design has 9 Current Starved Invertor stages, and a Bias circuit. It also has a CMOS invertor at the output stage to drive the subsequent stage. A list of the associated files are

Circuit	**File**	**Description**
Invertor	Inv.asc	CMOS Invertor circuit file.
	Inv.asy	Associated symbol for Inv.asc
Current Starved Invertor	CS_Inv.asc	Current Starved Invertor circuit file for the operation of VCO.
	CS_Inv.asy	Associated symbol for CS_Inv.asc
Biasing	Bias.asc	Biasing circuit for linearizing VCO gain. Has a wide transistor M3 which controls the current mirrored in the current starved invertor.
	Bias.asy	Associated symbol for Bias.asc

VCO	VCO_9.asc	The circuit file containing the topology of the 9 stage CSVCRO and the output invertor and biasing circuit. This is the top level design of the CSVCRO.
	VCO_9.asy	Associated symbol to VCO_9.asc
Main	VCO9.asc	Top level circuit file with specifications of transistor sizes, voltages and simulation commands. MATLAB/Octave uses this file as the handle for optimization.

This circuit has 42 transistors and we optimize three transistors namely $\{W_n, W_p, W_{nb}\}$. The circuit and the variables are described in the figure below



V_{DD}
V_{INVCO}
W_{nb}
$R = 10k\Omega$
W_p
W_n
V_O
Biasing Circuit for VCO – Bias.asc
9 stage Current Starved VCO – VCO_9.asc
Output - Inv.asc

Figure 1: Topology of CSVCRO

The fixed parameters in this exercise :

1. Length of transistors : $L = 65nm$
2. Supply Voltage: $V_{DD} = 1.1V$
3. Minimum control voltage: $V_{cont} = 0.3V$ which determines the nominal frequency f.
   This is done because the NMOS turns on around $V_{on} \geq V_{Tn} \approx 0.28V$.
4. Simulation time for transient simulation : $T_{max} = 1.5\mu s$.
   The FFT evaluator in the optimization algorithm computes the FFT for the last 60% of simulation time to allow for VCO to stabilize.

MATLAB/Octave programs:

LITE_MAIN : This is the main user interface script. It reads in the spreadsheet "LITE_header.xlsx", which contains user-specified metric targets, as well as other circuit parameters and tolerances. The spreadsheet file must be edited beforehand by the user. This top-level script calls the function pipeline() which follows through the design flow.

pipeline(): This is the main function which carries out the optimization flow of the VCO. It progresses in 3 stages indicated by 'pipe_switch'.

When **pipe_switch=1**, the algorithm searches the database of designs for a solution within the specified tolerance by the user using **database_search()**. If a solution Sol is found from within the database, the associated circuit is simulated and presented to the user.

If no solution is found, then we set **pipe_switch=2**. This initiates the local convex optimization algorithm, where a list S is first populated with 10 closest solutions, namely $S = \{S(1), \ldots, S(10)\}$. Then we begin at $S(1)$ and perform local multi-objective Optimization using the function **nesterov_gradient_descent()**. If the function gets stuck in a local minimum outside our acceptable tolerance, we then proceed to $S(2)$ and so on. When a solution is found post simulation we turn **pipe_switch=3**, and the associated circuit is simulated and presented to the user and the solution is saved to the database **database_VCO9.mat** of simulation based KGDs.

If after the whole list S, if the optimizer cannot find a solution satisfying user requirements, we turn **pipe_switch=4** and inform the user that no solution can be formed by the optimizer for the given inputs. Then control is returned to **LITE_MAIN** when the program terminates. The optimization is carried out based on simulation using LTspice, and we use a forward difference method to estimate the partial derivatives.

distance(): This function computes the cost, or distance, between the set of current metric values from an edited and evaluated schematic (in this case, the nominal frequency f_0 and tuning range Δf) and the set of desired, or target, values.

nesterov_gradient_descent(): This function takes the local solution $S(i) \in S$, where $S(i)$ has the transistor widths $\{W_n, W_p, W_{nb}\}$ and the associated output metrics namely $\{f, \Delta f\}$. Using our distance metric, we first determine the local Lipschitz constant around the solution $S(i)$. Then using the **grad()** function we estimate the gradient of multi-objective function, using which we perform a multi objective Nesterov-based gradient descent to determine a solution Sol. If we end up in a local minimum then we proceed to $S(i+1)$. Nesterov's gradient descent takes a *gamble* → *correction* approach, i.e. Nesterov's method (1) first makes a big jump in the direction of the previous accumulated gradient, then (2) measures the gradient at the new location and corrects accordingly.

grad(): This computes the Jacobian of the multi-objective function given a set of coordinates $\{W_n, W_p, W_{nb}\}$. It uses an incremental change preset in computing the forward finite difference of the partial derivatives.

edit_extract_index(): It is a function which edits the top level netlist VCO9.asc, and then calls LTspice to execute the netlist. Once LTspice has executed the netlist, it collects data from LTspice and converts it to a form workable within MATLAB/Octave.

metric_extract(): This is a function which is used to read the output data from LTspice execution and present it in a workable format within MATLAB/Octave. The two metrics it produces are $f \ and \ \Delta f$.

freq_extract(): This is a function that finds the output frequency at a given control voltage. It converts the VCO9.raw file of the transient simulation executed by LTspice and then uses the last 60% of the data to find the FFT, determining the output oscillating frequency. It returns the fundamental harmonic of the data. The goal of using the last 60% of data is to allow for the VCO to settle into oscillations.

database_search(): This function is used to find solutions in the database within user specifications and when such a solution does not exist it generates the list S.

final_schem_gen(): This function is essentially identical to the "edit" portion of **edit_extract_index()**; it is used at the very end of the main **pipeline()** function to perform the final edits to either the VCO or PLL schematics (and therefore parameter values in the corresponding netlists). It is called just before the final design is outputted by MATLAB at the end of **pipeline()**.

database_VCO9.mat: This is a database of simulated KGDs structured as

Nominal frequency f	Tuning Range Δf	W_n	W_p	W_{nb}