**Description**

MCML gate optimization process requires simultaneous alteration in design variables represented in Table I of [1] to satisfy the performance and power objectives formalized in design parameters of previous section. HSPICE environment allows an easy way of MCML circuit netlist representation; the internal topology is built upon the realistic n/p-type MOSFET transistor definitions (can be found with a standard CMOS cell library models) interconnected with the other active/passive HSPICE components represented as the circuit abstraction models, also called SUBCIRCUITS. The optimization process starts with the definition of variable parameters and their limits, the optimization specifications (with the .PARAM parameter=OPTx(init, min, max) format) and the optimization goals (provided as .MEASURE statements with the keyword GOAL stating limits) to be achieved. We performed the transient type of analysis during the optimization process (.TRAN type analysis) and stored the output values for the future best-case input/output parameter selection.

The automated optimization process consists of two global steps: 1) MCML cell library design and optimization and 2) the topological circuit design, which considers interconnection of basic cells into complex circuits, which meet the timing objectives.

For a complete description of MCML cell library design and optimization please refer to [1].

References:

[1] Y. Audzevich, P.W. Watts, A. West, et.al. “Power Optimized Transceivers for Future Switched Networks”, accepted to IEEE TVLSI, 2013.

**Single-level MCML logic optimization**

**/PMA/mcml\_optimization/MCML\_LEV1/** folder contains examples, which can be useful for single-level MCML cell library design and optimization. An example of an MCML inverter cell is presented in Figure 1.

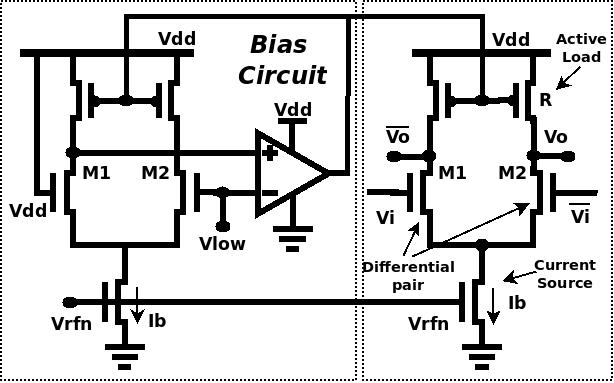


Figure 1. MCML inverter cell

MCML single-level optimization design has the following structure:

1. **cells.spi** – contains topological representation of single-level MCML cells under analysis. Please use HSPICE compliant subcircuit (.subckt -> .ends) statements to create cell abstractions. N-type and P-type MOSFET transistor model locations should be defined inside this file (or standard CMOS cell library location, which contain Spice transistor model descriptions).
2. **params.spi** – keeps the combination of all parameters used during the optimization process, including fixed parameters and parameters that need optimization. The optimization parameters should be defined using **optw()** function and should be commented when a fixed parameter is used instead. A transient type of analysis is used during the circuit optimization.
3. **av\_test.spi** fileallows a differential mid-swing voltage gain analysis via running DC type of HSPICE analysis. The file contains simulation options, analysed MCML circuit topology, voltage source definitions, measurement statements and design parameters, and, can be executed in standalone manner.
4. **nm\_test.spi** file examines the Noise Margin levels in single-level MCML cells. DC type of analysis is performed as in previous case. Can be run is standalone manner.
5. **test.spi** – assembles all dependencies needed for optimization process; defines the model of built-in optimizer; performs instantaneous signal measurements and optimization goals detection.

To initiate MCML gate parameters optimization, please make the following steps:

1. create an abstraction for every MCML cell under analysis. Substitute all fixed variables in every cell with corresponding optimization variables. Please make sure the identical variable names are used in params.spi file.
2. create an arbitrary input stimuli file (test.dsi) using DSI tool. Attach all input signals to the MCML gates under analysis. Add some loads and vary fanout value for MCML cells to make simulation more realistic (update **test.spi**).
3. Update the optimization parameter list and sweep limits (**params.spi**). Larger optimization sweeps will result in slowing down the optimization process.
4. Update the measurement statements and optimization goals inside the main file (**test.spi**). Specify optimization variables to HSPICE optimizer.
5. Run **run\_hspice.sh** script file to initiate optimization process. The process will stop when either optimization goals have been reached or optimization failed. In the latter case, please reduce the sweep value and the total number of parameters analysed simultaneously.
6. If optimization successful, verify results with optimization parameter disabled. Record results for the future use.
7. Run voltage gain and noise margin simulations separately based on the obtained optimization results.

**Multiple-level MCML logic optimization**

**/PMA/mcml\_optimization/MCML\_LEV2/** folder contains examples, which can be useful for multiple-level MCML cell library design and optimization. An example of an MCML Master-Slave D Flip-Flop cell is presented in Figure 2.

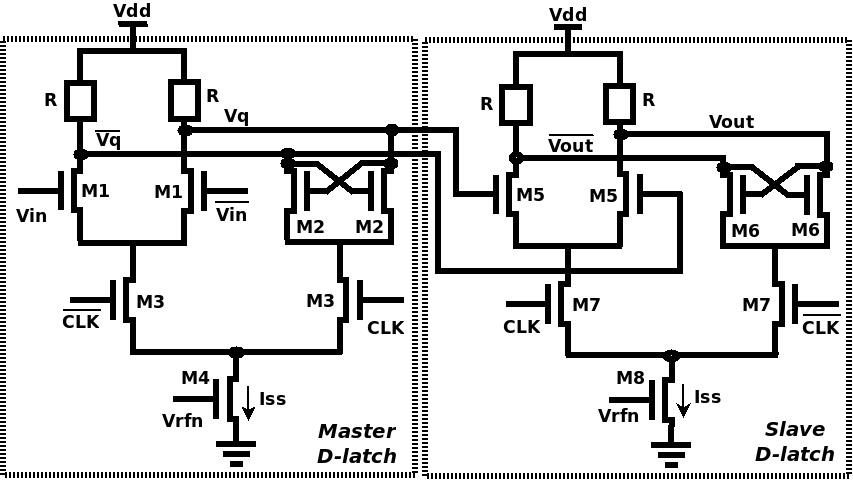


Figure 2. MCML Master-Slave D-FF cell

The optimization of multiple-level MCML cells is based on the optimization results obtained for the single-level logic. Some parameters, like supply voltage levels, voltage swing values, and others, are decisive for a stable interoperation of MCML cells with different logic levels and should be identical. The optimization procedure for all the other parameters is similar to the one described before for single-level logic.

To initiate MCML gate parameters optimization, please make the following steps:

1. create an abstraction for every MCML cell under analysis. Substitute all fixed variables in every cell with corresponding optimization variables. Please make sure the identical variable names are used in params.spi file.
2. create an arbitrary input stimuli file (**test.dsi**) using DSI tool. Attach all input signals to the MCML gates under analysis. Add some loads and vary fanout value for MCML cells to make simulation more realistic (update **test.spi**).
3. Update the optimization parameter list and sweep limits (**params.spi**). Larger optimization sweeps will result in slowing down the optimization process.
4. Choose the type of MCML cell for analysis. This feature is available for multiple-level MCML optimization process only and can simplify optimization process via re-use of already defined templates.
5. Update the measurement statements and optimization goals inside the main file (**test.spi**). Specify optimization variables to HSPICE optimizer (**RESULTS=** statement).
6. Run **run\_hspice.sh** script file to initiate optimization process. The process will stop when either optimization goals have been reached or optimization failed. In the latter case, please reduce the sweep value and the total number of parameters analysed simultaneously. Re-run the procedure.
7. If optimization successful, verify results with optimization parameter disabled. Record results for the future use.

Once the correct variable set is obtained, please verify cell performance with different loads and setups. In majority of cases a similar set of optimization parameters will be suitable for all the other cells with the same logic depth.

Although the maximum depth of cells used in **MCML\_LEV2/** examples is equal to two, the optimization process can be extended to cells with any number of logic-level depths.

**CMOS to MCML conversion circuits and optimization**

The optimization folder **/PMA/mcml\_optimization/MCML\_CONV/** contains a similar set of files and structures as the other MCML optimization procedures described earlier. Please use these as a reference.

**MCML circuit timing closure**

Once the MCML cell library is optimized and tested, the topological MCML circuit design is performed. The novelty of automated MCML cell library design procedure does not allow performing timing analysis in toolkit-assisted way as it is usually done in CMOS case. An example of fully-custom critical timing analysis for a 2:1 MCML-based multiplexer is shown in Figure 3.

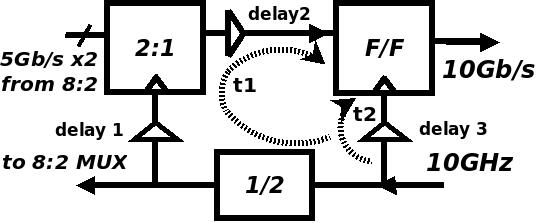


Figure 3. Critical timing analysis performed in 2:1 MCML multiplexer

A critical path *t*1 is formed through the frequency divider and selector circuits, with the additional impact introduced by a delay line marked as 1. Since the input signal sampling points for the flip-flop and selector circuits are not identical (related to the direction of clocking signal propagation), the setup time of the D-type flip-flop input is severely reduced. To compensate this instability, clocking signal propagation times are adjusted by variation in the corresponding delay line size or characteristics (marked as 1,2 and 3). In particular, the delay value of path *t*2 is enlarged to compensate the extra shift introduced by path *t*1. A similar procedure for timing closure is used at other stages of the binary-tree-like structure of MCML-based multiplexer as well as demultiplexer designs, and, in the latter case, the direction of clock and data signal propagation is identical.

The custom MCML timing closure analysis was performed by means of Synopsys CosmosScope tool after the MCML cell library had been optimized.