# xcell: a cell library characterizer for combinational and state-holding gates

Rajit Manohar
*Computer Systems Lab*
*Yale University*
New Haven, CT, USA
rajit.manohar@yale.edu

***Abstract*—We present an open-source cell library characterizer suitable for characterizing combinational logic as well as general state-holding gates that are used in asynchronous logic as well as clocked circuit families like pre-charged logic. The output of the characterizer uses the Synopsys `.lib` format, and has been used as the input to an existing asynchronous static timing and power analysis tool for multiple technologies and multiple cell libraries.**

***Index Terms*—cell characterization, timing estimation, power estimation.**

## I. Introduction

Static timing analysis and power analysis are an integral part of any electronic design automation (EDA) flow. The two key inputs to analysis tools are the user design specified as a collection of components (standard cells and macros) and their interconnections, and a timing/power library that is typically provided by the standard cell library vendor in Synopsys' open-source Liberty (`.lib`) file format [1].

We have implemented an open-source tool called `xcell`[1], designed to automate the generation of Synopsys `.lib` files from a gate-level cell library. `xcell` is designed for the requirements of asynchronous logic, and can also be used to characterize combinational logic for synchronous logic. `xcell` summarizes delay and power information using "non-linear delay model" (NLDM) tables.

For synchronous logic, a Liberty file uses different syntax when specifying timing and power information for combinational logic gates as compared to the syntax used for different categories of state-holding gates (e.g. flip-flops, latches, and tri-state drivers). Since asynchronous logic uses a much broader class of state-holding gates, `xcell` treats all state-holding gates in a uniform fashion.

In order to construct the `.lib` file, `xcell` runs a large number of SPICE simulations to compute the different values needed to construct the `.lib` file: leakage power, input capacitance, delay values and transition time for timing arcs, and internal power tables for power estimation. Two external SPICE simulators are currently supported: `Xyce`, the open-source simulator from Sandia National Labs [2], and `hspice`, a proprietary simulator from Synopsys. (We support both because open-source simulators do not always support the process design kits in advanced technology nodes.) The operation of the characterizer is controlled by a configuration file that specifies a number of parameters used to characterize the cell library, such as operating voltage, temperature, the path containing the SPICE models for the technology being used, as well as the input transition time and output load capacitance values to be used to characterize the delay and power for each cell.

The characterizer is designed to operate with the open-source ACT toolkit for asynchronous logic[2], and leverages several utility functions provided by the ACT infrastructure including: (i) the ability to represent a cell library in the ACT syntax; (ii) automated generation of a SPICE netlist from the ACT definition of a cell; and (iii) the ability to read both raw SPICE trace file formats, as well as an older `hspice` trace file format.

## II. Specifying the cell library

The following is an example of a buffer cell specified in the ACT syntax.

```
defcell BUFX2 (bool? A; bool! Y)
{
  bool _Y;
  prs {
    A => _Y-
   _Y => Y-
  }
  sizing { _Y{-1}; Y{-2} }
}
```

The name of the cell is `BUFX2`, with an input signal `A` and an output signal `Y`. Internally, the cell also has signal `_Y` but this is not exposed via its port list. The circuit is specified using the *production rule* syntax. Production rules can specify arbitrary pull-up and pull-down networks via the Boolean logic expression corresponding to the pull-up/pull-down switching network. The rule `A => _Y-` is an inverter; when `A` is true, `_Y` is set low. Similarly, `_Y => Y-` is another inverter, and so the circuit corresponds to two inverters in series. Finally, the sizing specification says that `_Y` should have unit drive strength, and `Y` should have twice the unit drive strength [3]. Values for width/length of transistors, the definition of what a

This work was supported by DARPA IDEA grant FA8650-18-2-7850.

[1]Available at https://github.com/asyncvlsi/xcell

[2]Available at https://github.com/asyncvlsi/act

unit drive strength is in terms of transistor sizing, the transistor model names, etc. are all contained in technology-dependent ACT configuration files. `xcell` leverages this existing infrastructure for specifying and generating circuits.

An example of a state-holding gate used in asynchronous logic, an inverting C-element, is shown below:

```
export
defcell gcelem2x0 (bool? in[2]; bool! out)
{
   prs {
     in[0]<10> & in[1] -> out-
     ~in[0]<12> & ~in[1] -> out+
   }
}
```

Here, the pull-down network for output signal `out` is given by the condition `in[0] & in[1]`. In addition, the width of the transistors used to implement the pull-down network is specified as $10\lambda$ units, where $\lambda$ is half the feature size. The pull-up network is the conjunction of the negation of `in[0]` and the negation of `in[1]`. Hence, when `in[0]` and `in[1]` disagree, the gate is state-holding. The core ACT library detects that the production rules correspond to a state-holding gate, and auto-generates a keeper for this cell to ensure that the output is never floating. (This behavior can be disabled, if necessary.)

A complete combinational logic cell library based on the OSU 180nm library for combinational logic [4] but re-written in a technology-independent format is available as part of the ACT standard library[3]. ACT configuration files that provide an example of how technology-specific parameters are specified for the open-source Skywater 130nm process are available as well.[4]

### A. Generating SPICE for a cell

ACT has a number of default rules to translate a cell definition into a transistor-level implementation, and there is a well-defined mapping from the circuit description and the transistor-level implementation that is inferred from the description.[5] ACT also permits a user to change transistor sizing specifiers, specify non-standard keepers, as well as individual transistors that may be needed to provide an accurate representation of a particular cell in the cell library.

Given a cell, ACT already includes methods to generate a SPICE netlist by combining the technology-specific information in ACT configuration files along with the directives specified by the user in the ACT cell description. The ACT library can also be used to determine the input and output ports for each cell. We leverage this existing infrastructure to generate the SPICE netlist for an individual cell during the characterization process.

In addition to using the core ACT SPICE generation support, `xcell` allows user-specified SPICE netlists that include additional extracted parasitics beyond the pure transistor-level netlist that is auto-generated by ACT.

[3]https://github.com/asyncvlsi/stdlib/blob/main/std/cells.act

[4]https://github.com/asyncvlsi/sky130l

[5]https://avlsi.csl.yale.edu/act/doku.php?id=language:langs:prs

## III. Characterization method

Given a cell described in the ACT syntax, we examine the logical definition of the cell in terms of the set of pull-up and pull-down networks within it. A cell is labeled as state-holding if any gate in the cell is state-holding; otherwise, it is a combinational cell. For state-holding cells, we identify every state-holding node within the cell, and identify all variables that are inputs to each state-holding gate. Finally, we construct the truth table for the pull-up and pull-down networks for each state-holding gate in the cell.

Cell characterization proceeds in four phases: (i) leakage power calculation; (ii) input capacitance calculation; (iii) switching scenario computation; and (iv) switching delay and internal power calculation.

### A. Leakage power

For leakage power, we create a SPICE file that applies all possible input combinations to a cell. To reduce the characterization time, we do this in a single SPICE run, thereby reducing the number of times we have to launch the SPICE simulator. SPICE `.measure` directives are used to compute the average leakage current over a window of time, and this is used to estimate the leakage power for each input scenario. Note that `hspice` and `Xyce` use slightly different syntax and output formats for their `.measure` directives, and `xcell` generates the appropriate syntax based on the selected simulator.

The ACT core library also has the capability to read certain SPICE simulation traces. We use this capability to determine the value of each output signal for the cell from the result of SPICE simulations. Recall that we previously recorded all input signal names for every state-holding gate in a cell; we also read the value of these input signals from the SPICE simulation trace. Using this information, we can ensure that we only record leakage information in legal scenarios for the state-holding gate (i.e. one where the circuit does not have a stable power/ground short).

### B. Input capacitance

The effective input capacitance is computed by using an RC estimate. A resistor is attached to the input port of interest, the input is changed from zero to one and one to zero, and the effective RC time constant is computed for both transitions. From this calculation and given that the external resistance is known, the average effective capacitance is computed and used as an estimate for the input rise and fall capacitance. This procedure is repeated for each input. All these scenarios are combined in a single SPICE simulation to reduce runtime.

### C. Switching scenarios

`xcell` must compute all the timing arcs for a cell. Since we have an ACT description of the cell, we analyze the circuit for the cell to classify it into one of the three different cases described below. Note that cells can have multiple outputs,

and for a cell to be in one of the categories described below it means that every output satisfies the constraints for the category.

A switching scenario is a sequence of assignments to the primary inputs of a cell that result in the output changing. For example, a switching scenario for the buffer example earlier would be the following: (a) Step 1: set the input `A` to zero, which results in the output being zero; (b) Step 2: set the input `A` to one, which causes the output to have a zero to one transition. This is a *two-step* scenario. The buffer also has a second (symmetric) switching scenario that causes the output to have a one to zero transition. To characterize a cell, we must determine all possible scenarios that cause the output to change.

*a) Cells with only combinational gates:* For combinational gates, we have already computed the truth-table information for the cell output from our analysis of the SPICE simulation trace file during leakage estimation. Hence, we can compute every possible switching scenario, and construct input scenarios that bring the output of the cell into a known state (step 1), and then switch an input signal to cause the output to change state while holding the other inputs fixed (step 2). We construct all possible "two-step" scenarios for the combinational cell being characterized, and record this in a "dynamic scenario" table for the cell.

*b) Cells with combinational or simple state-holding gates:* Computing switching scenarios for state-holding gates is more complex. A *simple* state-holding gate is one where all inputs to the state-holding gate are combinationally determined from the primary inputs of a cell, and the output of the state-holding gate and/or its inverted version is directly available as a primary output of the cell. We view these as *simple* because both the pull-up and pull-down network for any internal state-holding gate can be directly controlled by setting the primary inputs to the cell.

For simple state-holding gates, `xcell` can automatically compute all the switching scenarios for the gate. Switching scenarios for such gates consist of two, three, or four steps. In a two-step scenario, the inputs to the cell are set causing the output of the cell to be set to a fixed value (step 1). Followed by this, a particular input is switched, causing the output to also switch (step 2). This is similar to combinational cell characterization. However, this by itself may not capture all possible timing arcs for a simple state-holding gate.

A three-step scenario corresponds to the next level of complexity of identifying a switching scenario. It takes the following form:

- the inputs are set to a value that sets the output to a known value (step 1);
- an input is changed to bring the gate into a state-holding gate without any glitches on the output (step 2);
- finally, another input is changed to cause the output to switch (step 3).

An example of this is the inverting C-element: to characterize a zero to one transition on the output, we first set both inputs high (step 1); then, one of the inputs is set to low (step 2); finally, the second input is set to low that causes the output to make a zero to one transition (step 3).

In certain cases, it is possible that bringing the gate into a state-holding state (step 2) does not lead to a state where a single input change causes the output to switch (step 3). For such cases, we compute a four-step scenario: the first two steps are the same as the three-step scenario, but in step 3 we switch to a second state-holding state that preserves the output value. Finally, we change the input of interest that causes the output to change in step 4.

To summarize, for state-holding gates, we automatically compute trajectories through the state-space to identify all possible timing arcs. A trajectory may require two, three, or four steps for simple state-holding gates. These trajectories are recorded in a "dynamic scenario" table for the state-holding cell.

*c) Cells with complex state-holding cells:* For cells that correspond to circuits that are more complex than simple state-holding gates, we provide support for the user to specify the state-space trajectories to be used for the switching scenarios. This is done via the `xcell` configuration file. In our experience in designing asynchronous circuits and characterizing over a hundred different cell types (in terms of logical functionality—i.e. ignoring drive strength, gate sizing, transistor threshold voltage, etc. which would vastly increase the cell count), we have had to use the user-specified state-space trajectory feature for two cells.

### *D. Timing and power information*

The dynamic scenario table for a cell is used to create a SPICE scenario for timing and power characterization. The `xcell` configuration file specifies the set of input transition times and output load capacitance values to be used for the NLDM tables for timing and power. We construct a SPICE run consisting of all possible switching scenarios with all possible input transition times. We put a fixed load capacitance on the output of the cell, and then use the parameter sweep feature supported by both `hspice` and `Xyce` (albeit with different syntax) to sweep the output load capacitance. We measure the internal power and delay using `.measure` statements, and use this to populate the delay and power tables for the cell.

### *E. Final output*

`xcell` produces an output in the standard Synopsys Liberty file format. We have verified that the output generated by `xcell` can be read by both the Cyclone timing and power analysis engine for asynchronous logic [5], as well as a few commercial tools.

As should be clear from the previous description, we characterize state-holding gates in a manner similar to combinational logic gates. Hence, we report timing and power information using the same syntax as standard combinational logic. This is a departure from the standard way that `.lib` files are used, and hence our tool is suitable for synchronous library characterization only for combinational gates.

For combinational gates, the Synopsys Liberty file includes a "`function`" field that specifies the Boolean logic function corresponding to the output. We emit this field when generating the `.lib` file using the truth table information that was computed as part of the characterization process.

For state-holding gates, the output cannot be described as a simple function of the input. Using the inverting C-element example, the output `out` would be one if `in[0]` and `in[1]` are zero, zero if `in[0]` and `in[1]` are one, and state-holding otherwise. To capture this in the `function` field of the `.lib` file, we *include* the output pin as part of the Boolean expression in the `function` field. If the output pin is named `out`, the Boolean expression for the pull-up network is `U`, and the Boolean expression for the pull-down network is `D`, then the output function is reported as `(U) + out*!(D)`. The presence of the output pin name in the expression for the function can be used as a test for a state-holding gate.

### F. Runtime

We ran `xcell` using `Xyce` as the underlying circuit simulator on a 2020 Apple Macbook Air (M1 processor) with 8GB of memory. `xcell` generated a 1.01MB `.lib` file for a library with 49 cells under three minutes, where we used NLDM tables with 70 points per table for each delay and power table.

## IV. Related Work and Summary

We presented the implementation of `xcell`, a cell library characterizer suitable for generating Liberty files for combinational logic gates as well as state-holding gates for asynchronous logic. The `.lib` files generated by `xcell` have been as the input to the Cyclone static timing and power analysis engine for asynchronous logic [5]. We have used `xcell` to characterize cells by using `Xyce` in Skywater 130nm and TSMC 65nm, and by using `hspice` in ST Micro's 28nm FDSOI process. The `.lib` files generated for combinational logic have also been used as the input to the technology-mapper in `abc`[6], an open-source logic optimization engine.

The closest related work is the cell characterizer LiChEn [6], which can be used to automate the characterization of cell libraries including state-holding gates used for asynchronous logic. LiChEn is limited to cells that have a single output, and uses Cadence `spectre` as its SPICE simulator. ACT integration simplifies the user interface for `xcell`. For example, `xcell` can auto-generate SPICE netlists, determine I/O signal names, and automatically determine the logic function for each cell output; in LiChEn, all of these must be explicitly specified through a scripting interface.

There are a number of future extensions that would increase the utility of `xcell`: (i) support for flip-flop, latch, and tri-state-specific Liberty output formats could be added while reusing most of the code base for generating SPICE scenarios and collecting measurement results; (ii) additional characterization for on-chip variations could be incorporated; and (iii) better delay models beyond NLDM could be incorporated to improve accuracy.

[6]https://github.com/berkeley-abc/abc

## References


[1] Synopsys, "Liberty user guides and reference manual suite," http://www.opensourceliberty.org, 2017.

[2] E. R. Keiter, H. K. Thornquist, R. J. Hoekstra, T. V. Russo, R. L. Schiek, and E. L. Rankin, "Parallel transistor-level circuit simulation," in *Simulation and Verification of Electronic and Biological Systems*. Springer, 2011, pp. 1–21.

[3] R. F. Sproull and I. E. Sutherland, "Logical effort: Designing for speed on the back of an envelope," in *Conference on Advanced Research in VLSI (ARVLSI)*. MIT Press, 1991, pp. 1–16.

[4] J. E. Stine, "System-on-chip designs for SCMOS MOSIS AMI 0.6um, AMI 0.35um, TSMC 0.25um, and TSMC 0.18um," https://vlsiarch.ecen.okstate.edu/flows/MOSIS_SCMOS/osu_soc_v2.7/, 2017.

[5] W. Hua, Y.-S. Lu, K. Pingali, and R. Manohar, "Cyclone: a static timing and power engine for asynchronous circuits," in *IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE, 2020, pp. 11–19.

[6] M. T. Moreira, C. H. M. Oliveira, N. L. V. Calazans, and L. C. Ost, "Lichen: Automated electrical characterization of asynchronous standard cell libraries," in *2013 Euromicro Conference on Digital System Design*. IEEE, 2013, pp. 933–940.