Power Estimation for VTR

Jeffrey Goeders

May 26, 2013

The latest version of this document can be obtained here: `http://vtr-verilog-to-routing.googlecode.com/svn/trunk/doc/power/power_manual.pdf`

Contents

1 Overview

This document describes the power estimation tool for VTR. This tool provides transistor-level dynamic and static power estimates for a given architecture and circuit.

Figure 1 illustrates how the VTR flow is modified to add power estimation. The actual power estimation is performed within the VPR executable; however, additional files must be provided. In addition to the circuit and architecture files, power estimation requires files detailing the signal activities and technology properties.

Section 2 details how to run power estimation for VTR. Section 3 provides details on the supporting tools that are used to generate the signal activities and technology properties files. Section 4 provides details about how the tool models architectures, including different options of modelling methods. Section 5 provides more advanced configuration options.

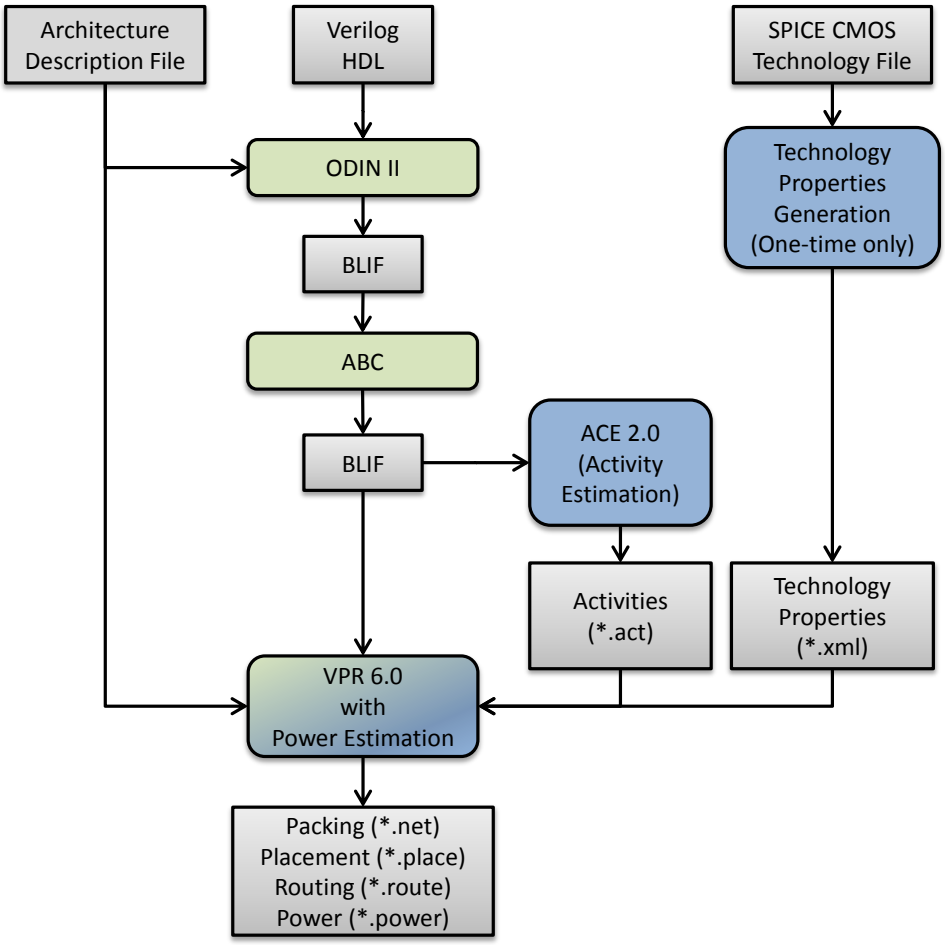

Figure 1: Power Estimation in the VTR Flow

2 Running VTR with Power Estimation

2.1 VTR Flow

The easiest way to run the VTR flow is to use the `run_vtr_flow.pl` script. See `http://code.google.com/p/vtr-verilog-to-routing/wiki/Run_VTR_Flow` for instructions on how to use the script.

In order to perform power estimation, you must add the following options:

```
-power
-cmos_tech <cmos_technology_properties_file.xml>
```

The CMOS technology properties file is an XML file that contains relevant process-dependent information needed for power estimation. XML files for 22 nm, 45 nm, and 130 nm PTM models can be found here:

```
<vtr>/vtr_flow/tech/.../*.xml
```

See Section 3.1 for information on how to generate an XML file for your own SPICE technology model.

2.2 VPR 6.0

The easiest way to perform power estimation is to use the `run_vtr_flow.pl` script. However, if you wish, you may run VPR directly. The following command-line options are added to the VPR executable to facilitate power estimation (all are required):
`--power`: Use this option to enable power estimation.
`--activity_file <activities.act>`: The activity file, produce by ACE 2.0, or another tool.
`--tech_properties <tech_properties.xml>`: The technology properties XML file.

This requires a activity file, which can be generated as described in Section 3.2.

3 Supporting Tools

3.1 Technology Properties Generation

Power estimation requires information detailing the properties of the CMOS technology. This information, which includes transistor capacitances, leakage currents, etc. is included in an `.xml` file, and provided as a parameter to VPR. This XML file is generated using a script which automatically runs HSPICE, performs multiple circuit simulations, and extract the necessary values.

Some of these technology XML files are included with the release, and are located here:

```
<vtr>/vtr_flow/tech/*
```

If the user wishes to use a different CMOS technology file, they must run the following script. HSPICE must be included in the $PATH$.

```
<vtr>/vtr_flow/scripts/generate_cmos_tech_data.pl <tech_file> <tech_size> <Vdd> <temp>
```

`<tech_file>`: A SPICE technology file, containing a `pmos` and `nmos` models.
`<tech_size>`: The technology size, in meters. For example, a 90nm technology would have the value `90e-9`.
`<Vdd>`: Supply voltage in Volts.
`<temp>`: Operating temperature, in Celcius.

3.2 ACE 2.0 Activity Estimation

Power estimation requires activity information for the entire netlist. This ativity information consists of two values:

1. The Signal Probability, P_1, is the long-term probability that a signal is logic-high. For example, a clock signal with a 50% duty cycle will have $P_1(clk) = 0.5$.

2. The Transition Density (or switching activity), A_S, is the average number of times the signal will switch during each clock cycle. For example, a clock has $A_S(clk) = 2$.

The default tool used to perform activity estimation in VTR is ACE 2.0 [1]. This tool was originally designed to work with the Berkeley SIS tool, which is now obsolte. ACE 2.0 was modifed to use ABC, and is included in the VTR package here:

```
<vtr>/ace2
```

The tool can be run using the following command-line arguments:

```
<vtr>/ace2/ace -b <abc.blif> -o <activities.act> -n <new.blif>
```

`<abc.blif>`: The input BLIF file produced by ABC.
`<activities.act>`: The activity file to be created.
`<new.blif>`: The new BLIF file. This will be identical in function to the ABC blif; however, since ABC does not maintain internal node names, a new BLIF must be produced with node names that match the activity file.

User's may with to use their own activity estimation tool. The produced activity file must contain one line for each net in the BLIF file, in the following format:

```
<net name> <signal probability> <transistion density>
```

[1] Julien Lamoureux and Steven J.E. Wilton. "Activity Estimation for Field-Programmable Gate Arrays." In: International Conference on Field Programmable Logic and Applications. Aug. 2006, pp. 1–8

4 Architecture Modelling

The following section describes the architectural assumptions made by the power model, and the related parameters in the architecture file.

4.1 Complex Blocks

The VTR architecture description language supports a hierarchichal description of blocks. In the architecture file, each block is described as a `pb_type`, which may includes one or more children of type `pb_type`, and interconnect structures to connect them. The power estimation algorithm traverses this hierarchy recursively, and performs power estimation for each `pb_type`. The power model supports multiple power estimation methods, and the user specifies the desired method in the architecture file:

```
<pb_type>
        <power method="<est-method>">
</pb_type>
```

The following is a list of valid estimation methods. Detailed descriptions of each type are provided in the following sections. The methods are listed in order from most accurate to least accurate.

1. `specify-size`: Detailed transistor level modelleling. The user supplies all buffer sizes and wire-lengths. Any not provided by the user are ignored.
2. `auto-size`: Detailed transistor level modelleling. The user can supply buffer sizes and wire-lengths; however, they will be automatically inserted when not provided.
3. `pin-toggle`: Higher-level modelling. The user specifies energy per toggle of the pins. Static power provided as an absolute.
4. `C-internal`: Higher-level modelling. The user supplies the internal capacitance of the block. Static power provided as an absolute.
5. `absolute`: Highest-level modelling. The user supplies both dynamic and static power as absolutes.

Other methods of estimation:

6. `ignore`: The power of the `pb_type` is ignored, including any children.
7. `sum-of-children`: Power of `pb_type` is solely the sum of all children `pb_types`; interconnect between the `pb_type` and its children is ignored.

If no estimation method is provided, it is inherited from the parent `pb_type`. ***If the top-level `pb_type` has no estimation method, `auto-size` is assumed.***

4.1.1 `specify-size`

This estimation method provides a detailed transistor level modelling of CLBs, and will provide the most accurate power estimations. For each `pb_type`, power estimation accounts for the following components (See Figure 2).

- Interconnect multiplexers
- Buffers and wire capacitances
- Child `pb_types`

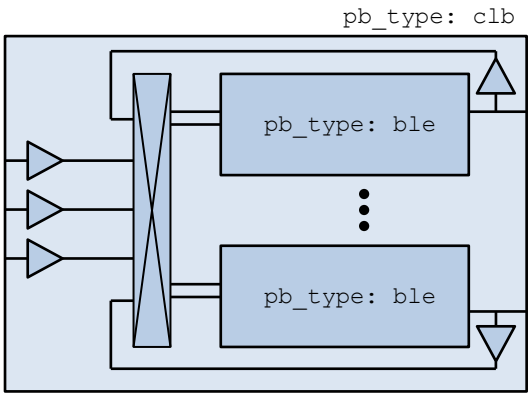

Figure 2: Sample Block

Multiplexers Interconnect multiplexers are modelled as 2-level pass-transistor multiplexers, comprised of minimum-size NMOS transistors. Their size is determined automatically from the `<interconnect/>` structures in the architecture description file.

Buffers and Wires Buffers and wire capacitances are not defined in the architecture file, and must be explicitly added by the user. They are assigned on a per port basis using the following construct:

```
<pb_type>
        <input name="my_input" num_pins="1">
                <power ...options.../>
        </input>
</pb_type>
```

The wire and buffer attributes can be set using the following options. If no options are set, it is assumed that the wire capacitance is zero, and there are no buffers present. Keep in mind that the port construct allows for multiple pins per port. These attributes will be applied to each pin in the port. If necessary, the user can seperate a port into multiple ports with different wire/buffer properties.

- `wire_capacitance="1.0e-15"`: The absolute capacitance of the wire, in Farads.
- `wire_length="1.0e-7"`: The absolute length of the wire, in meters. The local interconnect capacitance option must be specified, as described in Section 5.3.
- `wire_length="auto"`: The wirelength is automatically sized. See Section 5.1.
- `buffer_size="2.0"`: The size of the buffer at this pin. See Section 5.2 for more information.
- `buffer_size="auto"`: The size of the buffer is automatically sized, assuming it drives the above wire capacitance and a single multiplexer. See Section 5.2 for more information.

Primitives For all child `pb_types`, the algorithm performs a recursive call. Eventually `pb_types` will be reached that have no children. These are primitives, such as flip-flops, LUTs, or other hard-blocks. The power model includes functions to perform transistor-level power estimation for flip-flops and LUTs. If the user wishes to use a design with other primitive types (memories, multipliers, etc), they must provide an equivalent function. If the user makes such a function, the `power_calc_primitive` function should be modified to call it. Alternatively, these blocks can be configured to use higher-level power estimation methods.

4.1.2 auto-size

This estimation method also performs detailed transistor-level modelling. It is almost identical to the `specify-size` method described above. The only difference is that the local wire capacitance and buffers are automatically inserted for all pins, when necessary. This is equivalent to using the `specify-size` method with the `wire_length="auto"` and `buffer_size="auto"` options for every port.

This is the default power estimation method. Although not as accurate as user-provided buffer and wire sizes, it is capable of automatically capturing trends in power dissipation as architectures are modified.

4.1.3 pin-toggle

This method allows users to specify the dynamic power of a block in terms of the energy per toggle (in Joules) of each input, output or clock pin for the `pb_type`. The static power is provided as an absolute (in Watts). This is done using the following construct:

```
<pb_type>
        ...
        <power method="pin-toggle">
                <port name="A" energy_per_toggle="1.0e-12"/>
                <port name="B[3:2]" energy_per_toggle="1.0e-12"/>
                <port name="C" energy_per_toggle="1.0e-12" scaled_by_static_prob="en1"/>
                <port name="D" energy_per_toggle="1.0e-12" scaled_by_static_prob_n="en2"/>
                <static_power power_per_instance="1.0e-6"/>
        </power>
</pb_type>
```

Keep in mind that the port construct allows for multiple pins per port. Unless an subset index is provided, the energy per toggle will be applied to each pin in the port. The energy per toggle can be scaled by another signal using the `scaled_by_static_prob`. For example, you could scale the energy of a memory block by the read enable pin. If the read enable were high 80% of the time, then the energy would be scaled by the *signal_probability*, 0.8. Alternatively `scaled_by_static_prob_n` can be used for active low signals, and the energy will be scaled by $(1 - signal\_probability)$.

This method does not perform any transistor-level estimations; the entire power estimation is performed using the above values. It is assumed that the power usage specified here includes power of all child `pb_types`. No further recursive power estimation will be performed.

4.1.4 C-internal

This method allows the users to specify the dynamic power of a block in terms of the internal capacitance of the block. The activity will be averaged across all of the input pins, and will be supplied with the internal capacitance to the standard equation $P_{dyn} = 1/2\alpha CV^2$. Again, the static power is provided as an absolute (in Watts). This is done using the following construct:

```
<pb_type>
        <power method="c-internal">
                <dynamic_power C_internal="1.0e-16"/>
                <static_power power_per_instance="1.0e-6"/>
        </power>
</pb_type>
```

It is assumed that the power usage specified here includes power of all child `pb_types`. No further recursive power estimation will be performed.

4.1.5 `absolute`

This method is the most basic power estimation method, and allows users to specify both the dynamic and static power of a block as absolute values (in Watts). This is done using the following construct:

```
<pb_type>
        <power method="absolute">
                <dynamic_power power_per_instance="1.0e-6"/>
                <static_power power_per_instance="1.0e-6"/>
        </power>
</pb_type>
```

It is assumed that the power usage specified here includes power of all child `pb_types`. No further recursive power estimation will be performed.

4.2 Global Routing

Global routing consists of switch boxes and input connection boxes.

4.2.1 Switch Boxes

Switch boxes are modelled as the following components (Figure 3):

1. Multiplexer
2. Buffer
3. Wire capacitance

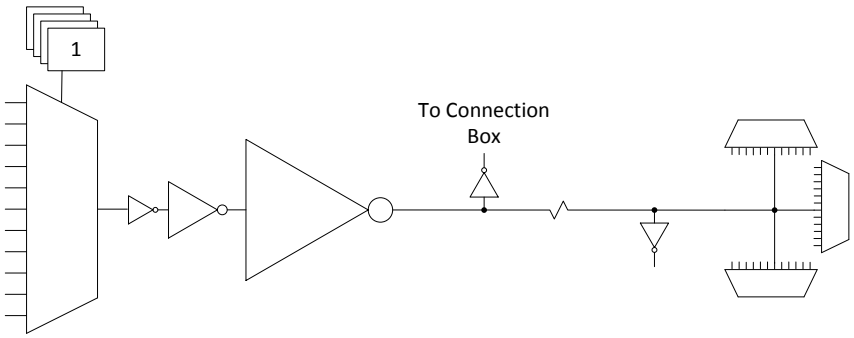

Figure 3: Switch Box

Multiplexer The multiplexer is modelled as 2-level pass-transistor multiplexer, comprised of minimum-size NMOS transistors. The number of inputs to the multiplexer is automatically determined.

Buffer The buffer is a multistage CMOS buffer. The buffer size is determined based upon output capacitance provided in the architecture file:

```
<switchlist>
        <switch type="mux" ... C_out="XXX"
</switchlist>
```

The user may override this method by providing the buffer size as shown below:

```
<switchlist>
        <switch type="mux" ... power_buf_size="16.0"
</switchlist>
```

The size is the drive strength of the buffer, relative to a minimum-sized inverter.

4.3 Input Connection Boxes

Input connection boxes are modelled as the following components (Figure 4):

- One buffer per routing track, sized to drive the load of all input multiplexers to which the buffer is connected (For buffer sizing see Section 5.2).
- One multiplexer per block input pin, sized according to the number of routing tracks that connect to the pin.

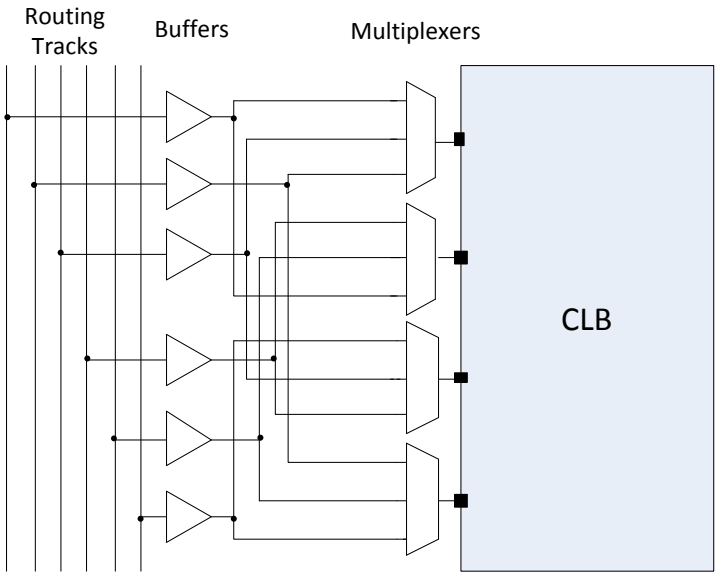

Figure 4: Conneciton Box

4.4 Clock Network

The clock network modelled is a four quadrant spine and rib design, as illustrated in Figure 5. At this time, the power model only supports a single clock. The model assumes that the entire spine and rib clock network will contain buffers separated in distance by the length of a grid tile. The buffer sizes and wire capacitances are specified in the architecture file using the following construct:

```
<clocks>
        <clock ... clock_options .../>
</clocks>
```

The following clock options are supported:

- `C_wire="1e-16"`: The absolute capacitance, in fards, of the wire between each clock buffer.
- `C_wire_per_m="1e-12"`: The wire capacitance, in fards per m. The capacitance is calculated using an automatically determined wirelength, based on the area of a tile in the FPGA.

- `buffer_size="2.0"`: The size of each clock buffer. This can be replaced with the "auto" keyword. See Section 5.2 for more information on buffer sizing.

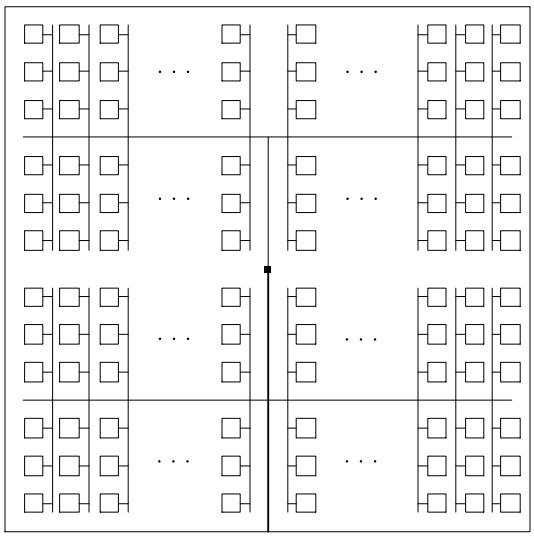

Figure 5: The clock network. Squares represent CLBs, and the wires represent the clock network.

5 Other Architecture Options & Techniques

5.1 Local Wire Auto-Sizing

Due to the significant user effort required to provide local buffer and wire sizes, we developed an algorithm to estimate them automatically. This algorithm recursively calculates the area of all entities within a CLB, which consists of the area of primitives and the area of local interconnect multiplexers. If an architecture uses new primitives in CLBs, it should include a function that returns the transistor count. This function should be called from within `power_count_transistors_primitive()`.

In order to determine the wire length that connects a parent entity to its children, the following assumptions are made:

- Assumption 1: All components (CLB entities, multiplexers, crossbars) are assumed to be contained in a square-shaped area.
- Assumption 2: All wires connecting a parent entity to its child pass through the interconnect square, which is the sum area of all interconnect multiplexers belonging to the parent entity.

Figure 6 provides an illustration of a parent entity connected to its child entities, containing one of each interconnect type (direct, many-to-1, and complete). In this figure, the square on the left represents the area used by the transistors of the interconnect multiplexers. It is assumed that all connections from parent to child will pass through this area. Real wire lengths could me more or less than this estimate; some pins in the parent may be directly adjacent to child entities, or they may have to traverse a distance greater than just the interconnect area. Unfortuantely, a more rigorous estimation would require some information about the transistor layout.

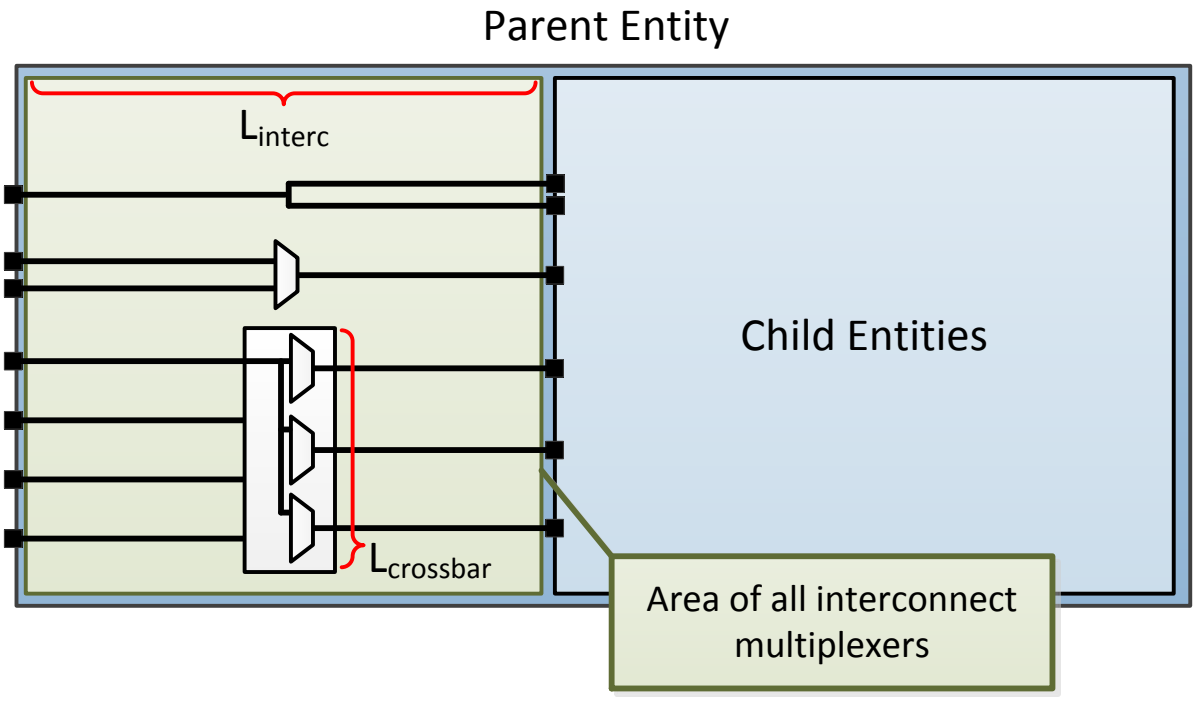

Figure 6: Local interconnect wirelength.

Connection from Entity Pin to:	Estimated Wirelength	Transistor Capacitance
Direct (Input or Output)	$0.5 \cdot L_{interc}$	0
Many-to-1 (Input or Output)	$0.5 \cdot L_{interc}$	C_{INV}
Complete m:n (Input)	$0.5 \cdot L_{interc} + L_{crossbar}$	$n \cdot C_{INV}$
Complete m:n (Output)	$0.5 \cdot L_{interc}$	C_{INV}

Table 1: Local interconnect wirelength and capacitance. (C_{INV} is the input capacitance of a minimum-sized inverter.)

Table 1 details how local wire lengths are determined as a function of entity and interconnect areas. It is assumed that each wire connecting a pin of a `pb_type` to an interconnect structure is of length $0.5 \cdot L_{interc}$. In reality, this length depends on the actual transistor layout, and may be much larger or much smaller than the estimated value. If desired, the user can override the 0.5 constant in the architecture file:

```
<architecture>
        <power>
                <local_interconnect factor="0.5">
        </power>
</architecture>
```

5.2 Buffer Sizing

In the power estimator, a buffer size refers to the size of the final stage of multi-stage buffer (if small, only a single stage is used). The specified size is the (W/L) of the NMOS transistor. The PMOS transistor will automatically be sized larger. Generally, buffers are sized depending on the load capacitance, using the following equation:

$$\text{Buffer Size} = \frac{1}{2 \cdot f_{LE}} * \frac{C_{Load}}{C_{INV}} \tag{1}$$

In this equation, C_{INV} is the input capacitance of a minimum-sized inverter, and f_{LE} is the logical effort factor. The logical effort factor is the gain between stages of the multi-stage buffer, which by default is 4 (minimal delay). The term $(2 \cdot f_{LE})$ is used so that the ratio of the final stage to the driven capacitance is smaller. This produces a much lower-area, lower-power buffer that is still close to the optimal delay, more representative of common design practises. The logical effort factor can be modified in the architecture file:

```
<architecture>
        <power>
                <buffers logical_effort_factor="4">
        </power>
</architecture>
```

5.3 Local Interconnect Capacitance

If using the `auto-size` or `wire-length` options (Section 4.1), the local interconnect capacitance must be specified. This is specified in the units of Farads/meter.

```
<architecture>
        <power>
                <local_interconnect C_wire="2.5e-15">
        </power>
</architecture>
```

6 Support

For support, please check `http://code.google.com/p/vtr-verilog-to-routing/wiki/Power`, or email vtr.power.estimation@gmail.com.