FPGA-SPICE User Manual

Xifan TANG

Email: xifan.tang@epfl.ch

Supervised by
Prof. Pierre-Emmanuel Gaillardon
and Prof. Giovanni De Micheli

Integrated Systems Laboratory
EPFL, Lausanne, Switzerland

October 26th, 2015

Table of Contents

I. MOTIVATION

The built-in timing and power analysis engines of VPR are based on analytical models [1]. Analytical model-based analysis can promise accuracy only on a limited number of circuit designs for which the model is valid. As the technology advancements create more opportunities on circuit designs and FPGA architectures, the analytical power model require to be updated to follow the new trends. However, without referring to simulation results, the analytical power models cannot prove their accuracy. SPICE simulators have the advantages on generality and accuracy over analytical models. For this reason, SPICE simulation results are often selected to check the accuracy of analytical models. Therefore, there is a strong need for a simulation-based power analysis approach for FPGAs, which can support general circuit designs.

It motivates us to develop FPGA-SPICE, an add-on for the current State-of-Art FPGA architecture exploration tools, VPR [2]. FPGA-SPICE aims at generating SPICE netlists and testbenches for the FPGA architectures supported by VPR. The SPICE netlists and testbenches are generated according to the placement and routing results of VPR. As a result, SPICE simulator can be used to perform accurate delay and power analysis. The SPICE simulation results are useful in three aspects: (1) it can provide accurate power analysis; (2) it helps improving the accuracy of built-in analytical models; and moreover (3) it creates opportunities in developing novel analytical models.

SPICE modeling for FPGA architectures requires a detailed transistor-level modeling for all the circuit elements within the considered FPGA architecture. However, current VPR architectural description language [3] does not offer enough transistor-level parameters to model the most common circuit modules, such as multiplexers and LUTs. Therefore, we develop an extension on the VPR architectural description language in order to model the transistor-level circuit designs.

In this manual, we will introduce how to use FPGA-SPICE to conduct accuracy power analysis. First, we give an overview on the design flow of FPGA-SPICE-based tool suites. Then, we show the command-line options of FPGA-SPICE. Afterwards, we introduce the extension on architectural language and the transistor-level design supports. Finally, we present how to simulate the generated SPICE netlists and testbenches.

In the appendix, we introduce the hierarchy of the generated SPICE netlists and testbenches, to help you customize the SPICE netlists. We also attach an example of architecture XML file for your interest.

The technical details can be found in our ICCD'15 paper [4].

II. OVERVIEW

As illustrated in Figure 1, FPGA-SPICE creates a modified VTR flow. All the input files for VPR do not need modifications except the architecture description XML. As simulation-based power analysis needs the transistor-level netlists, we extend the architecture description language to support transistor-level modeling (See details in Section V). FPGA-SPICE, embedded in VPR, outputs the SPICE netlists and testbenches according to placement and routing results, when enabled by command-line options. (See Section IV for details about command-line options.) Besides automatically generating all the SPICE netlists, FPGA-SPICE supports user-defined SPICE netlists for modules. We believe the support on user-defined SPICE netlists allows FPGA-SPICE to be general enough to support novel circuit designs and even technologies. (See Section VII for guidelines in customize your FPGA-SPICE compatible SPICE netlists.) With the dumped SPICE netlists and testbenches, a SPICE simulator, i.e. HSPICE, can be called to conduct power analysis. FPGA-SPICE automatically generates a shell script, which brings convenience for users to run all the simulations (See Section VIII).

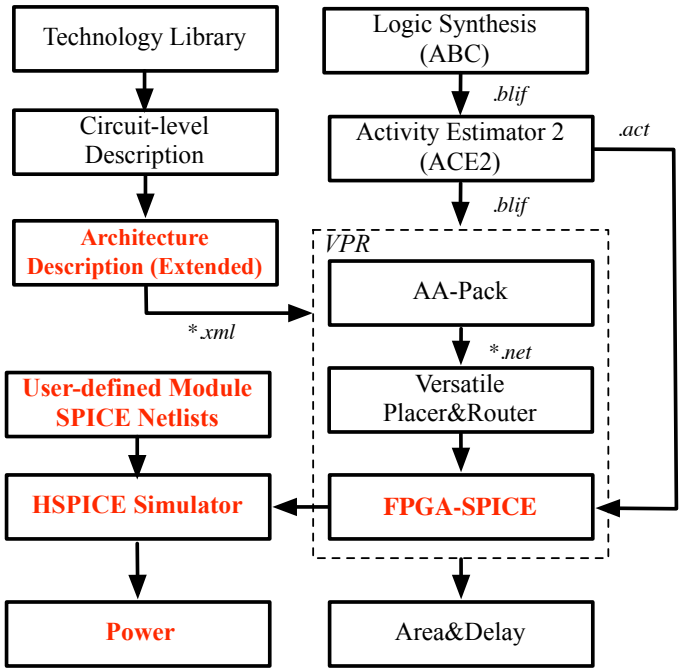

Figure 1. Modified VTR flow for FPGA-SPICE

III. HOW TO COMPILE

Running the Makefile in the released package can compile all the source codes. The released package includes a version of VPR with FPGA-SPICE support, as depicted in the dash-lined box in Figure 1. Full package of Logic Synthesis and Activity Estimator tools can be found in the VTR project.

Download link: http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html

IV. COMMAND LINE OPTIONS

All the command line options of FPGA-SPICE can be shown by calling the help menu of VPR. Here are all the FPGA-SPICE-related options that you can find:

Spice Support Options:

--fpga_spice
--spice_dir <directory_path_output_spice_netlists>
--print_spice_top_testbench
--print_spice_lut_testbench
--print_spice_dff_testbench
--print_spice_pb_mux_testbench
--print_spice_cb_mux_testbench
--print_spice_sb_mux_testbench
--print_spice_grid_testbench
--fpga_spice_leakage_only
--fpga_spice_parasitic_net_estimation_off

Note that FPGA-SPICE requires the input of activity estimation results (*.*act file*) from ACE2. Remember to use the option *--activity_file <act_file>* to read the activity file.

To dump full-chip-level testbenches, the option *--print_spice_top_testbench* should be enabled.

To dump grid-level testbenches, the options *--print_spice_grid_testbench, --print_spice_cb_mux_testbench* and *--print_spice_sb_mux_testbench* should be enabled.

To dump component-level testbenches, the options *--print_spice_lut_testbench, --print_spice_dff_testbench, --print_spice_cb_mux_testbe-nch* and *--print_spice_sb_mux_testbench* should be enabled.

Table 1 describes each option.

Table 1 Command-line options and their description.

--fpga_spice
Turn on the FPGA-SPICE.
--spice_dir <dir_path>
Specify the directory that all the SPICE netlists will be outputted to. <dir_path> is the destination directory.
--print_spice_top_testbench
Print the full-chip-level testbench for the FPGA.
--print_spice_lut_testbench
Print the testbenches for all the LUTs.
--print_spice_dff_testbench
Print the testbenches for all the FFs.
--print_spice_pb_mux_testbench
Print the testbenches for all the multiplexers in the logic blocks.
--print_spice_cb_mux_testbench
Print the testbenches for all the multiplexers in Connection Boxes.

--print_spice_sb_mux_testbench
Print the testbenches for all the multiplexers in Switch Blocks.
--print_spice_grid_testbench
Print the testbenches for the logic blocks.
--fpga_spice_leakage_only
FPGA-SPICE conduct power analysis on the leakage power only.
--fpga_spice_parasitic_net_estimation_off
Turn off the parasitic net estimation technique. The parasitic net estimation technique is used to analyze the parasitic net activities which improves the accuracy of power analysis. When turned off, the errors between the full-chip-level and grid/component-level testbenches will increase.

V. ARCHITECTURAL DESCRIPTION LANGUAGE

1. General Hierarchy

The extension of the VPR architectural description language is developed as an independent branch of the original one. Most of the FPGA-SPICE descriptions are located under a XML node called <*spice_settings*>, which is a child node under the root node <*architecture*>. Under the <spice_settings>, a number of child node are created for describing SPICE simulation settings, technology library and transistor-level modeling of circuit modules.

In the following sub-sections, we will introduce the structures of these XML nodes and the parameters provided.

2. Parameters for SPICE simulation settings

All the parameters that need to be defined in the HSPICE simulations are located under a child node called <parameters>, which is under its father node <spice_settings>.

The parameters are divided into three categories and can be defined in three XML nodes, <options>, <measure> and <stimulate>, respectively.

- *The XML node <options>*

<options sim_temp="int" post="string"captab="string" fast="string" />
These properties define the options that will be printed in the top SPICE netlists. ***sim_temp:*** specify the temperature which will be defined in SPICE netlists. In the top SPICE netlists, it will show as *.temp <int>*. ***post:*** [on|off]. Specify if the simulation waveforms should be printed out after SPICE simulations. In all the SPICE netlists, it will show as *.option POST* when turned *on*. **Note that when the SPICE netlists are large or a long simulation time period is defined, the post option is recommended to be *off*.** If not, huge disk space will be occupied by the waveform files. ***captab*:** [on|off]. Specify if the capacitances of all the nodes in the SPICE netlists will be printed out. In the top SPICE netlists, it will show as *.option CAPTAB* when turned *on*. When turned on, the SPICE simulation runtime may increase. ***fast:*** [on|off]. Specify if the fast simulation algorithm is turned on in SPICE. In all the SPICE netlists, it will show as *.option FAST* when turned *on*. **Note that when the SPICE netlists are large or a long simulation time period is defined, the fast option is recommended to be *on*, in order to reduce the runtime for an accuracy comprimise.**

- *The XML node <stimulate>*

Figure 2 depicts the definition of the slew and delays of signals and the parameters that can be supported by FPGA-SPICE.

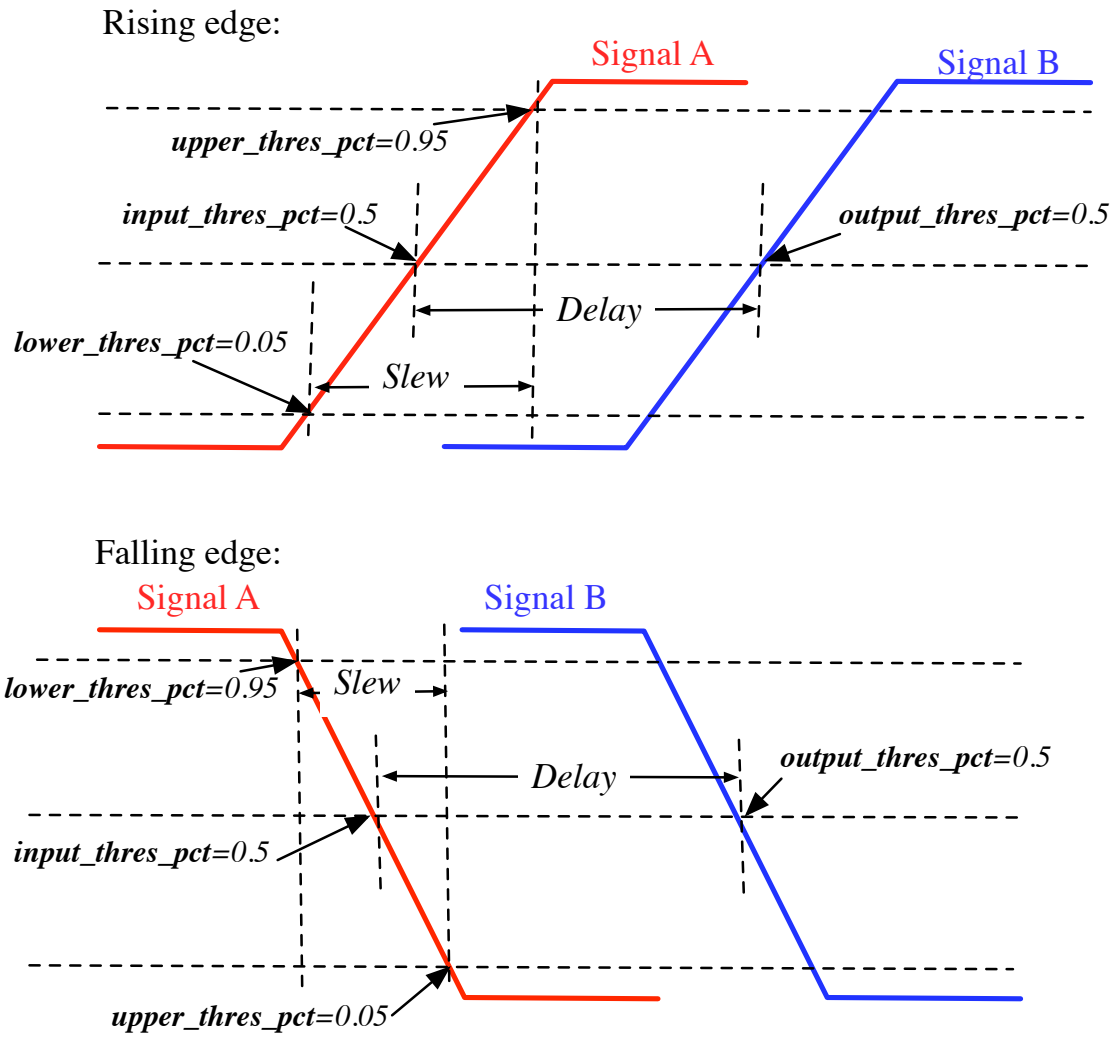

Figure 2 Parameter in measuring the slew and delay of signals

```
<stimulate>
  <clock freq="float" sim_slack="float">
    <rise slew_time="float" slew_type="string"/>
    <fall slew_time="float" slew_type="string"/>
  </clock>
</stimulate>
```

Define stimulates for the clock signal.

freq: [Hz] Specify the clock frequency that are used in SPICE simulations. If the clock frequency is specified. The *sim_slack* option is disgarded.

***sim_slack*:** add a slack to the critical path delay in the SPICE simulation. For example, sim_slack=0.2 implies that the clock period in SPICE simulations is 1.2 of the critical path delay reported by VPR. Only valid when option *freq* is not specified.

slew_type&slew_time: define the slew of clock signals at the rising/falling edge. Property *slew_type* can be either absolute or fractional [abs|frac].
The type of absolute implies that the slew time is the absolute value. For example, *slew_time*=20e-12, *slew_type*=abs means that the slew of a clock signal is 20*ps*.
The type of fractional means that the slew time is related to the time period (frequency) of the clock signal. For example, *slew_time*=0.05, *slew_type*=frac means that the slew of a clock signal takes 5% of the time period of the clock.

```
<stimulate>
  <input>
    <rise slew_time="float" slew_type="string"/>
    <fall slew_time="float" slew_type="string"/>
  </input>
</stimulate>
```

Define the slew of input signals at the rising/falling edge.

slew_type&slew_time: define the slew of all the input signals at the rising/falling edge. Property *slew_type* can be either absolute or fractional *[abs|frac]*.
The type of absolute implies that the slew time is the absolute value. For example, *slew_time*=20e-12, *slew_type*=abs means that the slew of a clock signal is 20*ps*.
The type of fractional means that the slew time is related to the time period (frequency) of the clock signal. For example, *slew_time*=0.05, *slew_type*=frac means that the slew of a clock signal takes 5% of the time period of the clock.
Note that these slew settings are valid for all the input signals of the testbenches in different complexity levels.

- *The XML node <measure>*

<measure sim_num_clock_cycle="int"accuracy="float"accuracy_type="string"/>
sim_num_clock_cycle: can be either "auto" or an integer. By setting to "auto", FPGA-SPICE automatically determines the number of clock cycles to simulate, which is related to the average of all the signal density in ACE2 results. When set to an integer, FPGA-SPICE will use the given number of clock cycles in the SPICE netlists. ***accuracy_type:*** *[abs|frac]*. Specify the type of transient step in SPICE simulation. When *abs* is selected, the accuracy should be the absolute value, such as 1e-12. When *frac* is selected, the accuracy is the number of simulation points in a clock cycle time period, for example, 100. ***accuracy:*** specify the transient step in SPICE simulation. Typically, the smaller the step is, the higher accuracy can be reached while the long simulation runtime is. The recommended accuracy is between 0.1ps and 0.01ps, which generates good accuracy and runtime is not significantly long.

Users can define the parameters in measuring the slew of signals, under a child node <slew> of the node <measure>.

<rise upper_thres_pct="float" lower_thres_pct="float"/>
Define the starting and ending point in measuring the slew of a rising edge of a signal. ***upper_thres_pct:*** *the ending point in measuring the slew of a rising edge. It is expressed as a percentage of the maximum voltage of a signal. For example, the meaning of upper_thres_pct=0.95 is depicted in* Figure 2. ***lower_thres_pct:*** *the starting point in measuring the slew of a rising edge. It is expressed as a percentage of the maximum voltage of a signal. For example, the meaning of lower_thres_pct=0.05 is depicted in* Figure 2.
<fall upper_thres_pct="float" lower_thres_pct="float"/>
upper_thres_pct: *the ending point in measuring the slew of a falling edge. It is expressed as a percentage of the maximum voltage of a signal. For example, the meaning of upper_thres_pct=0.05 is depicted in* Figure 2. ***lower_thres_pct:*** *the starting point in measuring the slew of a falling edge. It is expressed as a percentage of the maximum voltage of a signal. For example, the meaning of lower_thres_pct=0.95 is depicted in* Figure 2.

Users can define the parameters related to measurements of delays between signals, under a child node <delay> of the node <measure>.

<rise input_thres_pct="float" output_thres_pct="float"/>
Define the starting and ending point in measuring the delay between two signals when they are both at a rising edge. ***input_thres_pct:*** *the starting point in measuring the delay of a rising edge. It is expressed as a percentage of the maximum voltage of a signal. For example, the*

meaning of input_thres_pct=0.5 is depicted in Figure 2.

output_thres_pct: *the ending point in measuring the delay of a rising edge. It is expressed as a percentage of the maximum voltage of a signal. For example, the meaning of output_thres_pct=0.5 is depicted in* Figure 2.

<fall input_thres_pct="float" output_thres_pct="float"/>

Define the starting and ending point in measuring the delay between two signals when they are both at a falling edge.

input_thres_pct: *the starting point in measuring the delay of a falling edge. It is expressed as a percentage of the maximum voltage of a signal. For example, upper_thres_pct=0.5 is depicted in* Figure 2.

output_thres_pct: *the ending point in measuring the delay of a falling edge. It is expressed as a percentage of the maximum voltage of a signal. For example, lower_thres_pct=0. 5 is depicted in* Figure 2.

3. Technology library Declaration

<tech_lib lib_type="string" transistor_type="string" lib_path="string" nominal_vdd="float"/>

lib_type: can be either industry or academia [*industry*|*academia*]. For the industry library, a number of transistor types are available and the type of transistor should be declared in the property transistor_type.

transistor_type: This XML property specify the transistors to be used in the industry library. For example, the type of transistors can be "TT", "FF" etc.

lib_path: specify the path of the library. For example: *lib_path=/home/tech/45nm.pm*.

nominal_vdd: specify the working voltage for the technology. The voltage will be used as the supply voltage in all the SPICE netlist.

<transistors pn_ratio="float" model_ref="string"/>

pn_ratio: specify the ratio between *p*-type transistors and *n*-type transistors. The ratio will be used when building circuit structures such as inverters, buffers etc.

model_ref: specify the reference of in calling a transistor model. In SPICE netlist, define a transistor follows the convention: *<model_ref><trans_name> <ports> <model_name>*. The reference depends on the technology and the type of library. For example, PTM bulk model use "M" as the reference while PTM FinFET model use "X" as the reference.

<nmos model_name="string" chan_length="float" min_width="float"/>
<pmos model_name="string" chan_length="float" min_width="float"/>

model_name: specify the name of the p/n type transistor, which can be found in the manual of the technology provider.

chan_length: specify the channel length of p/n type transistor.

min_width: specify the minimum width of p/n type transistor. This parameter will be

used in building inverter, buffer and etc. as a base number for transistor sizing.

4. Define SPICE model for modules

For each module that appears in the FPGA architecture, a spice model should be defined. In the definition of a spice model, user can specify if the SPICE netlists of the module is either auto-generated or user-defined.

```
<module_spice_models>
  <spice_model type="string" name="string" prefix="string" is_default="int"
netlist="string">
    <transistor-level circuit design features>
  </spice_model>
  …
</module_spice_models>
```

module_spice_models: the father node for all the spice models. All the spice models should be defined under this XML node.

spice_model: the child node defining transistor-level modeling parameters.

type: can be [***mux***|***wire***|***chan_wire***|***sram***|***lut***|***ff***|***hard_logic***|***inpad***|***outpad***]. Specify the type of this spice model. The provided types cover all the modules in FPGAs. For the spice models in the type of *mux*/*wire*/*chan_wire*/*lut*, FPGA-SPICE can auto-generate SPICE netlists. For the rest, FPGA-SPICE requires a user-defined SPICE netlists.

name: define the name of this spice model. The name should be unique and will be used in create the sub-circuit of the spice model in SPICE netlists.
Note that for a customized SPICE netlist, the name defined here should be the name of the top-level sub-circuit in the customized SPICE netlist.
FPGA-SPICE will check if the given name is conflicted with any reserved words.

prefix: specify the name of the spice_model to shown in the auto-generated SPICE netlists. The prefix can be the same as the name defined above. And again, the prefix should be unique.

is_default: can be [1|0], corresponding to [true|false] respectively. Specify this spice model is the default one for some modules, such as multiplexers. If a module is not linked to any spice model by users, FPGA-SPICE will find the default spice model defined in the same type and link. For a spice model type, only one spice model can be set as default.

netlist: specify the path and file name of a customized SPICE netlist. For some modules such as SRAMs, FFs, inpad and outpads, FPGA-SPICE does not support auto-generation of the transistor-level sub-circuits because their circuit design are highly dependent on the technology nodes. These circuit designs should be specified by users. For the other modules that can be auto-generated by FPGA-SPICE, user can also define a custom netlist. **Multiplexers can not be user-defined.**
If netlist is not specified, FPGA-SPICE auto-generates the SPICE netlists for multiplexers, wires and LUTs.
Note that if the user-defined netlists, such as LUTs, the decoding methodology should comply with the auto-generated LUTs (See Section 4.3)

Under the XML node spice_model, the features of transistor-level designs can be defined. In the following table, we show the common features supported for all the modules. Then, we will introduce unique features supported only for some spice models types.

<spice_model type="string" name="string" prefix="string" is_default="int" netlist="string"> *<design_technology type="string"/>* *<input_buffer exist="string" type="string" size="int" tapered="string" tap_buf_level="int" f_per_stage="float"/>* *<output_buffer exist="string" type="string" size="int" tapered="string" tap_buf_level="int" f_per_stage="float"/>* *<pass_gate_logic type="string" nmos_size="float" pmos_size="float"/>* *<port type="string" prefix="string" size="int"/>* *</spice_model>*
design_technology : ***type:*** *[cmos|rram]. Specify the type of design technology of the spice_model. Currently, the RRAM-based designs are only supported for multiplexers.*
input_buffer and output_buffer: ***exist:*** [*on|off*]. Define the existence of the input_buffer or output_buffer. Note that the existence is valid for all the inputs and outputs. Note that if users want only part of the inputs (or outputs) to be buffered, this is not supported here. A solution can be building a user-defined SPICE netlist. ***type:*** [*inverter|buffer*]. Specify the type of input_buffer or output_buffer, can be inverter and buffer. ***size:*** Specify the driving strength of inverter/buffer. For a buffer, the size is the driving strength of the inverter at the second level. We consider a two-level structure for a buffer here. The support for multi-level structure of a buffer will be introduced in the tapered options. ***tapered:*** *[on|off]*. Define if the buffer is a tapered (multi-level) buffer. ***tap_buf_level:*** define the number of levels of a tapered buffer. This parameter is valid only when tapered is turned on. ***f_per_stage***: define the ratio of driving strength between the levels of a tapered buffer. This parameter is valid only when tapered is turned on. Default value is 4.
port: define the port list of a spice model. ***type:*** can be [input|output|sram|clock]. For programmable modules, such as multiplexers and LUTs, SRAM ports should be defined. For registers, such as FFs and memory banks, clock ports should be defined. ***prefix:*** the name of the port. Each port will be shown as *<prefix>[i], 0≤i<size* in SPICE netlists. ***size:*** bandwidth of the port.
pass_gate_logic: defined the parameters in pass-gates, which are used in building

multiplexers and LUTs.

type: *[transmission_gate|pass_transistor]. The transmission gate consists of a NMOS transistor and a PMOS transistor. The pass transistor consists of a NMOS transistor.*

nmos_size: the size of NMOS transistor in a transmission gate or pass_transistor, expressed in terms of the *min_width* defined in XML node *<transistors>*.

pmos_size: the size of PMOS transistor in a transmission gate, expressed in terms of the *min_width* defined in XML node *<transistors>*.

4.1. SRAMs

```
<spice_model type="sram" name="string" prefix="string" netlist="string"/>
   <design_technology type="cmos"/>
   <input_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
   <output_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
   <port type="input" prefix="string" size="int"/>
   <port type="output" prefix="string" size="int"/>
</spice_model>
```

Note: The circuit designs of SRAMs are highly dependent on the technology node and well optimized by engineers. Therefore, FPGA-SPICE requires users to provide their customized SRAM SPICE netlists. A sample SPICE netlist of SRAM can be found in the directory SpiceNetlists in the released package.

FPGA-SPICE assumes that all the LUTs and MUXes employ the SRAM circuit design. Therefore, currently only one SRAM type is allowed to be defined.

The information of input and output buffer should be clearly specified according to the customized SPICE netlist! The existence of input/output buffers will influence the decision in creating testbenches, which may leads to larger errors in power analysis.

4.2. Multiplexers and flexible routing architectures

```
<spice_model type="mux" name="string" prefix="string" is_default="int"/>
   <design_technology type="string" structure="string" num_level="int"
ron="float" roff="float" prog_transistor_size="float"/>
   <input_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
   <output_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
   <pass_gate_logic type="string" nmos_size="float" pmos_size="float"/>
   <port type="input" prefix="string" size="int"/>
   <port type="output" prefix="string" size="int"/>
   <port type="sram" prefix="string" size="int"/>
</spice_model>
```

Note: customized SPICE netlists are not currently supported for multiplexers.

design_technology:

structure: can be [tree|multi-level|one-level]. The structure options are valid for SRAM-based multiplexers. For RRAM-based multiplexers, currently we only support

the circuit design in [5].

num_level: specify the number of levels when multi-level structure is selected.

ron: valid only when the type of design technology is *rram*. Specify the on-resistance of the RRAM device used in the RRAM-based multiplexer.

roff: valid only when the type of design technology is *rram*. Specify the off-resistance of the RRAM device used in the RRAM-based multiplexer.

prog_transistor_size: valid only when the type of design technology is *rram*. Specify the size of programming transistors used in the RRAM-based multiplexer, we use only n-type transistor and the size should be expressed in terms of the *min_width* defined in XML node <*transistors*>.

port: for a multiplexer, the three types of ports, *input, output* and *sram* should be defined.

Figure 3 illustrates an example of multiplexer modelling, which consists of input/output buffers and a transmission-gate-based tree structure.

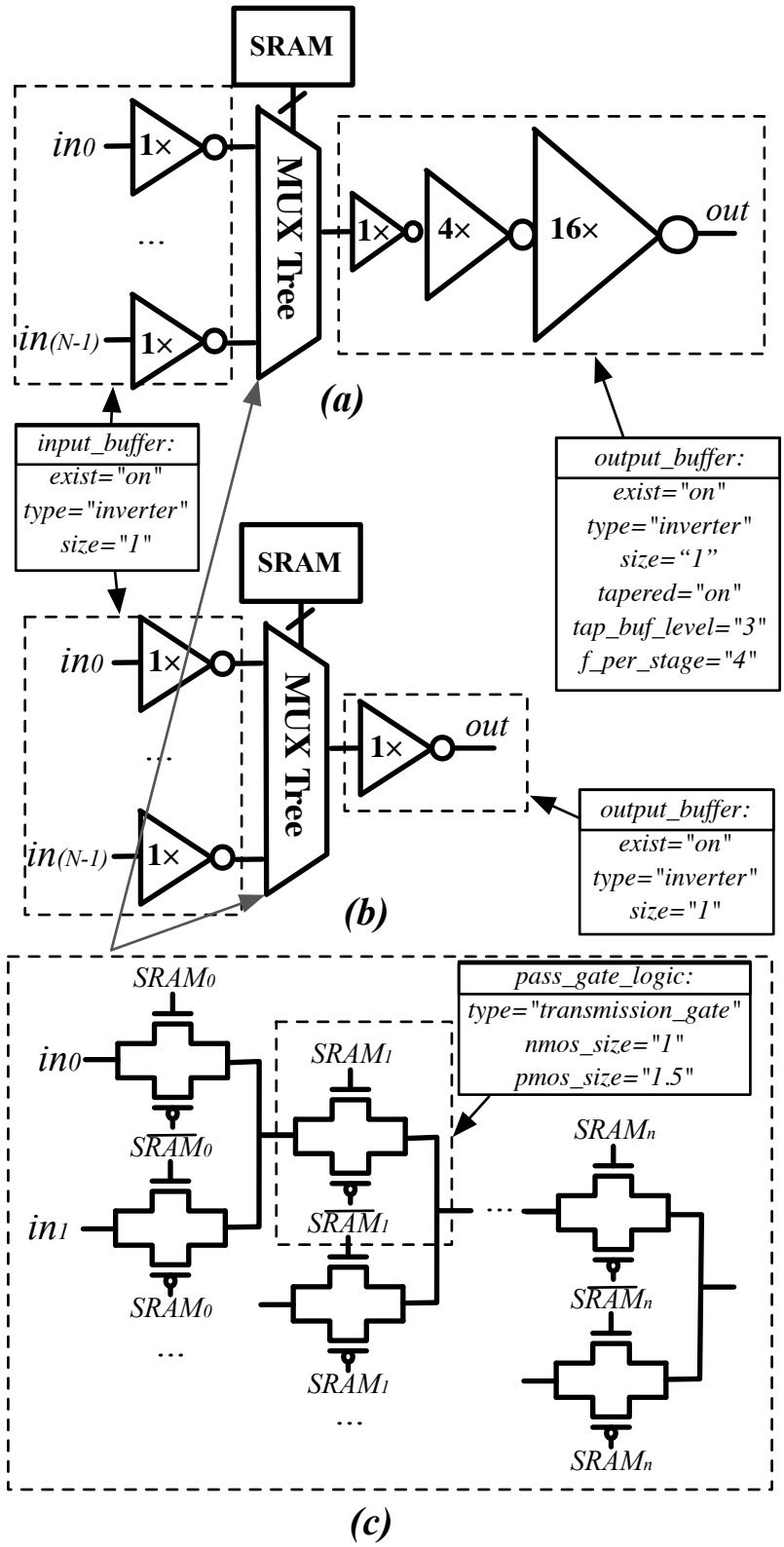

Figure 3 An example of a tree-like multiplexer with transistor-level design parameters

4.3. LUTs

```
<spice_model type="lut" name="string" prefix="string" is_default="int"
netlist="string"/>
   <design_technology type="cmos"/>
   <lut_input_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
   <input_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
   <output_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
   <pass_gate_logic type="string" nmos_size="float" pmos_size="float"/>
   <port type="input" prefix="string" size="int"/>
   <port type="output" prefix="string" size="int"/>
   <port type="sram" prefix="string" size="int"/>
</spice_model>
```

Note:
The SPICE netlists of LUT can be auto-generated or customized.
The auto-generated LUTs are based on a tree-like multiplexer, whose gates of the transistors are used as the inputs of LUTs and the drains/sources of the transistors are used for configurable memories (SRAMs).
The LUT provided in customized SPICE netlist should have the same decoding methodology as the traditional LUT.

Additional design parameters for LUTs:
lut_input_buffer : Specify the buffer for the inputs of a LUT (gates of the internal multiplexer).

Instructions of defining design parameters:
input_buffer: Specify the buffer/inverter that connects the SRAM outputs to the inputs of multiplexer.

pass_gate_logic: Specify the pass-gates of the internal multiplexer, the same as the multiplexers.

port: three types of ports (*input, output and sram*) should be defined. If the user provides an customized SPICE netlist, the bandwidth of ports should be defined to the same as the SPICE netlist.

Figure 4 illustrates an example of LUT modeling, which consists of input/output buffers and a transmission-gate-based tree structure.

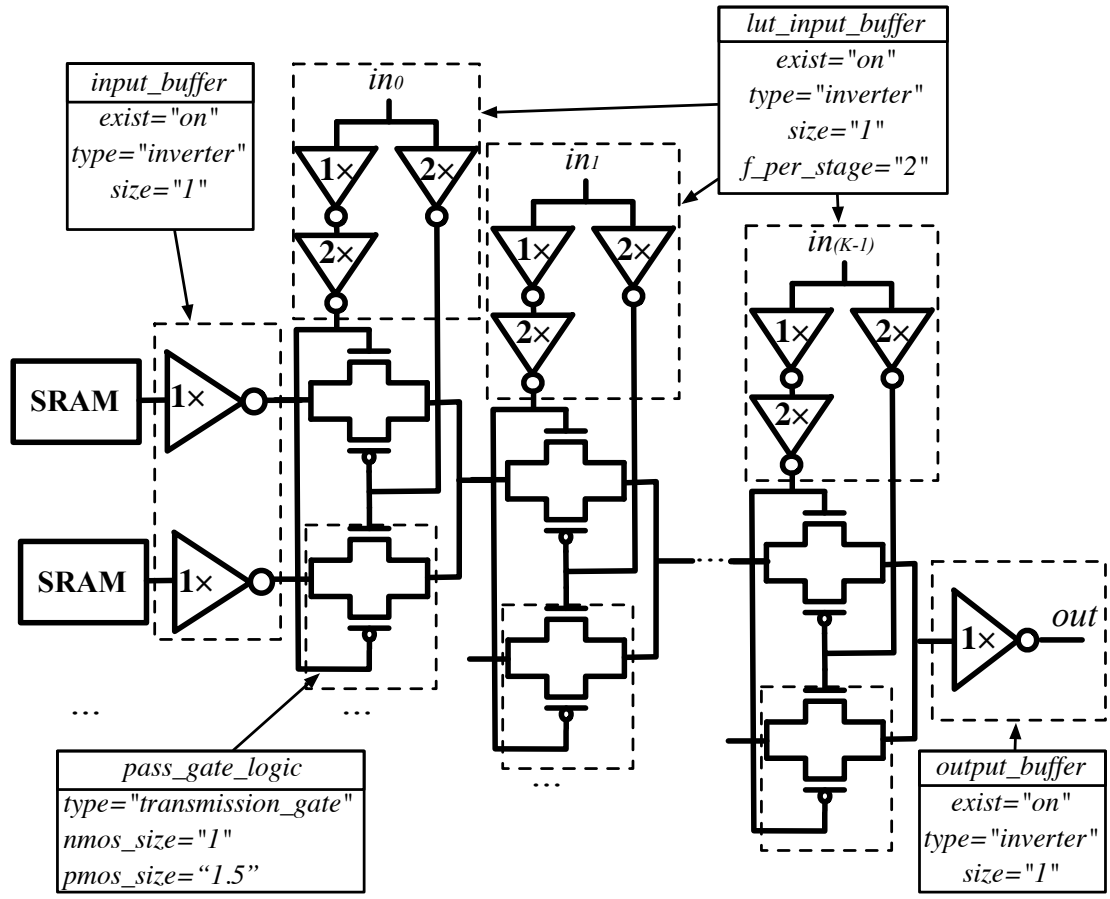

Figure 4 An example of a LUT with transistor-level design parameters.

4.4. FFs

```
<spice_model type="ff" name="string" prefix="string" netlist="string"/>
   <design_technology type="cmos"/>
   <input_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
   <output_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
   <port type="input" prefix="string" size="int"/>
   <port type="output" prefix="string" size="int"/>
   <port type="clock" prefix="string" size="int"/>
</spice_model>
```

Note: The circuit designs of flip-flops are highly dependent on the technology node and well optimized by engineers. Therefore, FPGA-SPICE requires users to provide their customized SRAM SPICE netlists. A sample SPICE netlist of FF can be found in the directory SpiceNetlists in the released package.

The information of input and output buffer should be clearly specified according to the customized SPICE netlist! The existence of input/output buffers will influence the decision in creating testbenches, which may leads to larger errors in power analysis.

FPGA-SPICE currently support only one clock domain in the FPGA. Therefore there should be only one clock port to be defined and the size of the clock port should be 1.

Instructions of defining design parameters:
port: three types of ports (*input, output and clock*) should be defined. If the user provides an customized SPICE netlist, the bandwidth of ports should be defined to the same as the SPICE netlist.

4.5. Hard Logics, inpad and outpad

```
<spice_model type="inpad|outpad|hardlogic" name="string" prefix="string"
netlist="string"/>
   <design_technology type="cmos"/>
   <input_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
   <output_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
   <port type="input" prefix="string" size="int"/>
   <port type="output" prefix="string" size="int"/>
</spice_model>
```

Note:
The circuit designs of inpad/output pad are highly dependent on the technology node and well optimized by engineers.
As more functional units are included in FPGA architecture, it is impossible to auto-generate these functional units [3].
Therefore, FPGA-SPICE requires users to provide their customized SPICE netlists. A sample SPICE netlist of inpad/outpad can be found in the directory SpiceNetlists in the released package.

The information of input and output buffer should be clearly specified according to the customized SPICE netlist! The existence of input/output buffers will influence the decision in creating testbenches, which may leads to larger errors in power analysis.

Instructions of defining design parameters:
port: two types of ports (*input* and *output*) should be defined. If the user provides a user-defined SPICE netlist, the bandwidth of ports should be defined to the same as the SPICE netlist.

4.6. Wire Segments

FPGA-SPICE provides two types of SPICE models for the wire segments in FPGA architecture. One type is called wire, which targets the local wires inside the logic blocks. The wire has one input and one output, directly connecting the output of a driver and the input of the downsteam unit, respectively

The other type is called chan_wire, especially targeting the channel wires. The channel wires have one input and two outputs, one of which is connected to the inputs of Connection Boxes while the other is connected to the inputs of Switch Boxes. Two outputs are created because from the view of layout, the inputs of Connection Boxes are typically connected to the middle point of channel wires,

which has less parasitic resistances and capacitances than connected to the ending point.

```
<spice_model type="string" name="string" prefix="string" netlist="string"/>
  <design_technology type="cmos"/>
  <input_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
  <output_buffer exist="string" type="string" size="int" tapered="string"
tap_buf_level="int" f_per_stage="float"/>
  <port type="input" prefix="string" size="int"/>
  <port type="output" prefix="string" size="int"/>
  <wire_param model_type="string" res_val="float" cap_val="float"
level="int"/>
</spice_model>
```

Note: FPGA-SPICE can auto-generate the SPICE model for wires while also allows users to provide their customized SPICE netlists.

The information of input and output buffer should be clearly specified according to the customized SPICE netlist! The existence of input/output buffers will influence the decision in creating testbenches, which may leads to larger errors in power analysis.

Instructions of defining design parameters:

spice_model:

type: can be [wire|chan_wire]. The SPICE model *wire* targets the local wire inside the logic block while the *chan_wire* targets the channel wires in global routing.

port: two types of ports (*input* and *output*) should be defined. If the user provides an customized SPICE netlist, the bandwidth of ports should be defined to the same as the SPICE netlist.

wire_param:

model_type: can be [pie|T], corresponding to the π-type and T-type RC wire models.

res_val: specify the total resistance of the wire

cap_val: specify the total capacitance of the wire.

level: specify the number of levels of the RC wire model.

Figure 5 depicts the modeling for a length-2 channel wire.

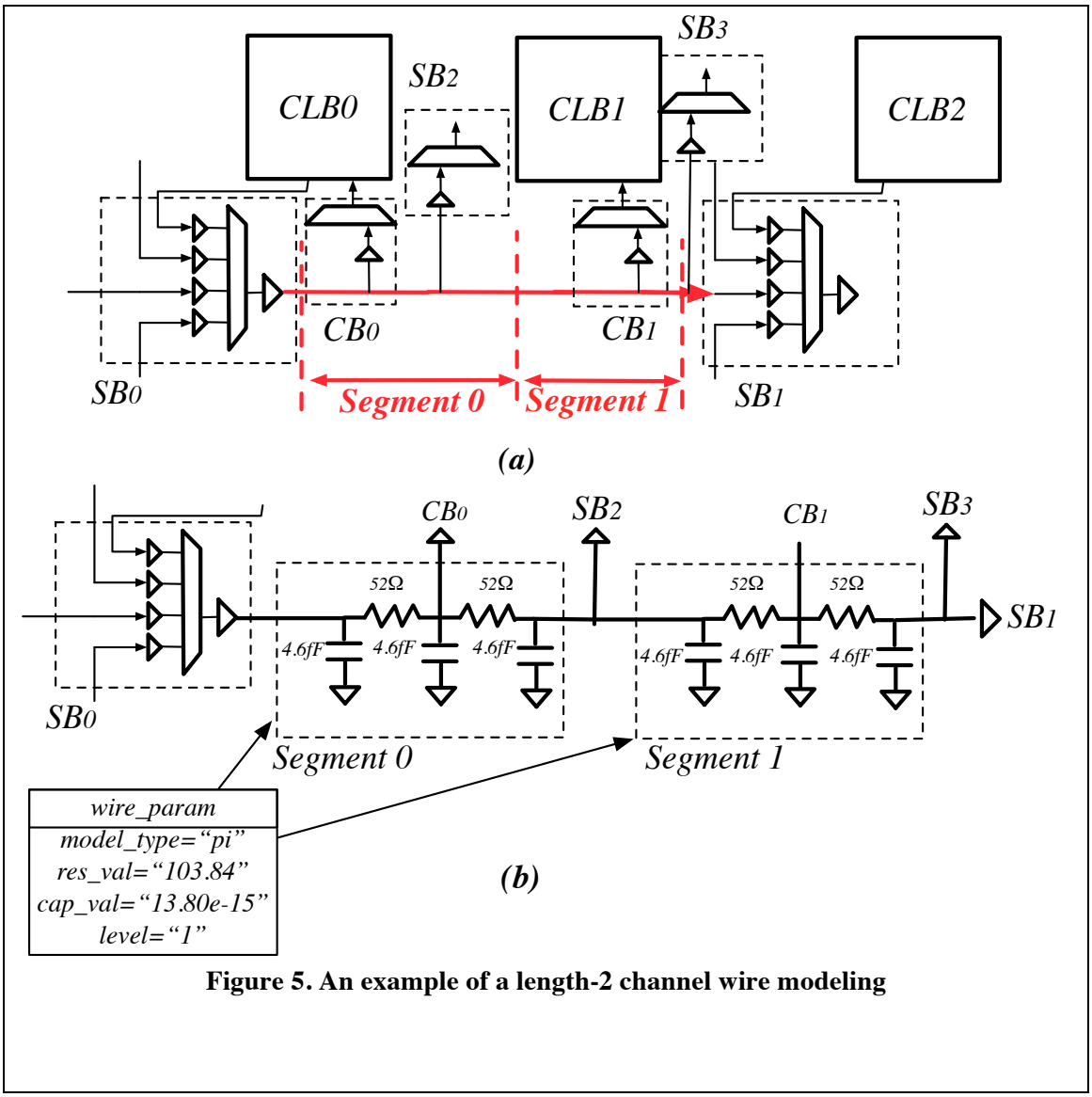

Figure 5. An example of a length-2 channel wire modeling

5. Link to Defined SPICE Models

Each defined SPICE model should be linked to a FPGA module defined in the original part of architecture descriptions. It helps FPGA-SPICE creating the SPICE netlists for logic/routing blocks. Since the original part lacks such support, we create a few XML properties to link to SPICE models.

5.1. SRAM

To link the defined SPICE model of SRAM into the FPGA architecture description, a new line in XML format should be added under the XML node *device*. The new XML node is named as *sram*, which defines the area of a SRAM and the name of SPICE model to be linked. And example is shown as follows:

<sram area="int" spice_model_name="string">

area is expressed in terms of the number of minimum width transistors. The SRAM area defined in this line is used in the area estimation of global routing multiplexers. *spice_model_name* should match the name of SPICE model that

have been defined under XML node *module_spice_model*. The type of the linked SPICE model should be *sram*.
Here is an example.

<sram area="4" spice_model_name="sram6T">

5.2. Switch Blocks

Original VPR architecture description contains a XML node called *switchlist* under which all the multiplexers of switch blocks are described.
To link a defined SPICE model to a multiplexer in the switch blocks, a new XML property *spice_model_name* should added to the descriptions.

Here is an example:

<switchlist> *<switch type="mux" name="string" R="float" Cin="float" Cout="float" Tdel="float" mux_trans_size="float" buf_size="float"* ***spice_model_name****="string"/>* *</switchlist>*
spice_model_name*:* should match a SPICE model whose type is *mux* defined under *module_spice_models*.

5.3. Connection Boxes

To link the defined SPICE model of multiplexer to the Connection Boxes, a spice_model_name should be added to the definition of Connection Boxes switches. However, the original architecture descriptions do not offer a switch description for connection boxes as they do for the switch blocks.

Therefore, FPGA-SPICE requires a new XML node called *cblock* under the **root** XML node *architecture*, where a switch for connection boxes can be defined.

Here is the example:

<cblock> *<switch type="mux" name="string" R="float" Cin="float" Cout="float" Tdel="float" mux_trans_size="float" buf_size="float"* ***spice_model_name****="string"/>* *</cblock>*
spice_model_name*:* should match a SPICE model whose type is *mux* defined under *module_spice_models*.

5.4. Channel Wire Segments

Simliar to *the SB and CB,* the channel wire segments in the original architecture descriptions can be adapted to provide a link to the defined SPICE model.

<segmentlist> *<segment freq="float" length="int" type="string" Rmetal="float" Cmetal="float"* ***spice_model_name****="string"/>* *</segmentlist>*
spice_model_name*:* should match a SPICE model whose type is *chan_wire* defined under *module_spice_models*.

5.5. Modules inside logic blocks (pb_type)

The architecture description employs a hierarchy of pb_types to depicting the sub modules and complex interconnections inside logic blocks. Each leaf node and interconnection in pb_type hierarchy should be linked to a SPICE model.

<pb_type name="string" blif_model="string" ***spice_model_name****="string"* ***idle_mode_name****="string">* *</pb_type>* *<interconnect>* *<direct name="string" input="string" output="string"* ***spice_model_name****="string"/>* *<complete name="string" input="string" output="string"* ***spice_model_name****="string"/>* *<mux name="string" input="string" output="string"* ***spice_model_name****="string"/>* *</interconnect>*
spice_model_name*:* should match a SPICE model defined under *module_spice_models*. For the interconnection type *direct*, the type of the linked SPICE model should be *wire*. For *mux*, the type of the linked SPICE model should be *mux*. For *complete*, the type of the linked SPICE model can be either *mux* or *wire*, depending on the case. **idle_mode_name:** tell the name of the mode that the pb_type is configured to be by default. This is critical in building SPICE netlists for unused logic blocks.

VI. RESERVED WORD LIST

FPGA-SPICE auto-generate the sub-circuits for NMOS, PMOS, inverter, buffer and pass-gates. With theses sub-circuits, FPGA-SPICE creates the sub-circuits for the multiplexers, LUTs and wires of the logic blocks and routing resources. Therefore, the name of these auto-generated sub-circuits becomes a reserved word list. There should not be a conflict on the names of sub-circuits between the auto-generated and customized SPICE netlists.

Table 2. Reserved Words

Sub-circuit Name	Usage
vpr_nmos	The NMOS transistor used in every SPICE netlist.
vpr_pmos	The PMOS transistor used in every SPICE netlist.
inv	Inverter
buf	Buffer
tapbuf_level<int>	Tapered buffer, the integer represents the number of levels.
cpt	Transmission gate
<clb_name><x><y> _mode[<mode_name>] _<pb_type_name> <index>… _mode[<mode_name>] _<pb_type_name> <index>	Sub-circuit name of each logic block and logic elements. *<clb_name>* is the name of a logic block in architecture description. *<x> and <y> are coordinates of the logic block.* *<mode_name>* is the name of operation mode of logic blocks/elements. *<pb_type_name>* is the name of a pb_type in architecture description. *<index>* is the index of a pb_type in the context of its farther pb_type. *<x>,<y> and <index>* follow the coordinate system of VPR.
cb_x[x][y]	Sub-circuit name of connection boxes that connect the horizontal routing tracks to logic blocks. *<x> and <y>* are coordinates of the connection box, following the coordinate system of VPR.
cb_y[x][y]	Sub-circuit name of connection boxes that connect the vertical routing tracks to logic blocks. *<x> and <y>* are coordinates of the connection box, following the coordinate system of VPR.
sb[x][y]	Sub-circuit name of switch blocks. *<x> and <y>* are coordinates of the switch block, following the coordinate system of VPR.
<name_spice_models>	The names of all the spice models in architectural description XML.

VII. GUIDELINES ON CREATING CUSTOMIZED SPICE NETLIST

To make sure the customized SPICE netlists can be correctly included in FPGA-SPICE, **the following rules should be fully respected**:

1. The customized SPICE netlists could contain multiple sub-circuits but the names of these sub-circuits should not be conflicted with any reserved words.. Here is an example of defining a sub-circuit in SPICE netlists. The <*subckt_name*> should be a unique one, which should not be conflicted with any reserved words.

   .subckt <subckt_name> <ports>

2. The ports of sub-circuit to be included should strictly follow the sequence:
   <input_ports> <output_ports> <sram_ports> <clock_ports> <vdd> <gnd>
   It is not necessary to keep the names of ports be the same with what is defined in the SPICE models. But the bandwidth of the ports should be consistent with what is defined in the SPICE models.

If the customized SPICE netlists includes inverters, buffers or transmission gates, it recommended to use those auto-generated by FPGA-SPICE. It is also recommend to use the transistor sub-circuit (*vpr_nmos* and *vpr_pmos*) auto-generated by FPGA-SPICE. In the appendix, we introduce how to use these useful sub-circuits.

VIII. HOW TO SIMULATE

1. Hierarchy of SPICE Directory

All the generated SPICE netlists are located in the <*spice_dir*> as you specify in the command-line options.
Under the <*spice_dir*>, FPGA-SPICE creates a number of folders: *include, subckt, lut_tb, dff_tb, grid_tb, pb_mux_tb, cb_mux_tb, sb_mux_tb, top_tb, results*. Under the <*spice_dir*>, FPGA-SPICE also creates a shell script called *run_hspice_sim.sh*, which run all the simulations for all the testbenches.
The folders contain the sub-circuits and testbenches and their contents are shown as following.

Folder	Content
include	The header files which contain the parameters for stimulate and measurement, as defined in <tech_lib>
subckt	Contain all the auto-generated sub-circuits, such as inverters, buffers, transmission gates, multiplexers, LUTs and even logic blocks, connection boxes and switch blocks.
lut_tb	Contain all the testbenches for LUTs. This folder is created only when option *print_spice_lut_testbench* is enabled.
dff_tb	Contain all the testbenches for FFs. This folder is created only when option *print_spice_dff_testbench* is enabled.
grid_tb	Contain all the testbenches for logic blocks (grid-level testbenches). This folder is created only when option *print_spice_grid_testbench* is enabled.
pb_mux_tb	Contain the testbenches for the multiplexers inside logic blocks. This folder is created only when option *print_spice_pbmux_test-bench* is enabled.
cb_mux_tb	Contain all the testbenches for the multiplexers inside connection boxes. This folder is created only when option *print_spice_cbmux_testbench* is enabled.
sb_mux_tb	Contain all the testbenches for the multiplexers inside switch blocks. This folder is created only when option *print_spice_sbmux_test-bench* is enabled.
top_tb	Contain the full-chip-level testbench. This folder is created only when option *print_spice_top_testbench* is enabled.
results	An empty folder when created. It stores all the simulation results by running the shell script *run_hspice_sim.sh*.

2. Simulation results

The HSPICE simulator creates a LIS file (*.lis) to store the results. In each LIS file, you can find the leakage power and dynamic power of each module, as well the total leakage power and the total dynamic power of all the modules in a SPICE netlist.
The following is an example of simulation results of a pb_mux testbench.

total_leakage_srams= -16.4425u
total_dynamic_srams= 83.0480u

total_energy_per_cycle_srams= 269.7773f
total_leakage_power_mux[0to76]=-140.1750u
total_energy_per_cycle_mux[0to76]= -37.5871p
total_leakage_power_pb_mux=-140.1750u
total_energy_per_cycle_pb_mux= -37.5871p

total_energy_per_cycle_srams represents the total energy per cycle of all the SRAMs of the multiplexers in this testbench, while *total_energy_per_cycle_pb_mux* is the total energy per cycle of all the multiplexer structures in this testbench.
Therefore, the total energy per cycle of all the multiplexers in this testbench should be the sum of *total_energy_per_cycle_srams* and *total_energy_per_cycle_pb_mux*.

Similiarly, the total leakage power of all the multiplexers in this testbench should the sum of *total_leakage_srams* and *total_leakage_power_pb_mux*.

The leakage power is measured for the first clock cycle, where FPGA-SPICE set all the voltage stimuli in constant voltage levels.
The total energy per cycle is measured for the rest of clock cycles (the 1st clock cycle is not included). The total power can be calculated by

$$total\_energy\_per\_cycle \cdot clock\_freq ,$$

where *clock_freq* is the clock frequency used in SPICE simulations.

IX. REFERENCE

[1]. J. B. Goeders *et al*., VersaPower: Power Estimation for Diverse FPGA Architectures, IEEE ICFPT, pp. 229 - 234, 2012.
[2]. J. Rose *et al*., The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing, FPGA, 2012, pp. 77-86.
[3]. J. Luu *et al*., Architecture Description and Packing for Logic Blocks with Hierarchy, Modes and Complex Interconnect, FPGA, pp. 227-236, 2011.
[4]. X. Tang *et al*., FPGA-SPICE: A Simulation-based Power Estimation Framework for FPGAs, ICCD 2015.
[5]. X. Tang et al., A High-Performance Low-Power Near-Vt RRAM-based FPGA, FPT, pp. 207-214, 2014.
[6]. V. Betz *et al*., Architecture and CAD for Deep-Submicron FPGAs, Springer Publisher 1999.

X. APPENDIX

1. Useful parameters in auto-generated SPICE Netlists

5.1. Global Parameters

Parameter	Description
nl	Channel length of the NMOS transistor
pl	Channel length of the PMOS transistor
wn	Minimum transistor width of the NMOS transistor
wp	Minimum transistor width of the PMOS transistor
clk_freq	Clock frequency
vsp	Working voltage

5.2. Basic Auto-generated Sub-circuits

In this part, we introduce how to use the sub-circuits auto-generated by FPGA-SPICE.

NMOS transistor
X<your_name> <drain> <gate> <source> <bulk> vpr_nmos L=<float> W=<float>
PMOS transistor
X<your_name> <drain> <gate> <source> <bulk> vpr_pmos L=<float> W=<float>
inverter
X<your_name> <input> <out> <vdd> <gnd> inv size=<float>
buffer
X<your_name> <input> <out> <vdd> <gnd> buf size=<float>
Transmission gate
X<your_name> <input> <out> <gate_nmos> <gate_pmos> <vdd> <gnd> cpt nmos_size=<float> pmos_size=<float>

2. An Example of Architecture XML

```
<architecture>
  <models>
  </models>
  <layout auto="1.0"/>
  <spice_settings>
    <parameters>
      <options sim_temp="25" post="off" captab="off" fast="on"/>
      <measure sim_num_clock_cycle="auto" accuracy="1e-13" accuracy_type="abs">
        <slew>
          <rise upper_thres_pct="0.95" lower_thres_pct="0.05"/>
          <fall upper_thres_pct="0.05" lower_thres_pct="0.95"/>
        </slew>
        <delay>
          <rise input_thres_pct="0.5" output_thres_pct="0.5"/>
          <fall input_thres_pct="0.5" output_thres_pct="0.5"/>
        </delay>
      </measure>
      <stimulate>
        <clock sim_slack="0.2">
```

```
        <rise slew_time="50e-12" slew_type="abs"/>
        <fall slew_time="50e-12" slew_type="abs"/>
      </clock>
      <input>
        <rise slew_time="100e-12" slew_type="abs"/>
        <fall slew_time="100e-12" slew_type="abs"/>
      </input>
    </stimulate>
  </parameters>
  <tech_lib lib_type="academia" transistor_type="TT"
lib_path="/home/xitang/tangxifan-eda-tools/branches/fpga_flow/tech/45nm_HP.pm"
nominal_vdd="1.0"/>
  <transistors pn_ratio="1.4" model_ref="M">
    <nmos model_name="nmos" chan_length="45e-9" min_width="160e-9"/>
    <pmos model_name="pmos" chan_length="45e-9" min_width="160e-9"/>
  </transistors>
  <module_spice_models>
    <spice_model type="chan_wire" name="chan_segment" prefix="track_seg"
is_default="1">
      <design_technology type="cmos"/>
      <input_buffer exist="off" type="inverter" size="4" tapered="off"/>
      <output_buffer exist="off" type="inverter" size="4" tapered="off"/>
      <port type="input" prefix="in" size="1"/>
      <port type="output" prefix="out" size="1"/>
      <wire_param model_type="pie" res_val="0" cap_val="0" level="1"/> <!--
model_type could be T, res_val and cap_val DON'T CARE -->
    </spice_model>
    <spice_model type="wire" name="direct_interc" prefix="direct_interc"
is_default="1">
      <design_technology type="cmos"/>
      <input_buffer exist="off" type="inverter" size="4" tapered="off"/>
      <output_buffer exist="off" type="inverter" size="4" tapered="off"/>
      <port type="input" prefix="in" size="1"/>
      <port type="output" prefix="out" size="1"/>
      <wire_param model_type="pie" res_val="0" cap_val="0" level="1"/> <!--
model_type could be T, res_val cap_val should be defined -->
    </spice_model>
    <spice_model type="mux" name="ble_mux_buffered" prefix="ble_mux_buf"
is_default="1">
      <design_technology type="cmos" structure="tree"/>
      <!--design_technology type="rram" ron="3e3" roff="1e6"
prog_transistor_size="1"/-->
      <input_buffer exist="on" type="inverter" size="1" tapered="off"/>
      <output_buffer exist="on" type="inverter" size="1" tapered="on"
tap_buf_level="3" f_per_stage="4"/>
      <!--mux2to1 subckt_name="mux2to1"/-->
      <pass_gate_logic type="transmission_gate" nmos_size="1" pmos_size="1.4"/>
      <port type="input" prefix="in" size="1"/>
      <port type="output" prefix="out" size="1"/>
      <port type="sram" prefix="sram" size="1"/>
```

```
    </spice_model>
    <spice_model type="mux" name="localrouting_mux_buffered"
prefix="localrouting_mux_buf">
      <design_technology type="cmos" structure="tree"/>
      <!--design_technology type="rram" ron="3e3" roff="1e6"
prog_transistor_size="1"/-->
      <input_buffer exist="on" type="inverter" size="1" tapered="off"/>
      <output_buffer exist="on" type="inverter" size="1" tapered="off"/>
      <!--mux2to1 subckt_name="mux2to1"/-->
      <pass_gate_logic type="transmission_gate" nmos_size="1" pmos_size="1.4"/>
      <port type="input" prefix="in" size="1"/>
      <port type="output" prefix="out" size="1"/>
      <port type="sram" prefix="sram" size="1"/>
    </spice_model>
    <spice_model type="mux" name="cb_mux_buffered" prefix="cbmux_buf">
      <design_technology type="cmos" structure="tree"/>
      <!--design_technology type="rram" ron="3e3" roff="1e6"
prog_transistor_size="1"/-->
      <input_buffer exist="on" type="inverter" size="1" tapered="off"/>
      <output_buffer exist="on" type="inverter" size="1" tapered="on"
tap_buf_level="3" f_per_stage="4"/>
      <!--mux2to1 subckt_name="mux2to1"/-->
      <pass_gate_logic type="transmission_gate" nmos_size="1" pmos_size="1.4"/>
      <port type="input" prefix="in" size="1"/>
      <port type="output" prefix="out" size="1"/>
      <port type="sram" prefix="sram" size="1"/>
    </spice_model>
    <spice_model type="mux" name="sb_mux_buffered" prefix="sbmux_buf">
      <design_technology type="cmos" structure="tree"/>
      <!--design_technology type="rram" ron="1e3" roff="1e6"
prog_transistor_size="3"/-->
      <input_buffer exist="on" type="inverter" size="1" tapered="off"/>
      <output_buffer exist="on" type="inverter" size="1" tapered="on"
tap_buf_level="4" f_per_stage="4"/>
      <pass_gate_logic type="transmission_gate" nmos_size="1" pmos_size="1.4"/>
      <port type="input" prefix="in" size="1"/>
      <port type="output" prefix="out" size="1"/>
      <port type="sram" prefix="sram" size="1"/>
    </spice_model>
    <!--DFF subckt ports should be defined as <D> <Q> <CLK> <RESET> <SET>
-->
    <spice_model type="ff" name="static_dff" prefix="dff"
netlist="/home/xitang/tangxifan-eda-tools/branches/fpga_flow/SpiceSubckts/ff.sp">
      <design_technology type="cmos"/>
      <!--design_technology type="rram" ron="1e3" roff="1e6"
prog_transistor_size="1"/-->
      <input_buffer exist="on" type="inverter" size="1" tapered="off"/>
      <output_buffer exist="on" type="inverter" size="1" tapered="off"/>
      <pass_gate_logic  type="transmission_gate"  nmos_size="1"  pmos_size="1.4"/>
      <port type="input" prefix="D" size="1"/>
```

```
        <port type="input" prefix="Set" size="1"/>
        <port type="input" prefix="Reset" size="1"/>
        <port type="output" prefix="Q" size="1"/>
        <port type="clock" prefix="clk" size="1"/>
      </spice_model>
      <spice_model type="lut" name="lut5" prefix="lut5">
        <design_technology type="cmos"/>
        <!--design_technology type="rram" ron="1e3" prog_transistor_size="1"/-->
        <lut_input_buffer exist="on" type="inverter" size="1" tapered="on"
tap_buf_level="2" f_per_stage="2"/>
        <input_buffer exist="on" type="inverter" size="1" tapered="off"/>
        <output_buffer exist="on" type="inverter" size="1" tapered="off"/>
        <pass_gate_logic type="transmission_gate" nmos_size="1" pmos_size="1.4"/>
        <port type="input" prefix="in" size="5"/>
        <port type="output" prefix="out" size="1"/>
        <port type="sram" prefix="sram" size="32"/>
      </spice_model>
      <spice_model type="lut" name="lut6" prefix="lut6">
        <design_technology type="cmos"/>
        <!--design_technology type="rram" ron="1e3" prog_transistor_size="1"/-->
        <lut_input_buffer exist="on" type="inverter" size="1" tapered="on"
tap_buf_level="2" f_per_stage="2"/>
        <input_buffer exist="on" type="inverter" size="1" tapered="off"/>
        <output_buffer exist="on" type="inverter" size="1" tapered="off"/>
        <pass_gate_logic type="transmission_gate" nmos_size="1" pmos_size="1.4"/>
        <port type="input" prefix="in" size="6"/>
        <port type="output" prefix="out" size="1"/>
        <port type="sram" prefix="sram" size="64"/>
      </spice_model>
      <spice_model type="sram" name="sram6T" prefix="sram"
netlist="/home/xitang/tangxifan-eda-tools/branches/fpga_flow/SpiceSubckts/sram.sp">
        <design_technology type="cmos"/>
        <input_buffer exist="on" type="inverter" size="1" tapered="off"/>
        <output_buffer exist="on" type="inverter" size="1" tapered="off"/>
        <pass_gate_logic type="transmission_gate" nmos_size="1" pmos_size="1.4"/>
        <port type="input" prefix="in" size="1"/>
        <port type="output" prefix="out" size="2"/>
      </spice_model>
      <spice_model type="inpad" name="inpad" prefix="inpad"
netlist="/home/xitang/tangxifan-eda-tools/branches/fpga_flow/SpiceSubckts/io.sp">
        <design_technology type="cmos"/>
        <input_buffer exist="on" type="inverter" size="1" tapered="off"/>
        <output_buffer exist="on" type="inverter" size="1" tapered="off"/>
        <pass_gate_logic type="transmission_gate" nmos_size="1" pmos_size="1.4"/>
        <port type="input" prefix="in" size="1"/>
        <port type="output" prefix="out" size="1"/>
      </spice_model>
      <spice_model type="outpad" name="outpad" prefix="outpad"
netlist="/home/xitang/tangxifan-eda-tools/branches/fpga_flow/SpiceSubckts/io.sp">
        <design_technology type="cmos"/>
```

```
      <input_buffer exist="on" type="inverter" size="1" tapered="off"/>
      <output_buffer exist="on" type="inverter" size="1" tapered="off"/>
      <pass_gate_logic type="transmission_gate" nmos_size="1" pmos_size="1.4"/>
      <port type="input" prefix="in" size="1"/>
      <port type="output" prefix="out" size="1"/>
     </spice_model>
   </module_spice_models>
  </spice_settings>
  <device>
   <sizing R_minW_nmos="8926" R_minW_pmos="16067"
ipin_mux_trans_size="2.4"/>
   <timing C_ipin_cblock="532e-18" T_ipin_cblock="9.4e-11"/>
   <area grid_logic_tile_area="30800.00"/>
   <sram area="8" spice_model_name="sram6T"/>
   <chan_width_distr>
    <io width="1.0"/>
    <x distr="uniform" peak="1.0"/>
    <y distr="uniform" peak="1.0"/>
   </chan_width_distr>
   <switch_block type="wilton" fs="3"/>
  </device>
  <cblocks>
   <switch type="mux" name="cb_mux" R="0" Cin="532e-18" Cout="0"
Tdel="9.4e-11" mux_trans_size="2.4" buf_size="50.4"
spice_model_name="cb_mux_buffered" power_buf_size="21">
   </switch>
  </cblocks>
  <switchlist>
   <switch type="mux" name="0" R="72.5" Cin="532e-18" Cout="3.69e-14"
Tdel="6.88e-11" mux_trans_size="2.4" buf_size="204"
spice_model_name="sb_mux_buffered" power_buf_size="85"/>
  </switchlist>
  <segmentlist>
   <segment freq="1.0" length="4" type="unidir" Rmetal="103.84"
Cmetal="13.80e-15" spice_model_name="chan_segment">
    <mux name="0"/>
    <sb type="pattern">
     1 1 1 1 1
    </sb>
    <cb type="pattern">
     1 1 1 1
    </cb>
   </segment>
  </segmentlist>
  <complexblocklist>
   <pb_type name="io" capacity="8" idle_mode_name="outpad">
    <input name="outpad" num_pins="1"/>
    <output name="inpad" num_pins="1"/>
    <clock name="clock" num_pins="1"/>
```

```
    <!-- IOs can operate as either inputs or outputs.
         Delays below come from Ian Kuon. They are small, so they should be interpreted as
         the delays to and from registers in the I/O (and generally I/Os are registered
         today and that is when you timing analyze them.
         -->
    <mode name="inpad">
      <pb_type name="inpad" blif_model=".input" num_pb="1" spice_model_name="inpad">
        <output name="inpad" num_pins="1"/>
      </pb_type>
      <interconnect>
        <direct name="inpad" input="inpad.inpad" output="io.inpad">
          <delay_constant max="0" in_port="inpad.inpad" out_port="io.inpad"/>
        </direct>
      </interconnect>

    </mode>
    <mode name="outpad">
      <pb_type name="outpad" blif_model=".output" num_pb="1" spice_model_name="outpad">
        <input name="outpad" num_pins="1"/>
      </pb_type>
      <interconnect>
        <direct name="outpad" input="io.outpad" output="outpad.outpad">
          <delay_constant max="0" in_port="io.outpad" out_port="outpad.outpad"/>
        </direct>
      </interconnect>
    </mode>

    <!-- Every input pin is driven by 15% of the tracks in a channel, every output pin is driven by 10% of the tracks in a channel -->
    <fc default_in_type="frac" default_in_val="0.15" default_out_type="frac" default_out_val="0.10"/>

    <!-- IOs go on the periphery of the FPGA, for consistency,
         make it physically equivalent on all sides so that only one definition of I/Os is needed.
         If I do not make a physically equivalent definition, then I need to define 4 different I/Os, one for each side of the FPGA
        -->
    <pinlocations pattern="custom">
      <loc side="left">io.outpad io.inpad io.clock</loc>
      <loc side="top">io.outpad io.inpad io.clock</loc>
      <loc side="right">io.outpad io.inpad io.clock</loc>
      <loc side="bottom">io.outpad io.inpad io.clock</loc>
    </pinlocations>

    <!-- Place I/Os on the sides of the FPGA -->
```

```
    <gridlocations>
      <loc type="perimeter" priority="10"/>
    </gridlocations>

    <power method="ignore"/>
  </pb_type>
  <pb_type name="clb">
    <power method="specify-size"/>
    <input name="I" num_pins="33" equivalent="true">
      <power wire_capacitance="0" buffer_size="60"/>
    </input>
    <output name="O" num_pins="20" equivalent="false">
      <power wire_capacitance="0" buffer_size="85"/>
    </output>
    <clock name="clk" num_pins="1"/>

    <!-- Describe fracturable logic element.
         Each fracturable logic element has a 6-LUT that can alternatively operate as
two 5-LUTs with shared inputs.
         The outputs of the fracturable logic element can be optionally registered
     -->
    <pb_type name="fle" num_pb="10" idle_mode_name="n2_lut5">
      <input name="in" num_pins="6">
        <power wire_capacitance="0" buffer_size="2"/>
      </input>
      <output name="out" num_pins="2">
        <power wire_capacitance="0" buffer_size="1"/>
      </output>
      <clock name="clk" num_pins="1"/>

      <!-- Dual 5-LUT mode definition begin -->
      <mode name="n2_lut5">
        <pb_type name="lut5inter" num_pb="1">
          <input name="in" num_pins="5"/>
          <output name="out" num_pins="2"/>
          <clock name="clk" num_pins="1"/>
          <pb_type name="ble5" num_pb="2">
            <input name="in" num_pins="5"/>
            <output name="out" num_pins="1"/>
            <clock name="clk" num_pins="1"/>

            <!-- Define the LUT -->
            <pb_type name="lut5" blif_model=".names" num_pb="1" class="lut"
spice_model_name="lut5">
              <input name="in" num_pins="5" port_class="lut_in"/>
              <output name="out" num_pins="1" port_class="lut_out"/>
              <!-- LUT timing using delay matrix -->
              <!-- These are the physical delay inputs on a Stratix IV LUT but because
VPR cannot do LUT rebalancing,
```

```
                we instead take the average of these numbers to get more stable
results
                82e-12
                173e-12
                261e-12
                263e-12
                398e-12
                -->
              <delay_matrix type="max" in_port="lut5.in" out_port="lut5.out">
               9.1e-11
               9.1e-11
               9.1e-11
               9.1e-11
               9.1e-11
              </delay_matrix>
              <power method="pin-toggle">
               <port name="in" energy_per_toggle="2.42e-14"/>
               <static_power power_per_instance="6.1e-8"/>
              </power>
             </pb_type>

             <!-- Define the flip-flop -->
             <pb_type name="ff" blif_model=".latch" num_pb="1" class="flipflop"
spice_model_name="static_dff">
              <input name="D" num_pins="1" port_class="D"/>
              <output name="Q" num_pins="1" port_class="Q"/>
              <clock name="clk" num_pins="1" port_class="clock"/>
              <T_setup value="4.68e-11" port="ff.D" clock="clk"/>
              <T_clock_to_Q max="2.7e-11" port="ff.Q" clock="clk"/>
              <power method="pin-toggle">
               <port name="Q" energy_per_toggle="5.6e-15"/>
               <static_power power_per_instance="2.95e-8"/>
              </power>
             </pb_type>

             <interconnect>
              <direct name="direct1" input="ble5.in[4:0]" output="lut5[0:0].in[4:0]"/>
              <direct name="direct2" input="lut5[0:0].out" output="ff[0:0].D">
               <!-- Advanced user option that tells CAD tool to find LUT+FF pairs in
netlist -->
               <pack_pattern name="ble5" in_port="lut5[0:0].out"
out_port="ff[0:0].D"/>
              </direct>
              <direct name="direct3" input="ble5.clk" output="ff[0:0].clk"/>
              <mux name="mux1" input="ff[0:0].Q lut5.out[0:0]"
output="ble5.out[0:0]" spice_model_name="ble_mux_buffered">
               <!-- LUT to output is faster than FF to output on a Stratix IV -->
               <delay_constant max="4.65e-11" in_port="lut5.out[0:0]"
out_port="ble5.out[0:0]" />
```

```
            <delay_constant max="4.65e-11" in_port="ff[0:0].Q"
out_port="ble5.out[0:0]" />
          </mux>
        </interconnect>
      </pb_type>
      <interconnect>
        <direct name="direct1" input="lut5inter.in" output="ble5[0:0].in"/>
        <direct name="direct2" input="lut5inter.in" output="ble5[1:1].in"/>
        <direct name="direct3" input="ble5[1:0].out" output="lut5inter.out"/>
        <complete name="complete1" input="lut5inter.clk" output="ble5[1:0].clk"/>
      </interconnect>
    </pb_type>
    <interconnect>
      <direct name="direct1" input="fle.in[4:0]" output="lut5inter.in"/>
      <direct name="direct2" input="lut5inter.out" output="fle.out"/>
      <direct name="direct3" input="fle.clk" output="lut5inter.clk"/>
    </interconnect>
  </mode>
  <!-- Dual 5-LUT mode definition end -->
  <!-- 6-LUT mode definition begin -->
  <mode name="n1_lut6">
    <!-- Define 6-LUT mode -->
    <pb_type name="ble6" num_pb="1">
      <input name="in" num_pins="6"/>
      <output name="out" num_pins="1"/>
      <clock name="clk" num_pins="1"/>

      <!-- Define LUT -->
      <pb_type name="lut6" blif_model=".names" num_pb="1" class="lut"
spice_model_name="lut6">
        <input name="in" num_pins="6" port_class="lut_in"/>
        <output name="out" num_pins="1" port_class="lut_out"/>
        <!-- LUT timing using delay matrix -->
        <!-- These are the physical delay inputs on a Stratix IV LUT but because
VPR cannot do LUT rebalancing,
             we instead take the average of these numbers to get more stable results
          82e-12
          173e-12
          261e-12
          263e-12
          398e-12
          397e-12
          -->
        <delay_matrix type="max" in_port="lut6.in" out_port="lut6.out">
          1.4e-10
          1.4e-10
          1.4e-10
          1.4e-10
          1.4e-10
          1.4e-10
```

```
          </delay_matrix>
          <power method="pin-toggle">
            <port name="in" energy_per_toggle="4.64e-14"/>
            <static_power power_per_instance="7.85e-8"/>
          </power>
        </pb_type>

        <!-- Define flip-flop -->
        <pb_type name="ff" blif_model=".latch" num_pb="1" class="flipflop"
spice_model_name="static_dff">
          <input name="D" num_pins="1" port_class="D"/>
          <output name="Q" num_pins="1" port_class="Q"/>
          <clock name="clk" num_pins="1" port_class="clock"/>
          <T_setup value="4.68e-11" port="ff.D" clock="clk"/>
          <T_clock_to_Q max="2.7e-11" port="ff.Q" clock="clk"/>
          <power method="pin-toggle">
            <port name="Q" energy_per_toggle="5.6e-15"/>
            <static_power power_per_instance="2.95e-8"/>
          </power>
        </pb_type>

        <interconnect>
          <direct name="direct1" input="ble6.in" output="lut6[0:0].in"/>
          <direct name="direct2" input="lut6.out" output="ff.D">
            <!-- Advanced user option that tells CAD tool to find LUT+FF pairs in
netlist -->
            <pack_pattern name="ble6" in_port="lut6.out" out_port="ff.D"/>
          </direct>
          <direct name="direct3" input="ble6.clk" output="ff.clk"/>
          <mux name="mux1" input="ff.Q lut6.out" output="ble6.out"
spice_model_name="ble_mux_buffered">
            <!-- LUT to output is faster than FF to output on a Stratix IV -->
            <delay_constant max="4.65e-11" in_port="lut6.out" out_port="ble6.out" />
            <delay_constant max="4.65e-11" in_port="ff.Q" out_port="ble6.out" />
          </mux>
        </interconnect>
      </pb_type>
      <interconnect>
        <direct name="direct1" input="fle.in" output="ble6.in"/>
        <direct name="direct2" input="ble6.out" output="fle.out[0:0]"/>
        <direct name="direct3" input="fle.clk" output="ble6.clk"/>
      </interconnect>
    </mode>
    <!-- 6-LUT mode definition end -->
  </pb_type>
  <interconnect>
    <!-- We use a full crossbar to get logical equivalence at inputs of CLB
                The delays below come from Stratix IV. the delay through a
connection block
```

```
                input mux + the crossbar in Stratix IV is 167 ps. We already have a
72 ps
                delay on the connection block input mux (modeled by Ian Kuon), so
the remaining
                delay within the crossbar is 95 ps.
                The delays of cluster feedbacks in Stratix IV is 100 ps, when driven
by a LUT.
                Since all our outputs LUT outputs go to a BLE output, and have a
delay of
                25 ps to do so, we subtract 25 ps from the 100 ps delay of a
feedback
                to get the part that should be marked on the crossbar.      -->
      <complete name="crossbar" input="clb.I fle[9:0].out" output="fle[9:0].in"
spice_model_name="localrouting_mux_buffered">
        <delay_constant max="1.05e-10" in_port="clb.I" out_port="fle[9:0].in" />
        <delay_constant max="1.05e-10" in_port="fle[9:0].out"
out_port="fle[9:0].in" />
      </complete>
      <complete name="clks" input="clb.clk" output="fle[9:0].clk">
      </complete>

      <!-- This way of specifying direct connection to clb outputs is important because
this architecture uses automatic spreading of opins.
           By grouping to output pins in this fashion, if a logic block is completely
filled by 6-LUTs,
           then the outputs those 6-LUTs take get evenly distributed across all four
sides of the CLB instead of clumped on two sides (which is what happens with a more
           naive specification).
        -->
      <direct name="clbouts1" input="fle[9:0].out[0:0]" output="clb.O[9:0]"/>
      <direct name="clbouts2" input="fle[9:0].out[1:1]" output="clb.O[19:10]"/>
    </interconnect>

    <fc default_in_type="frac" default_in_val="0.15" default_out_type="frac"
default_out_val="0.10"/>
    <pinlocations pattern="spread"/>
    <gridlocations>
      <loc type="fill" priority="1"/>
    </gridlocations>
  </pb_type>
 </complexblocklist>
 <power>
  <local_interconnect C_wire="0"/>
  <buffers logical_effort_factor="4"/>
  <mux_transistor_size mux_transistor_size="2.4"/>
 </power>
 <clocks>
  <clock buffer_size="0" C_wire="0"/>
 </clocks>
</architecture>
```

3. Coordinator System