Block Level Training
Characterization Concepts

Digital CAD

Introduction to AccuCore
Introduction to AccuCore

- High Performance SoC Timing Solution
- AccuCore Inputs and Outputs
- AccuCore Flow
  - Partitioning
  - Function Extraction
  - Vector Generation
  - Simulation and Characterization
  - Model Generation
- Setup and Debug of AccuCore Flow
High Performance SoC Timing Solution

AccuCore STA
Full-Chip
Static Timing Analysis

AccuCore
Block/Core Characterization and Static Timing Analysis

AccuCell
Cell Characterization
Silvaco supports both a top-down and bottom-up timing strategy to ensure accurate results and optimize timing closure.
**Digital CAD**

**In-Circuit Characterization Engine**

**Design Performance:**
- Highest accuracy
- Complete modeling

**Tool Performance:**
- Fastest Characterization
- Incremental Analysis

**Engineering Performance:**
- Automation
- Ease of use

**Static Timing Analysis**

- Timing Model Generation

- Automatic Circuit Partitioning
- Circuit Functional Extraction
- Automatic Vector Generation
- Ultra-Fast / Accurate Spice

- Functional Model Generation

- Power Characterization
AccuCell and AccuCore Characterization Flows

**AccuCell**

- Cell
  - Functional Extraction
  - Vector Generation
  - Dynamic Simulation
  - Model Generation

**Block/Core**

- Block/Core Partitioning
  - Characterization
  - Static Timing Analysis
  - Model Generation

**AccuCore**

- Model Generation

**Timing**

- Power Function
Silvaco’s characterization technology scales-up from standard cells to custom blocks to IP cores for use in Full-Chip STA.
Digital Circuit Verification Methods

- Non-Hierarchical True SPICE Simulation
  - Highest accuracy, but with limited capacity and too slow
  - Requires large manual vector generation effort and will not achieve 100% coverage

- Gate Level Simulation with SDF Back Annotation
  - Requires expensive dedicated emulation/acceleration hardware to simulate in reasonable timeframes
  - Some capacity limitations and reduced accuracy
  - Requires large manual vector generation effort and will not achieve 100% coverage
  - Requires detailed exhaustive library characterization effort

- Verilog RTL Simulation with Formal Verification and Static Timing Analysis (STA)
  - Fast functional verification with near 100% coverage
  - Dependent on some manual manual vector generation and has reduced timing accuracy for advanced designs and advanced processes
  - Limitations in design style for high performance (dynamic logic, asynchronous)
  - Requires detailed exhaustive library characterization effort

- Switch Level/Fast SPICE Simulation
  - Requires large manual vector generation effort and will not achieve 100% coverage
  - Adequate capacity but reduced accuracy
Register Transfer Level (RTL) is a Hardware Design Language (HDL) that specifies the functional characteristics of a design.

It is a simple design style that simulates fast.
- must follow a set of architecture rules that can limit performance (speed and power).
- requires excess logic and/or area to perform a given function.
  - as such designers turn to more advanced transistor level methods.

Such methods however require problematic and lengthy SPICE ONLY simulation.

AccuCore is ideally suited to characterize and analyze these advanced transistor level circuits.
- Easily
- quickly
- accurately
- with automation.
Verilog RTL Simulation with Formal Verification and Static Timing Analysis (STA) (2 of 3)

- Formal verification checks that a gate/cell level design matches the RTL design
- The RTL is typically logically synthesized into this gate/cell level design
- A regression simulation is a lengthy vector set based, gate level dynamic simulation with all timing characteristics
- Static Timing Analysis (STA) is a vector-less method of quickly and exhaustively verifying circuit timing
- Both timing paths and characteristics of a digital circuit undergo a pass / fail check
- Additionally, this check provides quantitative feedback of the degree of timing margin against a set of specified constraints and construction rules
- For the STA method to be valid, the circuit must utilize specific design styles which follow the required rules
- The STA, together with a matching RTL simulation guarantees that the circuit behavior is identical to the result of a regression simulation
In a synchronous digital system, data is supposed to move in lockstep, advancing one stage on each tick of the clock signal. This is enforced by synchronizing elements such as flip-flops or latches, which copy their input to their output when instructed to do so by the clock. On first order, only two kinds of timing errors are possible in such a system:

- A **hold time** violation, when a signal arrives too early, and advances one clock cycle before it should.
- A **setup time** violation, when a signal arrives too late, and misses the time when it should advance.
AccuCore - In-Circuit Block Level Characterization with Static Timing Analysis

- Limitations of Traditional Gate Level Cell Library-based STA
  - Long characterization time
    - ALL possible cells must be characterized under ALL possible output loads and ALL possible input slew conditions
    - Factor of N accuracy increase requires N^2 simulation time increase
  - Accuracy losses
    - Table interpolation errors
    - Real-world non-linear effects NOT modeled (PWL vs. actual stimulus, Miller Effects)
    - Elmore RC delay model limitations
    - Ignores actual chip routing dependent RC effects
    - Ignores Well Proximity Effects (WPE)
    - Ignores chip level IR drop effects

- Benefits of In-Circuit Block Level Characterization and STA
  - Uses actual circuit loads, actual circuit slews and actual circuit RCs for both signal and supplies
  - Rapid and accurate automated characterization with ALL in-circuit effects
AccuCore Inputs

- Gate-level Verilog netlist can include SDF with parasitic RC information for back-annotation
- SPICE netlist can include RCs or can be merged with SDF
- SPICE models – any public HSPICE or SPECTRE model
- Configuration File – to set all run-time options
- Tcl Script – to start AccuCore or customize flow or reports
AccuCore Outputs

- Synopsys Liberty™ (.lib)
- Verilog connectivity netlist file is created for the cells in the design
- SPICE deck of critical path for clock tree detailed SPICE analysis
- STA Path reports include path analysis (slack) and constraint check analysis
AccuCore Bidirectional Interfaces

- AccuCore uses SmartSpice through an API for fast characterization
- AccuCore supports external SPICE simulators including SmartSpice, HSPICE, SPECTRE and ELDO
- AccuCore Uses an SQL driven relational database to store incremental characterization results
AccuCore Flow - Read Input Files

- AccuCore operates on the following set of input files:
- SPICE netlist file for the block to be characterized
- SPICE model file
- Configuration file
- TCL script file
AccuCore Flow - Read Input Files (cont’d)

- SPICE netlist file defines the connectivity device size type and RC parasitics
- Optional DSPF file provides information on parasitics
- SPICE model file
- TCL file is used to control the execution of AccuCore
- Configuration file is used to define control parameters for the design characterization and point to other data that elements in the characterization process
  - File names
  - Pin information
  - Circuit simulator type
  - Library model type
  - Transistor data
  - Process data
  - There should be a configuration file for each block
AccuCore Flow - Partitioning

- AccuCore begins partitioning after reading input files
- AccuCore divides the design into cells or smaller transistor groups to facilitate more efficient simulation and timing analysis
AccuCore Flow – Function Generation

- AccuCore derives the logic functions directly from the SPICE description.
- Once a function is identified, AccuCore automatically generates optimized vectors for SPICE simulation of the netlist to reduce runtime.
AccuCore Flow – Characterization

- During characterization, the process files, partitioned netlists, optimized vectors and timing measurement requirements are passed to the SPICE simulator specified in the configuration file, such as SmartSpice, to perform dynamic simulation and return measured result values.
- The key to AccuCore's timing accuracy is:
  - All results are generated by a full SPICE analysis of each and every part of the block by “in-circuit” means.
  - Actual circuit transistors, NOT an approximated model.
**AccuCore Flow – Modeling**

- In the modeling step, AccuCore generates the data required for the requested models.
- All the measured timing data created during characterization is stored in an all paths internal model using the Silvaco generic model format.
- The output files are created using the specified model types and formats.
AccuCore Flow – Static Timing Analysis

- Static Timing Analysis (STA) uses all of the characterization information to analyze the design
- Libraries are used in conjunction with a structural Verilog netlist to establish connectivity and timing of the design
- Paths are traced and compared with expected arrival times
Introduction to AccuCore

Partitioning

- Partitioning reduces the size of a circuit by separating it into cells
- This allows function extraction, vector generation and other phases of characterization to be performed on smaller, more manageable portions of the circuit
- It also preserves design integrity and enables more focused debugging since the software is working with smaller segments, rather than an entire circuit
- Although partitioning is automatic, manual overrides may be used when circumstances require that portions of a design remain intact
Introduction to AccuCore

Partitioning (cont’d)

- Automatic Partitioning Process
- Manual Partitioning
  - Hierarchy
  - Pattern Matching
  - BLACK_BOX and CUTBOX
- Partitioning Process
  - Cell formation
  - Levelizing
  - Complex and Large cell management
  - Flip-flops and Latches
  - Large Multiplexers
  - Small Multiplexers with Common Select Signals
  - Barrel Shifters
Automatic Partitioning Process

- Manually-defined partitions are also identified and their transistors marked so that they are not included in other partitions
- Flattening the SPICE netlist to rename ill-defined boundaries that could impair partitioning
- Channel connected components (CCC)s, collections of channel connected transistors, are identified and grouped together
- This is a critical step because at this point, the CCCs may be determined to be a cell and if so, partitioning is complete
- If not, the CCC is further evaluated to determine if the electrical relationships are strong enough to join it with other CCCs to form one single larger cell
- After each run, individual partitions may be reviewed and debugged in one of two ways:
  - As a file, ending with the suffix .out by default, showing each cell’s netlist at transistor-level; or
  - As a statistical representation in the log or .sum file
- If a partition is very large or if repartitioning is required to ensure circuit functionality, then manual partitioning should be used
- Leveling orders the sequence of the cells during the characterization process
Manual Partitioning - Hierarchy

- Circuits such as glitch latches and devices with three state points may require manual partitioning.
- This may be done by using hierarchy or pattern matching
  - Hierarchical structures are crucial in the creation of sophisticated circuit designs
    - When the netlist is flattened, these relationships are also flattened and the definitions of all subcircuits are lost
    - Subcircuits can be retained with the KEEP_SUBCKT command which is very important when related structures must be kept intact in order to preserve the design
Manual Partitioning – Pattern Matching

- A partition may need to transcend the established hierarchy, or it may require further separation into multiple cells
- For these devices, pattern matching is first used to find all occurrences of a subcircuit within the flattened netlist
- Then, a manual partition is created by grouping all identified devices together which establishes a new level of hierarchy that supersedes the original structure
- All patterns should be provided in the form of a SPICE netlist with the gate of the transistor identified as an input, whenever possible
- It’s important to consider that patterns are matched in the order specified by the configuration file
- Patterns should be chosen carefully because if the inputs are transistor source/drains, inaccurate matching will inadvertently occur
  - For example, specifying a 2-input NMOS mux will lead to inaccurate matching because the patterns will match part of the pulldown tree of an AOI gate as shown at left
Manual Partitioning – BLACK_BOX and CUTBOX

- Analog circuits, sense-amps or any other devices that are not digital in nature must be manually partitioned and removed since downstream characterization processes cannot analyze them.

- The BLACK_BOX command removes a subcircuit from the design while maintaining the input capacitance and connectivity in the overall circuit.
  - The driver to the BLACK_BOXed circuit will still ‘see’ the circuit, but it will not be included in the analysis.
  - If there is no hierarchy defined through subcircuit commands in the SPICE netlist, then FIND_BLACKBOX can be used in conjunction with a SPICE netlist of the desired partition.

- The CUTBOX command completely removes a structure from the circuit and is used for circuits such as memories that are characterized separately and there is no impact from removing them.
Partitioning Process – Cell Function

- AccuCore forms cells during the partitioning process
  - A DC is a set of one or more CCCs that functionally or electrically interact with one another
- Cells are created by identifying the state-point (a loop that’s either a keeper or feedback device) and verifying that all of the following conditions exist:
  - The number of state points within the group of CCCs does not exceed 3, the default
  - Both logical ‘1’s and ‘0’s can be overwritten
  - There is an explicit or implied clock controlling the partitions
- If any of the conditions above are not met, the CCCs remain independent and a cell is formed. This is the basis for a sequential device
- Cells are then collapsed into a single partition to insure accuracy during the function extraction phase of characterization
Partitioning Process – Levelizing

- Each partition is dependent on its predecessor for slope and its successor for loading.
- The order in which the cells are characterized is extremely important.
- As the final step in partitioning the cells are levelized (ordered) starting from the primary inputs to the outputs of the design.
- This leveling process enables AccuCore to simulate a cell with the actual slew for each input and actual load for each output with the actual RC parasitics of that cell.
Partitioning Process – Complex and Large Cells

- Certain circuits with transmission gates (pass transistors) may create complex or very large cells during partitioning.
- If not managed, these circuits can increase runtime and negatively impact the efficiency and accuracy of subsequent characterization processes.
- Each circuit type will subsequently be addressed:
  - Flip-flops and Latches
  - Large Multiplexers
  - Small Multiplexers with Common Select Signals
  - Barrel Shifters
Partitioning Process – Flip-flops and Latches

- Flip flops and latches must be grouped as a single cell because of the relationship between clocking and data.
- Naturally, this may result in a complex cell simply because of the grouping as shown at right.
- Most latches and flip-flops are partitioned automatically, but some may require user intervention. The FIND_SUBCKT and KEEP_SUBCKT commands may be needed to identify and retain the intended hierarchies and prevent the cells from becoming too large.
- The process begins by identifying the CCC as a state point and then conducting a search from that point to the surrounding CCCs to determine if that state point could also be a latch. If so, the cell is formed and the process continues to determine if the latch is actually a flip-flop.

Partitioning of a Latch
In the process of partitioning, all transistors in the multiplexer (mux) along with all of the logic structures driving the inputs are grouped together into one large cell. As the number of inputs increases, the equation for the cell increases further in size and complexity which in turn significantly impacts vector generation.

- You can manage the complexity of large muxes with the `ONE_HOT` or `ONE_COLD` commands.
- These commands reduce the vector set for characterization and set each of the select pins in a mutually exclusive state.
- If these commands are not used, the mux can go into an unknown state at the output.
- You can also use `FIND_SUBCKT` and `KEEP_SUBCKT` which serve to identify and then retain the subcircuit hierarchy, just as with flip-flops and latches.
- Finally, a table or an equation may also be used to further describe the function of the circuit, particularly if vector generation is causing a long runtime.
Partitioning Process – Small Multiplexers with Common Select Signals

- When a group of small multiplexers such as those shown at left use common select signals, the result is a complex cell which may also be very large.

- Fortunately, the ONE_HOT or ONE_COLD commands work automatically on most of these circuits, but for those where function extraction fails, the commands may be applied directly to specific signals.

- You can also use BDD_ORDER_SIZE to streamline the function extraction and vector generation processes by applying an additional algorithm.

- The FIND_SUBCKT and KEEP_SUBCKT commands can also be used to retain hierarchies and restrict the size of the cell.
Barrel shifters are frequently used to implement designs with arithmetic shifting, logical shifting and rotations.

Their cells are often complex and potentially large if the circuit includes transmission gates and single-pass transistors.

The single most common problem with these circuits is large CCCs that in turn form one very large cell.

The MAX_DCDEPTH command restricts analysis to the real paths by setting a limit on the transistor path tracing for vector generation and instructs the tool to stop after tracing at most MAX_DCDEPTH <# of transistors>
Function Extraction

- Automatic Function Extraction
  - Printing Generated Equations
  - Functional Equation Syntax
  - Cell Classifications
- Manual Function Extraction
  - Combinational Devices
  - Examples of Combinational Devices
  - Sequential Devices
- Controlling Function Extraction
  - Restricting Inflation of BDDs
  - Sneak Path
Automatic Function Extraction

- AccuCore uses a unique algorithmic approach to produce accurate and comprehensive circuit function equations.
- AccuCore analyzes all partitions in a design including those for complex devices such as flip-flops, latches, and large footless and footed dominoes.
- The process of function extraction begins by identifying weak and strong static drivers through a fast duality check between pull-up and pull-down paths for each node to correctly identify signal flow direction.
- Four equations are created for each output and internal state of a cell where the output is the name of the output pin as follows:
  - output.0 when output will equal zero
  - output.1 when output equals one
  - output.z when output will be left undriven
  - output.x when output will be unknown
- These four equations for each output are then analyzed to determine the classification for each cell.
Function Extraction - Printing Generated Equations

- It is critical that the correct function be identified for each cell before proceeding to the next step
- Print the equations to ensure expectations are met and for potential debugging
- All equations are written to the log file when the PRINT_EQNS command is enabled with a value of ‘1’ in the configuration file

Functional equation syntax

```plaintext
output_name.value := Input_name (operation)
Input_name;

OUTPUT.0 := INPUT_1 & INPUT_2;
OUTPUT.1 := INPUT_1 | INPUT_2;
```
Function Extraction - Functional Equation Syntax

- Whether created automatically or manually, all equations use a specified syntax, including operators with parentheses, to determine the order of processing.
- For states 0 and 1, equations are expressed in terms of the primary inputs to the cell, possibly other states such as that encountered with flip-flops, and the previous value of the state, denoted as the name of the state prefixed with a ‘-’ sign.
- The logical operators used to represent a function are as follows:
  - := Assignment
  - ! Negation
  - | Logical OR (union of terms)
  - & Logical AND
  - // Comments
  - ~ Invert
  - + Rising edge
  - - Falling edge
Function Extraction – Cell Classifications

- Cells are classified to determine the characterization routine to be used.
- Cells are grouped either combinational or sequential:
  - Combinational cells do not require clocks for their normal operation so determining their function is a straightforward process.
  - Sequential cells contain user-defined clocks and require a more exhaustive check to ensure the proper functional description is derived.
- Each of the classifications is further divided into specific categories:
  - Combinational devices are broken down into static devices, muxes, and tristates.
  - Sequential devices are categorized into latches, flip-flops, and footed and footless dominos.
Manual Function Extraction

- Some cells cannot be processed automatically
- AccuCore accepts manual overrides in the form of a functional equation or simulation vector for characterization
- Logical operators with parentheses must be used to specify the order of operation
- In all cases, manually written equations take precedence over those generated automatically
- Using a simple NOR gate as an example, a ‘0’ value may be achieved if either A or B are ‘1’. Conversely, to drive the output to a “1” value, the inputs A and B must both be ‘0’

\[
\text{out.0} = a \lor b; \\
\text{out.1} = \neg a \land \neg b;
\]

NOR gates with inputs A and B and equation
Introduction to AccuCore

Manual Function Extraction – Combinational Devices

- Transition Low
  - For a two-input NAND gate with two inputs (a, b) and one output (y), a completely specified vector that causes the output to go low is (1, 1). The equation is:
    - y.0 := a & b;
  - If there are additional zero vectors, append them to the equation list using "|"
  - For an AOI gate that implements the function \( y = a \& b \mid c \), the equation for three zero vectors looks like:
    - y.0 := (~a & b & ~c)|(a & ~b & ~c)|(~a & ~b & ~c);
  - There is no limit to the number of zero vectors the tool can consider because it selects the worst-case and best-case scenarios from the user-defined equation set
Manual Function Extraction – Combinational Devices (cont’d)

- Transition High

  For transition high, specify as many one vectors as desired. For the two-input NAND gate, these vectors would be:
  - y.1 := (~a & b) | (a & ~b) | (~a & ~b);

  In order to circumvent the creation of delays for a specific input pin, specify a constant, fixed value in both the zero and one vectors. For example, if pin “a” of a NAND2 should not be characterized, specify:
  - y.0 := a & b;
  - y.1 := a & ~b;

  In this case, ‘a’ does not switch when going from y.0 to y.1 or from y.1 to y.0
Manual Function Extraction – Examples of Combinational Devices

Tristate BUFFER & equation

Multiplexer & Equation

\[
\begin{align*}
\text{q.0} &= \text{en} \& \text{d}; \\
\text{q.1} &= \text{en} \& \text{d}; \\
\text{q.z} &= \text{!en}; \\
\text{out.0} &= (!\text{sel1} \& \text{sel0} \& \text{!in0}) \lor (\text{sel1} \& \text{!in1} \& \text{!sel0}) \\
& \quad \lor (\text{!in1} \& \text{sel0} \& \text{!in0}); \quad \text{// out is 0} \\
\text{out.1} &= (\text{in1} \& \text{sel0} \& \text{in0}) \lor (\text{!sel1} \& \text{sel0} \& \text{in0}) \\
& \quad \lor (\text{sel1} \& \text{!in1} \& \text{!sel0}); \quad \text{// out is 1} \\
\text{out.z} &= (!\text{sel1} \& \text{!sel0}); \quad \text{// out is floating} \\
\text{out.x} &= (\text{sel1} \& \text{!in1} \& \text{sel0} \& \text{!in0}) \lor (\text{sel1} \& \text{!in1} \& \text{sel0} \& \text{in0}); \quad \text{// out is unknown}
\end{align*}
\]
Manual Function Extraction – Sequential Cells

- For sequential cells, the CLOCKS command must be included in the configuration file.
- Cells must then be grouped by domain and within a domain, identified by phases.
- Set and Reset Lines
  - Flip-flops with asynchronous set and clear lines can also be specified using equations.
  - Consider a flip-flop with an active high signal set and active high signal clear. Asserting set causes the flop to produce a ‘1’ on output, but asserting clr causes it to produce a ‘0’.
  - In the example below, d and the clocks are irrelevant for the set and clr terms.

\[
\begin{align*}
q_0 & := \neg d \land \text{main}^+ \land \neg \text{set} \land \neg \text{clr} \lor \text{clr;} \\
q_1 & := d \land \text{main}^+ \land \neg \text{set} \land (\text{clr} \lor \text{set} \land \neg \text{clr});
\end{align*}
\]

Flip-flop with asynchronous set and clear and equation.
Controlling Function Extraction - Restricting Inflation of BDDs

- The most accurate way to characterize custom CMOS circuits that contain large pass transistor structures such as barrel shifters, cross-bar switching, and m-of-n logic trees is to keep them intact as a single cell.
- To limit the inflation of BDDs, insert the command `BDD_ORDER_SIZE` in the configuration file and specify a value that is less than the number of inputs plus clocks of the DCs with long runtimes.
- The command `BDD_ORDER_SIZE` is not recommended for DCs with fewer than 15 inputs as it may increase runtimes without adding any real benefit.
- If the specified value is higher than what is actually required, dynamic reordering will occur and the ability to specify `ONE_HOT` and `ONE_COLD` will be lost.
Managing Complex Large Cells with ONE_HOT and ONE_COLD

- Can be applied to both internal and primary inputs
- Reduce the vector set by specifying mutually exclusive states
- Apply directly to signals to manually override auto function extract results
- ONE_HOT (1 active high per group)
- ONE_COLD (1 active low per group)
Controlling Function Extraction – Sneak Path

- The command `MAX_STRONG_PATHS_RATIO` enables some control on finding strong static drivers while simultaneously avoiding potential sneak paths, both of which ultimately effect the final function extraction results.
- The default value for this command is 15, the number should be reduced until the correct functions are produced.

![Sneak path in three-to-one mux](image-url)
Vector Generation

- Automatic vector and measurement generation
- Manual vector generation
- VECTOR_BUFFER and AUTO_VECTOR_BUFFER
Manual Vector Generation with Auto Measurement

- AccuCore also accepts manually generated vectors
  - Manual vectors override the auto generated vectors
  - Vectors created manually are placed in a table file and are referenced in the configuration file
  - The manual approach may be used to achieve a specific measurement
  - Measurements will be automatically generated based on characteristics options of vector category controls for manual specified supplemented or replacement vectors
Automatic Vector Measurement and Generation

- The vectors and measurements needed to simulate and characterize the cell are defined by the functional equations and characterization options.
- The vector and measurement types include controls to transition the output net to measure delay to calculate setup and hold, and/or to identify other characterization values.
- The vector set is optimized to remove redundancy and condensed into a single PWL or series of PWL.
- The measurement statements are added to the SPICE file to capture and communicate characterization from the simulation results.
A table file can be used to manually specify the input vectors as an alternative to automatic vector generation.

The AccuCore Users Manual provides the complete syntax, semantics and many examples to enable you to create table files or to understand the effects of automatically generated vectors.

- A table file consists of a sequence of vectors that establish a required transition at the output.
- The final two vectors should only allow a single input value to change.
- If more than two input values change, the output measurement cannot be associated with one unique pin.

```plaintext
table DPPHQQX9
  inputs : D, RN : data ;
  inputs : CK : clock ;
  outputs : Q : flipflop ;
  property : name=Q , pos=RN, non=CK, clear_low=RN, store=D C r ;
  cap :  
    arc : D , CK ;
    [010;011;111;111] : r ;
    arc : RN ;
    [110;111;101] : f ;
  delay :  
    [110;111;011;011] : f ;
    [010;011;111;111] : r ;
    [110;111;101] : f ;
  set_hold :  
    [010;011;011;111] : r ;
    [110;111;101] : f ;
  res_rst :  
    [100;111] : f ;
  mpu :  
    [010;011;010;111] : r ;
    [110;111;111] : f ;
  max_cap :  
    [110;111;010;011] : f ;
    [010;011;010;111] : r ;
    [110;111;111] : f ;
  hidden_power :  
    [110;111;010;011] : - ;
    [010;011;010;111] : - ;
end_table
```

Table File Example
VECTOR_BUFFER and AUTOVECTOR_BUFFER

- For each measurement in the SPICE analysis, the transition time should be within a known range.
- If the transition exceeds the default transition ranges, also known as a Vector Buffer, then the measurements will be inaccurate and require recalculation.
- The default value for this parameter is 4 ns, but it may be modified with the VECTOR_BUFFER command.

Example of PWL and vector buffer
Introduction to AccuCore

Simulation and Characterization

- Specifying Simulator Options
  - SPICE Simulator Type
  - Manipulating Simulations
  - Adjusting the Simulation Corner
  - Specifying Rise and Fall Slew Rates
  - Specifying Slope Tables
  - Measuring Slew Rate
  - Slope Thresholds
  - Reporting Large Slew Rates
  - Measuring Transition Delays
  - Specifying Capacitive Load
  - Measuring Input Capacitance
- Integration Method
  - Measuring Setup and Hold Times
  - Tristate Modeling
  - Conductance Method
  - Current Method
  - Controlling Simulations
  - Modeling Without Simulations
  - Modeling Without Setup and Hold
  - Limiting Transient Analysis Steps
  - Limiting Simulation Time for Many Inputs
  - Simulating Only Specified Cells
Simulation and Characterization (cont’d)

- After function extraction AccuCore uses dynamic transistor simulation to characterize delay, slew rate, setup time, hold time, and primary input capacitance over a range of input slew rates and output load values.
- AccuCore automatically generates stimulus data in the form of piecewise linear waveforms (PWLs), and adds the PWLs to the SPICE deck of a cell.
- After simulating a design successfully, AccuCore gathers the results and generates the specified models.
Specifying Simulator Options

- AccuCore supports the following simulators for design characterization:
  - SMARTSPICE
  - HSPICE™
  - Spectre™
  - ELDO™
Specifying Simulator Options (cont’d)

- AccuCore uses the configuration file to choose the simulator to use and each simulator supports its own options
  - **SPICE_TYPE**: Selects simulator (smartspice, hspice, eldo, spectre)
  - **SMARTSPICE_OPTIONS**: A list of options for SMARTSPICE simulations
  - **HSPICE_OPTIONS**: A list of options for HSPICE simulations
  - **ELDO_OPTIONS**: A list of options for ELDO simulations
  - **SUPPLY_V_HIGH**: Specifies the maximum voltage any signal can reach during simulation
  - **TEMP**: Sets the temperature used in simulation
  - **LIB_CMD**: Used when SPICE simulation models are stored in a library
  - **SCALE_FACTOR**: Sets the transistor’s scaling factor if not set directly in the SPICE netlist
  - **INC_CMD**: Used when SPICE process models are added via an include file
Introduction to AccuCore

The options set via SMARTSPICE_OPTIONS appear as option constructs in the SPICE deck
- The post=2 and probe options were added automatically
- The .temp variable was set as a result of the TEMP command in the configuration file
- AccuCore renames each instance and net because some simulators restrict the length of instance and net names (cross referenced as comments at the head of the SPICE deck)

Example SPICE deck:

```
INPUTS a
OUTPUTS y
POWERS vdd
GROUNDS gnd 0

# parameters
TOP_VLOG_MODULE inv
IN_FILE_NAME inv.cir
DEBUG_FILE_NAME inv.out
MODEL_TYPE synthesis tlf
PRINT_EQNS 1

# smartspice parameters
SPICE_TYPE smartspice
SMARTSPICE_OPTIONS {accurate scale-1u scalm-1}
SUPPLY_V_HIGH 5.0
TEMP 85
LIB_CMD 'ählt/models/process.h' TT
SCALE_FACTOR 1.5e-06
MOSFET_TYPE p pmos
MOSFET_TYPE n nmos

* spice deck for dc_2
* inputs
* a net:12
* outputs
* y net:11
* inputs
* powers
* vdd net:12
* grounds
* 0 net:10
* clocks

m20 n11 n12 n13 p l=3.500000e-001
w=8.000000e+000
m17 n11 n13 0 0 n l=3.500000e-001
w=4.000000e+000

v23 n13 0 dc 5.0v
v24 n12 0 pw/0.000000ns 5.000000v 4.000000ns
5.000000v 4.200000ns
+ 0.000000v 8.000000ns 0.000000v 8.200000ns
5.000000v)

.transform 0.050000ns 12ns
.print v(n11)
.options post=2 probe
.temp 85
.options accurate
.options scale=1u
.options scalm=1
.lib 'ält/models/process' TT
```

SmartSpice SPICE deck for an inverter
Manipulating Simulations

- AccuCore includes a number of configuration commands to manipulate a specific simulation to:
  - Adjust the simulation corner
  - Specify slew rates, delays, and capacitance
  - Adjust the measurement of setup and hold times

- Adjusting the Simulation Corner

  - `SUPPLY_V_HIGH` specifies the maximum voltage that any internal signal can reach during simulation. The default value is 5.0
  - `TEMP` sets the temperature at which the simulation is run. This command corresponds to the common temp SPICE option
  - `LIB_CMD` is used when the SPICE simulation models are stored in a library
  - `INC_CMD` is used when the SPICE process models are added into the simulation via an .include file. AccuCore does not parse the files you list under `INC_CMD` but passes them directly to the simulator
Manipulating Simulations - Specifying Rise and Fall Slew Rates

- SPICE stimuli require slew rate to be defined for each input pin
- You can define the slew rate individually for all the primary input pins, or as a slew table for the whole block in the configuration file
  - `RISE_SLOPE`: The rate at which inputs specified in the command will rise from `SUPPLY_V_LOW` to `SUPPLY_V_HIGH`. Overrides `DEFAULT_RISE_SLOPE`
  - `FALL_SLOPE`: The rate at which inputs specified in the command will fall from `SUPPLY_V_HIGH` to `SUPPLY_V_LOW`
    - Overrides `DEFAULT_FALL_SLOPE`
  - `DEFAULT_RISE_SLOPE`: Rise value if `RISE_SLOPE` is not defined
  - `DEFAULT_FALL_SLOPE`: Fall value if `RISE_SLOPE` is not defined
  - `SLOPE_TABLE`: Specifies a specific table of slew rates
    - For specific rise and fall slew rates use the commands `RISE_SLOPE_TABLE` and `FALL_SLOPE_TABLE`
Manipulating Simulations - Specifying Rise and Fall Slew Rates (cont’d)

- **SLOPE_TABLE** increases run time significantly
  - All design clusters which are connected to primary inputs will be simulated as many times as the number of entries in **SLOPE_TABLE**
  - If a design is being characterized for the first time, first characterize a design without **SLOPE_TABLE**, to test the configuration file and resolve any other issues
    - The final model set can then be built with **SLOPE_TABLE**
Specifying Slope Tables

- The \texttt{SLOPE\_TABLE} command specifies a table of slew rates to be used on the primary inputs of a block.
- AccuCore simulates each input, running a simulation for each rate in the table that includes both rise and fall transitions.
- If a config file includes \texttt{SLOPE\_TABLE}, the specified table overrides any other slew rate commands in the file.
- In the example on the next page, all the inputs inherit the specified slew table; the default and input specific slew constructs are ignored.
- The slew table is modified for cells whose inputs also include internal nets.
Measuring Slew Rate

- AccuCore measures the slew rate at all outputs of a cell
- The output of one cell may become the input of another
- The measured slew rate is applied as the input slew rate of subsequent clusters
- In the following diagram, each primary input had a set slew rate of 0.2
- A rising slew of 0.21 and a falling slew of 0.23 were measured for the internal net z (output for dc_2)
- This slew rate of {0.21, 0.23} was in turn used as the input slew of dc_3
- Finally, a slew measurement is performed for the primary output y
Slope Thresholds

- The values of SLOPE_UPPER_THR and SLOPE_LOWER_THR specify the voltages between which to measure the slew rate for a signal transition and are expressed as a percentage of the supply voltage
  - SLOPE_LOWER_THR: Specifies the lower threshold voltage to measure slew rate for signal transition
  - SLOPE_UPPER_THR: Specifies the upper threshold voltage to measure slew rate for signal transition
  - MIN_RISE_SLOPE: Minimal rising slew rate if output is less than this value
  - MIN_FALL_SLOPE: Minimal falling slew rate if output is less than this value
  - REPORT_SLOW_TRANSITIONS: Reports all transitions greater than a user-defined threshold value
  - TOP_VLOG_MODULE: Assigns the module name in the Verilog netlist
  - SWITCH_THR: Voltage level at which delays are measured

- When the rising slew rate on an output of a cell falls below the value defined by MIN_RISE_SLOPE, then the slew rate is automatically clamped to MIN_RISE_SLOPE. MIN_FALL_SLOPE is used to clamp very sharp falling slews

- To resolve discrepancies between AccuCore results and the SPICE simulator results for the simulated block, check the clamping of slopes
Slope Threshold Measurements

- The example below shows the interaction between the slew measurement parameters and measured slew rates if the supply voltage is 2.0v, \texttt{SLOPE\_UPPER\_THR} is 80\% of supply and \texttt{SLOPE\_LOWER\_THR} is 20\% of supply
- At those percentages, the slew rates would be measured at 1.6v and 0.4v
Reporting Large Slew Rates

- Checking slew rate at the output of a cell is a good way to monitor AccuCore results
- Very large input slew rates can be due to heavily loaded nets, incorrect vectors, or an inappropriate cell
- The command `REPORT_SLOW_TRANSITIONS` is used to track anomalous slew rates. For example:
  - `REPORT_SLOW_TRANSITIONS 1.0`
- Directs AccuCore to report any slew rate greater than 1.0 nanoseconds to `TOP_VLOG_MODULE_slow_tran.txt`, where `TOP_VLOG_MODULE` is the value for `TOP_VLOG_MODULE` in the configuration file
Measuring Transition Delays

- AccuCore measures delays between rising and falling transitions at the outputs of a design cluster.
- `SWITCH_THR` specifies the voltage at which to measure the delay expressed as a percentage of the supply voltage.

![Diagram showing Measuring Transition Delay](image-url)
Specifying Capacitive Load

- By default AccuCore does not add any capacitive load at the primary outputs of a block.
- AccuCore can add loads (capacitors or capacitors and resistors) during simulation to the primary outputs of a block with the CAP_LOAD command.
- You can specify any number of nets for a CAP_LOAD statement, and use multiple CAP_LOAD statements to set different capacitance values to the outputs.
- Use the command RC_LOAD to specify a capacitor and resistor load at the outputs.
- The same output may not be specified for both CAP_LOAD and RC_LOAD in the same configuration file.
- By default, AccuCore uses pico-farads for units of capacitance and kilo-ohms for resistance.
- To change the units for capacitance, add CAP_SCALE to the configuration file to specify scale applied in the circuit design before it is passed to the simulator.
- To change the units for resistance, use RES_SCALE in the configuration file to specify a scaling factor for resistors the way CAP_SCALE does it for capacitance.
- When CAP_TABLE is specified all other loading commands are ignored.
- To verify that capacitance or resistance was added to a net, check the file specified for DEBUG_OUT_FILE, and search for the output to which the load was added. By default, DEBUG_OUT_FILE is TOP_VLOG_MODULE.out.
Measuring Input Capacitance - Integration Method

- Integration Method
  - AccuCore automatically generates the vectors for characterizing the input capacitance
  - AccuCore applies a pulse to the input pin for which the capacitance is being generated and measures the average current around a rising and falling transition over a specified period of time
  - All other input pins are set to a constant value
  - The capacitance is determined by the following equation:
    - \[ I = C \left( \frac{dv}{dt} \right) \]
    - where \( I \) = current, \( C \) = effective capacitance, \( v \) = voltage, and \( t \) = time
AccuCore sets pin b to supply voltage and applies a pulse to pin a and measures the current for both the rising and falling transitions to obtain the average of the rising and falling capacitance.

Next, AccuCore calculates the effective input capacitance of the NAND gate pin b which is set to ground and the same pulse is applied to pin a.

Another average capacitance for the rise and fall transitions is generated and the capacitance is calculated for each set of vectors.

Then the maximum capacitance is retained for pin a.

This same method is used to measure the input capacitance on pin b.

To manually adjust the transition period, use the CUR_MEAS_PERIOD variable in the configuration file to force input capacitance measurement for a more accurate measurement.

For measuring input capacitance, specify the pulse rise and fall slew rates via the C_EFF_RISE_SLOPE and C_EFF_FALL_SLOPE variables.
Measuring Setup and Hold Times

- AccuCore characterizes a setup and hold time for all related input pins and clock pins
- The setup and hold time simulations require long run times and should only be turned on as the final step during block characterization
- The clock is swept through the stationary data transition to determine the setup and hold time.
- The algorithm uses logarithmic bisection to find the metastable point and continues bisecting until it meets the tolerance specified via BISECT_ACCURACY or SETHOLD_ACCUR
- For each data/clock pair there may be several vectors. In each case AccuCore performs the two setup simulations, one for data rising and one for data falling. Similarly, AccuCore also performs two hold simulations
Measuring Setup and Hold Times Example

- As the storage node of a flip-flop nears failure, the waveform starts to perform incomplete transitions, as illustrated below.

- To be recognized as a setup or hold failure, a transition must cross within thresholds set by `SH_UPPER_THR` and `SH_LOWER_THR` commands specified as percentages of the supply voltage.

- At time T1 the storage node begins a transition but is unable to complete it.

- Technically, this qualifies as a failure, but not a fully qualified failure; T1 is classified as the last time at which the storage node did not fail.
  - Consequently, this separation between the clock and the data pin is reported as the setup time.
  - At time T2, the transition falls below `SH_UPPER_THR` but does not fall below `SH_LOWER_THR`.
  - This transition qualifies as a setup failure.

- A similar simulation is performed for hold time.
  - The clock starts after the data has switched and moves towards the data.
AccuCore models the output delays of tristate design clusters based on the configuration file command TRISTATE_METHOD.

Conductance Method

- Measures tristate delays when the conductance of the driving transistors reaches below the value set by the command GDS_CUTOFF_THR.
- Once the conductance crosses that threshold, the transistor is considered off and the delay is reported to be crossing some percentage of VDD to the “off time”.
- The default value of GDS_CUTOFF_THR is 1000, indicating that when the GDS value goes below 1/1000th of the maximum value, the cut off point is reached.
- The GDS method results will not change with respect to varying load capacitance.
  - The SPICE deck below shows how such a measurement will be done for a 0 to Z transition.
  - There is no slew rate measurement for this method.

SPICE deck for a 0 to Z transition:

```
* spice deck to measure 0 to Z transition
.include 'netlist.sp'
.lib 'models' TT
.param sup=2.75
voe oe 0 pwl(0.0ns sup 5.0ns sup 5.2ns 0.0v)
va a 0 dc 0.0v
.MEAS del_MPO_gds TRIG v(oe2) VAL='sup*0.5'
CROSS=1
+ TARG 1x3(mp0) VAL=1.3e-5 CROSS=1
.end
```
Controlling Simulations

Using CONST_DELAY to turn off simulation

```plaintext
INPUTS a b c
OUTPUTS y z
POWERS vdd
GROUNDs gnd 0
# parameters
TOP_VLOG_MODULE and
IN_FILE_NAME and.cir
MODEL_TYPE synthesis
PRINT_EQNS 1
#skip simulation
CONST_DELAY 0.5
SCALE_FACTOR 1.0e-06
MOSFET_TYPE p pmos
MOSFET_TYPE n nmos
```

The above example sets all delays to 0.5ns.

- Creating a working configuration file is the most important aspect for the characterization process with AccuCore.
- The parameters inside the configuration file control or restrict simulations and give you important insight into the vector generation and simulation time of a cell.
- Modeling Without Simulations
  - Simulation time is the single biggest contributor to the run time in AccuCore.
  - Add the command CONST_DELAY to the configuration file to turn simulation off.
  - AccuCore then goes through the partitioning, function extraction, vector generation and model creation process without ever calling a simulator and instead uses a constant delay for all transitions.
Modeling Without Setup and Hold

- To quickly establish an analysis environment and setup your configuration file, you can get quick answers by skipping the characterization of setup and hold times.

- The following commands direct AccuCore to model a design without simulating setup or hold times for latches or flip-flops by specifying default values expressed in nanoseconds:
  - DEFAULT_FF_SETUP: Forces AccuCore not to calculate a setup time for flip-flops.
  - DEFAULT_FF_HOLD: Forces AccuCore not to calculate a hold time for flip-flops.
  - DEFAULT_LATCH_SETUP: Forces AccuCore not to calculate setup time for latches.
  - DEFAULT_LATCH_HOLD: Forces AccuCore not to calculate hold time for latches.

- After you have a set of structurally correct models, rerun the analysis with setup and hold simulation turned on for the most accurate results.
Limiting Transient Analysis Steps

- In AccuCore, you can set the step size of the transient analysis with \texttt{TRAN\_ANALYSIS\_STEP} that has a default value is 0.01ns.
- With a smaller step size such as 0.001ns (1ps), the run time will increase and the accuracy may also increase.
- While creating an AccuCore configuration file, work with the default value to speed up the simulation process and provide a good result.
- The step size can be modified to achieve more accurate results if necessary.
- During partitioning and vector generation, AccuCore may generate some design clusters with very long transient analysis times.
- A limit can be set for the total simulation time to be spent in the transient analysis of a cluster via the \texttt{TRAN\_ANALYSIS\_LIMIT} command.
  - This is useful to speed up the creation of a configuration file.
Limiting Transient Analysis Steps (cont’d)

In the configuration file at left, all simulations for the cell use a transient analysis step of 0.01 and any cell with a total transient analysis time greater than or equal to 10ns is skipped, no model is generated for such a cell.

The DEBUG_FILE_NAME should be reviewed to understand the origin of such a cell.

By changing appropriate partitioning commands in the configuration file, this cell may disappear.

If the cell with a long analysis time proves to be valid, then remove TRAN_ANALYSIS_LIMIT from the configuration file and run the simulation to include that cell.

The time used to simulate a cell can be limited via MAX_CHAR_INPUTS.

If a cell has more inputs than specified in this command, then AccuCore creates a vector which characterizes only a single input to output relationship.

---

**Configuration file with transient analysis parameters**

```
INPUTS  a b c
OUTPUTS y
POWERS  vdd
GROUNDS gnd 0

# parameters
TOD_VLOG_MODULE and
IN_FILE_NAME and.cir
MODEL_TYPE synthesis
PRINT_BQNS1

# slew rate definition
RISE_SLOPE 0.3 a b
FALL_SLOPE 0.4 c
DEFAULT_RISE_SLOPE 0.25
DEFAULT_FALL_SLOPE 0.20
SLOPE_TABLE [0.1 0.25 0.3 0.4 0.45]

TRAN_ANALYSIS_LIMIT 0.01

# spice parameters
SPICE_TYPE smartspice
SMARTSPICE_OPTIONS {accurate scale=1u scalm-1}
SUPPLY_V_HIGH 5.0
TEMP 85
LIB_CMD "/models/process.h' TT"

SCALE_FACTOR 1.0e-06
MOSFET_TYPE p pmos
MOSFET_TYPE n nmos
```

---
Limiting Simulation Time for Many Inputs

- The transient analysis time is roughly proportional to the number of inputs times the number of outputs as AccuCore must calculate one rising and one falling transition per input/output pair.
- Cells with more than 10 inputs will result in very long transient analysis times, and correspondingly long simulation times.
- If the number of inputs + clock pins for a design cluster is greater than or equal to \texttt{MAX\_CHAR\_INPUTS} then, AccuCore generates one SPICE deck for each input/output pair in the cell.
- This is in contrast to a single, long PWL that would result without the benefit of \texttt{MAX\_CHAR\_INPUTS}. The result is a large number of smaller SPICE simulations, often resulting in a speedup of the overall simulation.
- For example, if a design cluster has 75 inputs and 30 outputs, the tendency is to perform 75 x 30 small simulations. If you set \texttt{MAX\_CHAR\_INPUTS} to 40, then AccuCore would only perform one small simulation for the cell.
- Then AccuCore would use the resulting vector to approximate the delays of the other transitions in the cell.
Simulating Only Specified Cells

- During an AccuCore run, some cells may not produce the correct function or may have problems with simulation.
- Such a cell may be the thousandth cell in a design with 20,000 cells.
  - Debugging such a problem involves finding the suspect cell.
- SIMULATE_SCCS takes an output net or a list of output nets as its argument.
- If any net specified in SIMULATE_SCCS appears as an output net in a cell then that cell will be characterized.
- All other cell will be created but will not be characterized so AccuCore will simulate only the suspect cells.
Introduction to AccuCore

Model Generation

- Model Output Files
- Setting Up “All Path” Model Generation
- Naming the Module
- Defining Model Format
- Defining Pins for the Output Verilog Connectivity Netlist
- Formatting Names in the Verilog Connectivity Netlist
- Naming Hierarchies
- Preserving Subcircuits
- Preserving Buses in Netlists
- Pin Name Case
- Pin Order
- Setting Up “High Level” Model Generation
Model Generation

- AccuCore generates models at the end of the characterization phase and at the end of the static timing analysis phase.
- The models generated at the end of the AccuCore characterization phase are referred to as “All Path” models and the models generated by AccuCore -STA are “High Level” models.
- The models output in Liberty .lib format.
- AccuCore can generate one or more library formats in a single run.
- For the “All Path” models AccuCore also generates a verilog netlist file with cell connectivity inside the block.
- This section provides information on the following:
  - Model output files
  - Setup for “All Path” model generation
  - Setup for “High Level” model generation
Model Output Files

- During an AccuCore run, some cells may not produce the correct function or may have problems with simulation.
- Such a cell may be the thousandth cell in a design with 20,000 cells.
  - Debugging such a problem involves finding the suspect cell.
- `SIMULATE_SCCS` takes an output net or a list of output nets as its argument.
- If any net specified in `SIMULATE_SCCS` appears as an output net in a cell then that cell will be characterized.
- All other cell will be created but will not be characterized so AccuCore will simulate only the suspect cells.
Setting Up “All Path” Model Generation

- “All Path” models are created at the end of the AccuCore Characterization phase.
- Commands in the AccuCore Configuration file define which model is to be created and are used to:
  - Specify the name of the module
  - Choose the model format
  - Specify the ports of the module
  - Specify the formatting for names in the verilog connectivity netlist
  - Specify the type of timing model (see STA section)
  - Specify the name of the timing report file (see STA section)
Naming the Module

- Use the **TOP_VLOG_MODULE** command in a block-level configuration file to name the top level verilog module of the generated netlist, set the prefix for cell names and output files and the default value of many other parameters.

- For example if **TOP_VLOG_MODULE** is set to “tutorial”, the naming convention for the “All Path” models is as follows:
  - tutorial_svc.lib: Silvaco proprietary library
  - tutorial_sps.lib: Synopsys Liberty library
  - tutorial.class: indicates the type of cells in the block
  - tutorial.v: verilog netlist file with cell connectivity
  - tutorial.xrf: cross reference file that shows mapping between user defined names and generated names
  - tutorial.keep: shows files in the "kept" subcircuit
  - tutorial.fb: lists feedback loops found in the design
Defining Model Formats to Generate

- MODEL_TYPE specifies the “All Path” library models that need to be generated for timing analysis by including the keywords listed below as arguments.

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Liberty Format (.lib)</td>
<td>synthesis</td>
</tr>
<tr>
<td>Verilog Library Format</td>
<td>verilog</td>
</tr>
<tr>
<td>AccuCore Library Format</td>
<td>svc</td>
</tr>
</tbody>
</table>
Defining Pins for the Output Verilog Connectivity Netlist

- You MUST specify all primary inputs, outputs, bidirectionals, powers, grounds, and clocks for AccuCore to model the design.
- Primary ports are specified by listing them using the following commands in the configuration file:
  - inouts Defines primary bidirectionals
  - inputs Defines primary inputs
  - outputs Defines primary outputs
  - powers Defines all the power supply nets listed in the input SPICE file
  - grounds Defines all the ground supply nets listed in the input SPICE file
- A name should never be listed more than once
- Make sure that the only character following a separator is a newline
- Each design cluster appears as the keyword `TOP_VLOG_MODULE_dc` followed by an underscore and an integer
  - For example `simple_and_dc_2` is a design cluster
  - The instance name of a design cluster is created by appending "i_" in front of the dc name
  - The instance name for `simple_and_dc_2` is `i_simple_and_dc_2`
- AccuCore only creates verilog instances with named-port specifications, of the form `pin_name.(net_name)`
- AccuCore uses the corresponding net name the pin is connected to as pin names
- All net names are preserved from SPICE
Formatting Names in the Verilog Connectivity Netlist

- If several blocks are to be characterized, it is useful for the cell names of each block to be associated with that block.
- With `DC_PREFIX`, a prefix for the cell names can be set that AccuCore generates instead of the default, `TOP_VLOG_MODULE`.
- For example, with the following command:
  - `DC_PREFIX blk1`
  - the resulting netlist will have the following cell declarations:
    - `blk1_dc_2 i_blk1_dc_2( .z(z) , .a(a) , .b(b) );`
    - `blk1_dc_3 i_blk1_dc_3( .y(y) , .c(c) , .z(z) );`
Naming Hierarchies

- **INST_HIER_SEP** specifies which character or set of characters are to be used to separate the instances in the hierarchical description.
- **AccuCore** flattens the input netlist before operating on the design.
- The flattened names of nets and instances are generated with full hierarchical paths with a default separator of “!” between instances and “-” between nets.
- **NET_HIER_SEP** specifies which character or set of characters are to be used to separate the nets in the hierarchical description.
- With the default separators, the flattened net and instance names from a spice netlist would appear as:
  - mx1!x2!m5 x1-x2-n1 d x1-x2-n2 0 n l=0.35u w=4.0u
  - mx1!x1!m6 x1-x1-n2 c 0 0 n l=0.35u w=4u
- Now for example, if "/" is specified as the separator for net and instance separators, as follows:
  - **INST_HIER_SEP** /
  - **NET_HIER_SEP** /
- the same instance and net names would appear as:
  - mx1/x2/m5 x1/x2/n1 d x1/x2/n2 0 n l=0.35u w=4.0u
  - mx1/x1/m6 x1/x1/n2 c 0 0 n l=0.35u w=4u
- Instance names only appear in the debug output file, but the net names are present in the verilog netlist.
Preserving Subcircuits

- If COMPRESS_BUSES is set to 1, individual bits of a bus are replaced with their bus name in the resulting verilog netlist port definitions instantiation.
- KEEP_SUBCKT and FIND_SUBCKT can be used to force AccuCore clustering algorithm to preserve certain structures of interest and use.
- If PRESERVE_PORTS is set to 1, the port names for resulting design clusters will be based on the port names of the SPICE subcircuit that the instance was derived from.
- The port names from the KEEP_SUBCKT and FIND_SUBCKT commands will be used in the verilog netlist.

Preserving Buses in Netlists

- It is common to supply buses to a block via the configuration file.
- AccuCore breaks up the buses into individual bits during characterization.
- If such a block is instantiated in a netlist which uses buses rather than individual bits, then a port mismatch will occur.
- To overcome this problem, use COMPRESS_BUSES to direct AccuCore to create buses in the verilog netlist.
Pin Name and Pin Order

- Pin Name Case
  - When AccuCore generates a model for a cell, it converts pin names to lower case letters.
  - The original case can be retained by setting the `PRESERVE_CASE` command to 1.

- Pin Order
  - When AccuCore generates a model for a cell, it writes out pin names in the order in which they appear in the configuration file.
  - To specify a different order for pin names use the `PIN_ORDER` command.
Setup and Debug of AccuCore Flow

- `gen_model_setup` enables you to quickly find, analyze and correct critical problems (errors and warnings) associated with the design timing characterization using a standard configuration file.
- Set `SETUP_MODE_LEVEL` to 1, 2, or 3 in the configuration file for different debugging levels.
- When `SETUP_MODE_LEVEL` is 1, AccuCore will check design partitioning, function extraction (FE) and simulation vector generation (VG).
  - The Function Extraction/Vector Generation (FE/VG) procedures are completed just for unique netlist cells, that is just for one out of each set of identical cells produced.
- When `SETUP_MODE_LEVEL` is 2 AccuCore will perform simulations using default parasitics.
- When `SETUP_MODE_LEVEL` is 3 AccuCore will perform simulations using actual in-circuit loads and parasitics.
SETUP_MODE_LEVEL 1 Enables:

- Fast analysis and validation of design partitioning, function extraction and vector generation process.
- Improved reporting of the FE/VG errors
  - The setup run log file contains information about subcircuits (cells) in the original netlist related to the partitioning/function extraction errors (warnings)
  - You can use KEEP/FIND/BLACKBOX commands to localize, correct or skip the design parts associated with the errors and warnings
  - Information about the netlist subcircuits associated with correctly classified DC outputs is provided in a .class file.
- Gate-level netlist validation
  - Checking the netlist connectivity
  - Reporting nets without drivers and loads
  - Checking the consistency of path arcs
  - An improved error report provides information about subcircuits (cells) in the original netlist related to the errors (warnings)
- The Silvaco proprietary .lib file contains correct information about cell simulation vectors
  - These vectors are obtained during the cell function extraction/vector generation process
  - The information about generated vectors allows you to check and validate library cells as well as function extraction results before running an exhaustive simulation
- Enhanced managing of error/warning messages and output file writing
  - The command SUPPRESS_OUTPUT_FILES allows you to specify a list of output files that will be suppressed during a setup run
SETUP_MODE_LEVEL 2 and 3 Usage

- First fix any problems with design partitioning/function extraction/vector generation found with SETUP_MODE_LEVEL 1.

- If you have simulation issues running AccuCore then set SETUP_MODE_LEVEL to 2 so that AccuCore:
  - Makes the structural cell analysis to get a set of unique master cells to be simulated that includes topological, device size, and RC comparison of the cells.
  - Performs master cell simulation using a default or user specified slope/load table.

- If Level 2 runs OK but you still have simulation issues, run SETUP_MODE_LEVEL set to 3 so that AccuCore:
  - Performs master cell simulation using
    - Calculated effective capacitance range for all master cell loads and
    - Default or user specified slope table
    - The effective capacitance calculations for a master cell take into account all cells that are structurally identical to the master cell.

- Level-2 and level-3 setup runs will create simulation decks and/or netlist (repair) decks for the master cell failing during simulation.
**Conclusion**

- AccuCore Transistor and Gate Level Full-Chip STA with Automatic Block Characterization provides Static Timing Analysis (STA) of complex designs with mixed design styles. It gives designers the ability to characterize a multi-million transistor design with SmartSpice accuracy and perform block or full-chip static timing analysis.

- **Key Benefits**
  - Complete Static Timing Analysis (STA) environment quickly identifies timing bottlenecks leveraging state-dependent models.
  - Automated False Path removal addresses bi-directional transistors.
  - Easy setup enables mixing of custom and ASIC blocks in SoC environment.
  - Automatically partitions multi-million transistor, flat or hierarchical design with parasitics into cells for accurate SPICE level characterization.
  - Largest collection of calibrated SPICE models for CMOS and SOI, including BSIM3, BSIM3SOI, BSIM4, PSP, and HiSIM.
  - Produces timing models for Liberty .lib.
  - Generates fully-sensitized SPICE deck for critical paths and clock trees.