Powering the Future: Mastering IEEE 2416 System-Level Power Modeling Standard for Low-Power AI and AMS Designs

Organizers: Nagu Dhanwada (IBM),
Leigh Anne Clevenger (Si2)

Speakers:

Nagu Dhanwada, Akil Sutton, IBM Corporation

Eunju Hwang, Samsung

Tavares Forby, Qualcomm

Rhett Davis, North Carolina State University

Daniel Cross, Cadence Design Systems

62 THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY CEDA sigda

Nagu Dhanwada

IBM Corporation

Welcome and Agenda

- Session outline
- Speaker introductions
- Goals of the tutorial

Session Outline

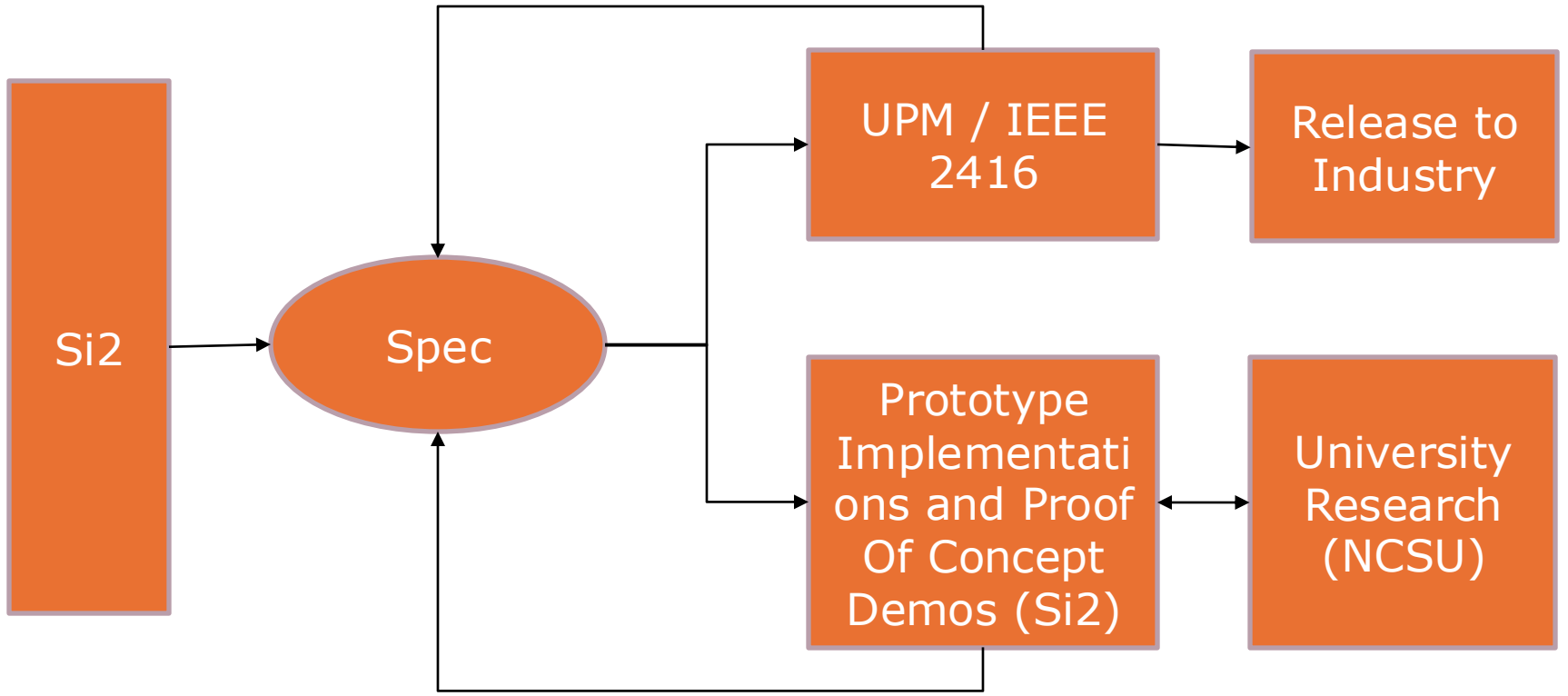
- Introduction (15 min), Nagu Dhanwada (IBM)
- Core Concepts of IEEE 2416-2025 – Digital and AMS Highlights (30 min), Akil Sutton (IBM)
- Real-World Applications – Industry Deep Dives (60 min), Eunju Hwang (Samsung), Tavares Forby (Qualcomm)
- System-Level Example: AI Accelerator with AMS Blocks (60 min), Daniel Cross (Cadence), Rhett Davis (NC State)
- Q&A (15 min)

IEEE 2416 Overview: Mission and Motivation

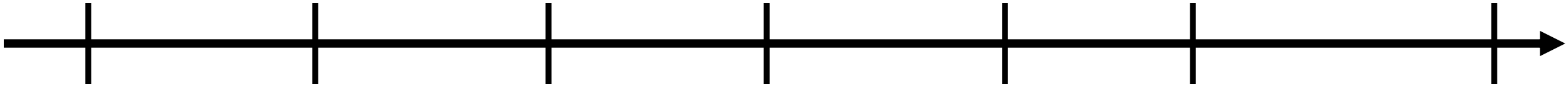
- Mission: Develop a standard for system level power modeling that has low barriers of adoption by the industry.

- Motivated by lack of a standard way to:
  - Formally represent power data at the system level
  - Represent IP block power data at multiple design levels
  - Represent power data for use at the System Level across a range of PVT points in a single model

How: Operational



Time-line (Big Picture)



Jan 2016	June 2017	June 2019	2023-2024	October 2024	July 2025	Sept – October 2025*
Approval of Complete 2416 Requirements Document v1.0	Power Modeling Contribution from Si2 V2.0 @DAC F2F	DAC 2019 F2F and announcement of 2416-2019 standard	Development of UPM2.0, 2.1 specs, adoption and real world examples development	Contribution of UPM 2.1 by Si2 to IEEE 2416	Ballot process, comment resolution, technical writing completed	Revcom submission, SASB approval, final release

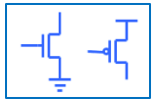
WHAT is the standard trying to address?

- IP and Component Abstraction
- PVT (Process, Voltage and Temperature) Independent Modeling
- Model Continuity and Inter-operability
  - Within Design Flows
  - Across Abstraction Levels

WHY do we need to address these?

- Power aware system simulations need efficient models
  - Abstraction essential for providing major efficiency gains
- Need for accurate temperature sensitive power models usable during early exploration and system design.
- Currently there is *no standard* way to:
  - Represent power data across a range of PVT points
  - Represent power data consistently across multiple design level
  - Supply power data to 1801 System Level Power Models
- PVT Independent modeling substantially reduces effort for
  - Multi-corner characterization
  - Inaccurate interpolation / extrapolation
- Power model re-use
  - Contributor modeling enables the use of a single power model from early spreadsheet-style modeling to detailed simulations during rest of design flow

How: Key Features Supporting Abstraction

- Multiple data representations
  - Scalars `7.51`
  - Multi-dimensional Table
    ```
    3.55 3.85 3.96 4.02
    3. 4.12 4.25 4.35 4.47
    4. 4.46 4.56 4.96 4.79
    4.46 4.56 4.96 4.79
    ```
  - Expressions `C*V^2*F`
  - Contributors 
- Separate representations for static and dynamic power
- process, Voltage, Temperature (pVT) independence – **1 model for many corners**
- It's designed for interoperability
  - It's an industry standard – UPM / IEEE 2416
  - It's built on standard languages – XML, Verilog-A
  - *It's designed to interoperate with UPF / IEEE 1801*

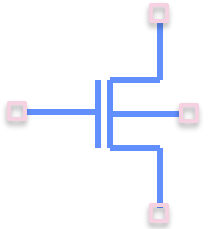
IEEE

<xml/>

UPF

How: Power Contributors as Power Proxies

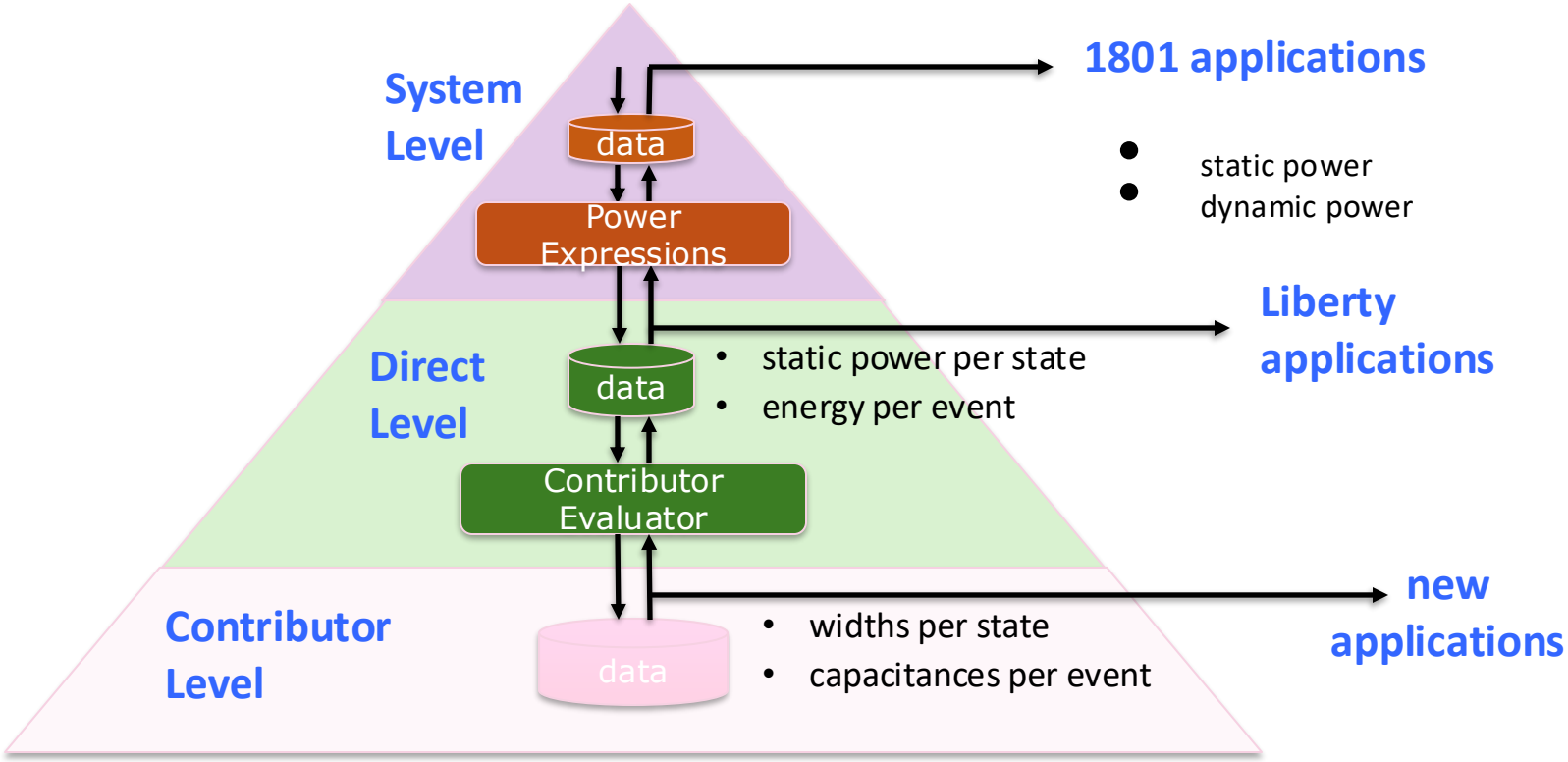
- Power data is NOT stored[1] in the model
  - Power *proxies* (power contributors) are stored instead
    - Storing proxies enables *PVT independent* modeling
- Leakage power proxy: Leaking transistor widths = W_{eff}
  - Leakage may be calculated from:
    - W_{eff} (stored in the model)
    - P, V, T, S (provided by user at time of evaluation)



- Energy proxy: Equivalent capacitance = C_{eff}
  - Energy (and dynamic power) may be calculated from:
    - C_{eff} (stored in the model)
    - V, T, S (provided by the user at time of evaluation)

[1] Actually, power data may be stored directly if desired for backwards compatibility. But doing so is PVT dependent and prevents the usage at other PVT corners.

How: UPM Multi-level Model for Model Continuity and Inter-operability



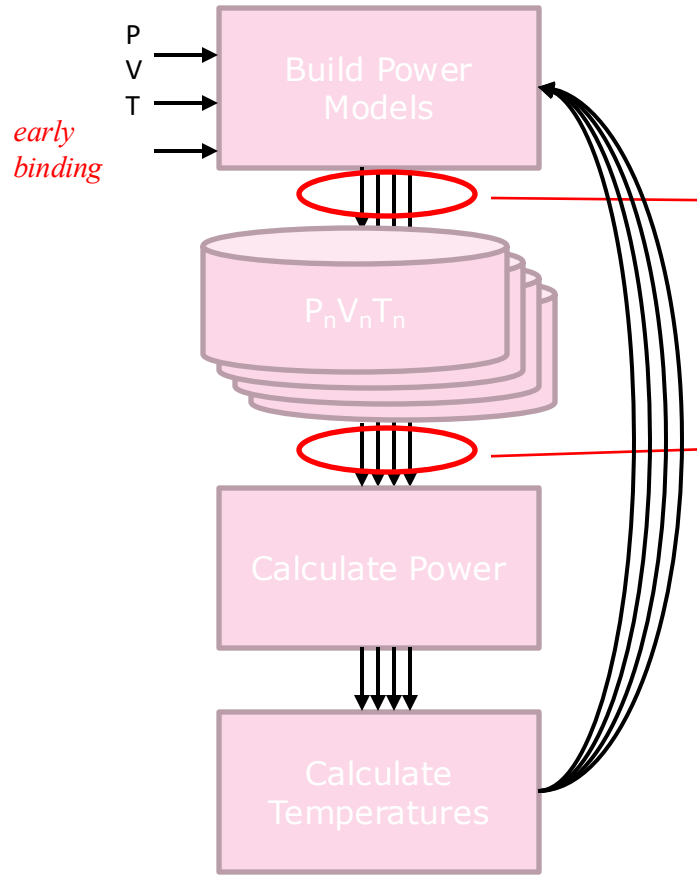
Data at a particular level may be stored there or retrieved and evaluated from a lower level.

Benefits of IEEE 2416

- Vendor-neutral modeling
- PVT-independence
- Design-space exploration

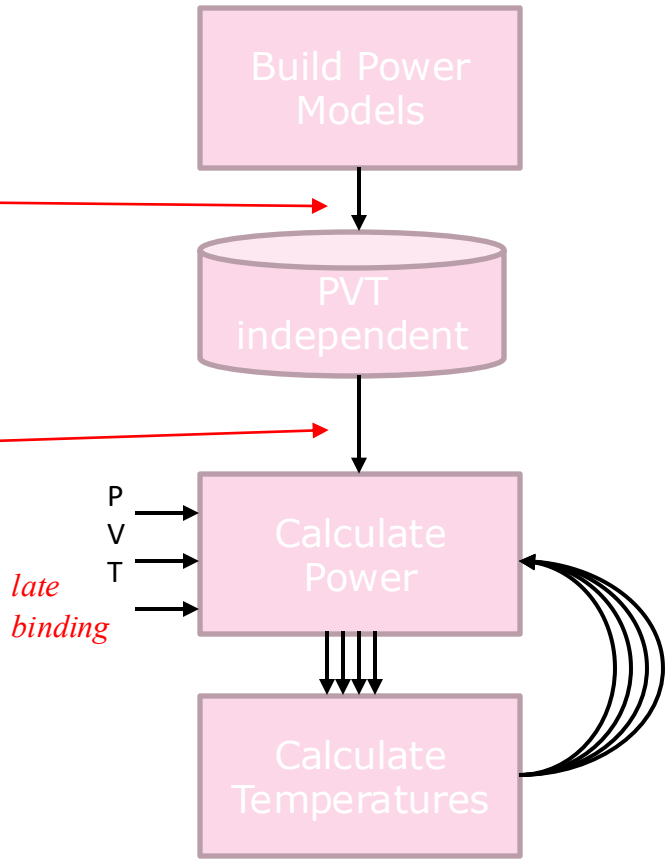
PVT Late Binding Improves Power Flows

Conventional library power flow



- Multiple libraries
- Long iterative loops
- Thermal gradient insensitive

Contributor library power flow



- Single library
- Fast, short iterative loops
- Thermal gradient capable

UPM Model Structure

```
<Cell name="cpu”>
  <CellParameters>
    <Parameter name="Loads" type="float"/>
    ...
  </CellParameters>

  <States>
    <State name=“Active">
      <DynamicPower>
        <Expression name="cpu_dynamic_power_linear_regression">
          0.5 * (((0.12*BP_mispredicts) + (4.78*Cache_missses) +
          (2.81*Loads) + (3.32*Stores) + 0.89)/1e12) * V * V * F
        </Expresssion>
      </DynamicPower>
      <StaticPower>
        <Expression name="cpu_static_power">
          0.128 * Leakage_Scaling_Expression()
        </Expression>
      </StaticPower>
    </State>
    ...
  </States>

<Pins>
    <Pin name="HCLK" direction="input" capacitance="9.9"/>
    ...
  </Pins>

  <Modes>
    ...
  </Modes>
  <Events>
    ...
  </Events>
```

Cell declaration

Cell parameters

State dependencies

System (TLM) level

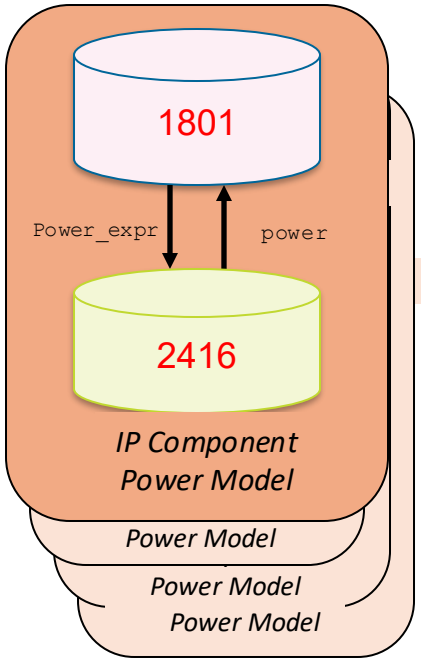
Power expression

Gate (bit) level

Self-Checking Model with XML Schema

- Model semantics are described in XML
- Model structure is described in XSD schema (formalization of model contents)
  - Validates model elements and attributes
  - Validates attribute values and value ranges
  - Validates relationships between parameters
- Schema enables automatic model structure validation
  - Basically, the XSD is an executable spec

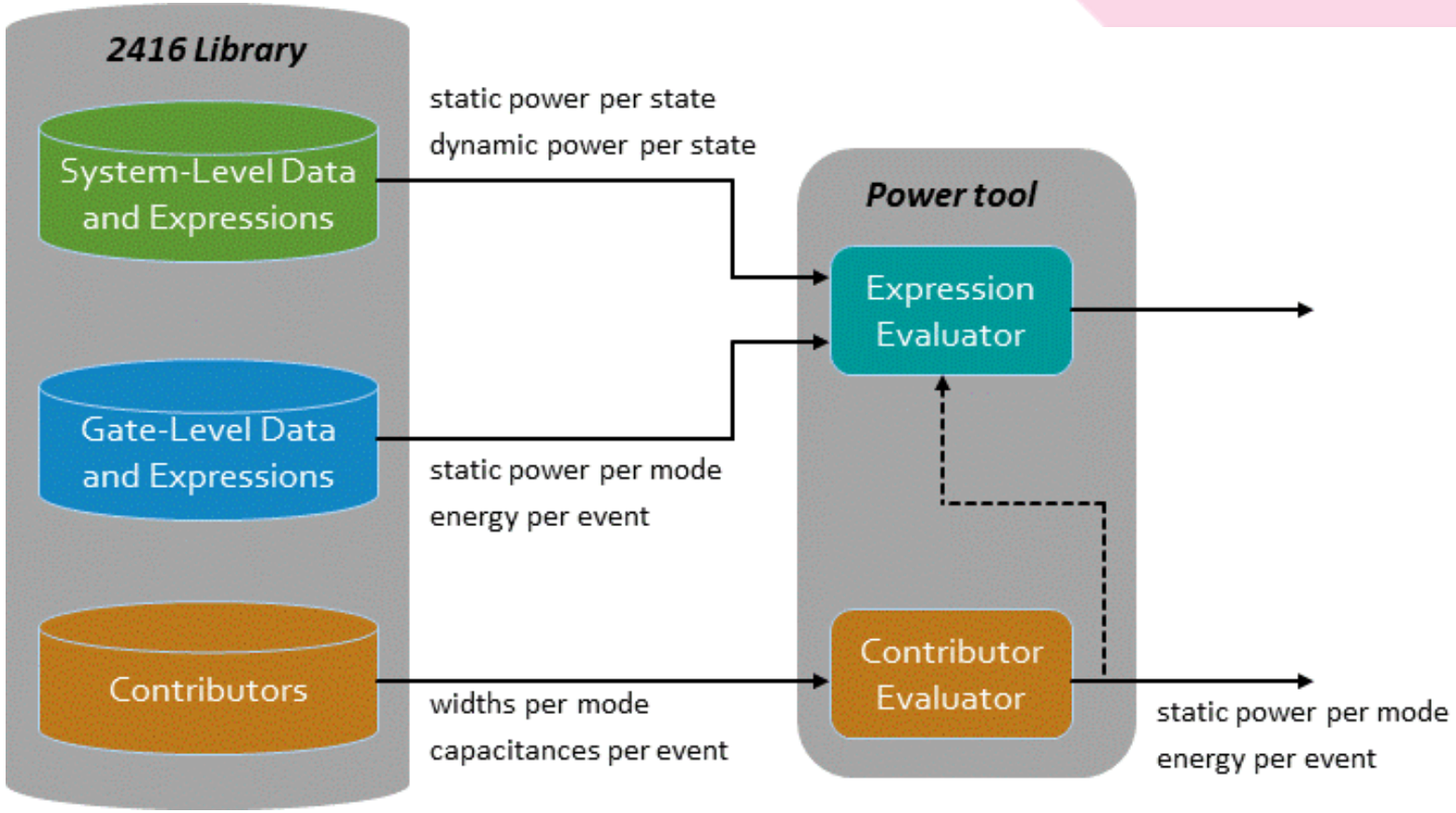
UPF (IEEE 1801) and UPM (IEEE 2416)



- Component Model Structure (1801-UPF)
  - Power states and power domains
  - Logic expressions determine current Power state
  - Parameters affect power in each P-state
  - Power expression w parameterized power function

- Component Power Data (2416-UPM)
  - Semantics for power abstraction of complex IP
  - Formalized PVT independence
    - Power contributors per (power & functional) mode
    - Energy contributors per mode transition
  - Computationally efficient PVT binding
  - Multiple data interfaces

2416-2019/2025 Overall Model Architecture



Session Outline

- Introduction (15 min), Nagu Dhanwada (IBM)
- Core Concepts of IEEE 2416-2025 – Digital and AMS Highlights (30 min), Akil Sutton (IBM)
- Real-World Applications – Industry Deep Dives (60 min), Eunju Hwang (Samsung), Tavares Forby (Qualcomm)
- System-Level Example: AI Accelerator with AMS Blocks (60 min), Daniel Cross (Cadence), Rhett Davis (NC State)
- Q&A (15 min)

Akil Sutton

IBM Corporation

IEEE P2416 Standard for Power Modeling: Latest Extensions to the Standard -- Analog and Mixed Signal (AMS) and Other Extensions

HIPS
STEMS
ERENCE

SPONSORED BY

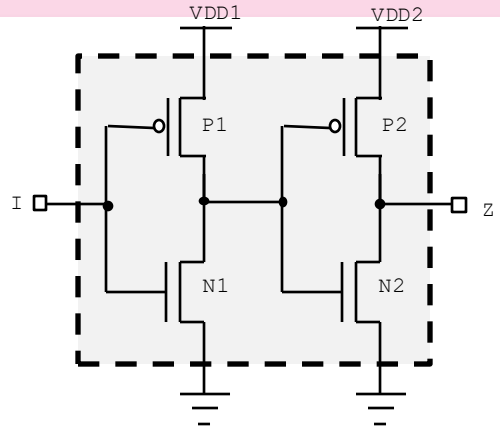
UPM 2.0 / IEEE 2416-2025 Enhancements

- Multiple supply modeling
  - `Cell` and `Pin` semantic enhancements
  - New `Source` and `Sink` semantics
  - Netlist level *current accounting* rules

- Power Categorization / Tagging
  - New `Tag` semantics for adding metadata to UPM power models

Cell and Pin Enhancements

- Integrated Level-Shifter used as example




```
<ModelConditions name="characterization"/>
    <Condition name="Process" value="TT"/>
    <Condition name="Temperature" value="373.15"/>
    <Condition name="v0" value="0.0"/>
    <Condition name="v1" value="1.1"/>
    <Condition name="v2" value="0.8"/>
</ModelConditions>
```

(deprecate!)

```
<Cell name="LS" condition="characterization" modelConditions="characterization">
   <Pins>
      <Pin name="I" direction="input" type="signal" relatedPower="VDD1" relatedGround="VSS"/>
      <Pin name="Z" direction="output" type="signal" relatedPower="VDD2" relatedGround="VSS"/>
      <Pin name="VDD1" direction="input" type="primary_power" condition="v1"/>
      <Pin name="VDD2" direction="input" type="primary_power" condition="v2"/>
      <Pin name="VSS" direction="output" type="primary_ground" condition="v0"/>
   </Pins>
   <Modes>
      ...
```

UPM 2.0 semantics shown in red

Pin Type Enhancements

- Add support for Liberty's pg_pin types (list below from Liberty v2017.06)
  - `primary_power` – primary power source (default if not specified)
  - `primary_ground` – primary ground source
  - `backup_power` – backup power source for retention registers, etc
  - `backup_ground` – backup ground source retention registers, etc
  - `internal_power` – internal power source for switch cells
  - `internal_ground` – internal ground source for switch cells
  - `nwell` – n-wells for substrate biasing
  - `pwell` – p-wells for substrate biasing
  - `deepnwell` – isolation n-wells for substrate-biasing
  - `deeppwell` - isolation p-wells for substrate-biasing
- Maintain other types as supported in UPM1.0
  - `secondary_power`
  - `secondary_ground`

UPM 2.0 semantics shown in red

Power/Energy Contributor Instance Enhancement

- Add Sink and Source specification (per contributor)

```
<PowerContributor name="…" quantity="…" width="…" length="…"
source="pin_name" sink="pin_name"/>

<EnergyContributor name="…" quantity="…" capacitance="…"
source="pin_name" sink="pin_name"/>
```

- Notes
  - The pin specified as a `source` or `sink` may be a signal pin or a supply pin
  - Current flow convention: current flows out of source, into sink
  - Currents can be either positive or negative
  - Supply connection(s) are implied by the contributor name

UPM 2.0 semantics shown in red

StaticPower & DynamicPower Enhancements

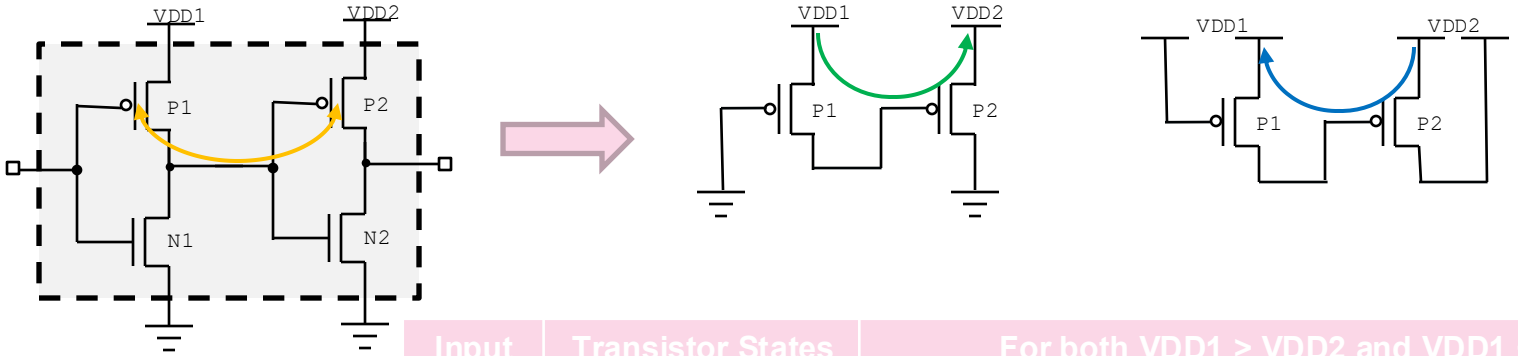
- Add Sink and Source specifications

```
<State name="ACTIVE">
    <DynamicPower source=”VDD” sink=”VSS”>
        <ExpressionParameters>
            <Parameter name=”Active_Energy” value=”6.64e-7”/>
        </ExpressionParameters>
        <Expression name="LFSR_ACTIVE_dpwr_VPG">
            Active_Energy * F
        </Expression>
     </DynamicPower>
     <StaticPower source=”VDD” sink=”VSS” value=”12.3”/>
</State>
```

UPM 2.0 semantics shown in red

Supply to Supply Current Paths

- 2 different contributors for supply to supply currents, dependent upon state



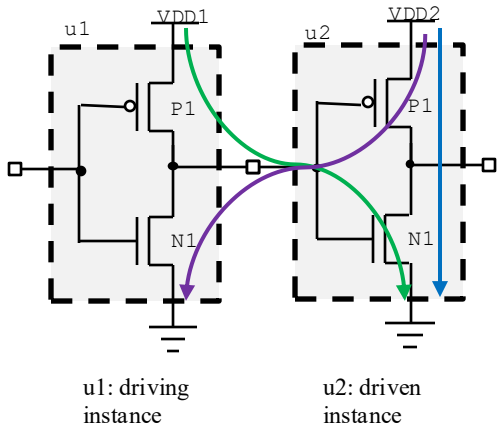
Input	Transistor States	For both VDD1 > VDD2 and VDD1 < VDD2
LOW	P1 ON, P2 OFF	P1's Active current sources P2's Gate-OFF leakage
HIGH	P1 OFF, P2 ON	N1 sinks P2's Gate-ON leakage

```
<Mode name="M0" when="(NOT I)">
    <PowerContributor name="SUPPLY1_TO_SUPPLY2" ... source="VDD1" sink="VDD2" />
</Mode>
<Mode name="M0" when="I">
    <PowerContributor name="SUPPLY2_TO_SUPPLY1" ... source="VDD2" sink="VDD1" />
</Mode>
```

UPM 2.0 semantics shown in red

Current Accounting Rules @ Netlist Level

- Assigning leakage currents properly to cells & supplies in UPM tools



- For gate leakage currents sourced by input pin (current flowing out of input pin)
  - Source current is assigned to `relatedPower` of driven instance
  - Sink current is assigned to `relatedGround` of driving instance
- For gate leakage currents sunk by input pin (current flowing into input pin)
  - Source current assigned to `relatedPower` of driving instance output
  - Sink current is assigned to `relatedGround` of driven instance input
- For channel (aka sub-threshold) leakage currents
  - Source and Sink currents are assigned to the supply of instance being analyzed

Power Categorization / Tagging in UPM 2.0

- A `Tag` may be used to add one or more labels to a UPM model
- UPM 2.0 defines several specific **standard `Tags`**
- Model creators may define and use **User Defined Tags**

Standard Tags

- Standard Tags shall have pre-defined names and typed values

```
<Tag name="tag_name" value="value" units="units"/>
```

where `name` and `value` are required and `units` is optional (the spec will need to specify what strings are valid as units)

- The following Standard Tags shall be added to the UPM2.0 spec
  - `category`: the value may be a comma separated list of labels
  - `extent`: the value shall be a pair of xy points (x_0, y_0, x_1, y_1)
    - Note that if units are not specified, the dimensional units as specified in the `Units` library level element will be used
  - `size`: the values shall be a length and width (l, w)
  - `UseCellOrigin`: the value shall be boolean
  - `location`: the value shall be a single xy point (x, y)
    - Note that if units are not specified, the dimensional units as specified in the `Units` library level element will be used
  - `layer`: the value shall be the name of the target layer (layer_name)
    - A commonly used set of layer names will need to specified in the spec

- It will be the responsibility of Tool Developers to interpret Standard Tags according to the specification

Standard Tag Valid Values

- Standard Tags shall take on only certain types and enumerated values
- Valid `category` values: any string or list of strings
- Valid `extent` values: two pairs of integers
- Valid `location` values: a single pair of integers
- Valid `layer` values: a string representing a pre-defined set of layer names
- Valid UseCellOrigin values: `TRUE | FALSE`

Examples of Standard Tags

- Cells

```
<Cell name="example_cell">
    <Tag name="category" value="combinatorial, lowVt"/>
    <Tag name="extent" value="0, 0, 397, 479" units="um"/>
    <Pins>
    ...
</Cell>
```

- Power Contributors

```
<PowerContributor name="contrib1" quantity="1" length="50" ...>
    <Tag name="category" value="analog, clock"/>
    <Tag name="location" value="20, 15" units="um"/>
    <Tag name="layer" value="M1"/>
    ...
</PowerContributor>
```

UPM 2.0 semantics shown in red

User Defined Tags

- Model creators may use User Defined Tags
- A User Defined Tag uses the same semantics as a standard Tag

```
<Tag name="non-std_name" value="value"/>
```

- It shall be the responsibility of the model creator to ensure that their target application is capable of understanding their User Defined Tags
  - A UPM compatible tool shall skip over User Defined Tags unless it has been enhanced to recognize and operate upon specific User Defined Tags

UPM 2.0 semantics shown in red

Examples of User Defined Tags

- Energy Contributors

```
<EnergyContributor name="switched_cap" quantity="1" ...>
    <Tag name="voltage_swing" value="0.5"/>
    <Tag name="num_transistors" value="138"/>
    ...
</EnergyContributor>
```

where `voltage_swing` and `num_transistors` are non-standard names

UPM 2.0 semantics shown in red

UPM 2.1 Key Enhancements

- AMS modeling primary motivation for changes
- Key enhancements
  - Stronger and more consistent datatyping
  - Expanded when conditions to support non-binary values
- Deprecated constructs
    - Cell condition attribute
    - When condition syntax
    - Type attribute

Datatyping Expanded in UPM2.1

- datatyping expanded and made more consistent in several Elements
  - `ModelParameters`
  - `CellParameters`
  - `ExpressionParameters`
  - `Pins`
  - `Array`
- Defined Datatypes
  - `bit, integer, float, string, enum.`
- Examples
  - `<Pin name="VDD" pinType="primary_power" … />`
  - `<Paramter name="enable" dataType="bit" … />`

Expanded when condition Support

- When condition enhancements needed to support non-binary values
- New when condition syntax
  - Old: `when="[AND [Not A] [Not B]"`
  - New: `when="A==0 && B==0 && VDD>=1.0"`
- Available when condition operators
  - Pre-existing logical operators: `!, &&, ||`
  - New algebraic operators: `<, <=, >, >=`

Deprecated Constructs in UPM2.1

- `Cell condition` attribute replaced
  - `condition` replaced with `modelConditions`

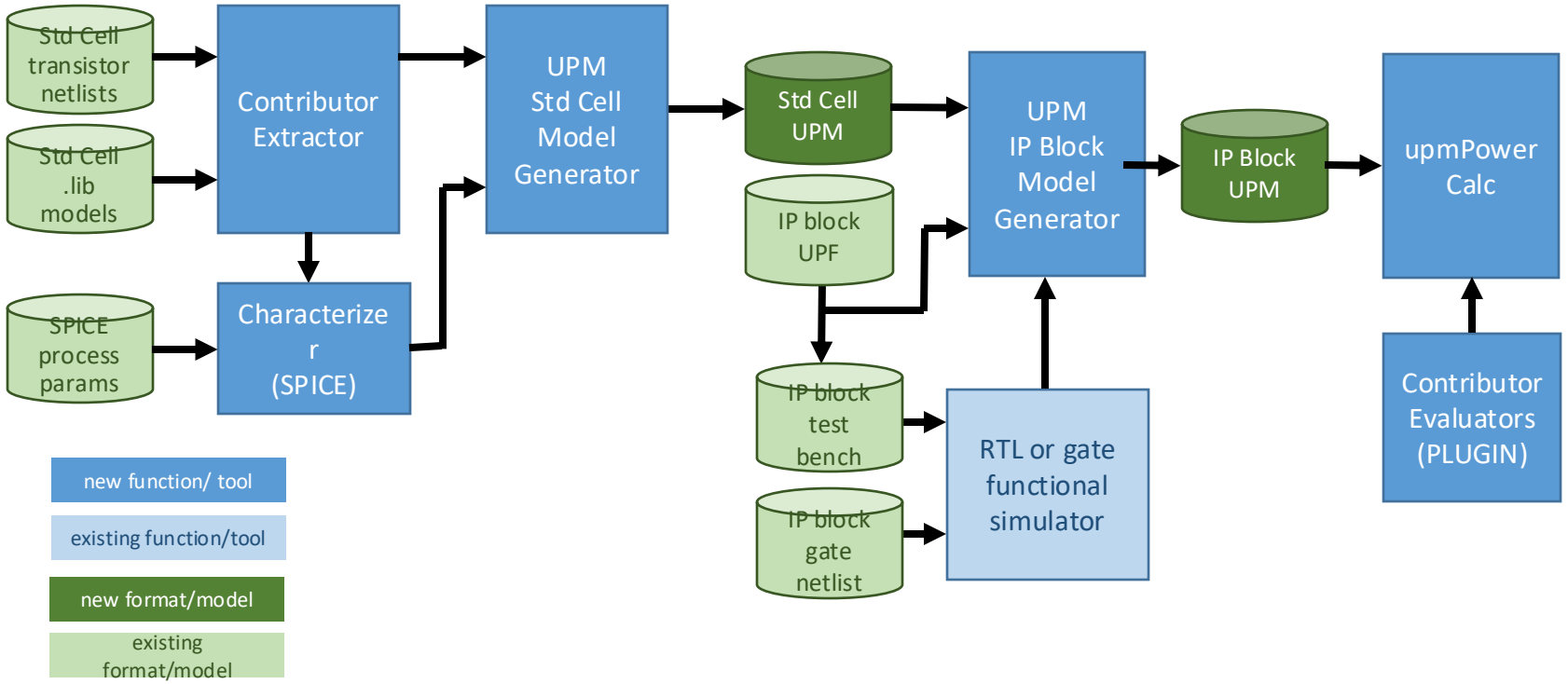
- `when condition` syntax replaced
  - Old: `when="[AND [Not A] [Not B]"`
  - New: `when="A==0 && B==0 && VDD>=1.0"`

- Type attribute replaced
  - Pin attribute `type` replaced with `pinType`
  - Parameter attribute `type` replaced with `dataType` in several Parameter subelements
  - Array attribute `valueType` replaced with `dataType`

UPM Prototype Toolset

Contributor Extractor	• Identifies and extracts Power Contributors from standard cells in library • Contributors are power proxies
Contributor Characterizer	• Generates Contributor characterization data for UPM Model generation • Permutates voltages across gate, source, drain, and body
UPM Standard Cell Model Generator	• Reads extracted Contributors and Contributor characterization data • Generates UPM models for standard cell primitives
upmPowerCalc	• Power calculator for UPM models • Calculates static and dynamic power • P, V, T values specified at calculation run-time
UPM IP Block Model Generator	• Optional auto generation of abstract IP Block model • Creates compressed abstract UPM power model • Rapid power calculation for complex IP blocks - no netlist required

UPM Model Generation & Analysis Flow



IP Block Abstract Power Model Generation

- UPM supports gate level models as well as abstract IP Block level models
  - IP Block UPM models can be used interchangeably across system level modeling
- Abstract IP Block created automatically from TESTMACRO netlist
  - UPM Standard enables creation of compressed abstract models from complex IP
  - IP-Block model uses aggregated V, T independent power contributors
  - Optional methodology for improved runtime analysis
- Netlist interconnectivity discarded in the abstraction process
  - Aids compression
  - Obfuscates block functionality
- Total runtime of 30 minutes to generate abstract model

RISC-V System Level Model w Expressions

```
<Cell name=”RocketTile”>
  <CellParameters>
    <Parameter name=”Num_LW”/>
    <Parameter name=”Num_SW"/>
    ...
  </CellParameters>
  <States>
    <State name=“*”>
      <DynamicPower>
        <ExpressionParameters>
          <Parameter name=”Cap_LW” value=“3.671”/>
          <Parameter name=”Cyc_LW” value=“2.863”/>
           ...
        </ExpressionParameters>
        <Expression name="cpu_dynamic_power">
          (Vi^2 / NumCyc) * (((Num_LW * Cap_LW) + ... )
        </Expression>
      </DynamicPower>
      ...
    </State>
</States>
</Cell>
```

Energy per Instruction (EPI) model

Voltage & Frequency Independent model

Cell parameters (values from workload instruction counts)

Expression parameters (from characterization)

Dynamic power expression

Agenda

1. Scope of Proposed AMS Extensions
2. New Functions
    a. LoadPower
    b. PowerSupplyContributor
3. Exemplary Use Case
    a. Power Delivery Network with Low Drop Out Regulator

Scope of Proposed AMS Extensions

1. Extend the power modeling capability of the IEEE P2416 standard to applications involving AMS components that deliver power to downstream functional and sub-circuit blocks.

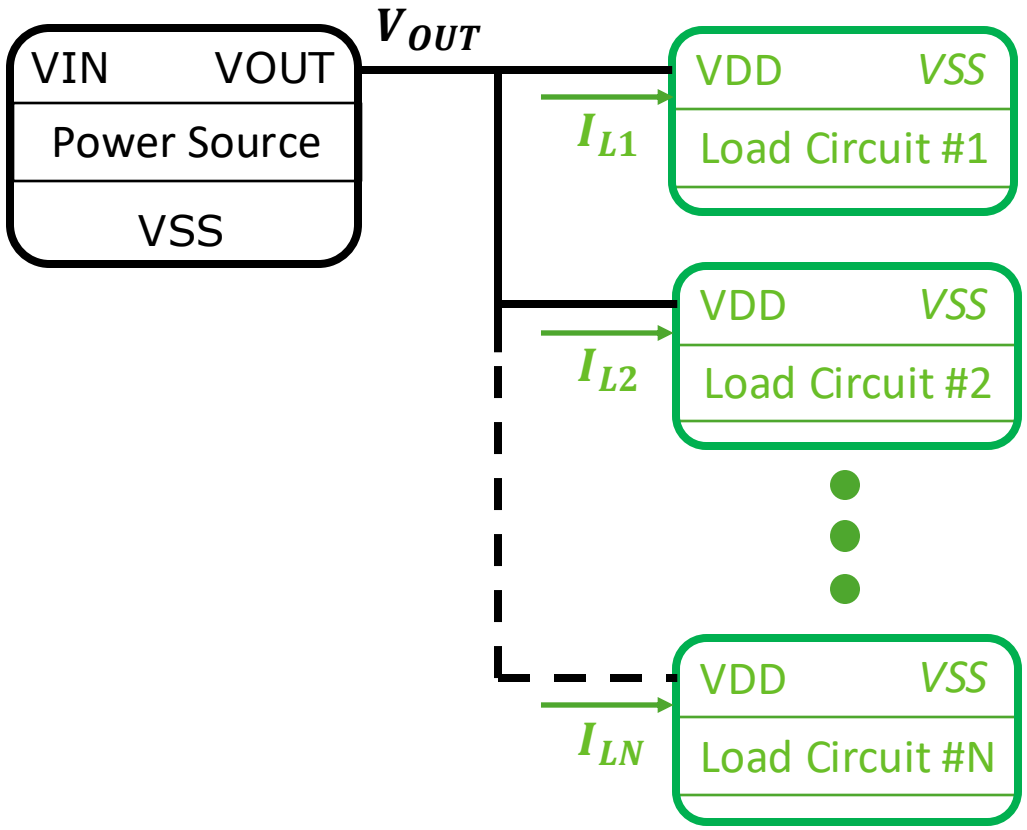
2. Does NOT address AMS functional and sub-circuit blocks that are not power sources

LoadPower

- Output: Total power being sourced from supply / output pin
- Input: Name of the supply pin
  - Supply pin must be declared with $direction = \text{“}inout\text{”}\ or\ \text{“}output\text{”}$ and $pinType = \text{“}internal\_power\text{”}$
- Power is returned in units as specified in the Units library element.

LoadPower equation

- $LoadPower(VOUT) = P_{OUT} = V_{OUT} \times \sum_{n=1}^{N} I_{Ln}$



PowerSupplyContributor()

- Output: Steady-state DC power consumed by an AMS power delivery component
- Inputs:
  - Contributor name
  - Input voltage
  - Output voltage
  - Overhead current from support circuitry (fixed)
  - Conversion losses (architecture and load dependent)
  - Relationship between input and output voltage
- Power is returned in units as specified in the Units library element.

PowerSupplyContributor parameters

- P_{IN} is the internal power consumed at the input pin, VIN
- V_{IN} is the DC voltage on input pin, VIN
- P_{OUT} is the total power demand at the output pin, $VOUT$
- V_{OUT} is the DC voltage on the output pin, $VOUT$
- I_{OH} isvthe fixed current consumption overhead of support circuits
- R_{EFF} captures the effective switch resistance in ON or OFF states
- η is the architecture and load-dependent power conversion loss
- α is selected based on whether the switching regulator is step-up or step-down

PowerSupplyContributor equations

- Three PowerSupplyContributor types are supported:
  - Linear Regulators: e.g., Low-Dropout Regulator

$$P_{IN} = V_{IN} \times \left[\left(\frac{\left(\frac{P_{OUT}}{V_{OUT}} \right)}{\eta} \right) + I_{OH} \right] - P_{OUT}$$

  - Switching Regulators: e.g., Buck or Boost Converter

$$P_{IN} = V_{IN} \times \left[\left(\frac{P_{OUT}}{V_{OUT}} \right) \times \left(\frac{\left(\frac{V_{IN}}{V_{OUT}} \right)^{\alpha}}{\eta} \right) + I_{OH} \right] - P_{OUT}$$

  - Load Switch
    - $P_{IN} = \left[\frac{(V_{IN} - V_{OUT})^2}{1/\eta} \right] + (V_{IN} \times I_{OH})$
- P_{IN} represents the internal power consumption of the contributor as the power supplied by the contributor (P_{OUT}) is subtracted to facilitate power roll-up consistent with digital contributors.

PowerSupplyContributor XML definition

- Linear regulator definition leveraging call to *LoadPower*:

```
<PowerSupplyContributor name="linearRegulator" source="VIN" sink="VSS" vin="VIN"
vout="VOUT" overhead="OVERHEAD_CURRENT" efficiency="EFFICIENCY" alpha="ALPHA">
  <Expression>
      V(VIN) * [((LoadPower(VOUT)/V(VOUT))/EFFICIENCY) + OVERHEAD_CURRENT] -
    LoadPower(VOUT)
  </Expression>
</PowerSupplyContributor>
```

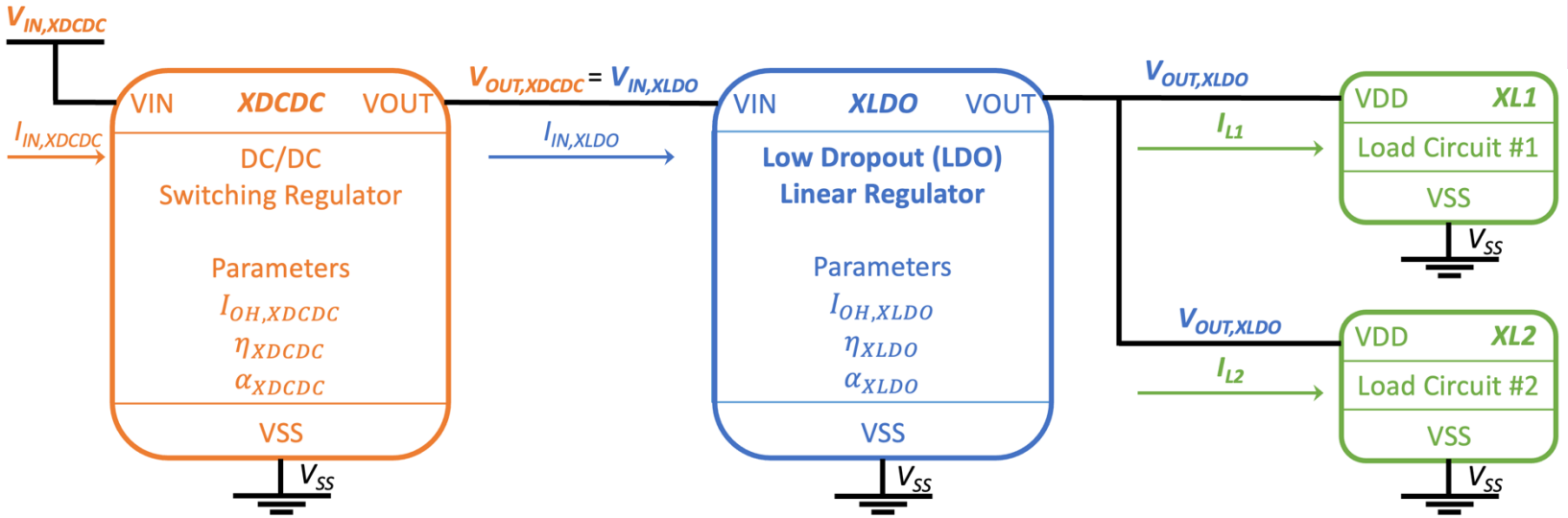
- Switching regulator definition leveraging call to *LoadPower*:

```
<PowerSupplyContributor name="switchingRegulator" source="VIN" sink="VSS" vin="VIN"
vout="VOUT" overhead="OVERHEAD_CURRENT" efficiency="EFFICIENCY" alpha="ALPHA">
  <Expression>
    V(VIN) * [(LoadPower(VOUT)/V(VOUT)) * (((V(VIN)/V(VOUT))^ALPHA)/EFFICIENCY)
                                + OVERHEAD_CURRENT] - LoadPower(VOUT)
  </Expression>
</PowerSupplyContributor>
```

- Load switch definition:

```
<PowerSupplyContributor name="loadSwitch" source="VIN" sink="VSS" vin="VIN" vout="VOUT"
overhead="OVERHEAD_CURRENT" efficiency="EFFICIENCY" alpha="ALPHA">
  <Expression>
     [((V(VIN) - V(VOUT))^2) × EFFICIENCY] + (V(VIN) × OVERHEAD_CURRENT)
  </Expression>
</PowerSupplyContributor>
```

Power Delivery Network with LDO



- Roll-up equations realized with *PowerSupplyContributor* and *LoadPower* functions

$$P_{TOTAL} = P_{XDCDC} = P_{IN,XDCDC} + P_{OUT,XDCDC} = P_{IN,XDCDC} + P_{XLDO} = P_{IN,XDCDC} + P_{IN,XLDO} + \sum_{n=1}^{N} P_{Ln}$$

$$\begin{aligned} P_{TOTAL} = P_{XDCDC} &= P_{IN,XDCDC} + P_{IN,XLDO} + LoadPower(V_{OUT,XLDO}) \\ &= PowerSupplyContributor(switching, V_{IN,XDCDC}, V_{OUT,XDCDC}, I_{OH,XDCDC}, \eta_{XDCDC}, \alpha_{XDCDC}) \\ &+ PowerSupplyContributor(linear, V_{IN,XLDO}, V_{OUT,XLDO}, I_{OH,XLDO}, \eta_{XLDO}, \alpha_{XLDO}) \\ &+ LoadPower(V_{REG,XLDO}) \end{aligned}$$

PowerSupplyContributor call (DCDC)

- Mode-dependent PowerSupplyContributor calls for *XDCDC* instance

```
<Cell name="XDCDC">
  <Pins>
    <Pin name="VIN" direction="input" type="primary_power"/>
    <Pin name="VOUT" direction="output" type="internal_power"/>
    <Pin name="VSS" direction="output" type="primary_ground"/>
  </Pins>
  <CellParameters>
    <Parameter name="OVERHEAD_CURRENT" datatype="float" select="internal" value="1E-6" units="A"/>
    <Parameter name="LEAKAGE_CURRENT" datatype="float" select="internal" value="1E-10" units="A"/>
    <Parameter name="BUCK_EFFICIENCY" datatype="float" select="internal" value="0.9"/>
    <Parameter name="BOOST_EFFICIENCY" datatype="float" select="internal" value="0.85"/>
    <Parameter name="SHUTDOWN_VOLTAGE" datatype="float" select="internal" value="2.5" units="V"/>
  </CellParameters>
  <Modes>
    <Mode name="BUCK" when="V(VIN) >= V(VOUT)">
      <PowerSupplyContributor name="switchingRegulator" source="VIN" sink="VSS" vin="VIN"
      vout="VOUT" overhead="OVERHEAD_CURRENT" efficiency="BUCK_EFFICIENCY" alpha="1"/>
    </Mode>
    <Mode name="BOOST" when="V(VIN) < V(VOUT)">
      <PowerSupplyContributor name="switchingRegulator" source="VIN" sink="VSS" vin="VIN"
      vout="VOUT" overhead="OVERHEAD_CURRENT" efficiency="BOOST_EFFICIENCY" alpha="-1"/>
    </Mode>
    <Mode name="SHUTDOWN" when="V(VIN) < SHUTDOWN_VOLTAGE">
      <Expression name="LEAKAGE_POWER" source="VIN" sink="VSS">
        V(VIN) * LEAKAGE_CURRENT
      </Expression>
    </Mode>
  </Modes>
</Cell>
```

PowerSupplyContributor call (LDO)

- Mode-dependent PowerSupplyContributor calls for *XLDO* instance

```
<Cell name="XLDO">
  <Pins>
    <Pin name="VIN" direction="input" type="primary_power"/>
    <Pin name="VOUT" direction="output" type="internal_power"/>
    <Pin name="VSS" direction="output" type="primary_ground"/>
  </Pins>
  <CellParameters>
    <Parameter name="OVERHEAD_CURRENT" datatype="float" select="internal" value="1E-6" units="A"/>
    <Parameter name="REGULATION_EFFICIENCY" datatype="float" select="internal" value="0.66"/>
    <Parameter name="DROPOUT_EFFICIENCY" datatype="float" select="internal" value="0.95"/>
    <Parameter name="REGULATION_VOLTAGE" datatype="float" select="internal" value="3.3" units="V"/>
    <Parameter name="DROPOUT_VOLTAGE" datatype="float" select="internal" value="0.1" units="V"/>
    <Parameter name="SHUTDOWN_VOLTAGE" datatype="float" select="internal" value="2.5" units="V"/>
  </CellParameters>
  <Modes>
    <Mode name="REGULATION" when="V(VIN) >= REGULATION_VOLTAGE + DROPOUT_VOLTAGE
      <PowerSupplyContributor name="linearRegulator" source="VIN" sink="VSS" vin="VIN"
       vout="VOUT" overhead="OVERHEAD_CURRENT" efficiency="REGULATION_EFFICIENCY"
       alpha="0"/>
    </Mode>
    <Mode name="DROPOUT" when="V(VIN) < REGULATION_OUTPUT + DROPOUT_VOLTAGE">
      <PowerSupplyContributor name="linearRegulator" source="VIN" sink="VSS" vin="VIN"
       vout="VOUT" overhead="OVERHEAD_CURRENT" efficiency="DROPOUT_EFFICIENCY"
       alpha="0"/>
    </Mode>
    <Mode name="SHUTDOWN" when="V(VIN) < SHUTDOWN_VOLTAGE">
      <Expression name="LEAKAGE_POWER" source="VIN" sink="VSS">
        V(VIN) * LEAKAGE_CURRENT
      </Expression>
    </Mode>
  </Modes>
</Cell>
```

Session Outline

- Introduction (15 min), Nagu Dhanwada (IBM)
- Core Concepts of IEEE 2416-2025 – Digital and AMS Highlights (30 min), Akil Sutton (IBM)
- Real-World Applications – Industry Deep Dives (60 min), Eunju Hwang (Samsung), Tavares Forby (Qualcomm)
- System-Level Example: AI Accelerator with AMS Blocks (60 min), Daniel Cross (Cadence), Rhett Davis (NC State)
- Q&A (15 min)

AI

Security

Systems

EDA

Design

62 THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY CEDA® IEEE Council on Electronic Design Automation

sigda acm

Eunju Hwang

Samsung Foundry

Samsung Industry Deep Dive: UPM/IEEE 2416

Eunju Hwang, Principal Engineer, Samsung Foundry

June 22, 2025

SPONSORED BY CEDA sigda

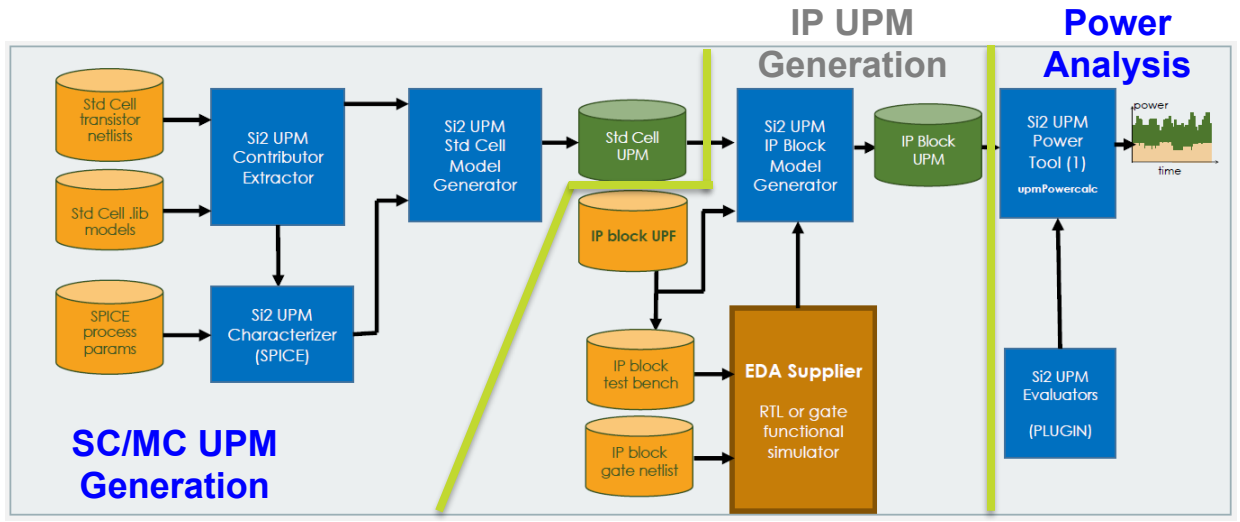
Motivation: Why Do We Need UPM?

- Too many PVT corners for power analysis
  - Different corners for timing analysis
  - Vary with customer application and scenarios
  - In-house power scaling utility: accuracy & maintenance issues
- Time consuming work for model changes
  - PDK/DK update scope → impact analysis with IP and design → feedback → PDK/DK update
  - Inefficient workflow
- Advanced thermal modeling
  - Power map generation / metal density map generation / thermal model creation
  - Power analysis for different scenarios should be done iteratively
  - Different temperature dependent leakage power modeling for each tool

We expect that UPM will reduce the library characterization cost/time and thus design TAT

Project Goals

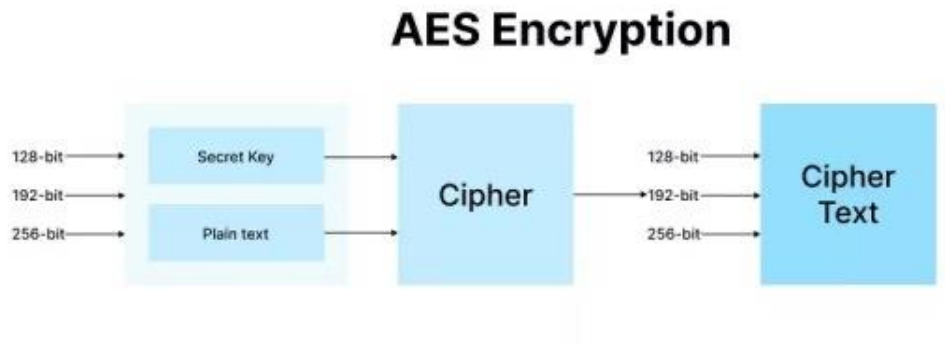
- Initial goal: To determine if UPM can replace the current library-based power analysis flow for standard cell leakage power in terms of accuracy



- 2-phase PoC for efficient flow enablement
  - Phase 1: UPM demonstration with an Open Core AES128 design and Nangate 15nm standard cell library – no intracell parasitics or Local Layout Effects (LLEs)
  - Phase 2: Extended demonstration with Samsung Foundry 14nm PDK and DK – includes intracell parasitics and LLEs

Benchmark Design Setup

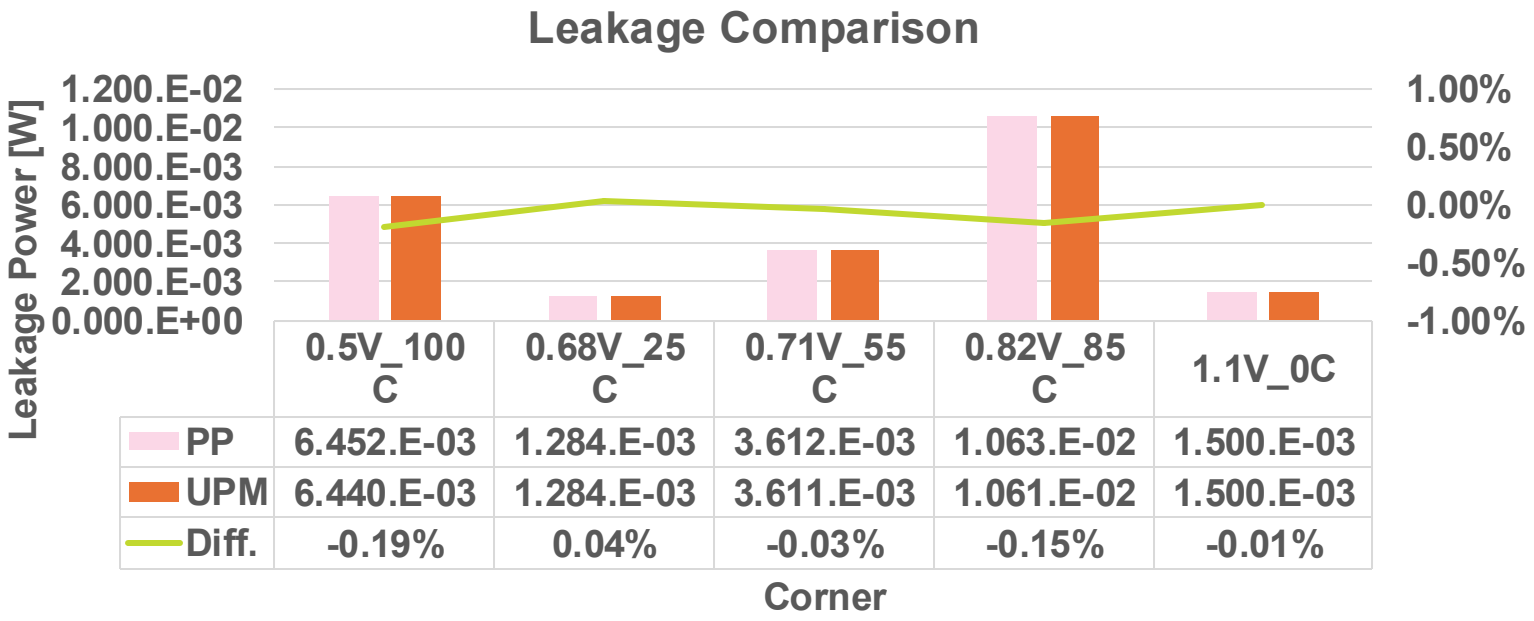
- Design: publicly available AES128 encryption core, standard cell count ~18k
  - Advanced Encryption Standard (Rijndael Algorithm) according to the NIST standard as documented in FIPS-197
  - https://opencores.org/projects/aes_crypto_core



- Synthesis and reference power analysis performed using commercial EDA tools
- UPM software suite used for UPM library generation and power calculation

Phase 1 Results: Leakage Power Accuracy

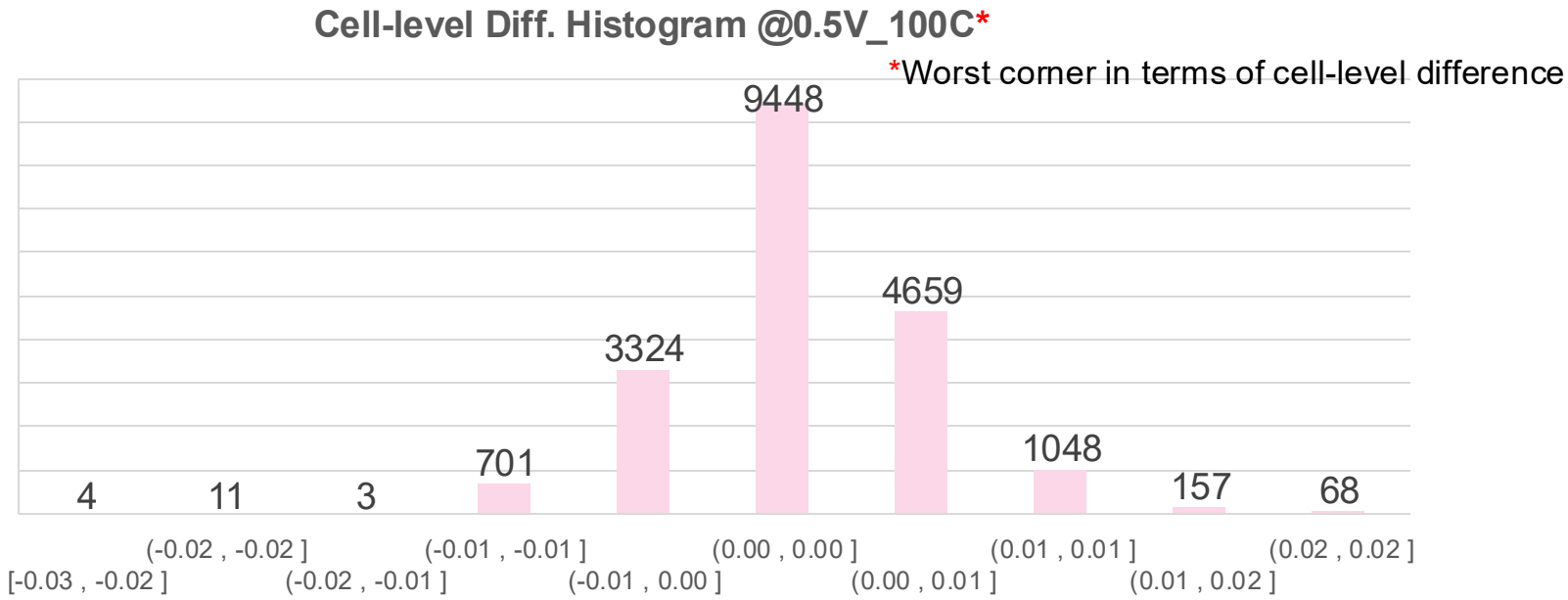
- Leakage power comparison between upmPowercalc and commercial sign-off power analysis tool
  - Voltage and temperature range for UPM: 0.5V – 1.1V in increments of 50mV, 0C – 100C in increments of 10C
  - Library corners for verification: TT / 0.5V_100C, 0.68V_25C, 0.71V_55C, 0.82V_85C, 1.1V_0C



Great accuracy for AES128 total leakage power: Max 0.19% and average 0.07% difference

Phase 1 Results: Per Cell Accuracy

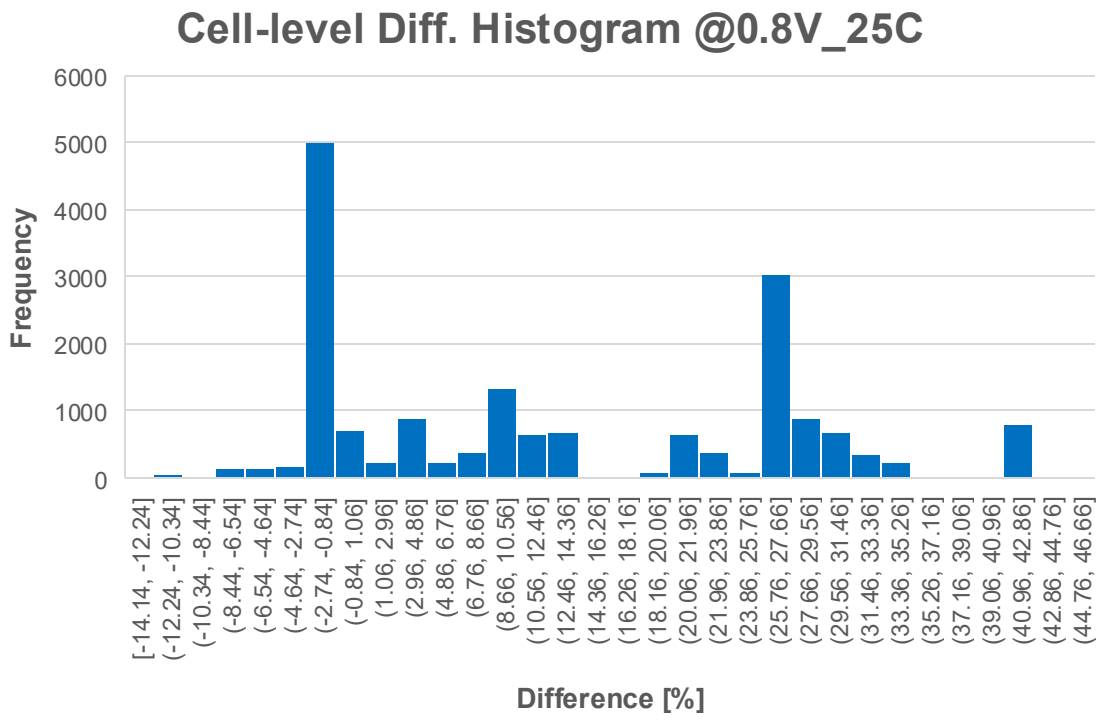
- Leakage power comparison between upmPowercalc and commercial sign-off power analysis tool
  - Voltage and temperature range for UPM: 0.5V – 1.1V in increments of 50mV, 0C – 100C in increments of 10C
  - Library corners for verification: TT / 0.5V_100C, 0.68V_25C, 0.71V_55C, 0.82V_85C, 1.1V_0C



Per instance difference: Relatively large differences up to 3.04%, observed at AND3/AND4/OR3/OR4
Small average and standard deviation: 0.079% and 0.53%

Phase 2: Initial Results with Samsung 14nm PDK/DK

- Initial results of the power comparison for AES 128 – not properly accounting for extracted parameters
  - The total power difference was about 2%, but the maximum difference was quite large: 44.92% when comparing on a per instance basis



LLE parameters should be properly modeled in the contributor netlists, but it is difficult to model the effects accurately without adding too much complexity

Phase 2: LLE Modeling Approaches

- Clustering (binning) of extracted parameters to limit their variation across the cell library
- Entire cell netlist as contributors
  - Accurate but less gain in library characterization cost

@TT/0.85V/25C

Approach	Contributors	Avg Error	Max Error	Std Dev
No LLE	~500	13.10%	44.92%	14.10%
Clustered Parameters	~11k	-0.59%	3.31%	0.97%
Full Netlist	~15k	0.005%	0.046%	0.016%

Phase 2 Results: Vth Dependency of Clustering

- Verification of the accuracy of the clustering approach for different V and T conditions @FFPG corner with **LVT cells** only case

	(V,T)	Min%	Max%	Avg%	StdDev%	Total%	Avg_Abs%	StdDev_Abs%
1	(0.52V, 83C)	-3.55	3.08	-0.27	0.68	0.03	0.61	0.40
2	(0.66V, 22C)	-4.06	3.56	-0.36	0.77	-0.01	0.71	0.47
3	(0.79V, 64C)	-3.73	3.17	-0.31	0.71	0.01	0.65	0.43
4	(0.84V, 148C)	-3.16	3.57	-0.18	0.59	-0.001	0.49	0.39
5	(0.86V, 57C)	-3.72	3.33	-0.22	0.74	0.04	0.64	0.44
6	(0.91V, -32C)	-4.09	3.44	-0.35	0.74	-0.08	0.64	0.50
7	(0.99V, 119C)	-3.31	2.73	-0.24	0.63	0.005	0.54	0.40

- Further investigation required for different threshold voltages
  - Multi-Vth design is common for low power
  - LLE level may be different for each threshold voltage device

Phase 2 Results: Vth Dependency of Clustering

- Verification of the accuracy of the clustering approach for different V and T conditions @FFPG corner with **HVT cells** only case

	(V,T)	Min%	Max%	Avg%	StdDev%	Total%	Avg_Abs%	StdDev_Abs%
1	(0.52V, 83C)	-3.47	3.68	0.28	1.12	1.17	0.84	0.79
2	(0.66V, 22C)	-3.06	3.42	0.06	0.87	0.51	0.65	0.57
3	(0.79V, 64C)	-3.04	2.76	0.23	0.93	0.90	0.70	0.64
4	(0.84V, 148C)	-2.86	5.91	0.44	0.81	0.995	0.66	0.65
5	(0.86V, 57C)	-1.94	4.60	1.45	1.03	1.79	1.46	1.02
6	(0.91V, -32C)	-5.62	2.26	-0.42	0.91	-1.12	0.47	0.89
7	(0.99V, 119C)	-2.62	3.33	0.19	0.64	0.63	0.49	0.46

Phase 2 Results: Vth Dependency of Clustering

- Verification of the accuracy of the clustering approach for different V and T conditions @FFPG corner with **RVT cells** only case

	(V,T)	Min%	Max%	Avg%	StdDev%	Total%	Avg_Abs%	StdDev_Abs%
1	(0.52V, 83C)	-3.68	2.72	-0.17	0.81	0.32	0.72	0.43
2	(0.66V, 22C)	-3.87	2.95	-0.13	0.94	0.46	0.81	0.50
3	(0.79V, 64C)	-3.65	2.70	-0.16	0.82	0.39	0.72	0.43
4	(0.84V, 148C)	-3.26	2.44	-0.11	0.68	0.24	0.29	0.35
5	(0.86V, 57C)	-3.45	3.62	0.47	1.1	0.70	1.09	0.49
6	(0.91V, -32C)	-1.61	2.21	-0.1	0.18	-0.20	0.14	0.15
7	(0.99V, 119C)	-3.32	2.01	-0.17	0.67	0.21	0.58	0.37

Phase 2 Results: Vth Dependency of Clustering

- Verification of the accuracy of the clustering approach for different V and T conditions @FFPG corner with **SLVT cells** only case

	(V,T)	Min%	Max%	Avg%	StdDev%	Total%	Avg_Abs%	StdDev_Abs%
1	(0.52V, 83C)	-3.17	5.60	-0.16	0.64	-0.09	0.47	0.46
2	(0.66V, 22C)	-3.61	3.25	-0.22	0.71	0.16	0.64	0.38
3	(0.79V, 64C)	-3.27	3.25	-0.17	0.61	0.09	0.52	0.36
4	(0.84V, 148C)	-5.01	9.38	-0.12	1.10	-0.59	0.79	0.78
5	(0.86V, 57C)	-3.28	3.00	-0.17	0.62	0.11	0.54	0.35
6	(0.91V, -32C)	-4.01	3.71	-0.31	0.79	0.10	0.72	0.44
7	(0.99V, 119C)	-3.08	6.13	-0.12	0.65	-0.17	0.49	0.44

- Clustering approach showed highly reliable results for different threshold voltages
  - |Min| < 5.7%, Max < 9.4%, |Avg|&Avg_Abs < 1.5%, |StdDev|&StdDev_Abs < 1.2%, |Total| < 1.8%

Phase 2: Interpolation Accuracy Case Study

- UPM libraries benefit from contributor concept for leakage characterization and ‘scaling’ using characterized values for given V and T conditions
  - Verification of the accuracy of the scaling method for different V and T conditions is required: @FFPG corner with SLVT and HVT cells only cases
  - Scaling: when the (V,T) condition does not exist in the characterized tables, **interpolation** is performed for leakage power analysis
- Different levels of sparseness of tables used in contributor models
  - Case A: use all voltage corners (from 0.5 to 1.0 with 50mV voltage resolution) and temperature corners (-40, -25, 0, 25, 55, 85, 105, 125, 150)
  - Case B: use 100mV voltage resolution and all temperature corners
  - Case C: use 150mV voltage resolution (0.5, 0.65, 0.8, 1) and all temperature corners
  - Case D: use 100mV voltage resolution and temperature resolution (-40, 0, 55, 105, 150)

Phase 2: Interpolation Accuracy Case Study

- Power calculation comparison across cases (**SLVT cells** only)

Condition (ffpg_)	Case	Min%	Max%	Avg%	Std Dev%	Avg_Abs%	Std Dev_Abs%	Ref. Total Power	PowerCalc Total Power	Diff%
0p5200v_83c	A	-3.202	5.531	-0.1908	0.6296	0.4713	0.459	0.001578	0.001576	-0.1188
	B	-3.212	5.531	-0.1976	0.6329	0.476	0.4616	0.001578	0.001576	-0.1229
	C	-3.223	5.531	-0.2036	0.6346	0.4785	0.4638	0.001578	0.001576	-0.1258
	D	-5.892	3.423	-3.045	0.9029	3.054	0.8717	0.001578	0.001536	-2.614
0p6600v_22c	A	-3.576	3.231	-0.2162	0.7121	0.6384	0.3825	0.0004659	0.0004666	0.1632
	B	-3.576	3.231	-0.2114	0.713	0.6378	0.3824	0.0004659	0.0004667	0.1729
	C	-3.576	3.247	-0.2099	0.7125	0.6368	0.3822	0.0004659	0.0004667	0.1732
	D	-2.365	6.78	2.116	1.355	2.117	1.353	0.0004659	0.0004749	1.947
0p7900v_64c	A	-3.273	3.198	-0.2497	0.6036	0.539	0.369	0.00184	0.00184	0.0242
	B	-3.273	3.251	-0.2459	0.6052	0.5386	0.3695	0.00184	0.00184	0.02824
	C	-3.273	3.251	-0.2453	0.6054	0.5386	0.3695	0.00184	0.00184	0.02878
	D	-4.462	2.197	-1.763	0.7212	1.769	0.7058	0.00184	0.001817	-1.243

Phase 2: Interpolation Accuracy Case Study

- Power calculation comparison across cases (**SLVT cells** only)

Condition (ffpg_)	Case	Min%	Max%	Avg%	Std Dev%	Avg_Abs%	Std Dev_Abs%	Ref. Total Power	PowerCalc Total Power	Diff%
0p8400v_148c	A	-4.706	9.701	0.1689	1.116	0.7551	0.8384	0.01069	0.01065	-0.3162
	B	-4.669	9.701	0.2017	1.126	0.7631	0.8522	0.01069	0.01065	-0.296
	C	-4.574	9.809	0.2865	1.125	0.7616	0.8765	0.01069	0.01066	-0.1954
	D	-4.365	9.916	0.4858	1.137	0.8468	0.9006	0.01069	0.01068	-0.02587
0p8600v_57c	A	-3.28	2.997	-0.1861	0.6199	0.5377	0.3602	0.001766	0.001768	0.1077
	B	-3.28	3.037	-0.1522	0.6266	0.5392	0.3537	0.001766	0.001768	0.1374
	C	-3.11	3.272	0.01042	0.6442	0.5478	0.339	0.001766	0.001771	0.2721
	D	-3.449	2.651	-0.4992	0.6082	0.6841	0.3887	0.001766	0.001763	-0.1514
0p9100v_m32c	A	-2.808	5.263	1.089	0.8172	1.109	0.7907	0.0001115	0.0001131	1.429
	B	-2.796	5.522	1.215	0.863	1.23	0.8419	0.0001115	0.0001132	1.479
	C	-2.522	12.04	2.616	2.064	2.617	2.063	0.0001115	0.0001138	2.052
	D	-0.3514	7.485	4.043	1.434	4.044	1.432	0.0001115	0.0001173	5.228
0p9900v_119c	A	-3.318	5.953	-0.3475	0.6446	0.5804	0.4465	0.008182	0.00815	-0.3832
	B	-3.318	5.966	-0.3368	0.6471	0.5759	0.4477	0.008182	0.008151	-0.3742
	C	-3.291	5.991	-0.3134	0.6447	0.5591	0.4487	0.008182	0.008155	-0.3328
	D	-0.7698	8.362	2.552	0.9442	2.553	0.9438	0.008182	0.008366	2.257

Phase 2: Interpolation Accuracy Case Study

- Power calculation comparison across cases (**HVT cells** only)

Condition (ffpg_)	Case	Min%	Max%	Avg%	Std Dev%	Avg_Abs%	Std Dev_Abs%	Ref. Total Power	PowerCalc Total Power	Diff%
0p5200v_83c	A	-3.581	3.403	0.1733	1.119	0.8575	0.7401	1.574e-06	1.59e-06	1.067
	B	-3.581	3.403	0.1733	1.119	0.8575	0.7401	1.574e-06	1.59e-06	1.067
	C	-3.423	3.962	0.3273	1.101	0.8266	0.7971	1.574e-06	1.592e-06	1.188
	D	-22.7	-6.233	-12.67	4.051	12.67	4.051	1.574e-06	1.392e-06	-11.54
0p6600v_22c	A	-2.722	3.569	0.3308	0.8239	0.6276	0.628	2.44e-07	2.457e-07	0.6996
	B	-2.722	3.569	0.3308	0.8239	0.6276	0.628	2.44e-07	2.457e-07	0.6996
	C	-1.482	6.367	2.02	1.212	2.025	1.205	2.44e-07	2.489e-07	2.035
	D	10.25	43.64	24.47	8.935	24.47	8.935	2.44e-07	3.03e-07	24.18
0p7900v_64c	A	-3.329	2.815	0.1119	0.9555	0.7634	0.5855	1.761e-06	1.774e-06	0.7192
	B	-3.329	2.815	0.1119	0.9555	0.7634	0.5855	1.761e-06	1.774e-06	0.7192
	C	-2.921	3.545	0.6305	0.9942	0.9437	0.7038	1.761e-06	1.781e-06	1.13
	D	-11.9	-1.878	-7.018	2.399	7.018	2.399	1.761e-06	1.643e-06	-6.693

Phase 2: Interpolation Accuracy Case Study

- Power calculation comparison across cases (**HVT cells** only)

Condition (ffpg_)	Case	Min%	Max%	Avg%	Std Dev%	Avg_Abs%	Std Dev_Abs%	Ref. Total Power	PowerCalc Total Power	Diff%
0p8400v_148c	A	-2.145	6.759	1.097	0.8161	1.104	0.807	2.395e-05	2.435e-05	1.674
	B	-2.145	6.759	1.097	0.8161	1.104	0.807	2.395e-05	2.435e-05	1.674
	C	-0.557	22.84	4.799	2.86	4.799	2.86	2.395e-05	2.502e-05	4.446
	D	-1.034	8.776	2.553	0.937	2.553	0.9367	2.395e-05	2.466e-05	2.949
0p8600v_57c	A	-2.195	3.493	1.064	0.9437	1.137	0.8551	1.844e-06	1.871e-06	1.445
	B	-2.195	3.493	1.064	0.9437	1.137	0.8551	1.844e-06	1.871e-06	1.445
	C	6.522	104.5	27.19	16.18	27.19	16.18	1.844e-06	2.29e-06	24.19
	D	-1.862	16.93	3.465	3.087	3.562	2.975	1.844e-06	1.904e-06	3.239
0p9100v_m32c	A	-2.613	5.87	3.131	1.588	3.133	1.584	4.309e-07	4.416e-07	2.469
	B	-2.613	5.87	3.131	1.588	3.133	1.584	4.309e-07	4.416e-07	2.469
	C	27.54	121.6	74.37	28.73	74.37	28.73	4.309e-07	7.522e-07	74.55
	D	-3.211	15.54	6.84	4.933	6.922	4.818	4.309e-07	4.571e-07	6.067
0p9900v_119c	A	-3.037	3.259	0.2232	0.8021	0.7322	0.3963	1.829e-05	1.84e-05	0.5533
	B	-3.037	3.259	0.2232	0.8021	0.7322	0.3963	1.829e-05	1.84e-05	0.5533
	C	-2.458	5.853	1.51	1.345	1.655	1.162	1.829e-05	1.86e-05	1.653
	D	4.247	11.05	7.32	0.9993	7.32	0.9993	1.829e-05	1.974e-05	7.908

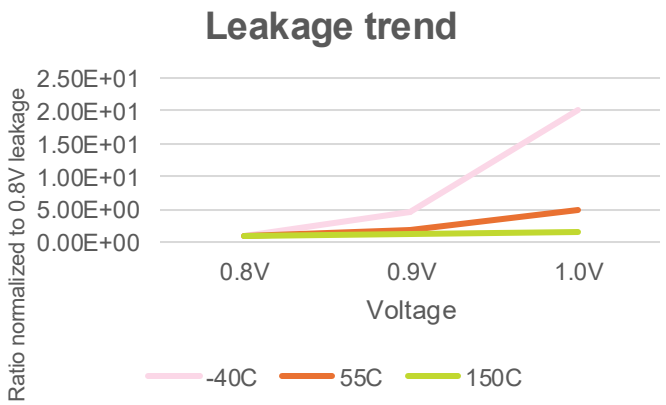
Nonlinear Leakage: Need for High-Order Interpolation

- It is shown that how the linear interpolation of voltage can lead to large errors under certain conditions
  - This highlights the limitations of scaling, but it can be improved by employing higher order interpolation schemes
- Next generation of upmPowerCalc will feature higher orders of interpolation for voltage and temperature
  - Example: Leakage represented by a contributor consisting of a single Off HVT NFET, displaying significant non-linear behavior with respect to voltage at -40C and 0.9V

Sample off NFET channel leakage currents:

Voltage (V)	-40C (A)	55C (A)	150C (A)
0.50	5.75261E-13	1.1354E-11	4.94604E-10
0.55	6.09293E-13	1.1835E-11	5.1432E-10
0.60	6.49543E-13	1.23444E-11	5.3485E-10
0.65	7.08685E-13	1.29127E-11	5.56491E-10
0.70	8.30666E-13	1.363E-11	5.79715E-10
0.75	1.14544E-12	1.47426E-11	6.05451E-10
0.80	1.98437E-12	1.68444E-11	6.35583E-10
0.85	4.09158E-12	2.12094E-11	6.73721E-10
0.90	8.96796E-12	3.03075E-11	7.26274E-10
0.95	1.93778E-11	4.85308E-11	8.03841E-10
1.00	4.00346E-11	8.31E-11	9.22859E-10

Note the 134% error in leakage by averaging currents at 0.8 and 1.0V compared to the value at 0.9V for the -40C case.



Ultimate Goal

- **Adding another EDA view would be a huge burden for ALL**
  - Users: flow enablement required for additional formats
  - Foundry/IP providers: increased maintenance and characterization costs
  - EDA vendors: additional support required for customer needs

- **Ultimate goal: To determine if UPM can replace the current library-based power and thermal analysis flow in terms of accuracy and efficiency**
  - IPs to be covered: standard cells, compiled memory
  - Dynamic and leakage power
  - Library extension model: e.g. Liberty library with a side-file containing augmented process sensitivity information representing the power behavior for a given physical SPICE parameter perturbation range

Future Steps

- Scope: UPM demonstration with Samsung Foundry 14nm PDK/DK
  - Phase 2-1: gate-level leakage power – complete
  - Phase 2-2: gate-level dynamic power – in progress
  - Test case: AES128

Task	Role
Library (internal_power, timing) delivery	Samsung Foundry
SAIF file generation	Samsung Foundry
UPM library generation for dynamic power	Si2
Power analysis with a commercial tool	Samsung Foundry
Power analysis with upmPowerCalc	Si2
Power comparison and modeling improvement	Si2 and Samsung Foundry

  - Phase 3: Leakage and dynamic power modeling of compiled SRAMs and development of the corresponding UPM-based flow for analyzing memory modules

Conclusion

- **Si2 and Samsung Foundry have evaluated a UPM-based power analysis flow using 14nm PDK/DK and AES128 design**
  - Focus: Gate-level leakage analysis for standard cells
  - Output: UPM library characterization and power analysis flow with proven accuracy
  - Demonstrated that using UPM, voltage and temperature scaling can be done more efficiently in terms of storage and computational requirements
- **The next step will focus on UPM-based dynamic power analysis of standard cells**
  - To achieve our adoption goals, it would be advantageous to have more proactive engagement from other UPTC members
- **The subsequent step will address leakage and dynamic power modeling of compiled SRAMs**

Case Study: Qualcomm SoCs

- AI workflow application
- Workload-aware modeling
- AMS interface integration

Qualcomm Case Study

UPM / IEEE 2416 Validation

Tavares Forby, Qualcomm

Acknowledgements

- Many thanks to Si2 for UPM partnership with Qualcomm.
- Acknowledging team members from Qualcomm and Si2 for contributions, support and regular status meetings to drive UPM adoption

- **Qualcomm Team**
  - Tavares Forby
  - Joe Kurian
  - Pritesh Johari
- **Si2 Team**
  - Leigh Anne Clevenger
  - Ali Sadigh
  - Jerry Frenkil

Motivation

- **Why Unified Power Modeling (UPM)?**
  - Qualcomm Technologies' Motivation
    - Bring about Standardization of **models** and **tools** used at system level
      - Power analysis currently performed at **system level** uses company-specific models/formats/tools
  - Benefits of UPM to Qualcomm Technologies Inc.
    - One Scalable Model:
      - Design Levels: from circuit to IP to HardMacro to Subsystem to SoC
      - Stages of Product: Architecture to RTL to Implementation Stages
      - Across PVT Corners
    - Non-proprietary format
    - Reduced complexity for scaling setup
    - Reduction in library characterization turn-around
    - Faster enablement of new PVT corners
    - Spice-accurate results (similar to Liberty)
    - Late binding PVT power analysis

Key Features of UPM

- **Unique Power Analysis Features**
  - Late binding PVT power analysis
  - Multiple data representations (scalar, tables, expressions, contributors)
  - Bottom-up approach to analyze power contributors for each cell
  - Separate representations for leakage (static) and dynamic power
  - Leakage power calculated with expressions and coefficients or leakage tables
- **Standards and Interoperability**
  - IEEE 2416 industry standard power modeling language for gate through system
  - Designed to interoperate with UPF / IEEE 1801
  - Built on XML

Use Models

- **Projection of Power KPIs**
  - As part of Architecture/Concept Exploration
  - Dynamic Power as function of Frequency, Bandwidth, PG Rail
  - Model as table for known parameters, and expression for parameter exploration
- **Estimate Days-of-Use (DoU)**
  - For ongoing projects, estimate DoU from the use case
  - Progressively improve accuracy, as UPM models from implementation roll in
- **Seamless Model Generation**
  - Standardization facilitates continuity & ecosystem development
  - Directly from EDA tool/flow
  - Stitched to up/down-stream

UPM Validation Plan

- Goal
  - Qualcomm Technologies & Si2 to demonstrate System Level Power can be represented by IEEE 2416
    - Represent ALL use-cases of two subsystems using IEEE2416
      - Show that ALL parameters can be represented through available knobs (expressions, tables, user defined params etc)
    - Create a simple utility to show rollup of power at system level for all the available subsystems
  - Secondary Goals: Show that the power model can be used to scale across corners
- Three Phase Validation Plan
  - Cell Level Validation
  - IP Block Level Validation
  - System Level Validation

Task	Dependency
Create Utilities to create template – Python library	Qualcomm Technologies, Si2
Understand & Gather various use cases	Qualcomm Technologies
Represent all use cases	Qualcomm Technologies, Si2
Check with Design Team	Qualcomm Technologies
Develop Utility to compute rollup power (PowerCalc)	Qualcomm Technologies, Si2
Testing for rollup Power – demonstrate Power Vector Match	Qualcomm Technologies, Si2

Cell Level Validation

Cell Level Validation

- Test vehicle: LFSR (Linear Feedback Shift Register)
- Each cell in LFSR was tested exhaustively
- UPM results compared to Liberty models

Cell Level Setup

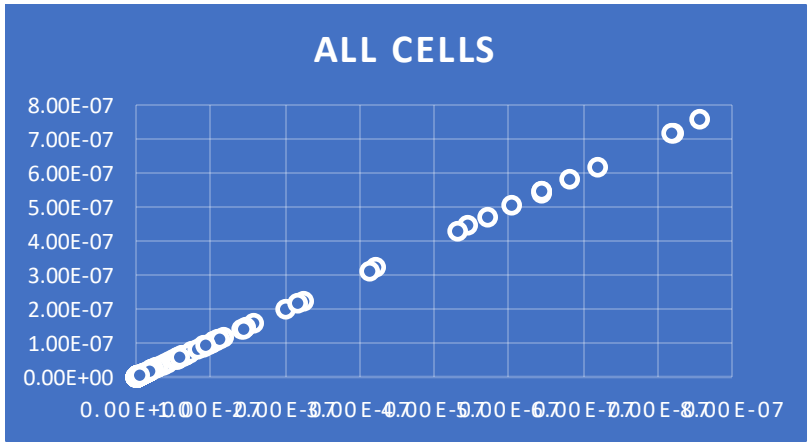
- Goal: Establish accuracy for 6 primitive cells used in LFSR under all logical and electrical conditions
- Contributors characterized for 10x9 V,T matrix
  - **VOLTAGES:** 0.505,0.559,0.613,0.667,0.721,0.775,0.829,0.883,0.937,0.991
  - **TEMPERATURES:** 0,15,30,45,60,75,90,105,125
- Leakage calculated for all possible states in various corners
  - 0.6v_0c
  - 0.750v_25c
  - 0.750v_60c
  - 0.94v_60c
  - 0.9v_125c

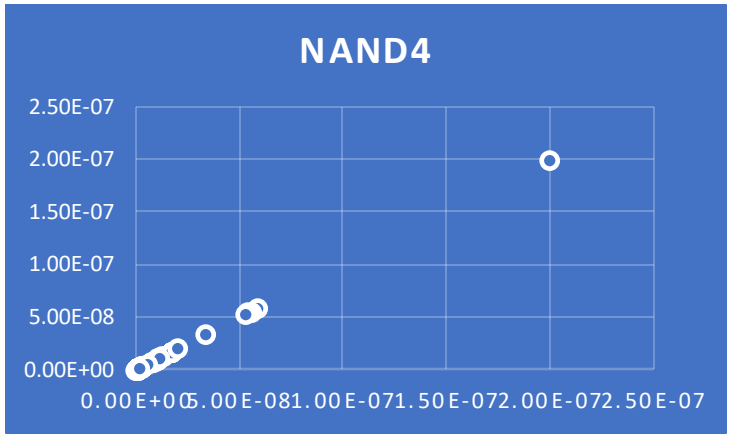
Cell	# of Modes
AND2	4
DFFR	16
INV	2
NAND4	16
NAND2	4
XOR2	4

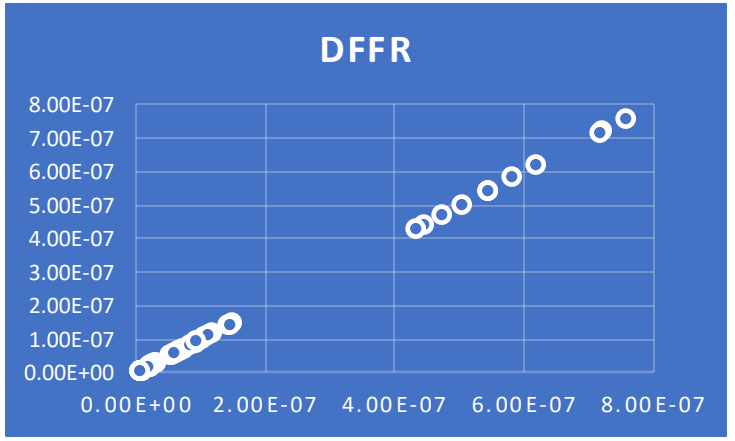
Cell Level Results

- upmPowerCalc results compared against Liberty data

Percentage Difference Statistics for All Cells			
Min	Max	Average	STDEV
-0.185	0.507	0.049	0.068







IP Block Validation

Qualcomm Technologies, Inc.

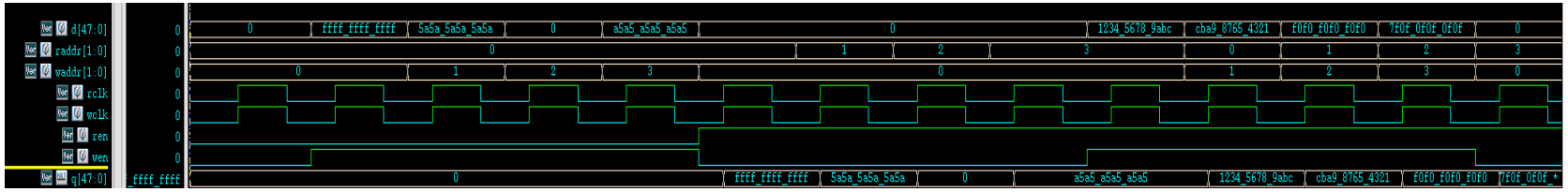
SPONSORED BY CEDA sigda

Block Level Validation

- Test vehicle: “TESTBLOCK” (Asynchronous FIFO)
- Larger design for more thorough, robust evaluation of end-to-end flow
- 1371 instance design (correlation checked for each individual instance)

IP Block Flow with TESTMACRO Gate Netlist

- Goal: Confirm accuracy using a production design IP block
- TESTMACRO
  - Netlist contains 1371 instances (User standard cells)
  - Two separate libraries used in design
- Functional Simulation
  - Simulated several different operating read/write modes on each FIFO port

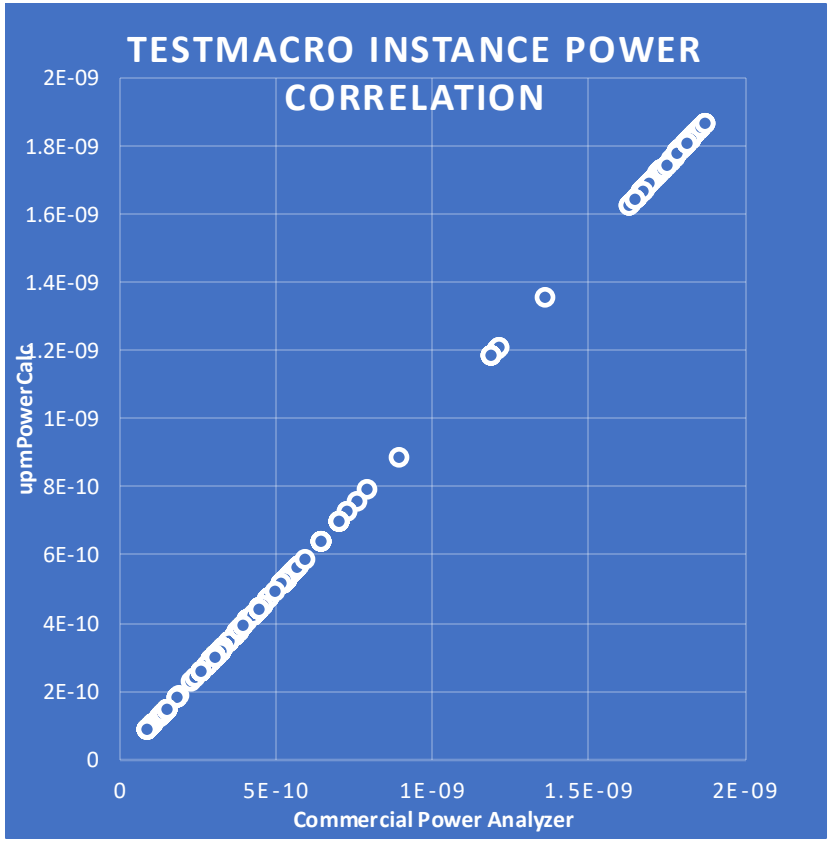


- Netlist results compared against a commercial power analyzer

IP Block Results for TESTMACRO Netlist

- upmPowerCalc results compared against a commercial power analysis tool
- Instance by instance and total Power comparison

Percentage Difference Statistics for All Instances				
Min	Max	Average	STDEV	Total Netlist Power
-0.06%	0.03%	-0.0006%	0.005%	0.0005%

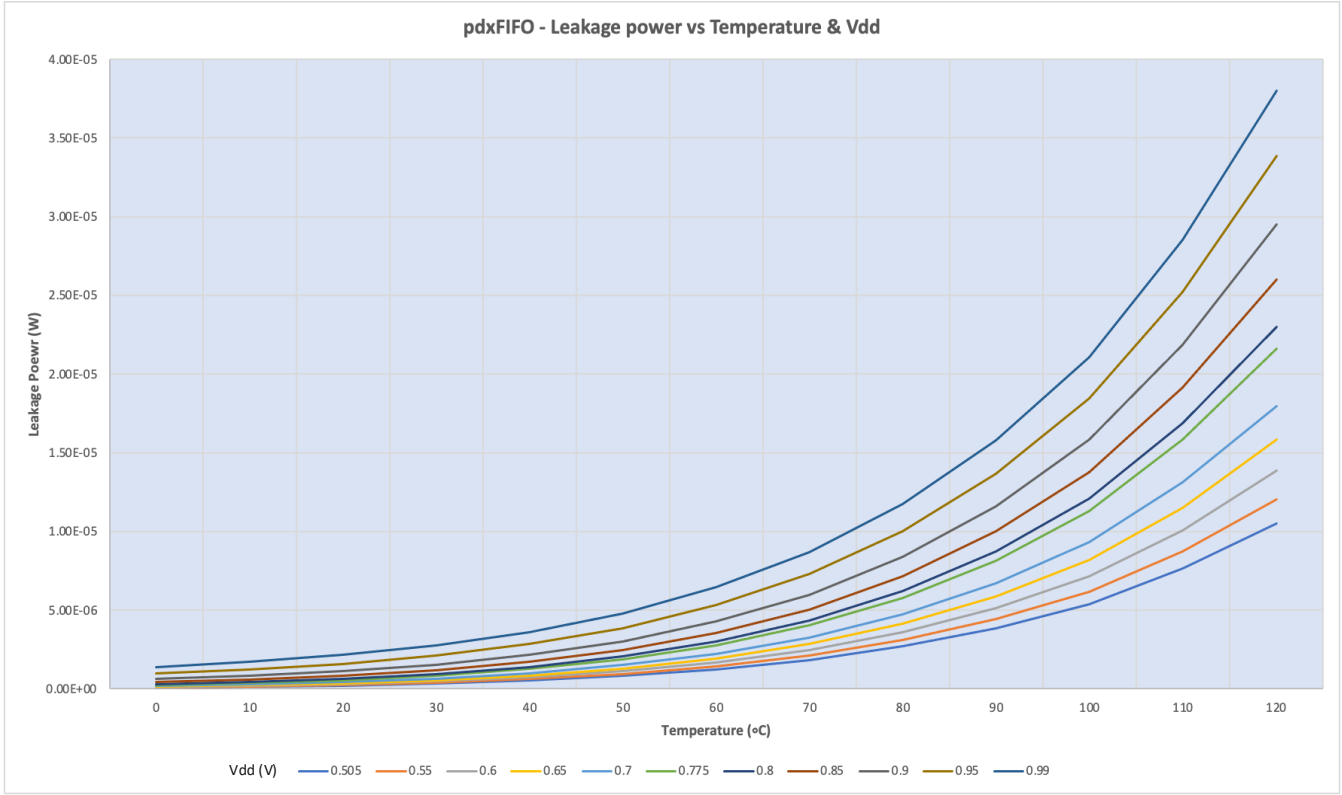


IP Block Abstract Power Model Generation

- UPM supports gate level models as well as abstract IP Block level models
  - IP Block UPM models can be used interchangeably across system level modeling
- Abstract IP Block created automatically from TESTMACRO netlist
  - UPM Standard enables creation of compressed abstract models from complex IP
  - IP-Block model uses aggregated V, T independent power contributors
  - Optional methodology for improved runtime analysis
- Netlist interconnectivity discarded in the abstraction process
  - Aids compression
  - Obfuscates block functionality
- Total runtime of 7 minutes to generate abstract model

TESTMACRO 143 Point Leakage Characterization Plot

- Entire characterization run using a single V,T independent model
  - 13 different Temperatures
  - 11 different Voltages
- Easy to generate
  - upmPowerCalc scripted to iterate over V,T array
- Fast time to results
  - Each datapoint took 4s to evaluate



System Level Validation

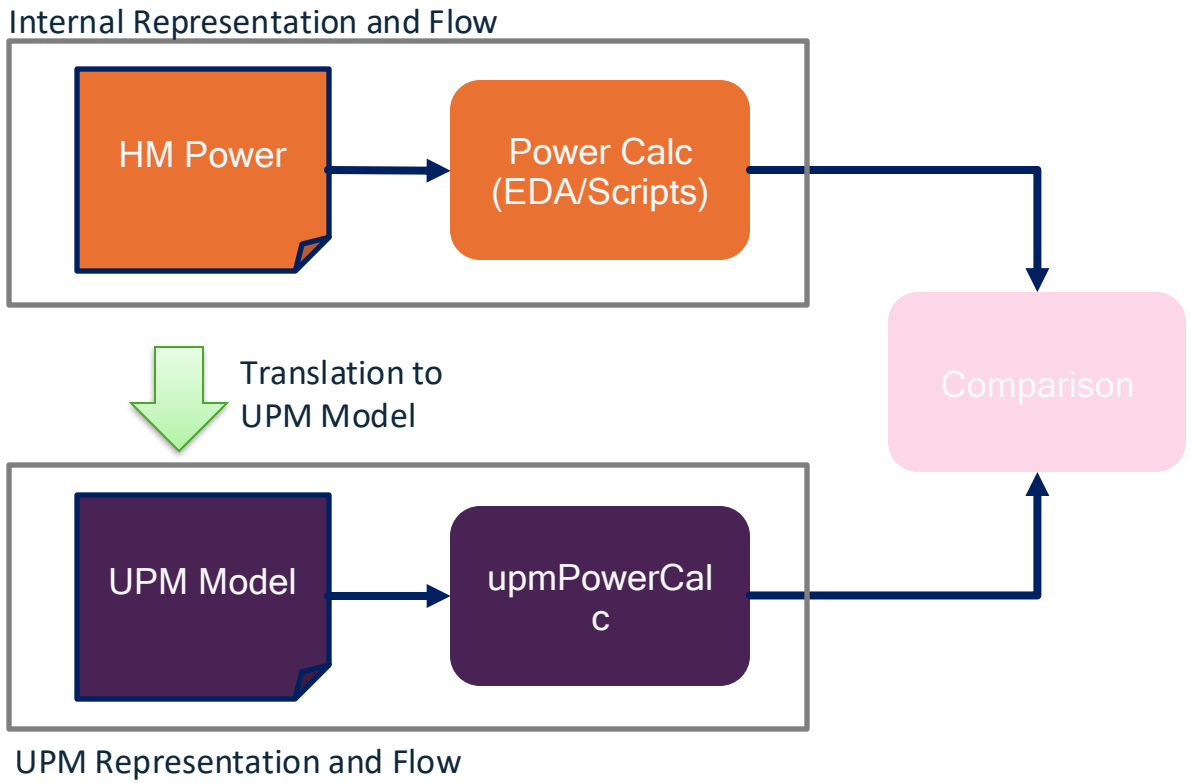
Qualcomm Technologies, Inc.

SPONSORED BY CEDA sigda

System Level Validation

- HardMacro and Subsystem Level Design Validation (Advanced Tech Node)
- Design size ranging from 1M-2M cells with large number of macros.
- HM parameterized power and state dependent power captured in UPM
- HM Power Calculation validated against upmPowerCalc.

System Level Validation



Validation Summary

Parameter	Design A	Design B
Std cell count	1.7 m	900k
Net count	1.8 m	1 m
Macro count	130	51
Area	0.2 mm sq	0.9 mm sq
No. of power domains	12	1
Internal power model	Equation based representation	Table based representation
UPM model	Equation based representation	Scalar representation
Validation Status	• Power closely matching with internal power model for most user input combinations with UPM base power calculation. • Apples to apples power correlation couldn't be achieved due to complexity of converting internal model to UPM format manually.	• Power per state matching exactly between internal model and UPM based power calculation. • State names need to be modified as UPM does not support special characters for the same.

Example UPM Power Expression

```
<ModelExpressions>
   <Expression name="global_exp_1"> 1.0 + ff_percent/100. </Expression>
   <Expression name="global_exp_2">
      (cf1 + (cf2 + cf3)*jumper_setting)
   </Expression>
   <Expression name="global_exp_3"> (cf1*cf2 - cf3*cf4/cf5) </Expression>
</ModelExpressions>
```

```
<State name="State_A">
   <DynamicPower>
     <ExpressionParameters>
       <Parameter name="Ld" value="-0.2"/>
     </ExpressionParameters>
     <Expression name="dyn_pwr">
       ((global_exp_2(cf1=1.5, cf2=2.1, cf3=0.003) &lt; global_exp_1()) ? 0 :
         global_exp_1(ff_percent=Ld))
     </Expression>
   </DynamicPower>
   <StaticPower>
     <Expression>
       (cparm3 > 0) ? (cparm1 - cparm2) : (cparm4 == 1) ?
         (global_exp_2(cf1=1.5, cf2=2.1, cf3=0.003) +
         global_exp_3(cf1=0.6702, cf2=1.15, cf3=1.0, cf4=0.96, cf5=0.4018)) :
         (global_exp_2(cf1=3.6, cf2=5.1, cf3=0.22) +
         global_exp_3(cf1=1.19, cf2=2.13, cf3=1.08, cf4=1.206, cf5=0.5198))
     </Expression>
   </StaticPower>
</State>
```

Example UPM Scalar Power Representation

```
<State name="Off">
    <StaticPower units="mW" value="0.0" />
    <DynamicPower units="mW" value="0.0" />
</State>
<State name="Standby">
    <StaticPower units="mW" value="0.01" />
    <DynamicPower units="mW" value="0.002" />
</State>
<State name="LowPower">
    <StaticPower units="mW" value="0.03" />
    <DynamicPower units="mW" value="0.1" />
</State>
<State name="Balanced">
    <StaticPower units="mW" value="0.04" />
    <DynamicPower units="mW" value="0.42" />
</State>
<State name="Performance">
    <StaticPower units="mW" value="0.05" />
    <DynamicPower units="mW" value="0.87" />
</State>
```

Conclusions: UPM Proof of Concept established

- **UPM power models auto-generated** for Qualcomm std cells
  - Each model is V and T independent: values for V and T set at runtime
- **Excellent Accuracy demonstrated** on two different designs
  - Accuracy demonstrated for overall design and instance by instance
- **Abstract UPM power model auto-generated** for TESTMACRO (Block)
  - Abstracted models reduce overall runtime and aid design security
- **System level UPM support validated**
  - Two complex design power model validated through UPM

Lesson Learnt

- Use of automation required to convert between internal scripts and UPM library. Manual translation of internal power models to UPM model is time consuming and error prone.
- Structures used in the testcase led to improvements in the IEEE 2416 specs
- Complex, nested “if” expressions were required, and are supported through use of ternary operators
- Power state names with special characters had to be modified as not currently supported in IEEE 2416 spec. Review in future, as changing state name is ok. All complex special character support is not must.
- Reconciling UPM power models at different design abstraction levels required review.
- Dynamic energy validation for gate level requires further steps. Although supported through UPM, current validation focused on leakage.
- Minor differences in system validation results are due to translation issues from internal scripts
  - Independent calculation of power based on same equations used in UPM libraries matched the results of upmPowerCalc

Next Steps

- Need EDA vendor tool support to write out UPM models at different design abstraction levels
  - Gate Level: Contributor extraction automation, Contributor simulation and UPM representation
  - System Level: UPM Model creation 1) extraction from block simulation 2) with user spec
- Need EDA vendor tool support to consume UPM power models in design flows.
  - Support mixed use of UPM and Liberty.
- Utilities to calculate on-demand power based on UPM library for a given PVT.
- Utilities for process variation power regression support based on UPM library
- State names should be able to support special characters (Although not currently gating).
- Automate translating internal power model to UPM format

Session Outline

- Introduction (15 min), Nagu Dhanwada (IBM)
- Core Concepts of IEEE 2416-2025 – Digital and AMS Highlights (30 min), Akil Sutton (IBM)
- Real-World Applications – Industry Deep Dives (60 min), Eunju Hwang (Samsung), Tavares Forby (Qualcomm)
- System-Level Example: AI Accelerator with AMS Blocks (60 min), Daniel Cross (Cadence), Rhett Davis (NC State)
- Q&A (15 min)

Full Example: AI Accelerator with AMS Blocks

- Target architecture
- Compute core, regulator, PLL

Rhett Davis

North Carolina State University

Daniel Cross

Cadence Design Systems

System-Level Power Examples for IEEE-2416/UPM

W. Rhett Davis (North Carolina State University) and Daniel Cross (Cadence Design Systems)

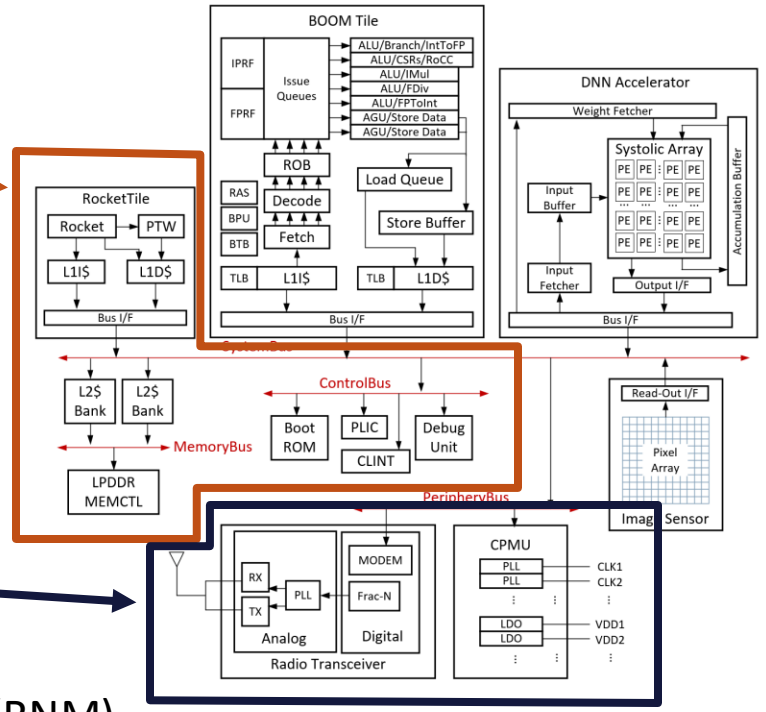
2025 Design Automation Conference

SPONSORED BY CEDA sigda

System Introduction

Varying levels of design model abstraction

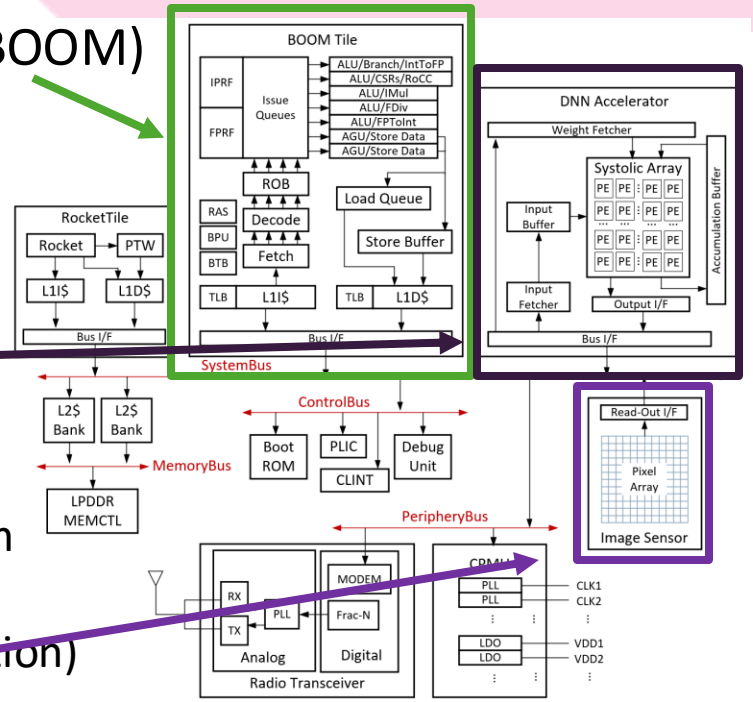
- Demonstration system to show the full range of UPM capabilities
- UCB RISCV RocketChip
  - Existing Digital Modeling
  - RTL and ISS (Spike) w/ SystemC wrapper and TLM socket
  - UPM Instruction-Level 15nm OCL Power Model extracted for DAC & ISLPED 2020
- RF Transceiver System
  - New Analog Modeling Capabilities
  - System-Verilog Real Number Model (RNM)
  - Radio model simplified from a proprietary model from Cadence
  - Clock & Power Management Unit included, due to similarities



Partially Included Aspects

Varying levels of design model abstraction

- UCB RISCV Out-of-Order Machine (BOOM)
  - RTL and ISS/SystemC TLM available
  - State-of-the-art Power Modeling with MCPAT (Z. Wang dissertation)
- Neural Network Accelerator
  - Small array HLS Model Available (Stanford Chimera)
  - Larger arrays simulated with Scale-Sim
  - Power model extracted for 45nm OCL and scaled to 15nm (Q. Zhao dissertation)
- Image Sensor
  - "What if" power model from literature review



Our Vision

System-level power estimation based on a variety of models and actual use-case scenarios.

- Common modeling language
- Common framework

Regardless of design abstraction

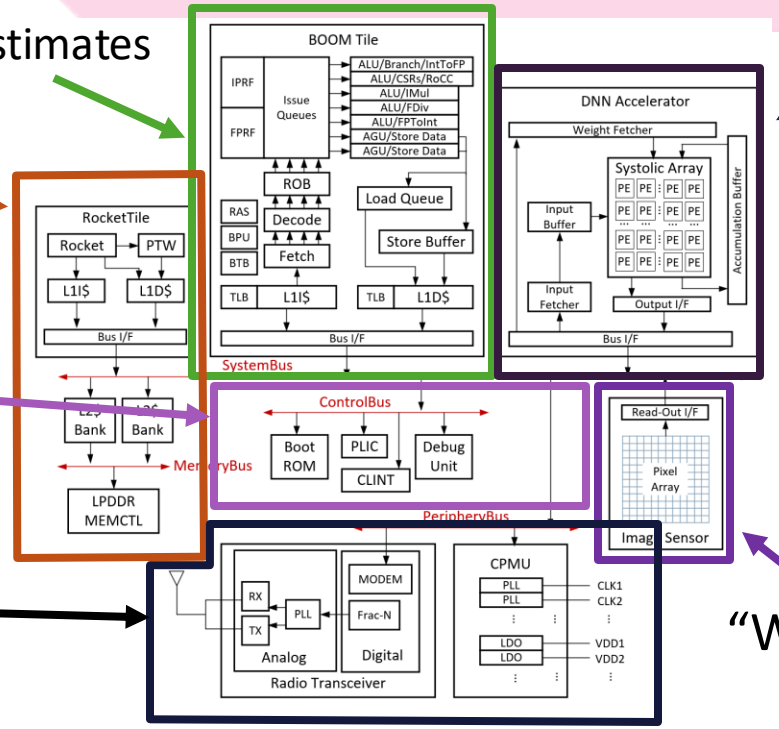
MCPAT Estimates

Per-instruction model

(SPIKE simulator)

Standard-cell derived models

Hand-coded model derived from SPICE simulations



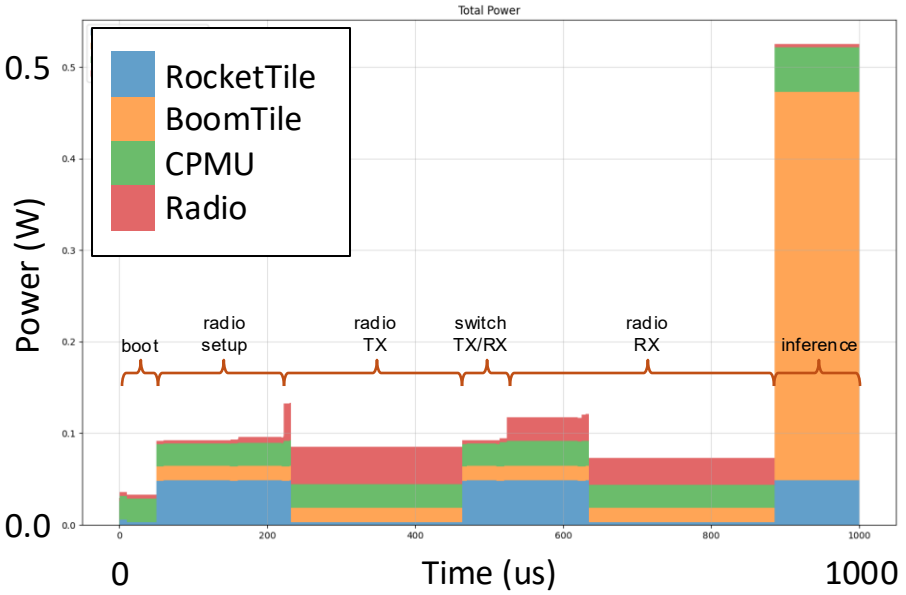
Scaled Zhao power model

"What if" model

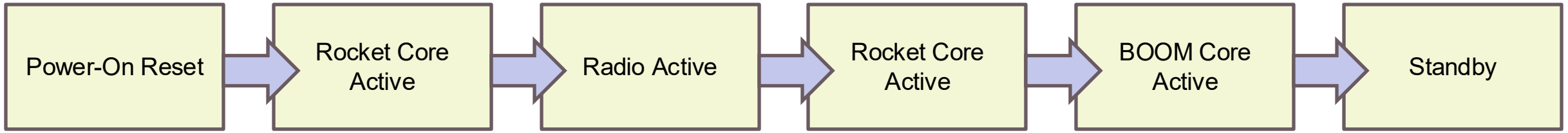
Explore power impacts of a variety of use cases and architectural options

Our Goals

- Create a high-performance (i.e. fast) system simulation that will allow rapid iteration of use cases, architecture choices, and sequencing strategies for our demonstration system.
- Include a variety of IEEE 2416 power models for the system components
- Use simulation output (VCD waveforms) to feed power estimates
- Include at least one realistic scenario
- Make the demonstration project available on favorable license terms



Our Status: System State and Power Sequencing



- Hold rest of system in reset
- Establish stable power and clock

- Set up Radio
- Schedule BOOM core and peripherals

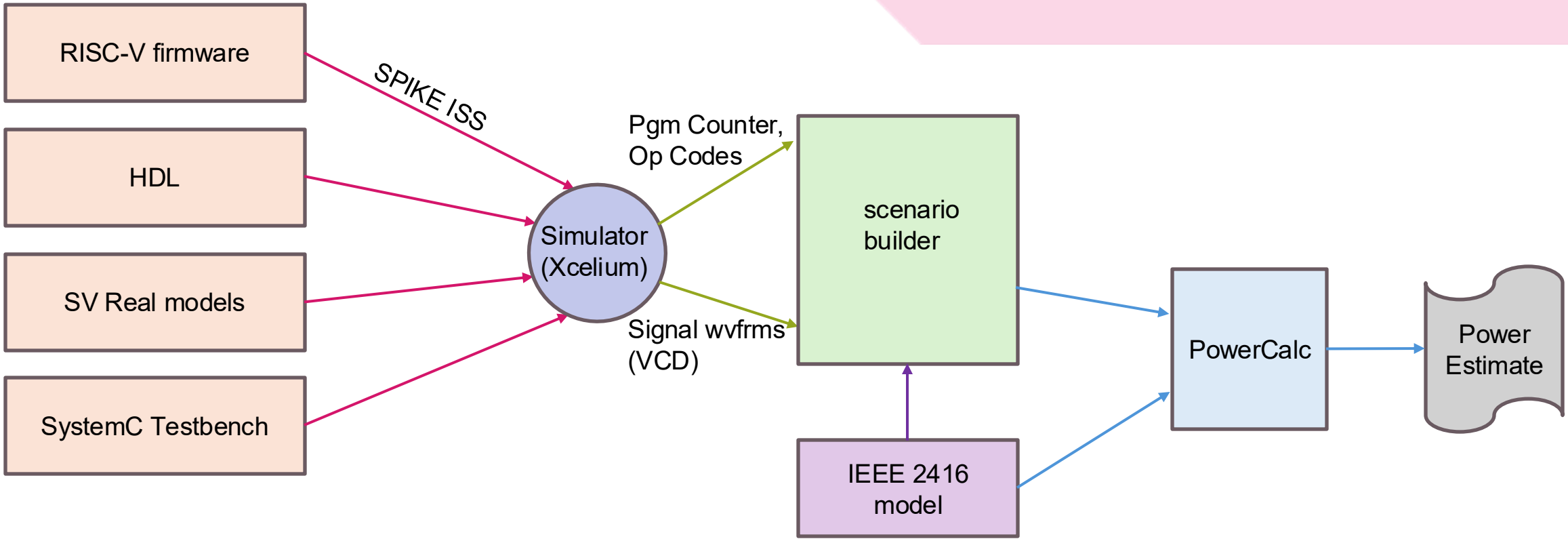
- Transmit request for data
- Receive response data
- Write data to memory

- Spool up PLL for BOOM Clock
- Pass data pointer to BOOM core

- Pre-process data
- Pass data to inference engine
- Update pixel display

- Rocket Core waits for new activity request

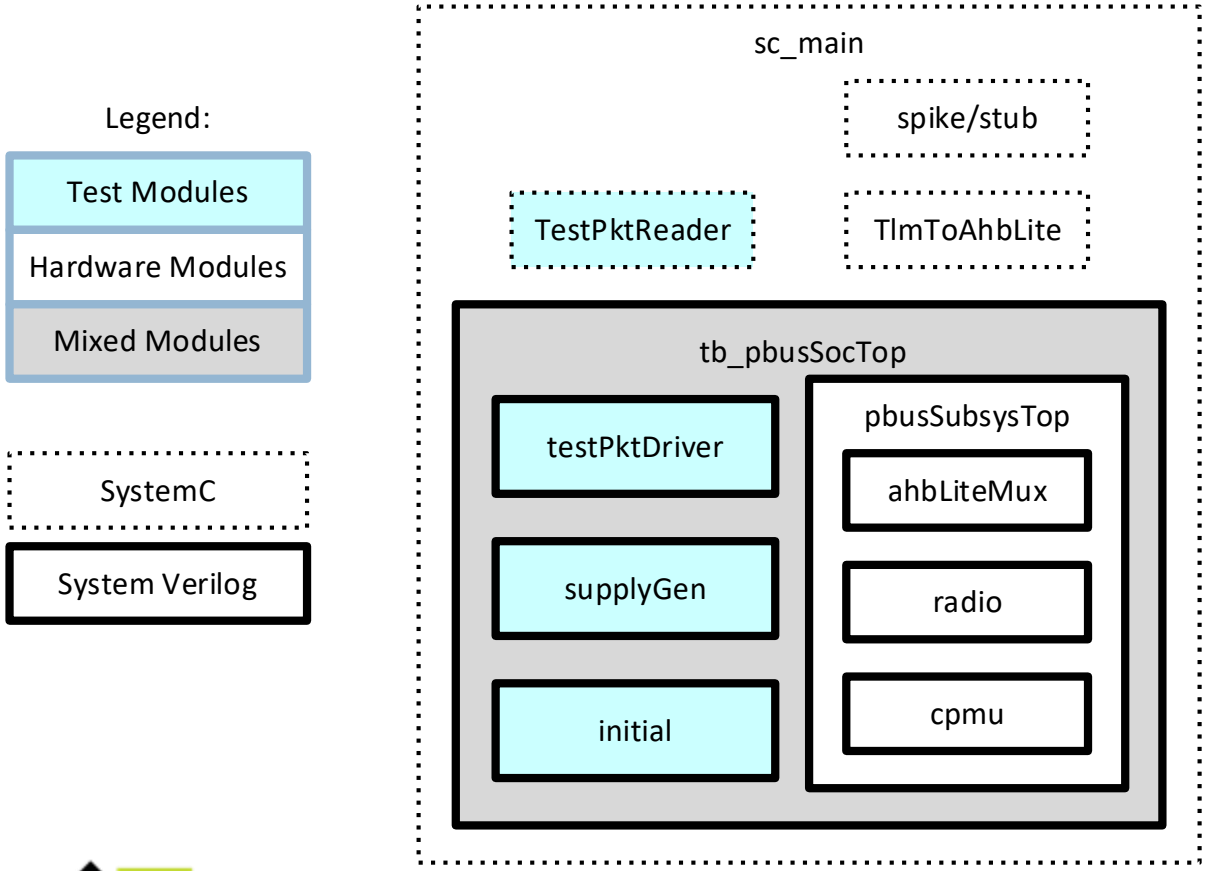
Our Status: Simulation



Simulate to generate activity based on system configuration and software.

Process against IEEE 2416 model to generate power estimates.

Simulation Structure



- Tests driven from SystemVerilog
- Top-Level in SystemC
- System-Level Models for SystemC aspects
  - Power expressions for states
  - Parameters changes in VCD
- Cell-Level Models for SystemVerilog RNM modules
  - Power expressions for modes
  - Energy expressions for events
  - Expressions triggered by “when” conditions
  - EvalMode() & EvalEvent() call cell-level models from system-level states

Real Number Behavioral Models for Analog Components

- Battery system power source (4V nominal)
- PowerOn Reset, Voltage Reference, Crystal clock reference (32MHz)
- Low DropOut voltage regulator (x5) – real valued supply nets
- Receiver, Transmitter (simple baseband models) RF->bits, bits->RF, 1 Mb/s
- PLL – VCO, XOR ph det, filter, divider
- Clock PLL – same as RF PLL

Power Models for Analog Components

rxFE.upm

```
<Modes>
 <Mode name="standby" when="AND ( (V(VDD) > 1.2)  (Not AND(rfEn adcEn)))">
      <Expression name="Standby_Pstatic1" source="VDD" sink="VSS">
          1e-9 + (Temperature-290)*(V(VDD)-1.2)*20e-12
      </Expression>
 </Mode>

 <Mode name="adcWarm" when="AND ( (V(VDD) > 1.2) adcEn)">
   <Expression name="warm_Pstatic1" source="VDD" sink="VSS">
      1.5e-3 + (Temperature-290)*(V(VDD)-1.2)*20e-4
   </Expression>
 </Mode>

 <Mode name="run" when="AND ( (V(VDD) > 1.2) adcEn rfEn)">
   <Expression name="run_Pstatic1" source="VDD" sink="VSS">
      6e-3 + (gain*50e-6) + (Temperature-290)*(V(VDD)-1.2)*1e-3
   </Expression>
 </Mode>
</Modes>

<Events>
 <Event name="adc_clk" when="AND ( (V(VDD)>1.2) adcEn " inputPin="clk" inputTransition="rising">
      <Expression name="adc_Pdyn" source="VDD" sink="VSS">
         V(VDD) * V(VDD) * 250e-12 + (Temperature-290)*(40e-9)
      </Expression>
 </Event>
 <Event name="lo_edge" when="AND ( (V(VDD)>1.2) rfEn )" inputPin="vco_out\" inputTransition="either">
      <Expression name="lo_edge_Pdyn" source="VDD" sink="VSS">
         V(VDD) * V(VDD) * 5.0e-12 + (Temperature-290)*(20e-12)
      </Expression>
 </Event>
</Events>
```

Power Models for Analog Components

rxFE.upm

```
<States>
<State name="adcWarm">
  <StaticPower>
    <Expression name="warm_Pstatic">
      EvalMode(adcWarm.warm_Pstatic1)
    </Expression>
  </StaticPower>
  <DynamicPower>
    <Expression name="warm_Pdyn">
      (EvalEvent(adc_clk.adc_Pdyn) * Fclk)
    </Expression>
  </DynamicPower>
</State>

<State name="run">
  <StaticPower>
    <Expression name="Sys_run_Pstatic">
      EvalMode(run.run_Pstatic1)
    </Expression>
  </StaticPower>
  <DynamicPower>
    <Expression name="Sys_run_Pdyn">
      (EvalEvent(adc_clk.adc_Pdyn) * Fclk) + (EvalEvent(lo_edge.lo_edge_Pdyn) * FLO)
    </Expression>
  </DynamicPower>
</State>
</States>
```

Use Modes and Events to compose State-based power calculations

Power Models for Analog Components

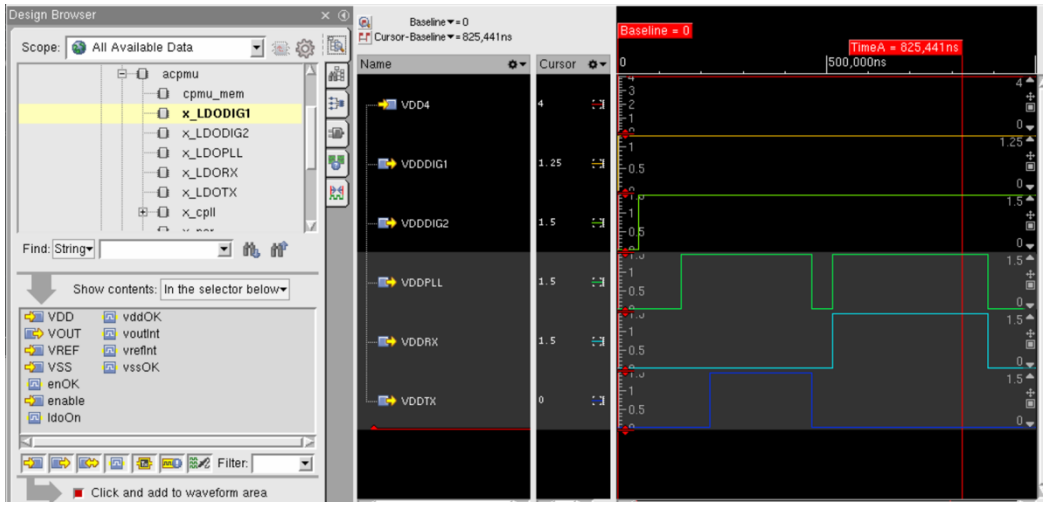
anaPLL.upm

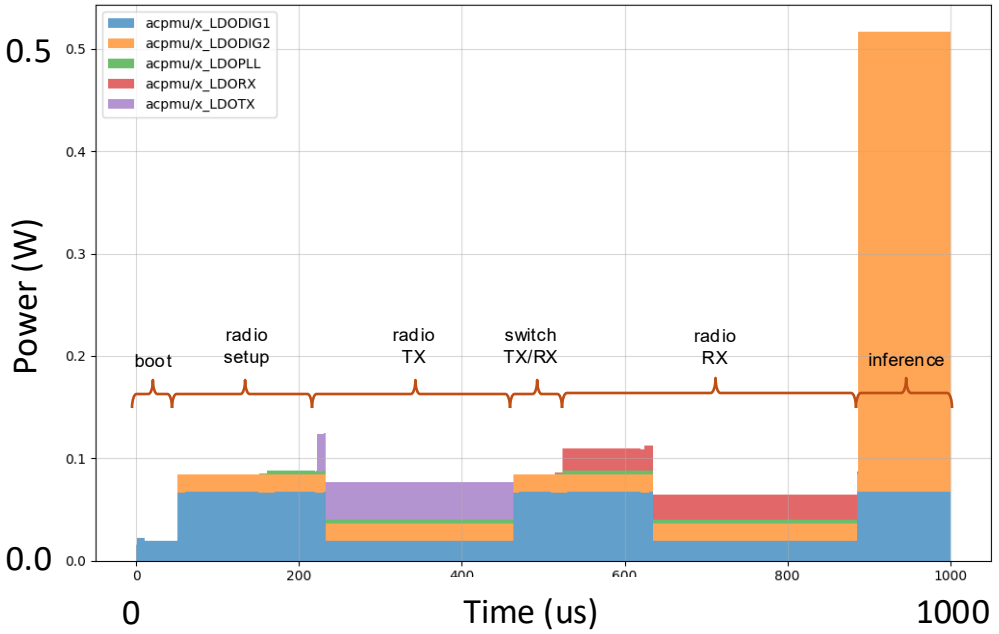
```
<Cell name="vco_rf">
   <Modes>
      <Mode name="Active" when="V(vdd) &gt; VddThreshold
&amp;&amp; vcoOn==1 &amp;&amp; supplyOK==1">
         <ExpressionParameters>
            <Parameter name="iActive" value="801e-6
         </ExpressionParameters>
         <Expression name="Active_Pstatic1" source="
sink="vss">
            iActive * V(vdd)
         </Expression>
      </Mode>
      <Mode name="Standby" when="V(vdd) &gt; VddThr
&amp;&amp; vcoOn==0 &amp;&amp; supplyOK==1">
         <ExpressionParameters>
            <Parameter name="iStandby" value="1e-6"
         </ExpressionParameters>
         <Expression name="Standby_Pstatic1" source=
sink="vss">
            iStandby * V(vdd)
         </Expression>
      </Mode>
   </Modes>
 </Cell >
```

```
<Cell name="xorPhDet">
   <Modes>
      <Mode name="Active" when="V(vdd) &gt; VddThreshold &amp;&amp; pdOn==1
&amp;&amp; supplyOK==1">
         <ExpressionParameters>
            <Parameter name="iActive" value="201e-6"/>
         </ExpressionParameters>
         <Expression name="Active_Pstatic1" source="vdd" sink="vss">
            iActive * V(vdd)
         </Expression>
      </Mode>
      <Mode name="Standby" when="V(vdd) &gt; VddThreshold &amp;&amp;
pdOn==0 &amp;&amp; supplyOK==1">
         <ExpressionParameters>
            <Parameter name="iStandby" value="1e-6"/>
         </ExpressionParameters>
         <Expression name="Standby_Pstatic1" source="vdd" sink="vss">
            iStandby * V(vdd)
         </Expression>
      </Mode>
  </Modes>
</Cell >
```

Voltage Regulator Modeling

- UPM also enables calculation of power delivered by regulators
- Demo system includes 5 regulators with power from a 4V battery
- Voltages from SystemVerilog RNM sim.

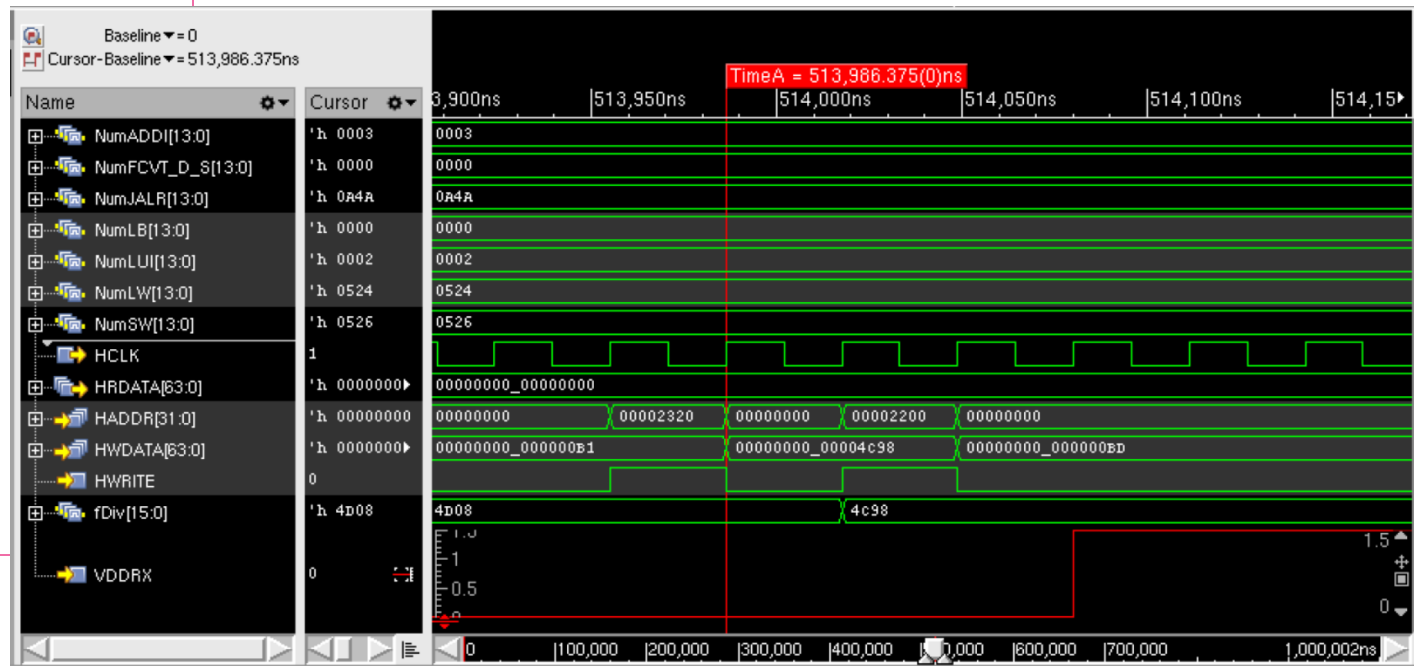




Simulating Actual Microcontroller Code

xact.c

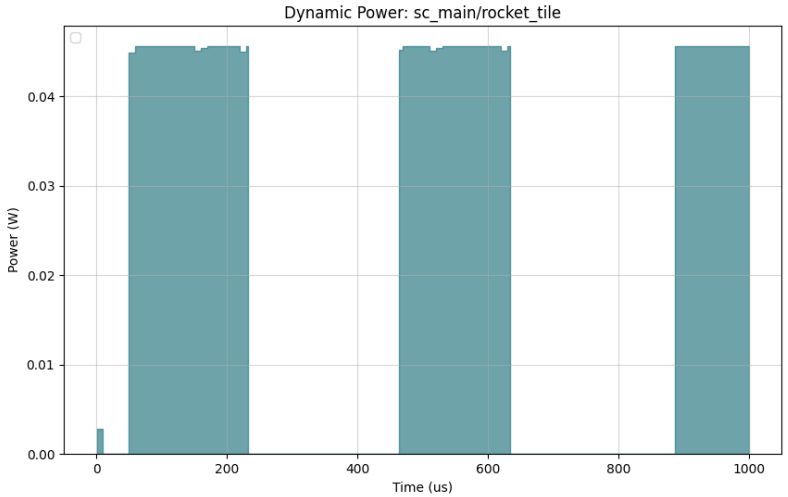
```
// @ 5 ns WRITE   0x1   0x2200 0xb1
cp=(volatile char *)0x60002200;
*cp=0xb1;

// + 50 us WRITE 0x8   0x2320 0x0000000000004c98
wait(50);
llp=(volatile long long *)0x60002320;
*llp=0x0000000000004c98;

// @ 5 ns WRITE   0x1   0x2200 0xbD
*cp=0xbD;

// + 10 us WRITE   0x1   0x2300 0x11
wait(10);
cp=(volatile char *)0x60002300;
*cp=0x11;

// + 100 us WRITE   0x1   0x2300 0x13
wait(100);
*cp=0x13;

// @ 5 ns WRITE   0x1   0x2310 0x08
cp=(volatile char *)0x60002310;
*cp=0x08;

// + 260 us  WRITE 0x1   0x2300 0x00
wait(260);
cp=(volatile char *)0x60002300;
*cp=0x00;
```



System-Level Dynamic Power Models

- Dynamic Power varies with workload but can be predicted with events (*e.g.* xacts., cache-fills)
- To predict power for a complex design:
  - decompose into a set of events
  - characterize the capacitances for each event (*i.e.* energy contributors)
  - count events, estimate power
- RocketTile model uses a capacitance-per-instruction model
- Other models currently use simple capacitance-per-cycle model

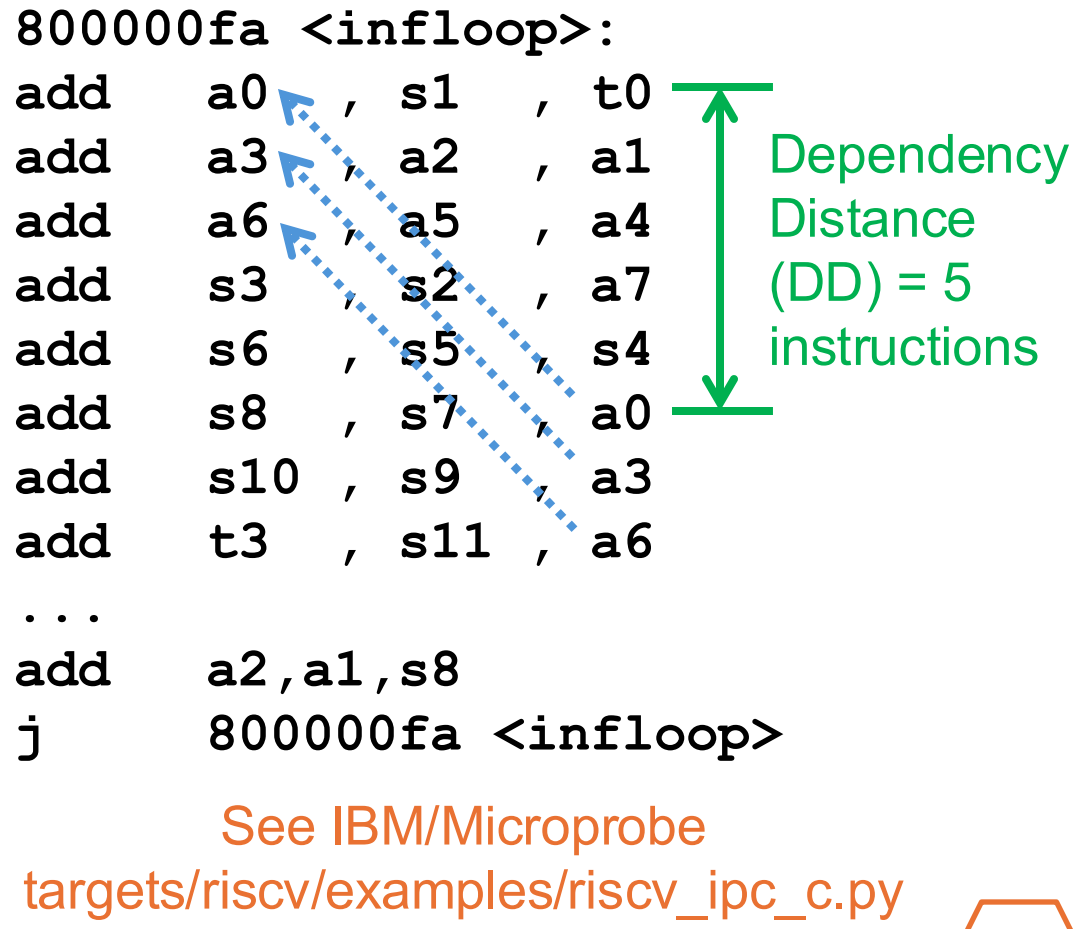
$$P_{avg} = \frac{1}{N} \sum_{j=0}^{M-1} C_j n_j(N)\, V_{dd}^2 f_{clk}$$



- RocketTile power of current system varies little, but what is in this model?

Instruction Benchmark Generation

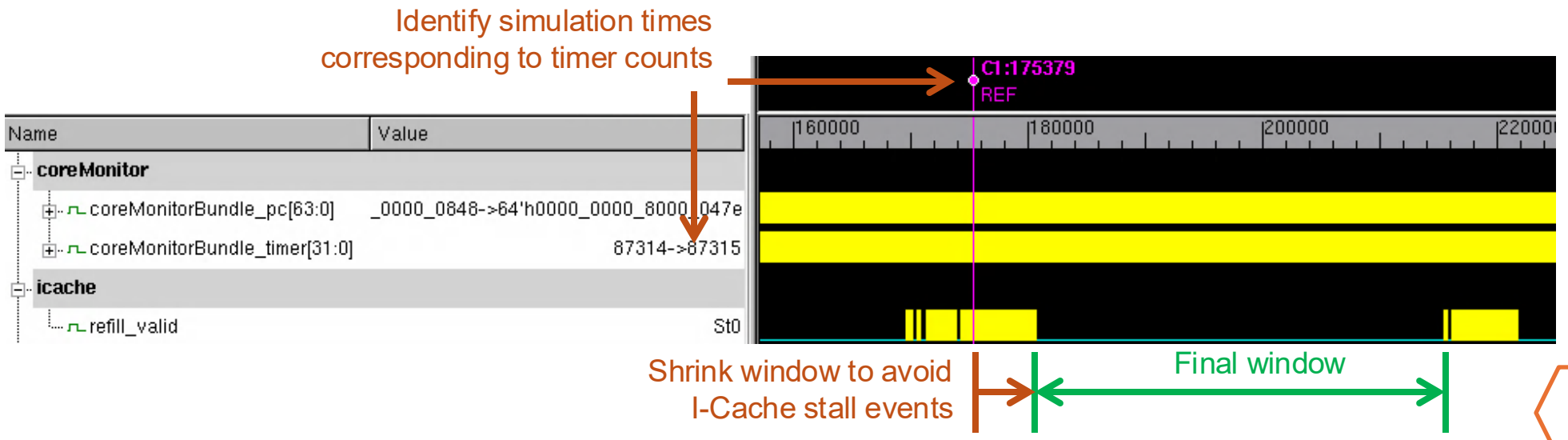
- Power data gathered from post-synthesis power analysis
- RTL simulations generated VCD files for activity propagation
- Microprobe tool from IBM used to generate workloads
- Sequences of 1000 instructions used to obtain average power
- Dependency Distance of 5 used to avoid pipeline stalls



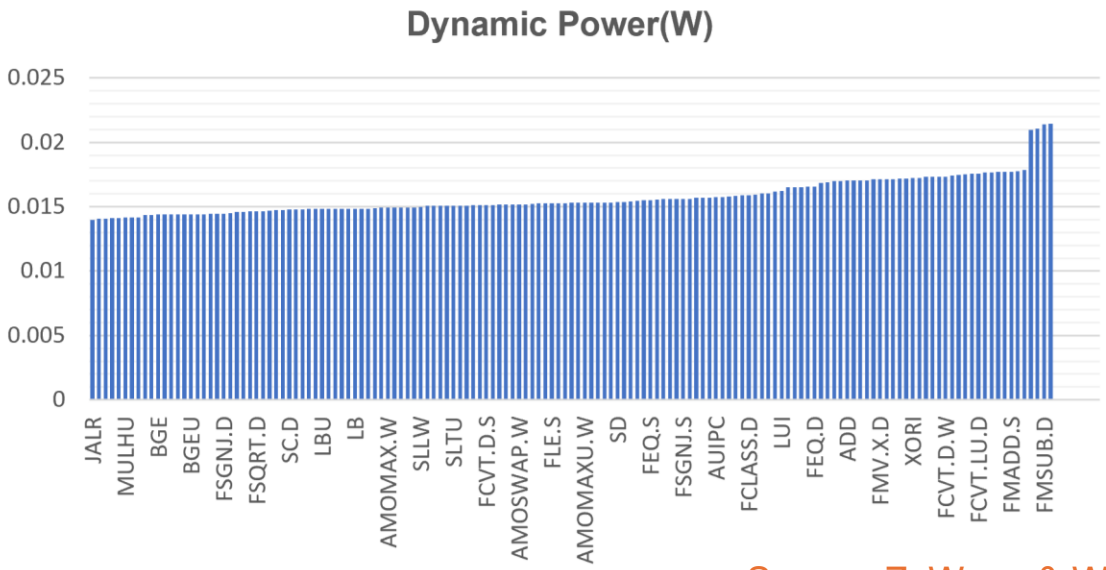
See IBM/Microprobe targets/riscv/examples/riscv_ipc_c.py

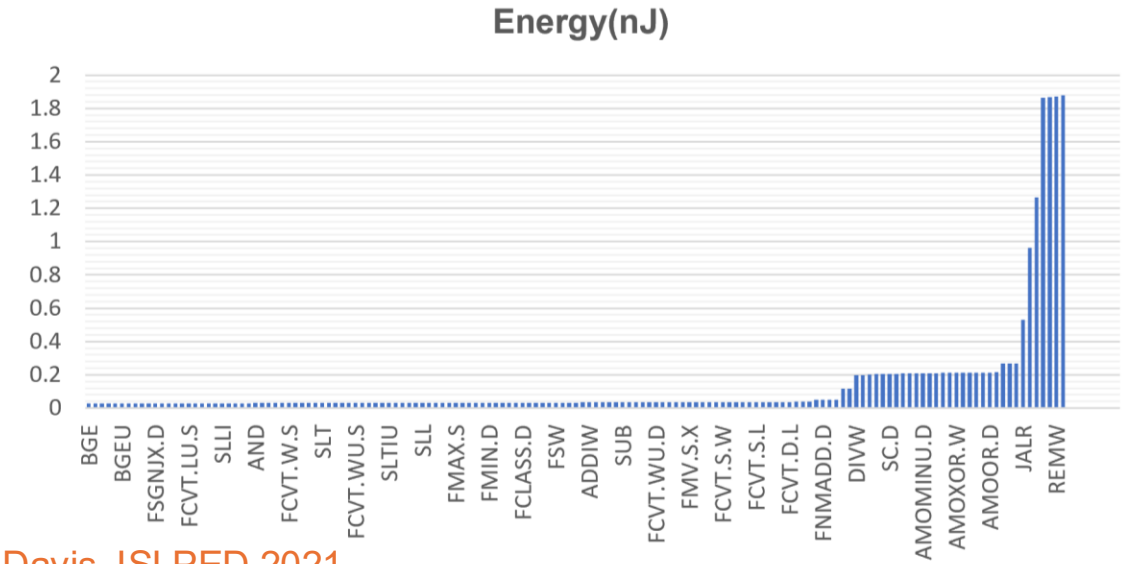
Finding Characterization Windows

- Switching activity must be parsed to determine characterization window
- Brief debug-controller interrupts every 1000 cycles evicted benchmark from I-cache, causing unwanted fill events
- Window with no extra events had to be found, instructions counted



Instruction characterization results



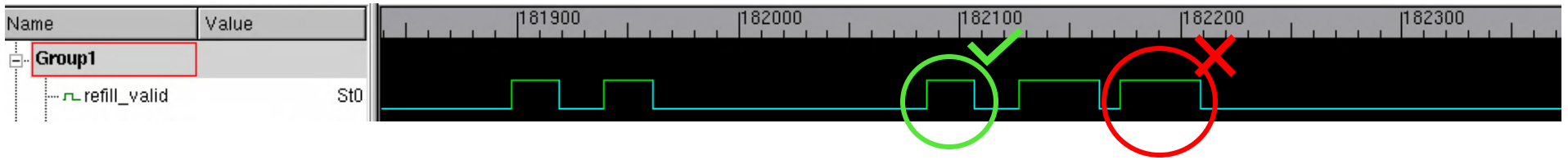


Source: Z. Wang & W. R. Davis, ISLPED 2021

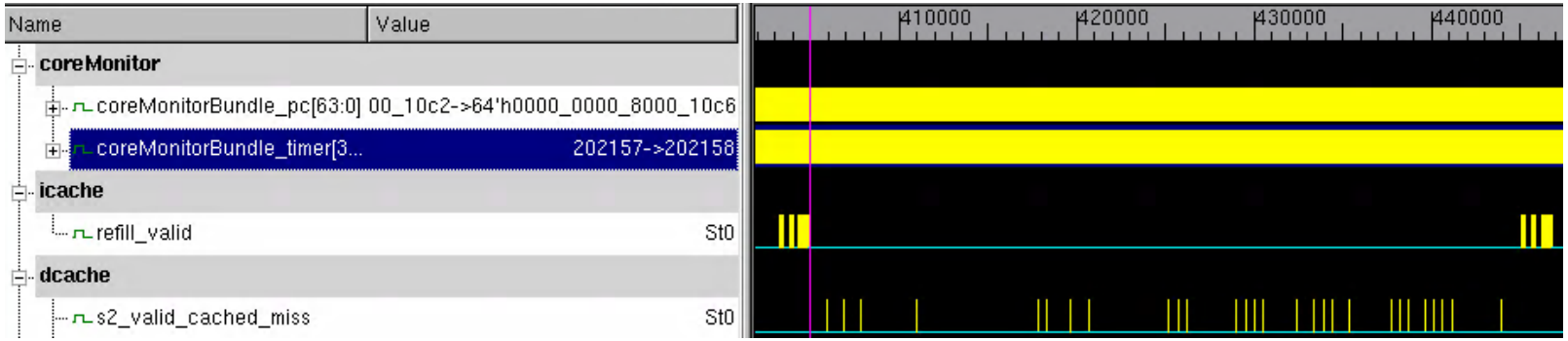
- 148 Instructions characterized
- Energy varies much more than power because of large variance in time per instruction (1 to 60 cycles)

Stall Energy Characterization

- Branching benchmark used to characterize pipeline stalls
- Instruction-Cache – shortest fill event characterized



- Data-Cache – Special benchmark required
  - Gauss Sum loops to access data cache, Increasing iterations forces fills



Final UPM RocketTile Model

- Model parameters
  - Supply Voltage
  - Frequency
  - Numbers for each event
  - Capacitance for each event
  - Total number of cycles

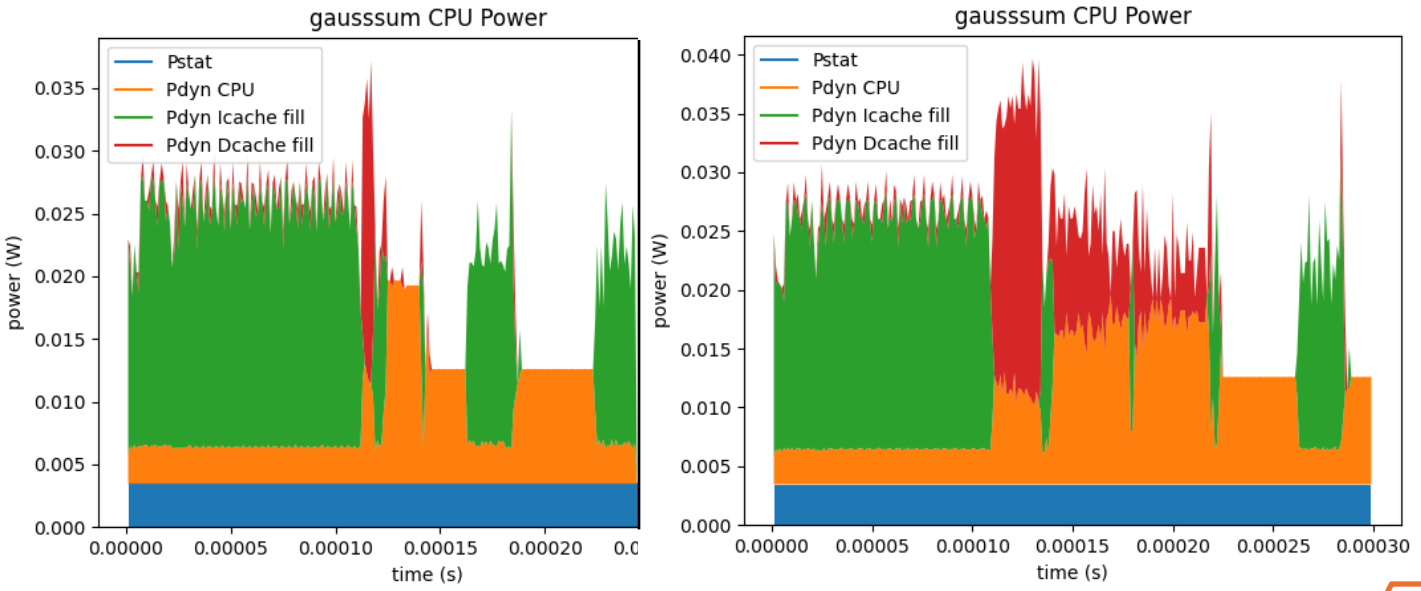
- Matches the desired form

$$P_{avg} = \frac{1}{N} \sum_{j=0}^{M-1} C_j n_j(N) V_{dd}^2 f_{clk}$$

```
<Cell name="RocketTile">
  <CellParameters>
    <Parameter name="Fclk" />
    <Parameter name="VDD" />
    <Parameter name="Temperature" />
    <Parameter name="NumREMUW" />
    <Parameter name="NumLR_D" />
          ...
    <Parameter name="Numdcache_fill" />
    <Parameter name="NumCyc" />
          ...
  <States units="W">
    <State name="on">
      <DynamicPower>
        <ExpressionParameters>
          <Parameter name="CapPI_REMUW" value="2.15044028036e-11" />
          <Parameter name="CapPI_LR_D" value="2.46584009186e-11" />
                      ...
        <Expression name="DynamicPower">
          ((VDD^2 * Fclk) / NumCyc) * (
          (NumREMUW * CapPI_REMUW)
          +(NumLR.D * CapPI_LR.D)
          +(NumAMOOR.D * CapPI_AMOOR.D)
          ...
          +(Numdcache_fill * CapPI_dcache_fill)
          )
```

RocketTile Power Accuracy

- Note that cache fills are responsible for significant power dissipation
- Shown below: two versions Gauss Sum benchmark with different D-cache fills and 1000-cycle averaging window
- More detailed CPU simulator than ISS must be used to gather these events



Other Blocks and Leakage Model

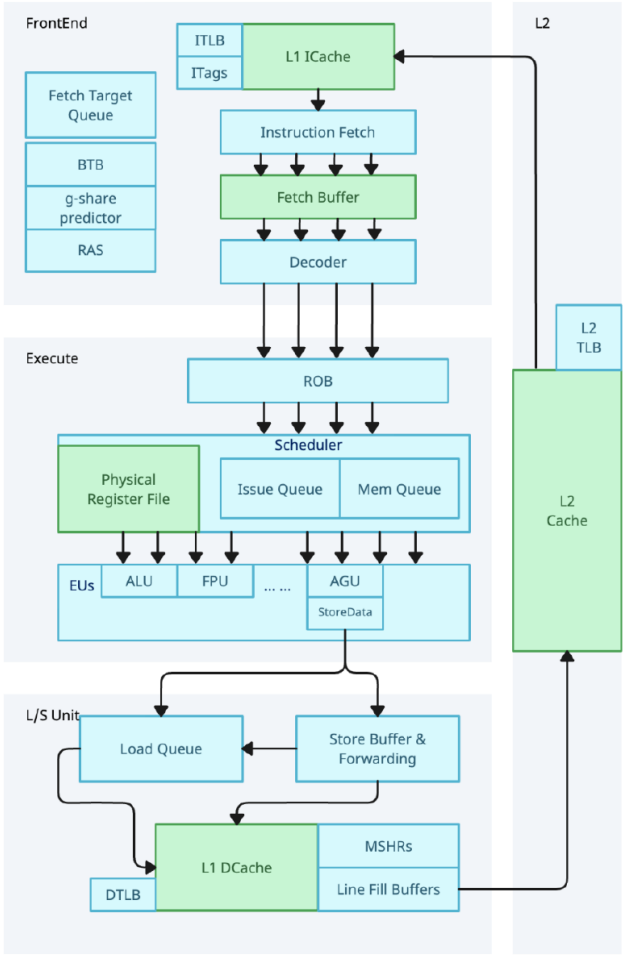
- Other blocks modeled using a simple model obtained from a “vectorless” power analysis of each block

```
<Cell name="SystemBus">
  <CellParameters>
    <Parameter name="VDD" />
    <Parameter name="Fclk" />
  </CellParameters>
  <States units="W">
    <State name="on">
      <DynamicPower>
        <Expression name="DynamicPower">
          V(VDD)^2 * Fclk * 6.14672540272e-1
        </Expression>
      </DynamicPower>
      <StaticPower>
        <Expression name="LeakagePower">
          V(VDD) * 0.000540088311074 * Leakage_Scaling_Expression()
        </Expression>
      </StaticPower>
    </State>
  </States>
</Cell>
```

```
<ModelExpressions>
    <Expression name="Leakage_Scaling_Expression">
        (VDD/Vnom) *
        ((C2K(Temperature)/Tnom)^0.5) *
        ((1/(0.4+(k*C2K(Temperature)/q))) / (1/(0.4+(k*Tnom/q))))^0.5 *
        (1-exp(-(VDD/(2*k*C2K(Temperature)/q))))/(1-exp(-(Vnom/(2*k*Tnom/q)))) *
        (exp(-(0.7+(0.4-((k*C2K(Temperature)/q)^0.5)+0.08))/
            (3*k* C2K(Temperature)/q)) /
         exp(-(0.7+(0.4-((k*Tnom/q)^0.5)+0.08))/(3*k*Tnom/q)))
    </Expression>
</ModelExpressions>
```

- Leakage power scaled with an expression derived from BSIM3 equations

Future Work: BoomTile Power Model



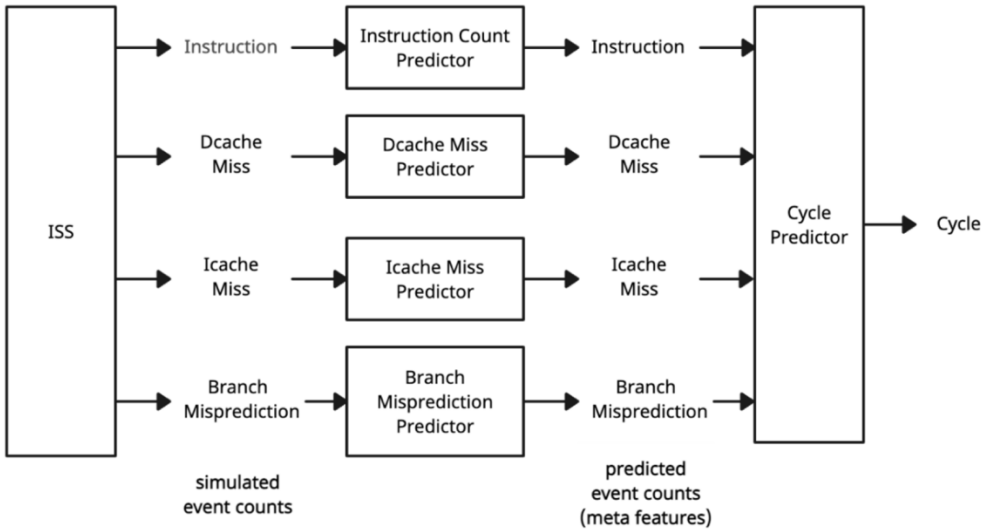
BoomTile diagram by Zhiping Wang

- A more typical CPU power model today uses events for large function-units
- Example: Popular public McPAT model has 121 energy-consuming events for
  - Main Pipeline
  - Instruction & Data Caches & Translation Lookaside Buffers
  - Branch Predictor & Target Buffer
- These could easily be converted to UPM
- Need a detailed CPU simulator to gather events and cycle counts

BoomTile Event Prediction

- Possible approach: Predict events and cycle-counts from ISS
- We investigated with a decision-tree estimator and Spike ISS augmented with cache and branch prediction simulators
- Trained with RTL simulations of BoomTile using 108 different configurations and 34 benchmarks

Processor			
Fetch Width	8		
Decode Width	3		
ROB Entries	96		
LSQ Entries	24		
Issue Width	int(3)	fp(1)	mem(1)
Issue Entries	int(32)	fp(16)	mem(24)
Physical Reg	int(100)	fp(96)	
Branch Predictor	type(g-share)	gbhl(16 or 32)	counter(1 or 2)
Memory			
L1	block size	$\log_2(nSets)$	$\log_2(nWays)$
DCache	64B	1::1::6	1::1::3
ICache	64B	1::1::6	1::1::3



Source: Zhiping Wang, NC State Dissertation, 2023

BoomTile Event Prediction Results

Benchmark	ISS+DTI (w/o g-share)	ISS+DTI (w/ g-share)	RTL Simulation (baseline)	Speed-up (w/o g-share)	Speed-up (w/ g-share)
hello_world	2.5s	2.9s	14min40s	352x	303x
specrand	7s	8.7s	154min17s	1322x	1064x
fibonacci	3.4s	3.6s	46min7s	814x	769x
insertion_sort	4.8s	5.2s	34min38s	433x	400x
deepsjeng	6.4s	8.9s	256min	2400x	1726x

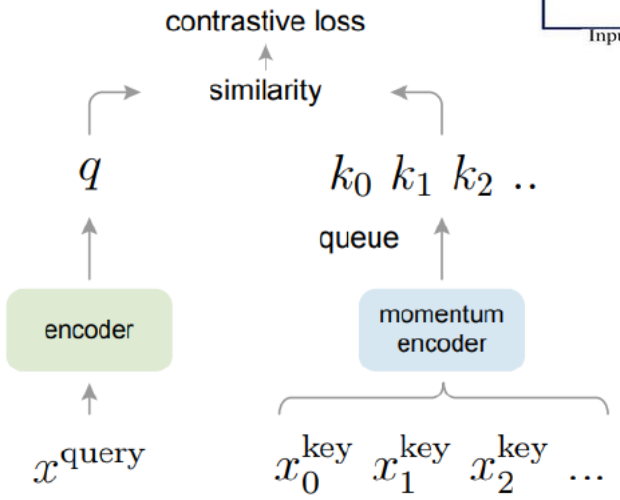
Benchmark	Dynamic Power		Total Energy	
	Baseline	Model	Baseline	Model
specrand_30_20	0.010%	0.001%	11.013%	0.067%
fibonacci_0x5000_5000	0.130%	0.017%	51.584%	11.758%
hello_world	0.217%	0.009%	60.242%	6.414%
gausssum_10	0.231%	0.002%	59.946%	1.269%
gausssum_900	0.168%	0.011%	52.469%	10.026%
bubble_sort_150_150	0.059%	0.000%	39.667%	0.163%
bubble_sort_350_350	0.003%	0.001%	5.623%	2.141%
bubble_sort_170_210	0.044%	0.004%	34.959%	4.345%
bubble_sort_420_600	0.000%	0.000%	0.513%	0.910%
insertion_sort_5	0.228%	0.012%	61.999%	8.839%
insertion_sort_100	0.175%	0.004%	57.388%	4.285%
insertion_sort_200	0.093%	0.003%	46.220%	2.105%

Source: Zhiping Wang, NC State Dissertation, 2023

- Simulation was 300X-2000X faster than detailed timing simulator
- Power Prediction is very accurate
- Energy prediction accuracy was good for some cases, but did not generalize as well as we hoped
- Could be built into SystemC model without much effort

Future Work: Incorporate AI Application

- Target Application: System reads out series of images from sensor and develops image classes
- Investigated Training with MoCo Self-Supervised Learning Algorithm from Kaiming He, Microsoft Research
- ResNet-50 used for encoder



Source: Kaiming He, *et al.*, CVPR 2020

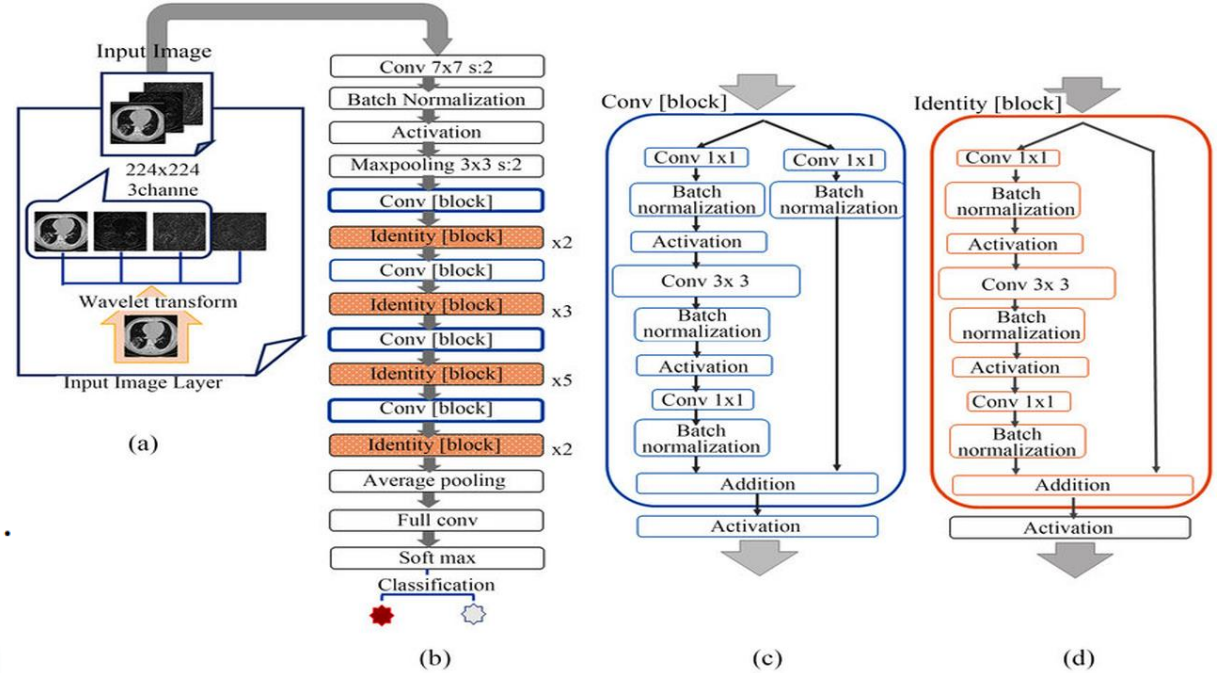
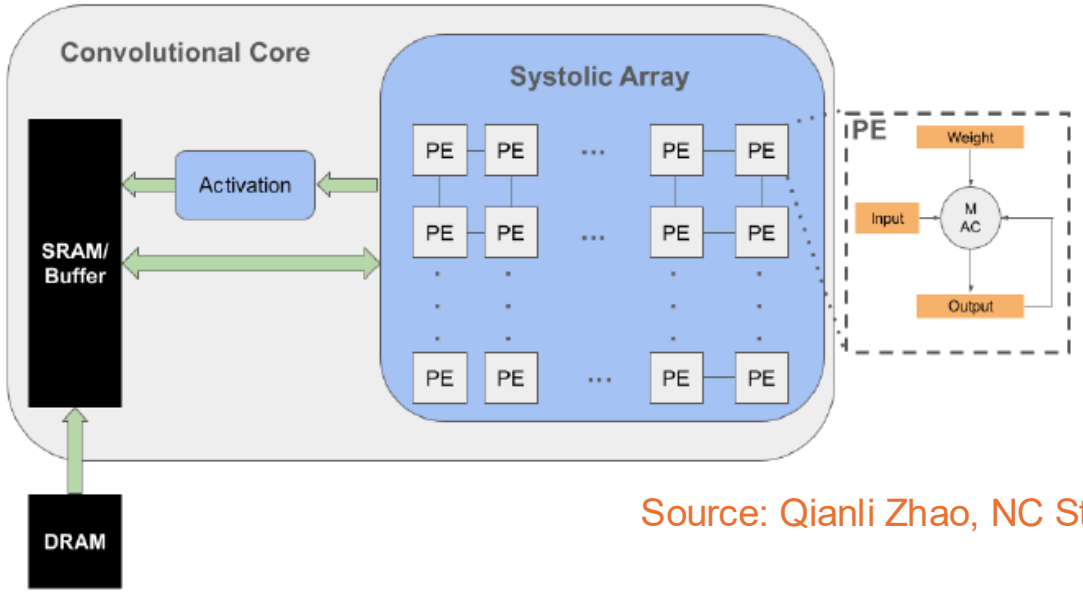
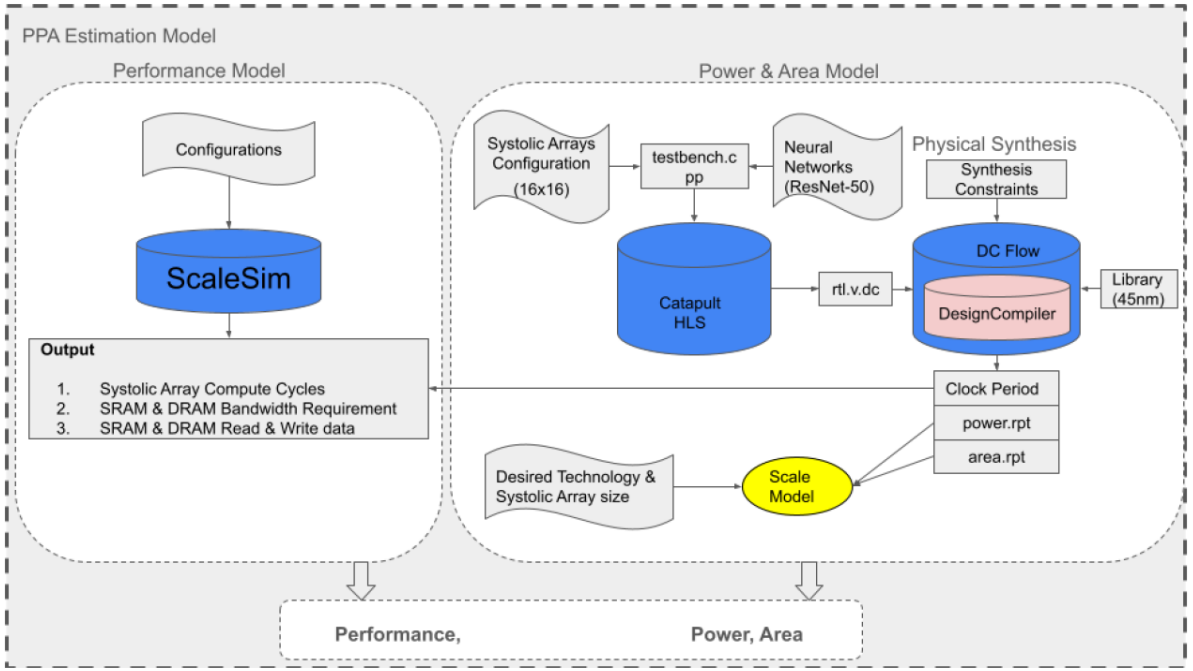


Image Source: wisdomml.in 2023

Power Models for AI Application

- Workload partitioned between RocketTile and Systolic Array
- Array model extracted from synthesized HLS code
- Scale-sim used for array timing

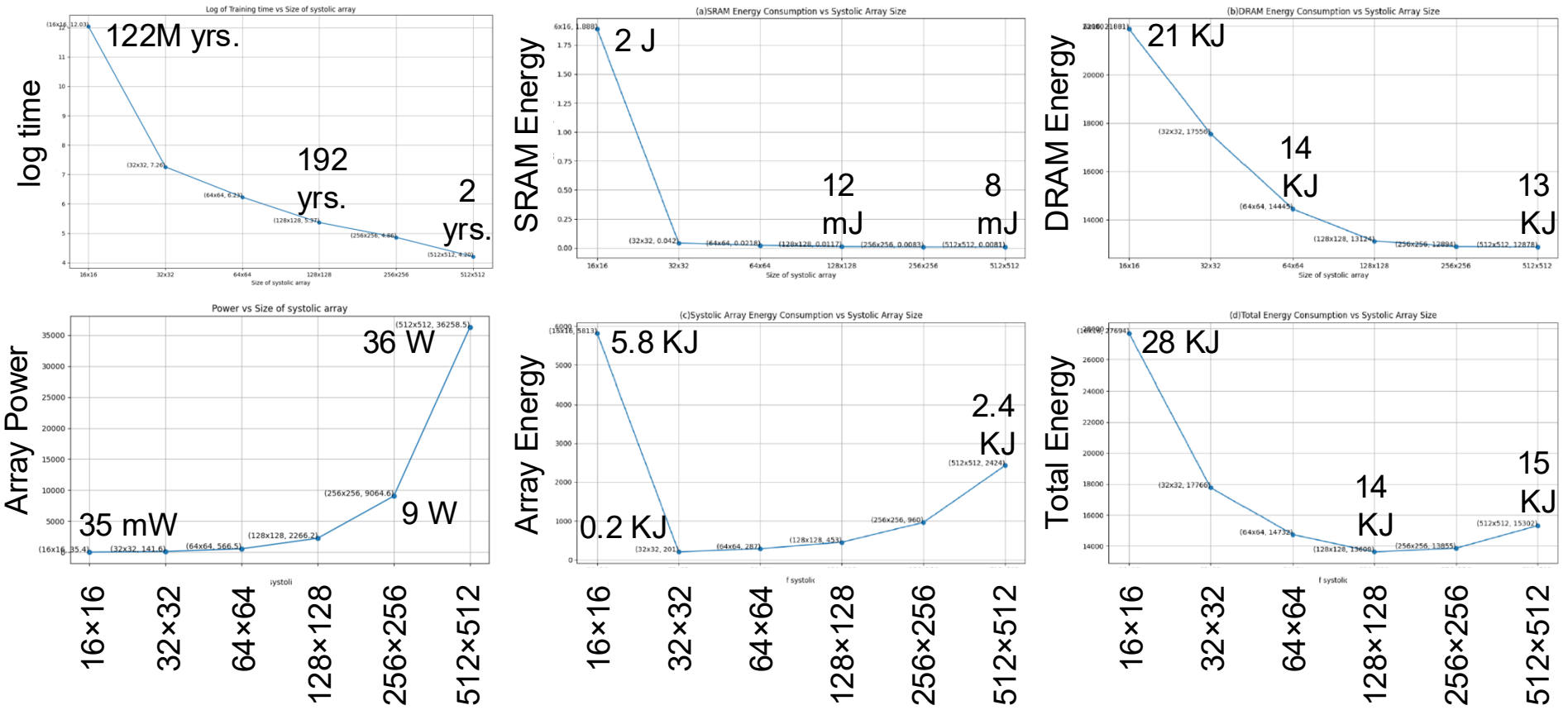




Source: Qianli Zhao, NC State Dissertation, 2024

Power/Energy Analysis for AI Application

- Model could be adapted for UPM
- Shows what is possible with UPM
- More software effort needed to complete



Source: Qianli Zhao, NC State Dissertation, 2024

Conclusions

- This System-Level Power Demo helps to illustrate how IEEE 2416 can aid in power planning, sequencing, and architecting for a complex mixed-signal system.
- Scenario generation and PowerCalc produce power traces with fine grained information about system power contributions and allow early refinement of power saving strategies.

- This work should be simulator-agnostic, and thus can be distributed to provide a platform for IEEE 2416 adoption.
- No More Spreadsheets!

Q&A + Discussion

Closing and Resources

- Join us for the IEEE 2416 Working Group open meeting, Tuesday June 24 at 10am in Room 3014
- 2416-2025 is in ballot. Voting has been extended – if you are in the pool, get your ballot and comments submitted!
- Thank you for attending

AI

Security

Systems

EDA

Design

62 THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY CEDA® IEEE Council on Electronic Design Automation

sigda acm