Integrating Power Models into Instruction Accurate Virtual Platforms for ARM-based MPSoCs

ARM TechCon 2016
26 October 2016

R. Görgen[2], D. Graham[1], K. Grüttner[2], L. Lapides[1], S. Schreiner[2]
[1]Imperas, [2]OFFIS

Agenda

- Current state of embedded software development
- Comparison of hardware-based and virtual platform-based methodologies
- Instruction accurate software timing simulation
- Power model with dynamic frequency and voltage scaling (DVFS) support
- Case study:
  - Simple power model for ARM Cortex-A9
  - Demo of case study

Agenda

- Current state of embedded software development
- Comparison of hardware-based and virtual platform-based methodologies
- Instruction accurate software timing simulation
- Power model with dynamic frequency and voltage scaling (DVFS) support
- Case study:
  - Simple power model for ARM Cortex-A9
  - Demo of case study

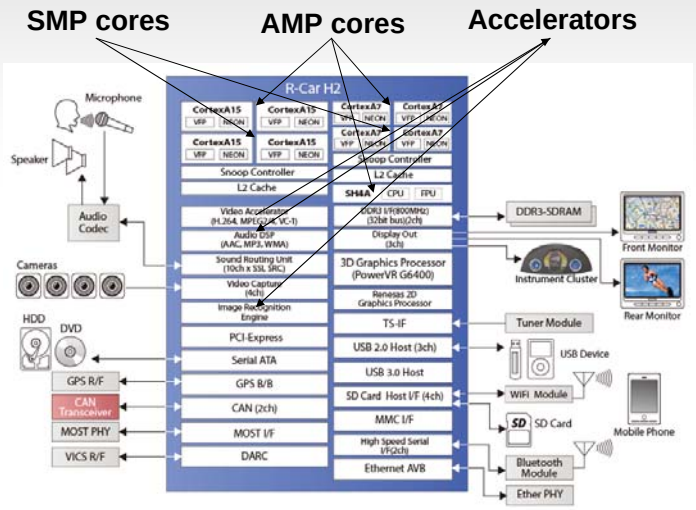
Embedded Software Development Issues
(in no specific order)

MULTICORE DESIGN SIMPLIFIED
imperas

- Schedule
- Quality
- Functionality
- Timing, power constraints
- Security / safety
- Predictability of the software engineering task: management accuracy on software resource and schedule requirements is +/- 50%
- Unknown / unmeasurable delivery risk

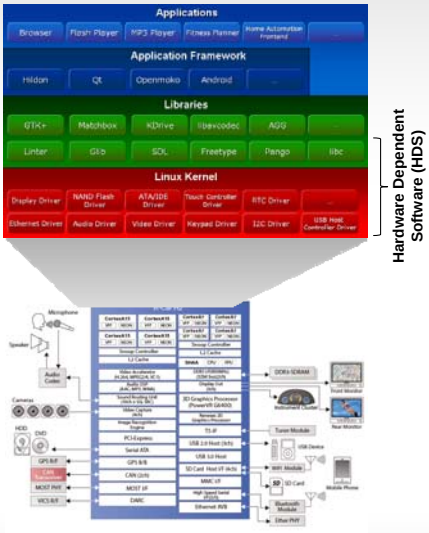
The last two bullets are management issues, and their importance should not be underestimated.

Modern SoCs Have Many Concurrent Processing Elements



Renesas R-Car H2: Automotive infotainment and ADAS

SW Verification Requirements



- Hardware Dependent Software (HDS)
  - Most complex foundation layer
    - Drivers, hypervisors, assembly libraries, operating system
  - Buried problems often appear elsewhere in a system, leading to misdirected analysis
  - Ripe for corner case type issues
  - Post development bugs hardest to fix
  - Testing needs to be platform centric not application centric
- Modern SoC SW verification is complex
  - SMP/AMP multicore interaction
  - Shared memory & devices
  - Extensive accelerators, peripherals
  - Externally authored, complex libraries
  - Complex SW/HW interaction (e.g. power)
    - **How to estimate power consumption and test power management strategies over a wide range of conditions?**

Hardware-Based Software Development

- Has timing/cycle accuracy
- JTAG-based debug, trace
- Traditional development board or hardware emulator based testing
  - Late to arrive
  - Limited physical system availability
  - Emulators are too slow to run enough system scenarios
  - Limited external test access (controllability)
  - Limited internal visibility
    - **How to observe power consumption?**
- To get around these limitations, software is modified
  - printf
  - Debug versions of OS kernels
  - Instrumentation for specific analytical tools, e.g. code coverage, profiling

- Modified software may not have the same behavior as clean source code

Agenda

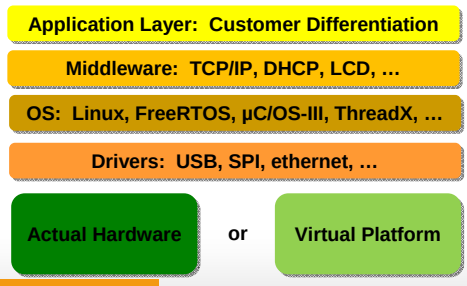
- Current state of embedded software development
- Comparison of hardware-based and virtual platform-based methodologies
- Instruction accurate software timing simulation
- Power model with dynamic frequency and voltage scaling (DVFS) support
- Case study:
  - Simple power model for ARM Cortex-A9
  - Demo of case study

Advantages of Virtual Platform Based Software Development
(Instruction Accurate Simulation)

- Earlier system availability
- Easy access for entire team
- Runs actual binaries, e.g. runs ARM executables on x86 host
- Fast, enables quick turnaround and comprehensive testing
- Full controllability of platform both from external ports and internal nodes
  - Corner cases can be tested
  - Errors can be made to happen
- Full visibility into platform: if an error occurs, it will be observed by the test environment
- Easy to replicate platform and test environment to support automated continuous integration (CI) and regression testing on compute farms

Virtual Platforms Complement Hardware-Based Software Development

- Current methodology employs testing on hardware
  - Proven methodology
  - Has limitations
  - We are at the breaking point
- Virtual platform based methodology delivers controllability, visibility, repeatability, automation



Virtual platforms – software simulation – provide a complementary technology to the current methodology

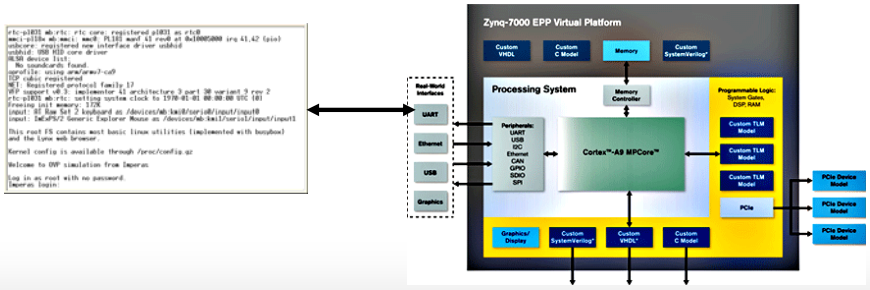
The same software stack can run on either the actual hardware or the virtual platform. This enables users to add virtual platform technology to their existing flow with minimal changes/risk, and achieve the benefits of virtual platforms.

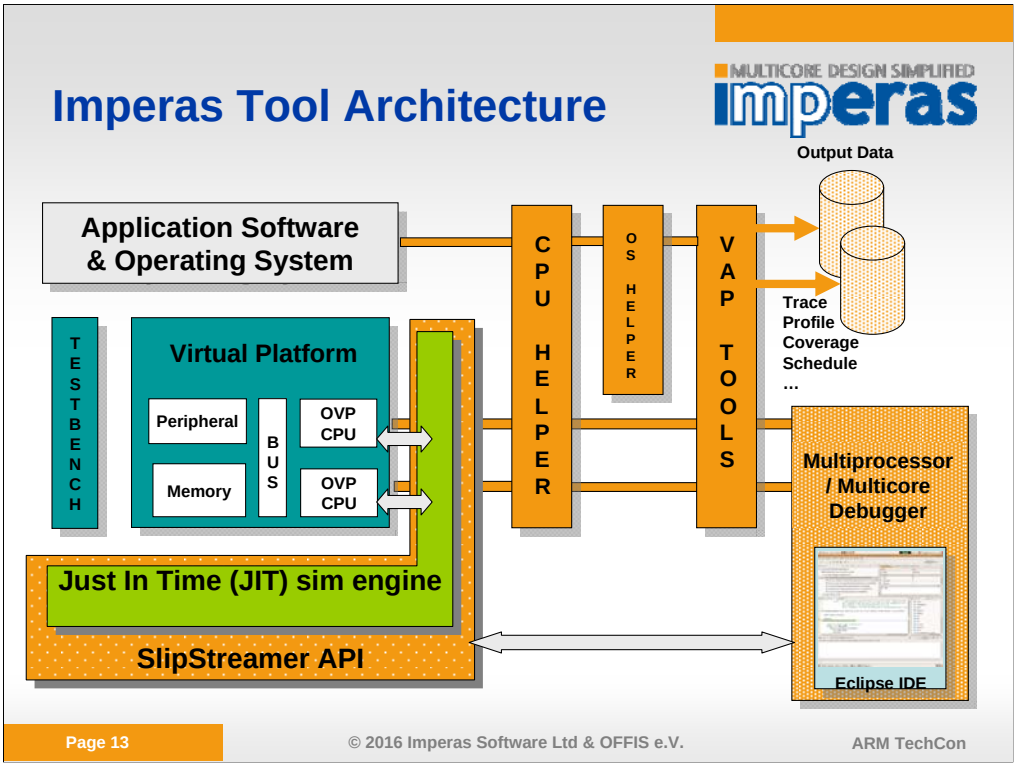
Agenda

- Current state of embedded software development
- Comparison of hardware-based and virtual platform-based methodologies
- Instruction accurate software timing simulation
- Power model with dynamic frequency and voltage scaling (DVFS) support
- Case study:
  - Simple power model for ARM Cortex-A9
  - Demo of case study

Building the Virtual Platform

- The virtual platform is a set of models that reflects the hardware on which the software will execute
  - Could be 1 SoC, multiple SoCs, board, system; no physical limitations
  - Functionally accurate, such that the software does not know that it is not running on the hardware
- Models are typically written in C or SystemC
- Models for individual components – interrupt controller, UART, ethernet, … – are connected just like in the hardware
- Peripheral components can be connected to the real world by using the host workstation resources: keyboard, mouse, screen, ethernet, USB, …



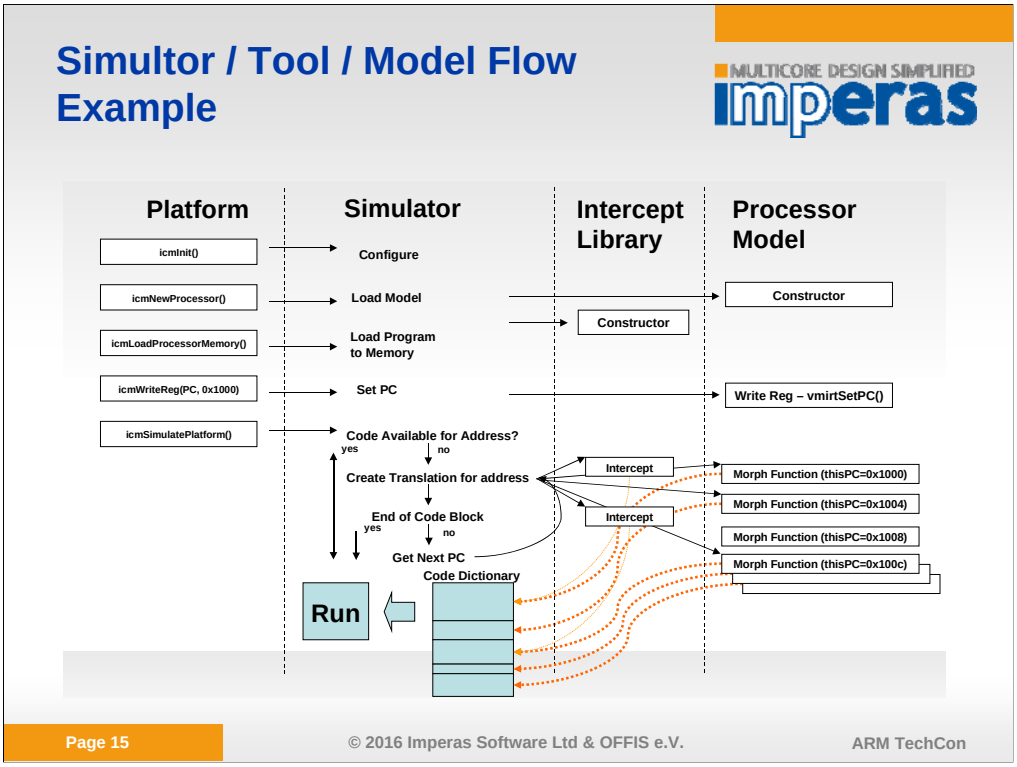


The SlipStreamer™ API enables the building of non-intrusive tools in the simulation environment. These tools include tracing (instructions, C functions, OS tasks), profiling, code coverage, OS scheduler analysis, memory analysis, and more. The SlipStreamer API is made available to both Imperas engineers and Imperas users, so that custom tools can be developed, such as the power analysis tools discussed in this presentation.

Simulator / Tool / Model Architecture

- Build environment elements separately
- Simulator engine uses Just In Time (JIT) binary translation (code morphing) technology to efficiently translate instructions for the target processor to x86
- SlipStreamer API enables tools to be built non-instrusively, i.e. no instrumentation or modification of software or operating systems
- Models – processors, peripherals, platforms – are built using the Open Virtual Platforms (OVP) APIs

- Software execution
  - For single core processor in the virtual platform, a block (“quantum”) of instructions, typically 1,000 – 100,000, is executed, then peripheral events are executed
  - For multicore processors, a quantum of instructions is executed in turn on each processor core; after each core has executed a quantum, the peripheral events are executed



The intercept library, created using the SlipStreamer API, is compiled for the x86 host (not the embedded processor target) and linked into the simulation environment.

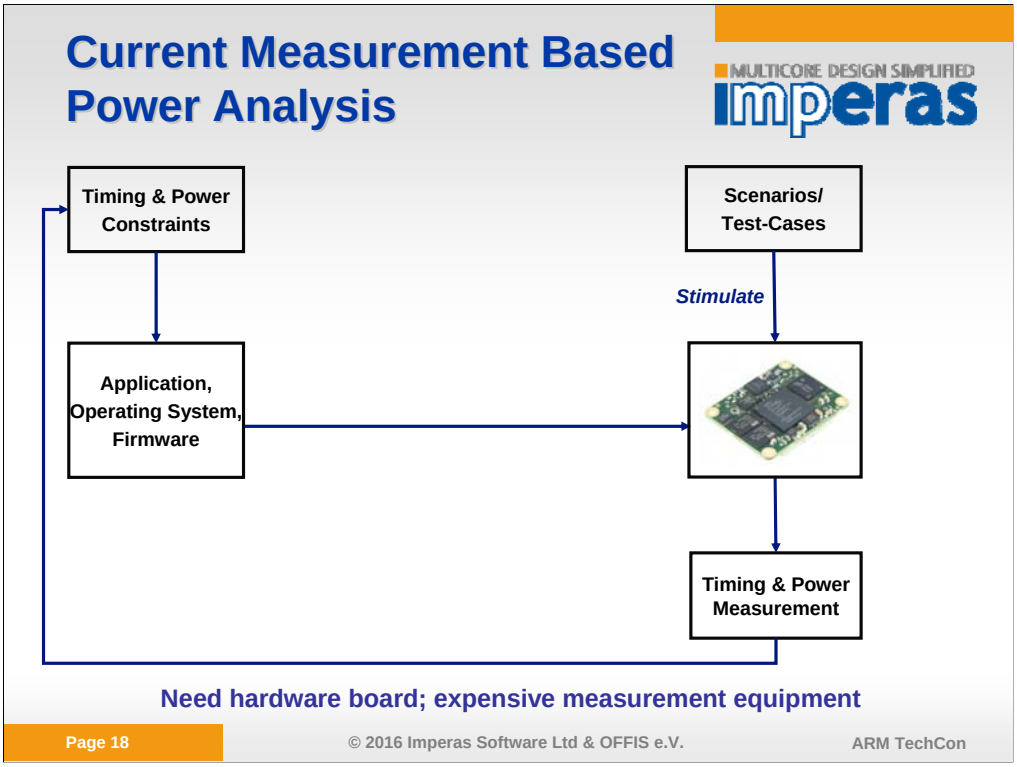
Timing Controls in Instruction Accurate Simulation

- Instruction accurate simulation is not timing or cycle accurate, however …
- The simulator and models have a sense of time
  - Timing assumption is 1 cycle per instruction
  - Processor models have an assumed speed in MIPS (millions of instructions per second)
  - Processor speed can be changed during simulation
  - Quantum size can be changed during simulation, so that artificial waits before peripheral events are reduced

The ability to dynamically change processor speed will be utilized later in this presentation when we talk about power management using DVFS.

Agenda

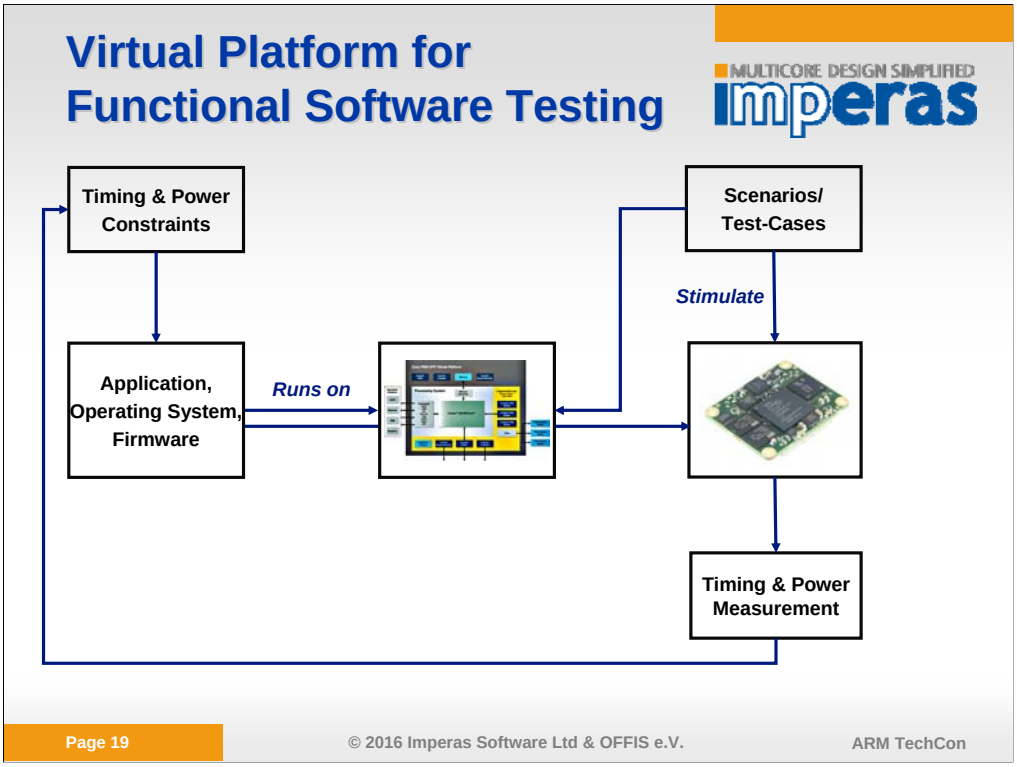
- Current state of embedded software development
- Comparison of hardware-based and virtual platform-based methodologies
- Instruction accurate software timing simulation
- Power model with dynamic frequency and voltage scaling (DVFS) support
- Case study:
  - Simple power model for ARM Cortex-A9
  - Demo of case study



How to obtain information about software power consumption?

State-of-the-art approach: Run application on development board and measure the power consumption in the laboratory (depending on the required measurement accuracy different (expensive) equipment is required)

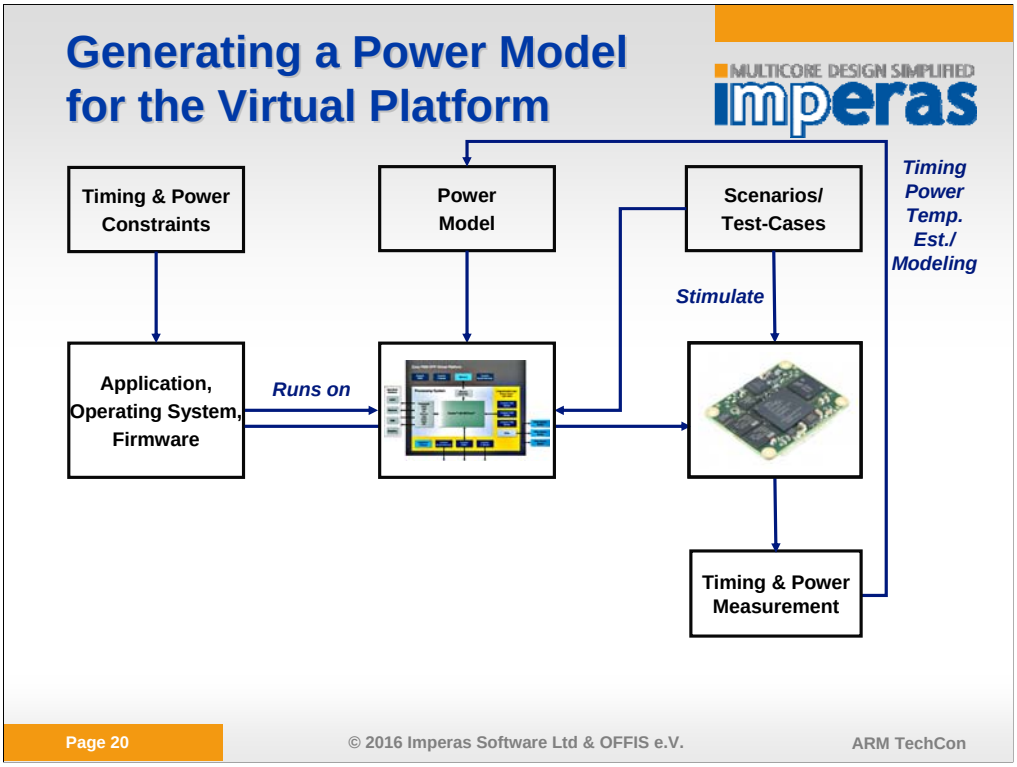
Obtained power measurement results can be compared against the specified power constraints.



With the availability of a virtual platform, the functional software testing can be performed without going down to the evaluation board.

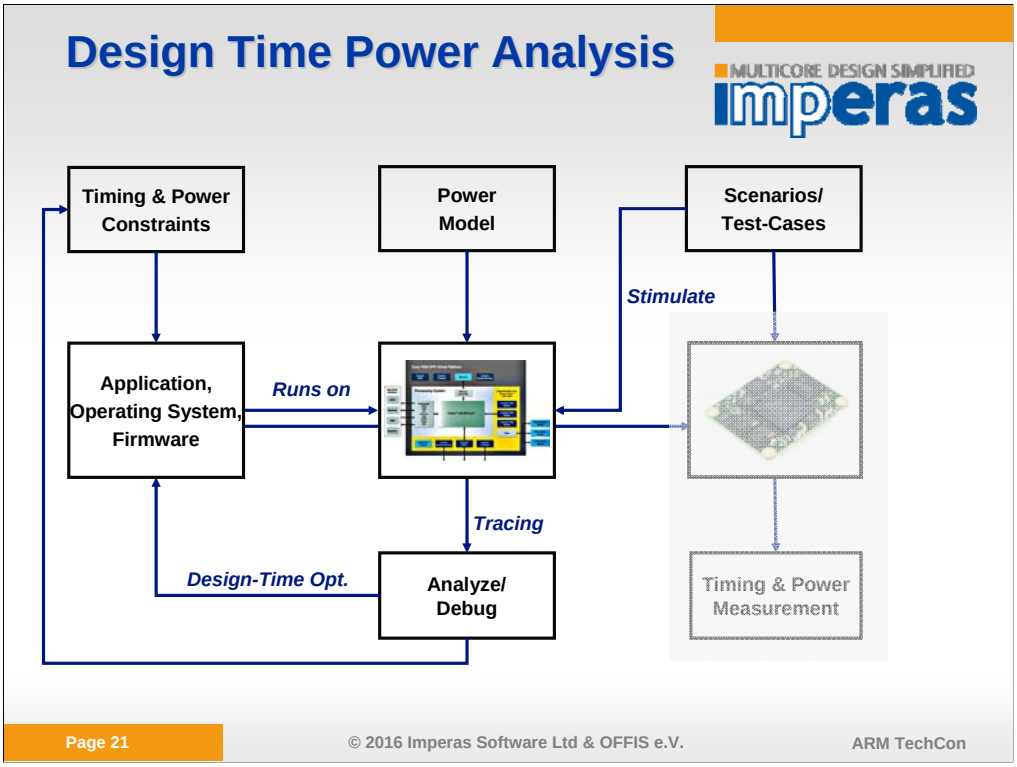
This approach is well suited for functional software (and to some extent for timing) software testing, BUT

Power measurement still performed on evaluation board.



The first step power analysis with virtual platforms is the power model generation/extraction process from measurement trails on the evaluation board.

Different approaches exist (micro-benchmarks, machine-based learning, …) exist, but this is out of the scope of this presentation.



After integration of the power model in the virtual platform (the following slides will show this), we can obtain

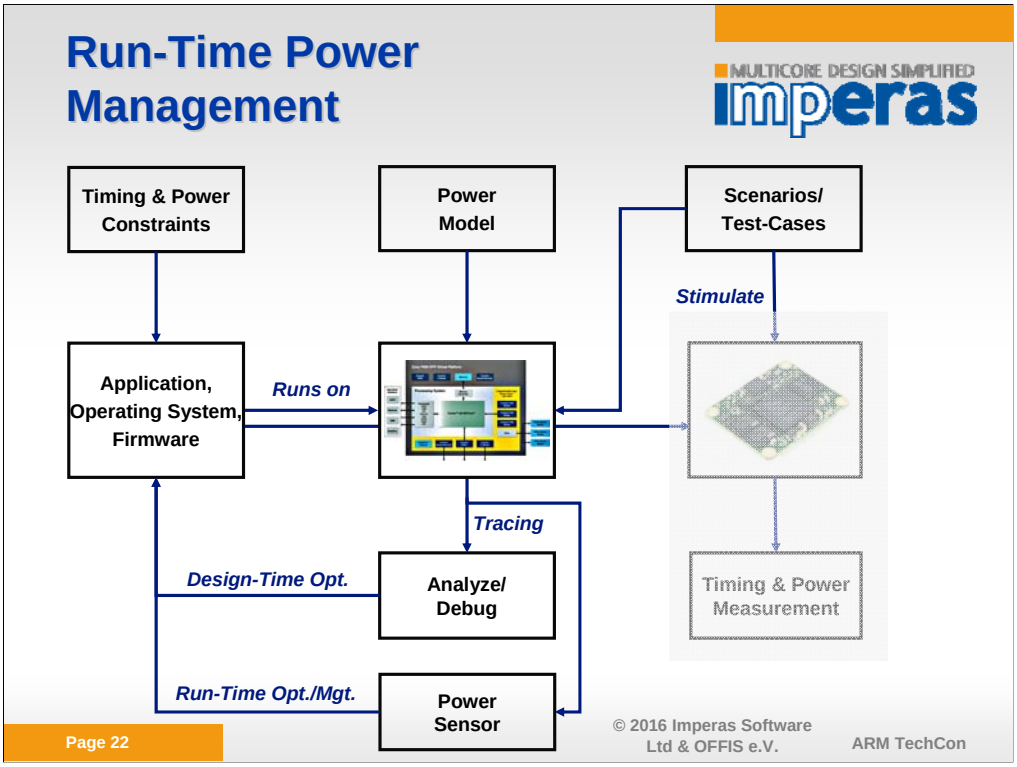
-Functional traces

-Timing traces

-AND (this is new) power traces over time

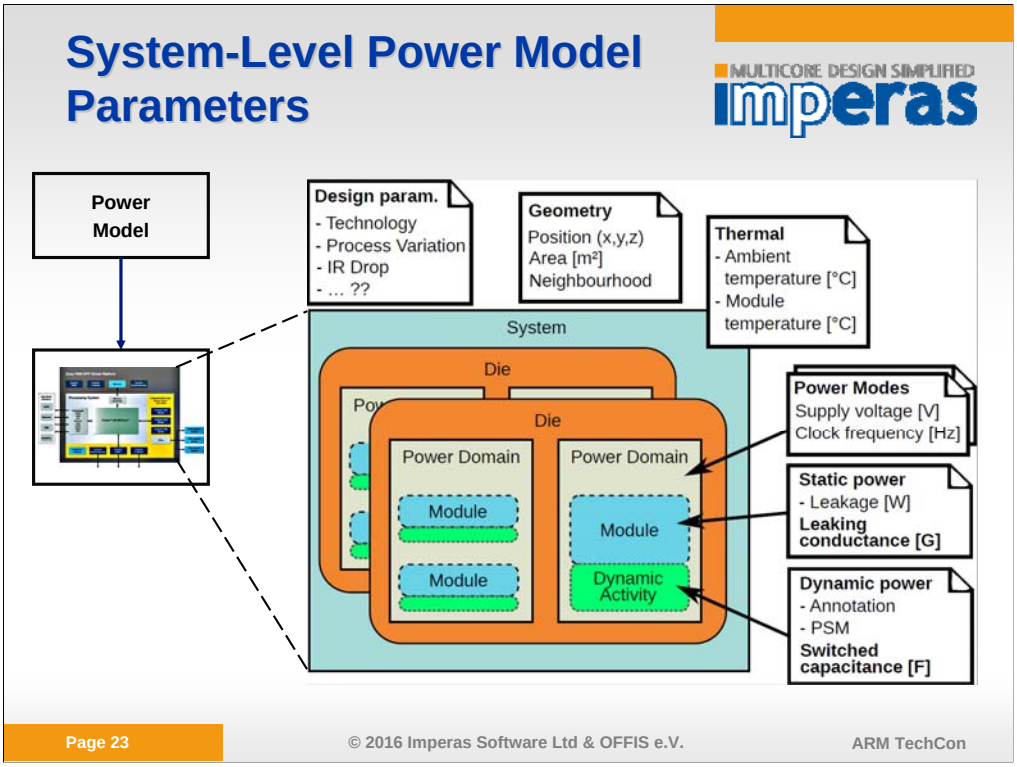
Depending on the spatial granularity of the power model, a hardware component power consumption breakdown can be supported.

Combining functional traces with component power traces, design time power optimization of the software can be performed.



Since many software already performs power management (usually based on temperature sensing).

Instead of writing the power information into an analysis trace (as before), we can also feed it into a power sensor that can be mapped into the address space of the hardware platform and thus allow software access to derive power management decisions at run-time.



Looking into the power model:

The applied power model has a hierarchical structure to represent different dies, power domains per die and different modules/functional hardware units per power domain.

The power consumption can be modeled at module level. It consists of:

-A dynamic part: depending on the actual usage of the component, expressed as average switched capacitance. The software dependent activity can be expressed as

- 1) Power State Machine (each state has a switched capacitance, transitions between states are triggered by the software or a power manager)
- 2) Annotation: Can be switched capacitance annotations in the processor model
- Our used power model is an annotation model. We are collection statistics during software execution (CPU load, number of memory read/write transactions, …) and transform them into a switched capacitance equivalent that is multiplied with the supply voltage and clock frequency to obtain the power consumption (see next slides).

-A static part: depending on the leaking conductance (area and technology dependent)

The actual power for the dynamic and the static part depends on the switching activity and the dynamic parameters of the associated power domain:

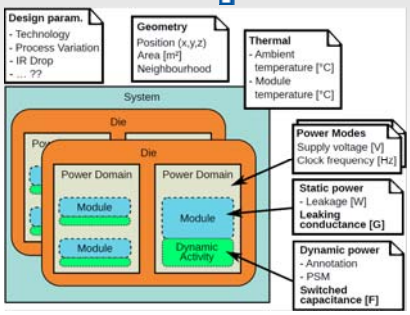
-Supply voltage

-Clock frequency

System-Level Power Model Parameters

MULTICORE DESIGN SIMPLIFIED imperas

- Building blocks for flexible power model
  - Static design parameters
  - Dynamic annotation/monitoring
- Overall power consumption can be computed from static parameters and observations



- **Hierarchical total power processing possible:**
  - $P(t) = V_{dd}^2(t) \cdot f(t) \cdot \bar{C}(t) + V_{dd}^2(t) \cdot G(\vartheta(t))$, where
    - $\bar{C}(t)$ average switching capacitance (or equivalent) per cycle (usually derived by dynamic annotation or performance counters)
    - $G(\vartheta(t))$ leakage conductance depending on temperature $\vartheta(t)$ (dynamic temperature model required)
    - $V_{dd}(t), f(t)$ supply voltage and frequency that can be changed over time (Dynamic Voltage and Frequency Scaling)

Overall power consumption P(t) based on:

-Dynamic part + static part

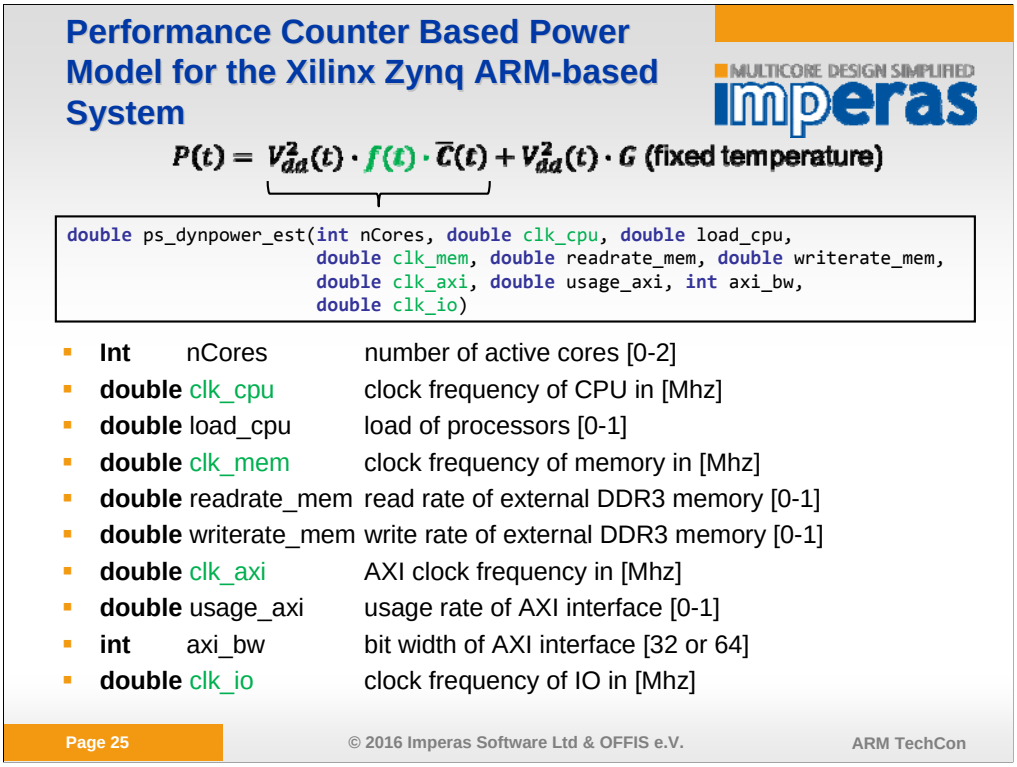
Important: All parameters can change over time:

-Vdd (supply voltage) can be changes by the software

-F (clock frequency) can be changed by the software

-C (average switched capacitance) is computed by a formula that takes different statistics during software execution (CPU load, number of memory read/write transactions, …) into consideration

-G(theta(t)) is not further taken into consideration. We assume a constant temperature (e.g. guaranteed by a sufficient cooling system)

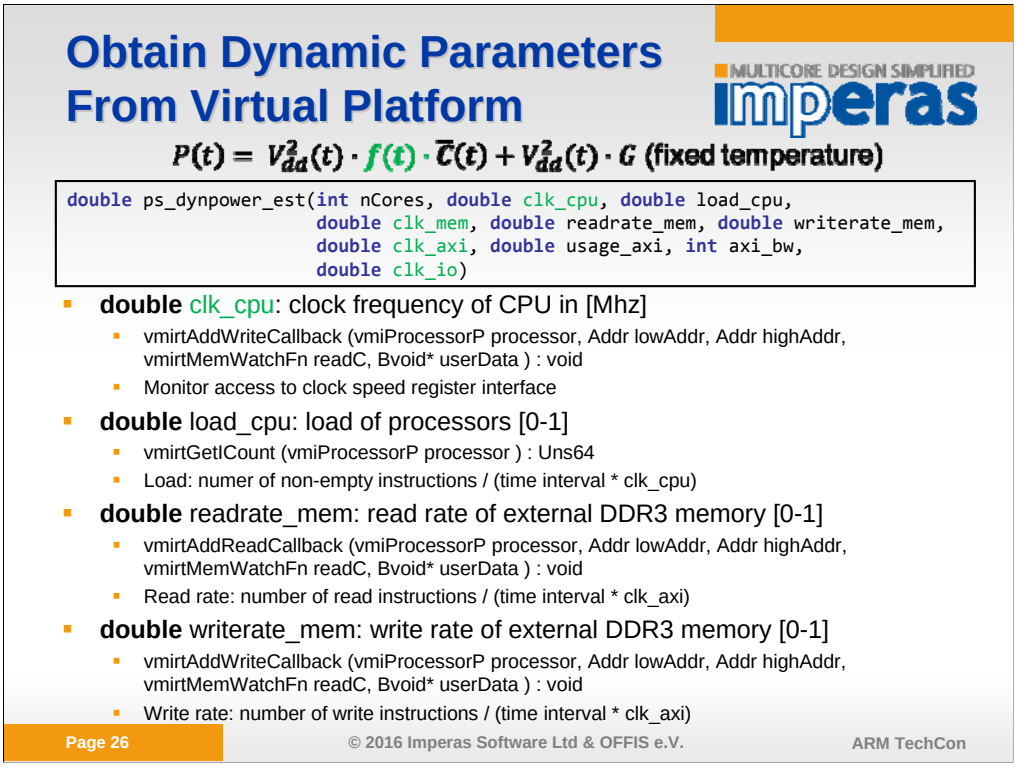


The function:

```
double ps_dynpower_est(int nCores, double clk_cpu, double load_cpu,
                       double clk_mem, double readrate_mem, double
writerate_mem,
                       double clk_axi, double usage_axi, int
axi_bw,
                       double clk_io)
```

Replaces the dynamic power term of the equation above.

The leakage term is assumed to be only dependent on the supply voltage.



Shows how some of the parameters of the function

```
double ps_dynpower_est(int nCores, double clk_cpu, double load_cpu,
                       double clk_mem, double readrate_mem, double
writerate_mem,
                       double clk_axi, double usage_axi, int
axi_bw,
                       double clk_io)
```

Are obtained through the OVP VP API.

Performance Counter Based Power Model for the Xilinx Zynq ARM-based System

MULTICORE DESIGN SIMPLIFIED imperas

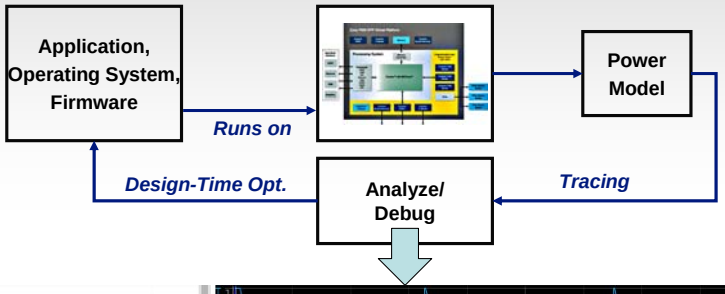
$$P(t) = \underbrace{V_{dd}^2(t) \cdot f(t) \cdot \bar{C}(t)} + V_{dd}^2(t) \cdot G$$

```
double ps_dynpower_est(int nCores, double clk_cpu, double load_cpu,
                       double clk_mem, double readrate_mem, double writerate_mem,
                       double clk_axi, double usage_axi, int axi_bw,
                       double clk_io) {
  P_Processor = VCCPINT / 1000.0 * (nCores * clk_cpu * load_cpu * 0.415 +
                (load_cpu<0.5 ? (0.5-load_cpu)*nCores*clk_cpu*0.1515 : 0));
  P_Processor_PLL = (((clk_cpu*2 > 0) ? 15.0 : 0) + clk_cpu*2*0.02)*VCCPAUX/1000.0;
  P_AXI = (VCCPINT/1000.0)*clk_axi*usage_axi*0.010417*axi_bw/8;
  P_Logic = P_Processor + P_Processor_PLL + P_AXI;
  ...
  P_DDR = P_Memory + P_Memory_PLL;
  ...
  P_Interfaces = P_USB + P_SD + 2*P_UART + P_I2C + P_SPI + 5*P_GPIO;
  ...
  P_IO = P_Interfaces + P_Interfaces_PLL;
  P_total = P_Logic + P_DDR + P_IO;
  return P_total;
}
```

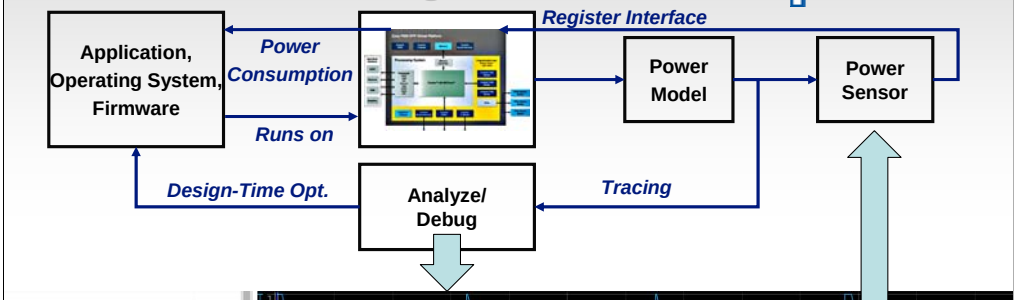

This slide just shows the complex power function for the dynamic power consumption of the PS part of the Xilinx Zynq SoC.

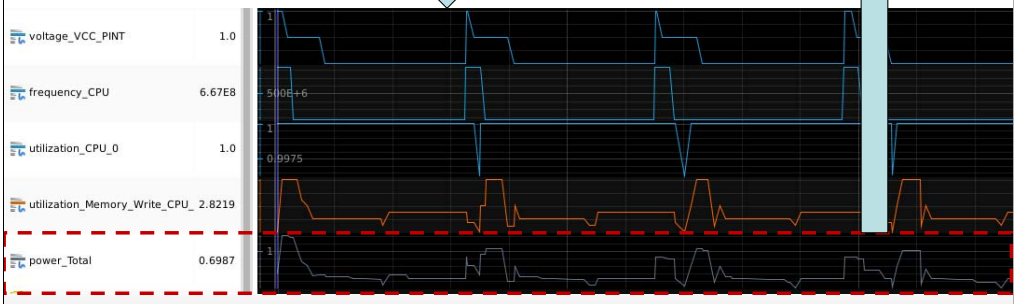
Timed Value Streams: Power Traces

MULTICORE DESIGN SIMPLIFIED
imperas



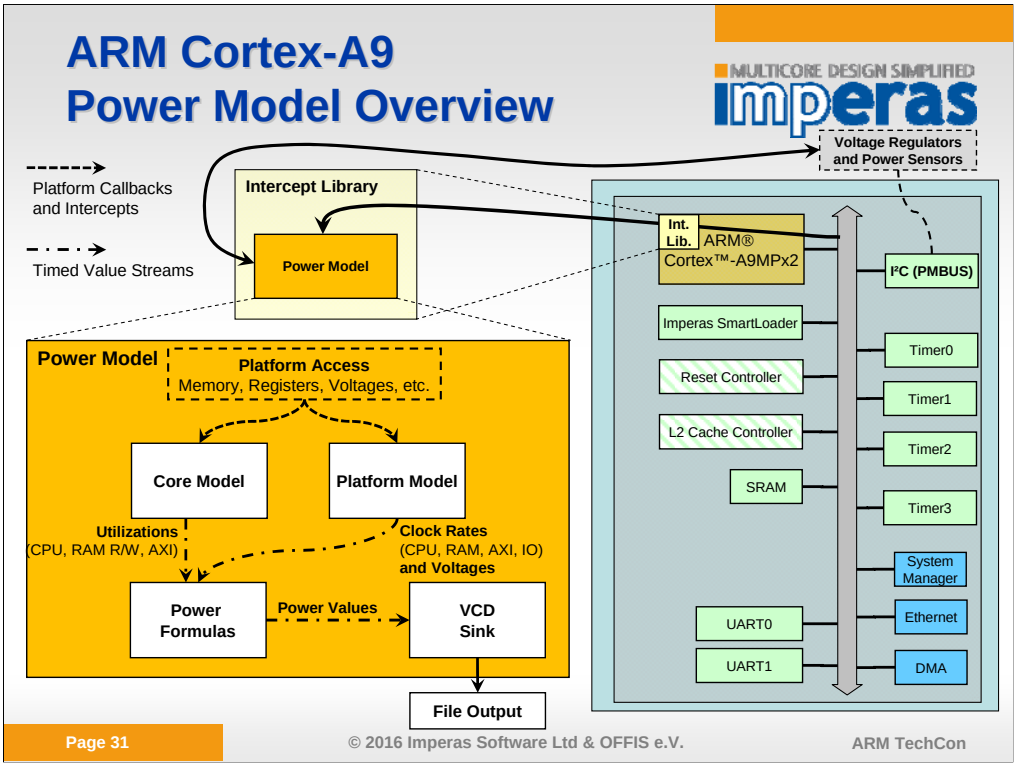
Timed Value Streams: Power Monitoring in SW





Agenda

- Current state of embedded software development
- Comparison of hardware-based and virtual platform-based methodologies
- Instruction accurate software timing simulation
- Power model with dynamic frequency and voltage scaling (DVFS) support
- Case study:
  - Simple power model for ARM Cortex-A9
  - Demo of case study



This slide shows an overview of the Power Model and the Zynq ARM Dual Core Platform

The Power Model is instantiated in the intercept library, it accesses the platform information via defined memory callbacks and I²C intercepts are used for transmitting new voltage parameters or returning power values.

The power model has 4 main parts which communicate via Timed Value Streams:

-The Platform model is responsible for recognizing all platform values, like frequencies and voltages, as well as the intercepted I²C communication with the Voltage Regulators and Power Sensors

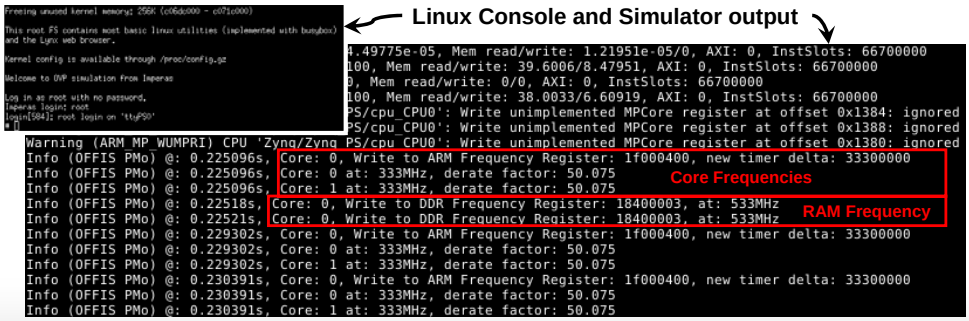
-Both Cores have one Core Model. It is responsible to calculate all Core specific data, like CPU utilization (with vmirtGetExecutedICount and vmirtGetICount), Memory Read and Write rates and AXI Load (both with memory read and write callbacks). All calculations for the utilizations are called periodicaly by a defined Model Timer

-The Power Formulas calculate all power streams for CPU, Memory, AXI, IO, Leakage, etc.

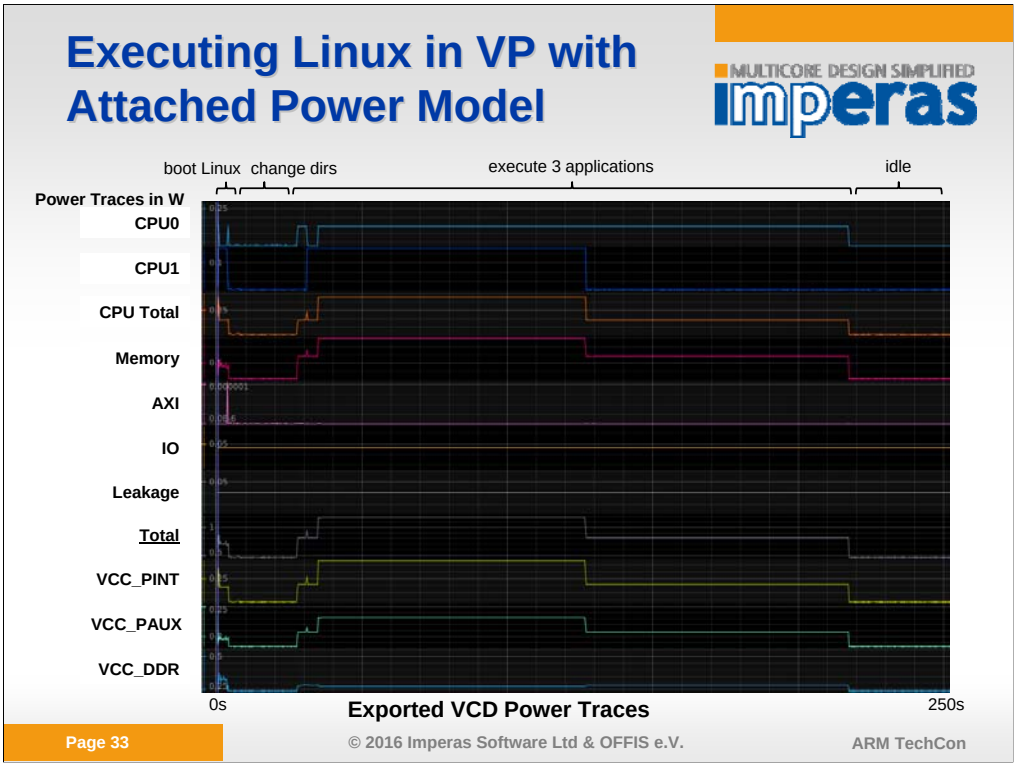
-The VCD Sink writes all traces and data to a trace file

Executing Linux in VP with Attached Power Model

- Virtual Platform (VP) executes Linux and Power Model recognizes changes:
  - Core frequencies are reconfigured to 333MHz
  - RAM frequency is configured to 533MHz
- Power Model reconfigures MIPS rate of both cores



The platform is able to boot Linux with the Power Model attached. In the screenshots you the reconfiguration of the core frequencies and the RAM frequency. Since the frequency of the Cores is reduced from 667MHz to 333MHz, also the MIPS rate in the platform is degrated from 667MIPS to 333MIPS with a factor of 50.075



This Slide presents all exported power Traces that are written to the VCD file

I booted Linux, switched to the directory „/benchmarks/“ and executed 3 instances of peakSpeed1.exe

Next to the power traces the following data is also written to the VCD file:

Group Power:

-CPU0

-CPU1

-CPU Total

-Memory

-AXI

-IO

-Leakage

-Total Power

-VCC_PINT

-VCC_PAUX

-VCC_DDR

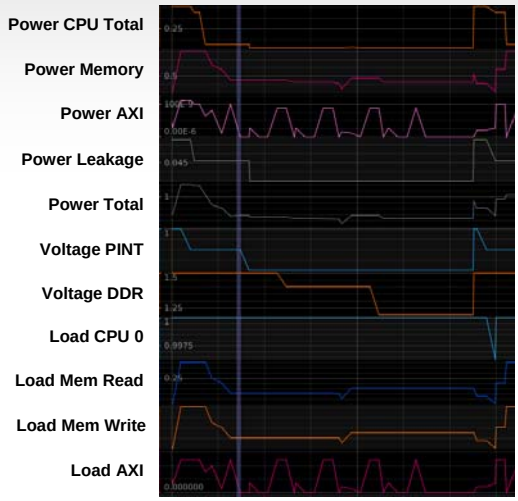
Group Frequency:

-CPU

-Memory

DVFS Bare Metal Example

- Application Binary is executed **bare metal** on ARM cores
- Switch **frequencies and voltages** for:
  - ARM cores
  - DDR memory
- **Power Model** recognizes changes

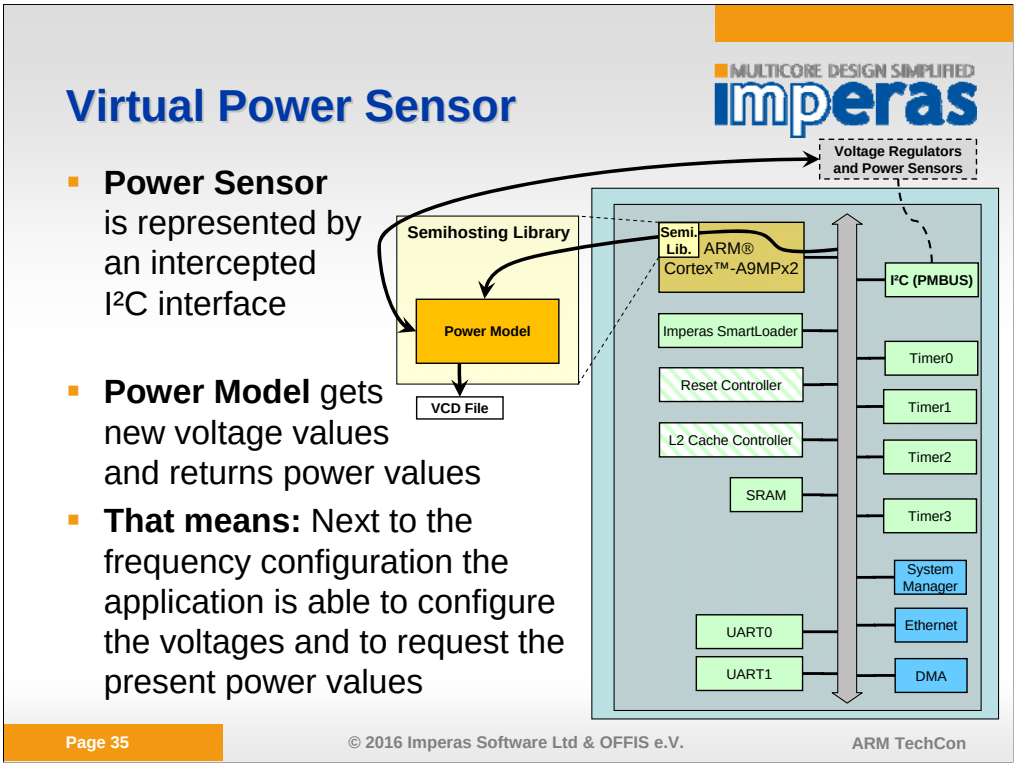


VCD Traces of DVFS example

The DVFS Bare Metal Example executes the following:

```
While(1)
{
        // First state
        scaleFrequency(DDR_CLK_CTRL, DDR_CLK_533_MHZ);
        scaleFrequency(ARM_CLK_CTRL, PS_CLK_667_MHZ);
        scaleVoltage(VCCPINT_DEVICE, VCCPINT_PAGE, 1.0);
        scaleVoltage(VCCPAUX_DEVICE, VCCPAUX_PAGE, 1.8);
        scaleVoltage(VCC1V5_PS_DEVICE, VCC1V5_PS_PAGE, 1.5); //VCC_DDR

        // Go throuh the changes
        for(i = 0; i < 15000000; i++);          // wait
        readMeasurementsTI();                   // read virtual sensor
        for(i = 0; i < 5000000; i++);           // wait
        scaleVoltage(VCCPINT_DEVICE, VCCPINT_PAGE, 0.9);
        for(i = 0; i < 5000000; i++);           // wait
        readMeasurementsTI();                   // read virtual sensor
        for(i = 0; i < 5000000; i++);           // wait
        scaleFrequency(ARM_CLK_CTRL, PS_CLK_333_MHZ);
        for(i = 0; i < 5000000; i++);           // wait
```



The executed application is able to request the current power values over an intercepted I²C communication from the power model. The communication protocol is nearly the same as of the TI chips located at the Zynq zc702 board.

As well as the application is able to reconfigure the frequncies of the cores and the DDR memory via the original register interface, there is the ability to configure also the voltages VCC_PINT, VCC_PAUX and VCC_DDR over the intercepted I²C communication.

In that way the platform and the power model fully supports DVFS.

Power Monitors in Action

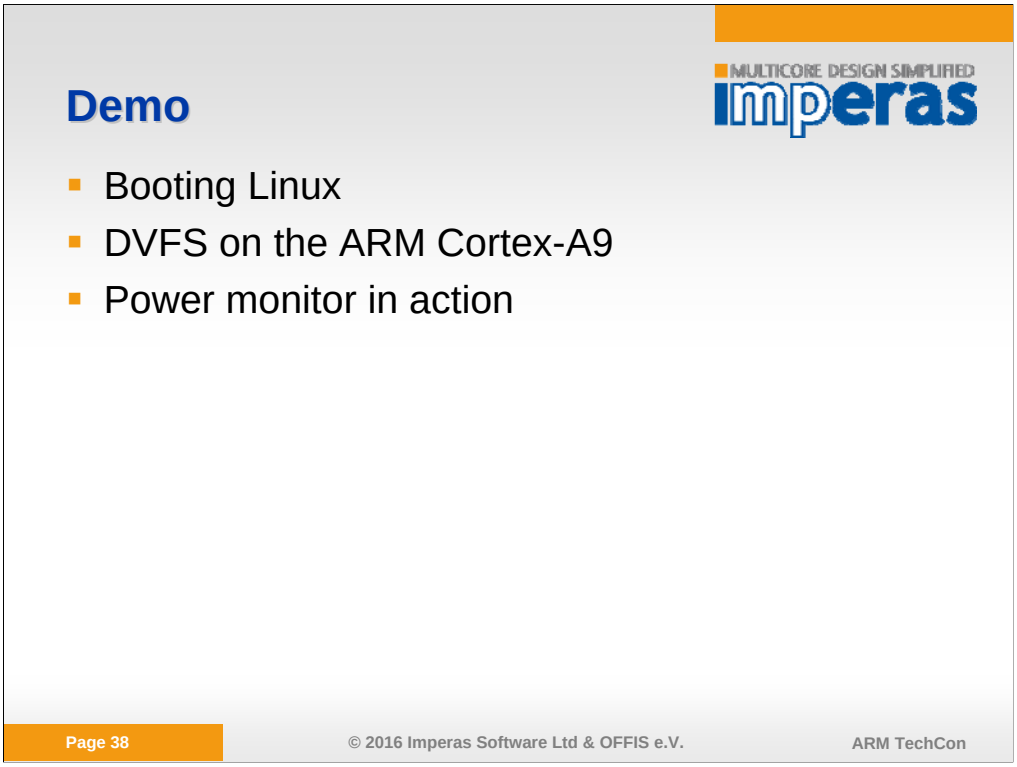
- Bare Metal DVFS example is executed in VP
- **VP Power Model** intercepts I²C communication of **Virtual Power Sensor**
  - Executed application is able to read power information
  - Values can be used in applications
  - **Example**: UART output of executed application:
    Simple read and print out of power values

```
|-------------------------------------------------------------|
|                        Power Monitor APP                    |
|RAIL           |  Voltage(V)   |   Current(mA)  |   Power(mW) |
|-------------------------------------------------------------|
|VccPInt        | 0.999912 V    | 312.320000     |   312.2925  |
|VccPAux        | 1.799744 V    | 118.584000     |   213.4208  |
|Vcc1V5_PS      | 1.499868 V    | 185.928000     |   278.8675  |
|-------------------------------------------------------------|
|TOTAL POWER  804.580831 mW       | MAX POWER  804.580831 mW |
```


The DVFS Bare Metal Example requests the power values (present voltages and currents) over the intercepted I²C communication and prints all grabbed information for the 3 power domains in its UART interface.

Agenda

- Current state of embedded software development
- Comparison of hardware-based and virtual platform-based methodologies
- Instruction accurate software timing simulation
- Power model with dynamic frequency and voltage scaling (DVFS) support
- Case study:
  - Simple power model for ARM Cortex-A9
  - Demo of case study



Video Demo of Zynq Platform with attached Power Model

1) Linux Demo:- Initialization: Power Model Init Outputs, Streams and other Models are initialized. Power Model outputs of Linux Booting Phase.

See the new derate factor that is set to 50.075 since the Frequency switches from 667MHz to 333MHz, as well as setting the DDR frequency to 533MHz.

The output sequence is the full Linux boot-up until the login comes.-

After it the same is shown with the uart1 output, here I login as root, switch the directory and execute two times the same peakSpeed benchmark-

Next I show the outputs of the power model again. First one task is already running (~0.8W) then the other one is started (~1.2W)-

As last I show the VCD Trace in the Viewer (impulse [http://toem.de/index.php/projects/impulse]). You see the boot phase at the beginning, in the power and utilization traces, as well as the execution of the two benchmarks (2 steps) in the power traces.

2) DVFS Demo:- Initialization: Power Model Init Outputs, Streams and other Models are initialized.

Power Model outputs are shown later again, here only for 2 seconds.

-Power Monitor: Application configures new frequencies and voltages, reads back voltages and currents to calculate power consumption on its own.

-Power Model Output: You see next to the core loads all switching activity of the frequencies and the voltages, as well as the readbacks (... Addr: 52, ...) of the currents. The procedure is described in the slide comments

-Power Model VCD Output: Here you see the swtiching activity in all traces.

Summary

- Virtual platforms – software simulation – provide a complementary technology to hardware-based testing of software
- Besides functional correctness, timing properties and power consumption are gaining importance
- Virtual platforms with a power model can help to
  - Test power management for low power or temperature sensitive systems and thus contribute to
    - Achieve higher quality software
    - Reduce development schedules
    - Increase software project predictability
    - Reduce delivery risk

Thank you

This work was partially funded by the European Union's Horizon2020 research and innovation programme under the grant agreement No 687902.
More information at
http://www.safepower-project.eu/