Fine-grained Energy/Power Instrumentation for Software-level Efficiency Optimization

spEEDO + PEHAM Project: Power estimation from high-level models

David J Greaves  
Klaus McDonald-Maier  
Milos Puzovic  
Ali Mustafa Zaidi  
Andrew Hopkins  

University of Cambridge  
Computer Laboratory + UltraSoC, ARM, Univ Kent  
Daresbury Laboratories.

FDL'15, Power Aware Modelling + Design Session.
spEEDO Project

• spEEDO: Energy Efficiency through Debug support

• University of Cambridge Computer Laboratory in Collaboration with UltraSoC Limited.

• Funded by the UK TSB (Innovate UK).

• Stage 1 October 2013. Stage 2 October 2015.
Power Aware Design

What battery life will I get?

Do I need to turn on another rack in my datacentre?

Should I offload this task to the GPU?

Is compiler option -Oblah helpful in terms of total energy for this task?

Will using single-way associativity in L1 for the stack segment save energy?

Should I use one core or four and at what clock frequency?
spEEDO 1 - Tagline

“Find out which thread on which core expended which picoJoule of energy on which IP block.”

Zynq 7010 device with PSU instrumentation that is binary compatible with our SystemC virtual platform (PRAZOR).

The spEEDO APIs are implemented in the virtual platform.

Currently we are implementing the `energy digestor' in the Zynq FPAG and also in a new RISC-V SoC.
spEEDO 1

• **Aim:** Develop a power API for three purposes:
  • Embedded software energy reflection API
  • Remote debugger energy accounting and logging
    - Extend GDB schemas for energy regs
  • Debug access to power-gated regions.

**Initial achievements:**

- Developed a strawman energy API for access to *On Chip Analytics*
- Trialed on SystemC virtual SoC
spEEDO – Overview/Solution

- Measure raw current and voltage at the regulator inputs and *instrument SMPSUs*.
- Use micro-architecture event counters to trace local energy expenditure but *these are banked*.
- Convey 'customer identifiers' over on-chip networks for remote accounting – *these say which banked register to increment*.
- *Energy digestor* remote reads event banks and PSU instruments to give up-to-date, fine-grain energy API.
- *Virtualise the digester via the O/S* to get per-thread energy consumption.
- Export debugging schema for viewing + calibration.
PC CPU Power Probe

Measures 12 volt rail to motherboard CPU socket.

Measures volts and amps at 10 Hz rate.

Accuracy: consistency of about 1 percent between runs (single-user mode or bare metal).
New Power Supply Monitors

Resistive shunt measurement

Linear measurement using resistive shunt

Measurement using switched-mode (SMPSU) duty cycle measurement
Switched-Mode PSU Controller

- SMPSU contains digital control logic that is easy to monitor.
- Provided we know the input rail voltage, we rely on the output rail being accurate and measure local duty cycle to get current.

Regulators can have complicated drive waveforms … but this matters not.
Basic Blade/PC Architecture

Software

S/W

Kernel

App1

App2

Hardware

H/W

spEEDO API

CPU core

caches local

caches local

CPU core

caches local

CPU core

PSU data abstraction

PSU

Input Output I/O

DRAM Main Memory

Network (ethernet)

Packet Counters for 802.11

Retired counters

Hit/Miss/Eviction counters

Hypertransport counters

Task control blocks
MSRs

Machine-Specific Registers:

Oprofile example.

Oprofile gives a uniform API to a wide variety of hardware platforms.

Listing shows monitorable event counters on AMD x86-Hammer
Intel's Power Gadget MSRs

Intel has implemented a Running Average Power Limit (RAPL) on Sandybridge processors.

A number of machine-specific registers are defined containing energy information:

SandyBridge:
- MSR_RAPL_POWER_UNIT
- MSR_PKG_POWER_LIMIT
- MSR_PKG_ENERGY_STATUS
- MSR_PP0_POLICY
- MSR_PP0_PERF_STATUS
- MSR_PKG_POWER_INFO
- MSR_PP0_POWER_LIMIT
- MSR_PP0_ENERGY_STATUS

Augmented Reference Architecture

Task control blocks contain energy accounts

Apps can link with a spEEDO library for energy information

Software S/W

Hardware H/W

Energy ‘digestor’

Calibration matrix

On-chip bus transactions are tagged with cust no.

Event counters are banked

Retired counters
Hit/Miss/Eviction counters
Hypertransport counters
DRAM counters
Packets for 802.11
Software Event Counts

Typical device driver stats:

eth0 Link encap:Ethernet  HWaddr 00:13:20:84:5d:81
inet6 addr: fe80::213:20ff:fe84:5d81/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:24110214 errors:0 dropped:0 overruns:0 frame:0
TX packets:15028627 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:100
RX bytes:3461755890 (3.4 GB) TX bytes:15455753259 (15.4 GB)

Existing event counters in device drivers and hardware can also be projected through a calibration matrix to give energy estimates.
Register Energy/Power ABI Strawman

// Typical hardware register to implement the spEED0 hardware API - unbanked version.

#define SPEED0_REG_MONICA 0  // Contains an identifying constant
#define SPEED0_REG_ABI 8  // Version number of the interface
#define SPEED0_REG_ENERGY_UNITS 16  // Energy units for the following
#define SPEED0_REG_CMD_STATUS 40  // Capability description and commands for res
#define SPEED0_REG_GLOBAL_ENERGY 48  // Running total energy in the units given - i
#define SPEED0_REG_TIME_UNITS 56  // Units for ticks in the time register.

#define SPEED0_REG_CTX0_BASE 512
#define SPEED0_REG_CTX1_BASE (512+256)

#define SPEED0_REFLECTION_URL0 1024  // First location of a canned URL giving further

// Each hardware context contains:

#define SPEED0_CTX_REG_LOCAL_ENERGY 8  // Running local energy in the units given
#define SPEED0_CTX_REG_LOCAL_TIME 16  // Running local time (if implemented) for the
Simplistic (Invasive) Energy Logging

A Hello World C app – a powerful step forward infact:

```c
#define SOCDAM_SPEEDO_REGS_BASE 0xFFFDO0000
#define READ_SPEEDO(X) (((unsigned int *)(SOCDAM_SPEEDO_REGS_BASE + X))[0])

int main(int argc, char *argv[])
{
    int j;
    printf("Hello World %x\n", READ_SPEEDO(SPEEDO_REG_MONICA));
    printf("Global energy units at start are %i\n", READ_SPEEDO(SPEEDO_REG_GLOBAL_ENERGY));
    for (j = 0; j < 10; j++)
    {
        int le = READ_SPEEDO(SPEEDO_REG_CTX0_BASE + SPEEDO_CTX_REG_LOCAL_ENERGY);
        printf("Core %i: Energy units are %i\n", SOCDAM_READ_PID_REG(0), le);
    }

    printf("Global energy units at end are %i\n", READ_SPEEDO(SPEEDO_REG_GLOBAL_ENERGY));
    _kllsim(0); // This makes a nice exit from SystemC - seems better at making or1ksmp exit!
}
```
Output from the very-simple C Program

(The C++ Figure 3 in the paper prints nicely in picoJoules.)
GDB/RSP Abuse/Extensions

GDB only understands uniform memory arch.
We wish to route register+mem reads to a specific core.
We abuse the thread select RSP command to address cores.

GDB

Debug session

$ info threads
$ thread 4
$ x32

Register descriptions

Canned XML or URI

Target

Core 0
Core 1
Core 2
Core 3
IP Block 4
IP Block 5

opaque customer number

H core id customer id

RSP 'H' command sets current thread for debug

JTAG or RSP over TCP
Baseline GDB energy reporting ...

Here a Python script reads the spEEDO API and prints one energy line.

Problems:
- Highly invasive,
- Simplistic,
- Static power while paused?
- No standard for automation,
- GDB poorly coded for generic extensions.
spEEDO API Stack

1) Application API
2) Machine-neutral multicontext API
3) Machine-neutral core API
4) Register API (H/W S/W ifc).

- Application Code
- O/S scheduler
- Software 'C' API library dll
- Machine-specific HAL (device driver)

Debug network interface (JTAG / USB / GDBserver)

VLSI & SoC Debug Network

Software

Hardware
Programmable FSMs

**Input events**

- Watchpoint trigger registers

**User-programmable finite-state machine (FSM)**

**Output events**

- spEEDO accounting context or on/off
- CPU Halt
- CPU Interrupt

**Trigger event counter file.**

*spEEDO account registers can be context switched by generalised watchpoint breakpoint and debug trace programmable FSMs.*
Banked Register Management

- At least one alternate energy register bank is needed for atomic snapshot on the live system.

- We extend this concept with customer numbers conveyed over the on-chip busses so that appropriate event counter bank can be credited in a peripheral.

- Debug tools and the 'energy digestor' require knowledge of bank to customer mapping - we provide this.

*The complete 'spEEDO package' will consist of H/W cells, debugger plugins and S/W library shims. These can be commercialised or open sourced.*
Tiny CLR demo.

VM on virtual platform!

.net/mono runtime

Gives energy use per CLR thread.

Perform profile-directed energy optimisations?

Compare results between H/W and virtual platform.

David J Greaves et al

FDL'15 Barcelona
## Energy Report With Customer Nos

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Statico Energy</th>
<th>Dynamic1 Energy</th>
<th>Wiring2 Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standalone modules:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...top.coreunit_0.core_0</td>
<td>9.997983e-05J</td>
<td>3.25128e-05J</td>
<td>1.3511615e-07J</td>
</tr>
<tr>
<td>Memory 0 (DRAM)</td>
<td>0.00866173075J</td>
<td>8.84e-07J</td>
<td>2.746e-12J</td>
</tr>
<tr>
<td>the_top_uart0</td>
<td>0J</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>Customer Accounts:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>anonymous</td>
<td>0.00866173075J</td>
<td>3.25128e-05J</td>
<td>2.6745349e-07J</td>
</tr>
<tr>
<td>busaccess_0</td>
<td>0J</td>
<td>0.00%</td>
<td>0J</td>
</tr>
<tr>
<td>TOP LEVEL++</td>
<td>0.00876171058J</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

Each line is for a separately-traced subsystem. These lines may be neither disjoint or complete. The TOP LEVEL figure is simply another line in the table that relates to the highest module found. Total energy used: 12900 uJ (12995854356318 fJ)

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Statico Power</th>
<th>Dynamic1 Power</th>
<th>Wiring2 Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standalone modules:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...top.coreunit_0.core_0</td>
<td>0.01W 75.38%</td>
<td>0.00325193592W</td>
<td>1.35143409e-05W</td>
</tr>
<tr>
<td>Memory 0 (DRAM)</td>
<td>0.866347818W 67.35%</td>
<td>0.42064464W 32.65%</td>
<td>1.3236129e-05W 0.00%</td>
</tr>
<tr>
<td>the_top_uart0</td>
<td>0W 0.00%</td>
<td>8.84178339e-05W 100.00%</td>
<td>2.74655e-10W 0.00%</td>
</tr>
<tr>
<td>Customer Accounts:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>anonymous</td>
<td>0.866347818W 99.62%</td>
<td>0.00325193592W 0.37%</td>
<td>2.67507446e-05W 0.00%</td>
</tr>
<tr>
<td>busaccess_0</td>
<td>0W 0.00%</td>
<td>0.420221111W 100.00%</td>
<td>0W 0.00%</td>
</tr>
<tr>
<td>TOP LEVEL++</td>
<td>0.876347818W 67.42%</td>
<td>0.423473047W 32.58%</td>
<td>2.67507446e-05W 0.00%</td>
</tr>
</tbody>
</table>

Each line is for a separately-traced subsystem. These lines may be neither disjoint or complete. The TOP LEVEL figure is simply another line in the table that relates to the highest module found. Average power used: 1290 mw (1299847614895725 fw)
Running on two cores...

<table>
<thead>
<tr>
<th>MODULE NAME</th>
<th>STATICO ENERGY</th>
<th>DYNAMIC1 ENERGY</th>
<th>WIRING2 ENERGY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standalone modules:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...top.coreunit_0.core_0</td>
<td>4.806e-08J</td>
<td>0.30%</td>
<td>1.3e-08J</td>
</tr>
<tr>
<td>...top.coreunit_1.core_1</td>
<td>4.806e-08J</td>
<td>0.30%</td>
<td>1.46e-08J</td>
</tr>
<tr>
<td>Memory 0 (DRAM)</td>
<td>1.04443197e-05J</td>
<td>64.51%</td>
<td>5.6217599e-06J</td>
</tr>
<tr>
<td>Customer Accounts:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>anonymous</td>
<td>1.04443197e-05J</td>
<td>64.51%</td>
<td>2.76e-08J</td>
</tr>
<tr>
<td>busaccess_0</td>
<td>0J</td>
<td>0.00%</td>
<td>2.89187835e-06J</td>
</tr>
<tr>
<td>busaccess_1</td>
<td>0J</td>
<td>0.00%</td>
<td>2.73060475e-06J</td>
</tr>
<tr>
<td>TOP LEVEL++</td>
<td>1.05404397e-05J</td>
<td>65.10%</td>
<td>5.6500831e-06J</td>
</tr>
</tbody>
</table>

Each line is for a separately-traced subsystem. These lines may be neither disjoint or complete. The TOP LEVEL figure is simply another line in the table that relates to the highest module found. Total energy used: 16100 nJ (16190844749 fJ)

<table>
<thead>
<tr>
<th>MODULE NAME</th>
<th>STATICO POWER</th>
<th>DYNAMIC1 POWER</th>
<th>WIRING2 POWER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standalone modules:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...top.coreunit_0.core_0</td>
<td>0.01W</td>
<td>78.59%</td>
<td>0.00270495214W</td>
</tr>
<tr>
<td>...top.coreunit_1.core_1</td>
<td>0.01W</td>
<td>76.60%</td>
<td>0.00303786933W</td>
</tr>
<tr>
<td>Memory 0 (DRAM)</td>
<td>2.17318346W</td>
<td>65.01%</td>
<td>1.16973781W</td>
</tr>
<tr>
<td>Customer Accounts:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>anonymous</td>
<td>2.17318346W</td>
<td>99.73%</td>
<td>0.00574282147W</td>
</tr>
<tr>
<td>busaccess_0</td>
<td>0W</td>
<td>0.00%</td>
<td>0.601722503W</td>
</tr>
<tr>
<td>busaccess_1</td>
<td>0W</td>
<td>0.00%</td>
<td>0.568165783W</td>
</tr>
<tr>
<td>TOP LEVEL++</td>
<td>2.19318346W</td>
<td>65.10%</td>
<td>1.17563111W</td>
</tr>
</tbody>
</table>
Future Work

- FPGA Implementation of the digestor.
- LowRISC RISC-V SoC implementation.
- Event/fluent/energy/power abstract modelling calculus, simulator and extrapolator.
- spEEDO-2 applied to UK gov for funding.
- Liase with industrial partners.
Current/power versus time plots.
(note monostable back edges)

Integer ALU

Floating Point ALU

Memory Access: Disjoint Footprints

Memory Access: Overlapping Footprint

Vertical bar -> 1 second.
Horizontal scale -> 100 Watts.

System has 6 cores sharing one DRAM bank.
<table>
<thead>
<tr>
<th>Event/Fluent/Energy/Power calculus+modelling language prims</th>
</tr>
</thead>
</table>

### EVENT SOURCE
- **1e7 0.5**
- **Param:** mean rate
- **Param:** coef var
- **Output:** event

### OR GATE
- **(events or fluents)**
- **Polymorphic Inputs:** Fluents
- **Output:** Fluents OR
- **Inputs:** events
- **Output:** event

### AND GATE
- **(fluents only unless common-src events)**
- **Inputs:** fluents
- **Output:** fluent

### MONOSTABLES
- **RETRIGGERABLE**
- **15ms**
- **Param:** delay
- **Input:** fluent or event
- **Output:** fluent

### TEMPORAL FORK
- **Input:** event
- **Outputs:** events

### TEMPORAL JOINS
- **EUREKA JOIN**
- **GATED JOIN**

### SUMMATION
- **(unobservable)**
- **5pJ**
- **Input:** event
- **Output:** energy

### EVENT DISSIPATOR
- **3mW**
- **Input:** fluent
- **Output:** power

### FLUENT DISSIPATOR
- **INERTIAL DELAY**
- **100us**
- **15ms**
- **Param:** delay time
- **Input:** fluent
- **Output:** event

### TRANSPORT DELAY
- **25ms**
- **Param:** delay time
- **Input:** fluent
- **Output:** fluent

### OBSERVABLE METRIC
- **M1**
- **Input:** any
  - logs energy or power
  - counts events
  - averages fluents
lowRiSC
RISC-V
Open Source SoC

- “lowRiSC is producing fully open hardware systems. From the processor core to the development board, our goal is to create a completely open computing eco-system.”

- spEEDO banked energy registers to be contributed.

- spEEDO energy digestor perhaps to run on one core or else be in H/W.

- See openrisc.io/orconf (Geneva 9-11th October 2015).
Thankyou for listening

David J Greaves
Ali M Zaidi
M Puzovic
Klaus McDonald-Maier
Andrew Hopkins

University of Cambridge
Computer Laboratory

FDL'15
Power Aware Modelling + Design Session.
BACKUP SLIDES NOW FOLLOW

....

TLM Modelling
and TLM POWER 3
SMP OpenRISC Demo Platform

1 to 64 cores (four shown)
Shared or split or no L1 Cache
Flexible cache architectures
L2 and L3 caches easily added

Each cache has power-annotated tag and data RAMs
SRAM parameters from CACTI
DRAM modelled by Univ Maryland DRAMSIM2

Verilated OpenRISC Core
Or fast ORSIM ISS
(Or MIPS64)
typedef unsigned int customer_t; // Value zero is reserved to denote the system global total.

extern customer_t get_local_customer_no();
extern int get_context_field(customer_t c);
extern int get_core_field(customer_t c);

int get_local_core_no() { return get_core_field(get_local_customer_no()); }
int get_local_context_no() { return get_context_field(get_local_customer_no()); }
Note that the roles of initiator and target do not necessarily relate to the sources and sinks of the data.

Infact, an initiator can commonly make both a read and a write transaction on a given target and so the direction of data transfer is dynamic.
### TLM: Loose Timing

<table>
<thead>
<tr>
<th>Naive Coding Style</th>
<th>Loosely-Timed Coding Style</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>b_putbyte(char d)</code></td>
<td></td>
</tr>
<tr>
<td>`{</td>
<td></td>
</tr>
<tr>
<td>printf(&quot;Byte '%c'\n&quot;, d);</td>
<td></td>
</tr>
<tr>
<td>wait(250, SC_NS);</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

|                     | Have a local variable 'delay' associated with each thread. |
|                     | `b_putbyte(char d, sc_time &delay)` |
|                     | `{                                 |
|                     | sc_time del(250, SC_NS);          |
|                     | printf("Byte '%c'\n", d);        |
|                     | delay += del;                     |
|                     | }                                 |

But, at any point, any thread can resynch itself with the kernel by performing:

```
// Resynch idiomatic form:
sc_wait(delay);
Delay = 0;
```

Simulation performance is reduced when there are frequent resynchs, but true transaction ordering will be modelled correctly.
Loosely-timed TLM Modelling: General Structure
Spatial Layout Support

- Every SC_MODULE has a chip/region designation.
- The area of a module is sum of
  - its children with the same chip/region name
  - its locally defined 'excess area'.
- Inter-module wiring lengths can be estimated using Rent's Rule on area of lowest-common-parent.
- Actual X-Y co-ordinates could be allocated by aplacer.
Report Formats (3: VCD)

- Each account and their summations can be plotted in various forms
  - 1: Ascii-art table format
  - 2: SYLK or CSV spreadsheet format
  - 3: VCD temporal display (using dirac impulse response or average over interval)
- A physical layout file is also written.
# Report Formats (2: Ascii-art text file)

---

```
# TLM POWER3 (Univ Cambridge, UK)
#
# Statistics file: energy/power consumption.
#
# For more information see the TLM POWER3 manual pdf.
#
# Creation Date: 17:27:22 -- 15/09/2012
```

Title: privmem-cln6000-dramsim-withcache-nile-gash-harvard
# Simulation duration: 24826590001096 ps
# Simulation duration: 24826590001096 ps

<table>
<thead>
<tr>
<th>MODULE NAME</th>
<th>STATIC$ ENERGY</th>
<th>DYNAMIC$ ENERGY</th>
<th>WIRING$ ENERGY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory 0 (DRAM)</td>
<td>0.1738739501J 3.49%</td>
<td>0.0875462788J 1.76%</td>
<td>4.48687512e-07J 0.00%</td>
</tr>
<tr>
<td>the_top_uart0</td>
<td>0J 0.00%</td>
<td>1.644e-06J 0.00%</td>
<td>6.7041e-11J 0.00%</td>
</tr>
<tr>
<td>the_top_busmux0</td>
<td>0J 0.00%</td>
<td>1.1905216e-05J 0.00%</td>
<td>0J 0.00%</td>
</tr>
<tr>
<td>the_top_dram=0</td>
<td>0.1738739501J 3.49%</td>
<td>0.0875462788J 1.76%</td>
<td>4.48687512e-07J 0.00%</td>
</tr>
<tr>
<td>...top.coreunit_0.core_0</td>
<td>0.2482659J 4.99%</td>
<td>0.0044012626J 0.09%</td>
<td>1.34648772e-05J 0.00%</td>
</tr>
<tr>
<td>...reunit_0.l1_d_cache_0</td>
<td>0J 0.00%</td>
<td>0.000594064671J 0.01%</td>
<td>6.14810556e-06J 0.00%</td>
</tr>
<tr>
<td>...l1_d_cache_0.Data_0</td>
<td>0.0333542257J 0.67%</td>
<td>0.000107935695J 0.00%</td>
<td>0J 0.00%</td>
</tr>
<tr>
<td>...l1_d_cache_0.Tags_0</td>
<td>0.0317907464J 0.64%</td>
<td>4.18042825e-05J 0.00%</td>
<td>0J 0.00%</td>
</tr>
<tr>
<td>...l1_d_cache_0.Data_1</td>
<td>0.0333542257J 0.67%</td>
<td>0.000105833853J 0.00%</td>
<td>0J 0.00%</td>
</tr>
<tr>
<td>...l1_d_cache_0.Tags_1</td>
<td>0.0317907464J 0.64%</td>
<td>3.37903219e-05J 0.00%</td>
<td>0J 0.00%</td>
</tr>
<tr>
<td>...l1_d_cache_0.Data_2</td>
<td>0.0333542257J 0.67%</td>
<td>0.000105435493J 0.00%</td>
<td>0J 0.00%</td>
</tr>
<tr>
<td>...l1_d_cache_0.Tags_2</td>
<td>0.0317907464J 0.64%</td>
<td>2.60627187e-05J 0.00%</td>
<td>0J 0.00%</td>
</tr>
<tr>
<td>...l1_d_cache_0.Data_3</td>
<td>0.0333542257J 0.67%</td>
<td>0.000108887529J 0.00%</td>
<td>0J 0.00%</td>
</tr>
<tr>
<td>...l1_d_cache_0.Tags_3</td>
<td>0.0317907464J 0.64%</td>
<td>1.83743234e-05J 0.00%</td>
<td>0J 0.00%</td>
</tr>
</tbody>
</table>

---

David J Greaves et al

FDL'15 Barcelona
Measured v Predicted: Runs 19-24 extrapolated from data fitting on 1-18.
C API – Registers via HAL

extern u32_t get_units();

extern u32_t get_local_energy(); // same as get_customer_energy(get_local_core_no());

extern u32_t get_customer_energy(customer_t customer_no);

extern u32_t get_global_energy();

extern const char *get_reflection_uri();

extern int reset_energy_counters(u32_t mask);
    // Returns 0 if ok.
    // Returns -ve error code if a selected register cannot be reset.

extern float report_average_power(customer_t no, int window_milliseconds) ... // TBD some
Power Estimation: Project Flow

Operating System Fragments

- SPLASH-2 benchmarks
- Application benchmarks
- OPEN MPI benchmarks

Real Hardware
- gcc-x86_64
- Various SMP Linux machines to hand

C Compilers
- gcc-x86_64 or gcc-MIPS
- llvm-gcc

High-level models
- ISS MIPS
- ISS x86
- DRAM
- HDD
- TLM-2.0 library
- TLM power library

SystemC kernel

POWER RESULTS

Very-high level model
- LLVM/VSDG interpreter
- Power modeller

POWER RESULTS

Web Data Upload
- Web Experiment Submit
- Web Data Download

PHP Infrastructure: Database, Simulation Platform and HTML/XML Remote Access

Greenfield/Rent Models
- Linear Regression fitting (etcetera)

Locally-Generated Power Results

Results from literature survey

Third party uploads

David J Greaves et al

FDL’15 Barcelona