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 ?

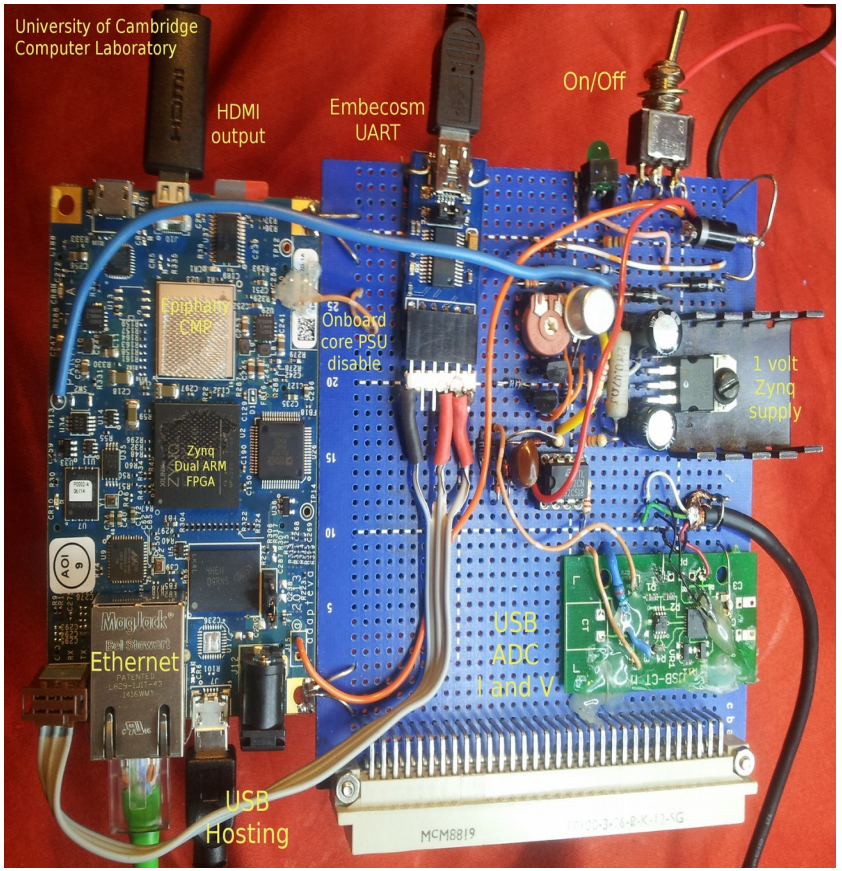
Is compiler option -Oblah helpful in terms of total energy for this task ?

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.

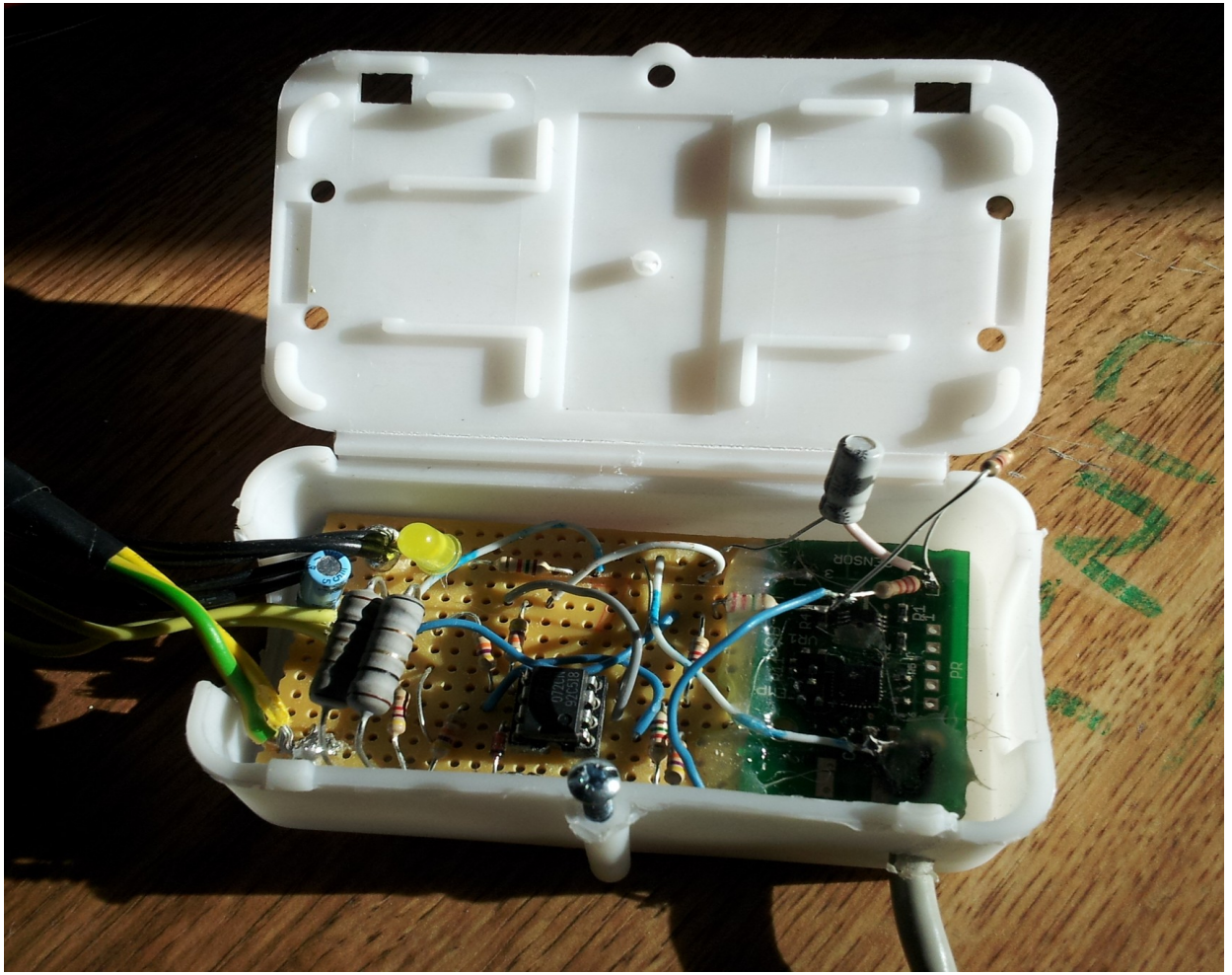
Initial achievements:

- 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

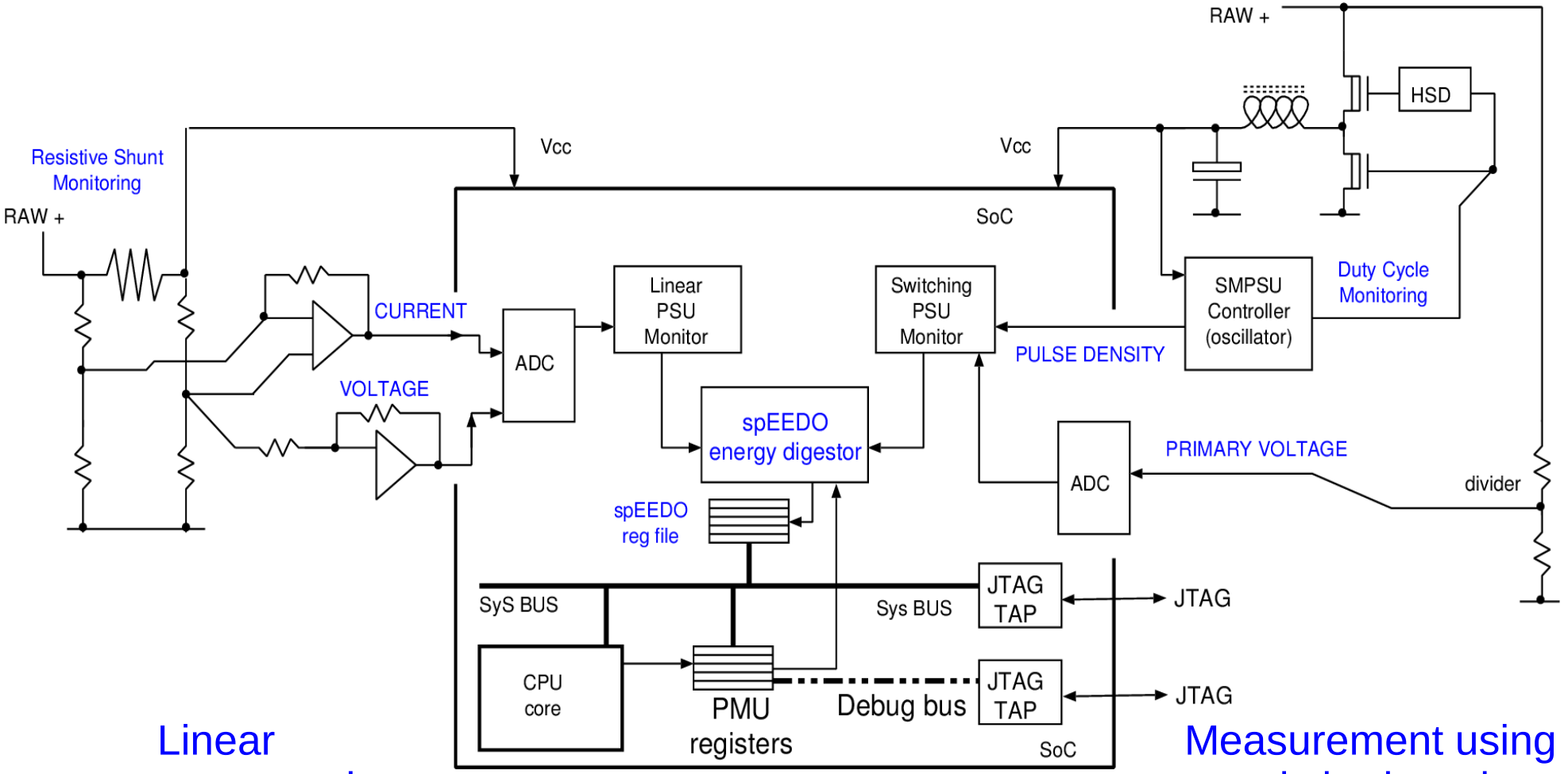


Measures 12 volt rail to motherboard CPU socket.

Measures volts and amps at 10 Hz rate.

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

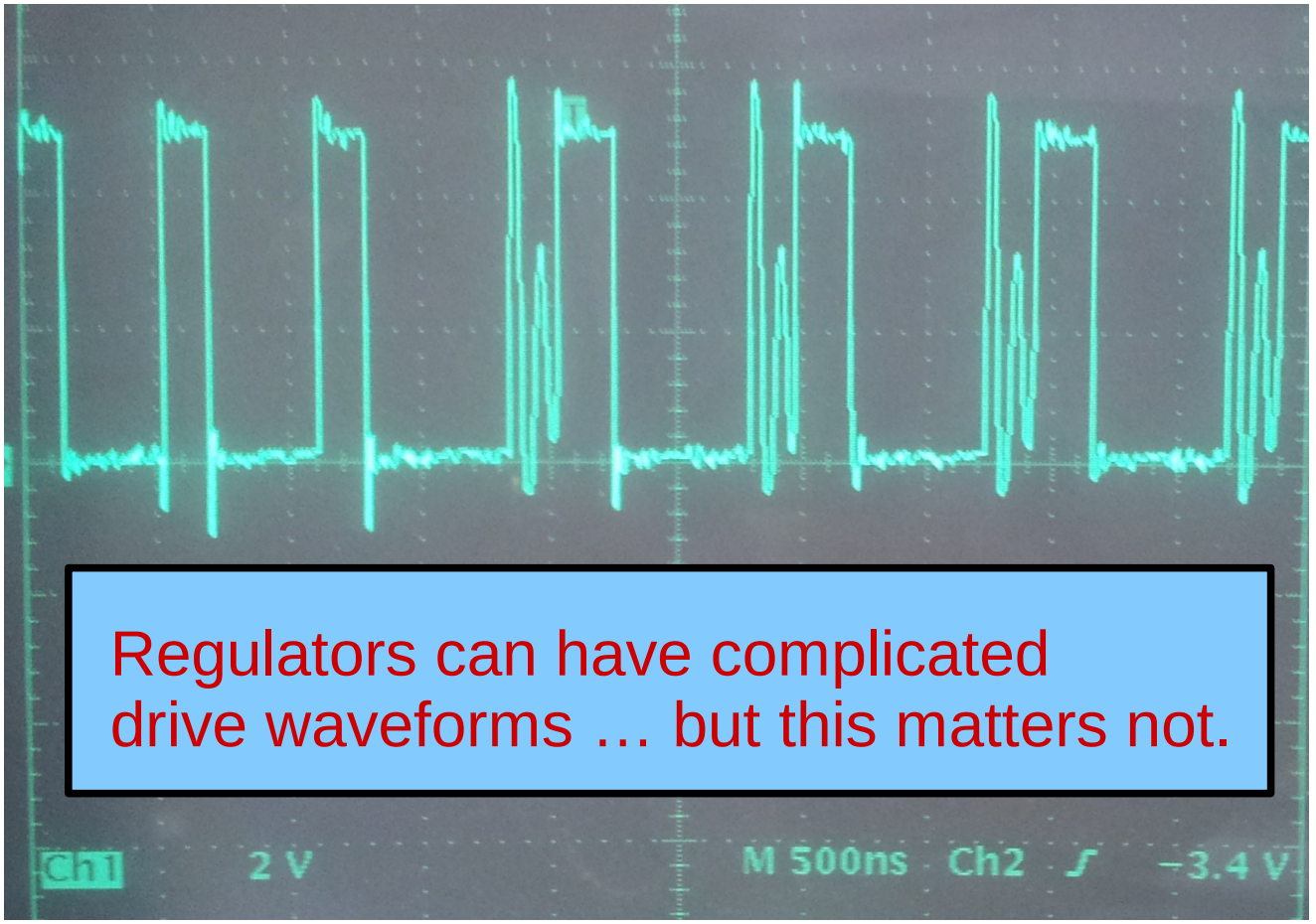
New Power Supply Monitors



Linear measurement using resistive shunt

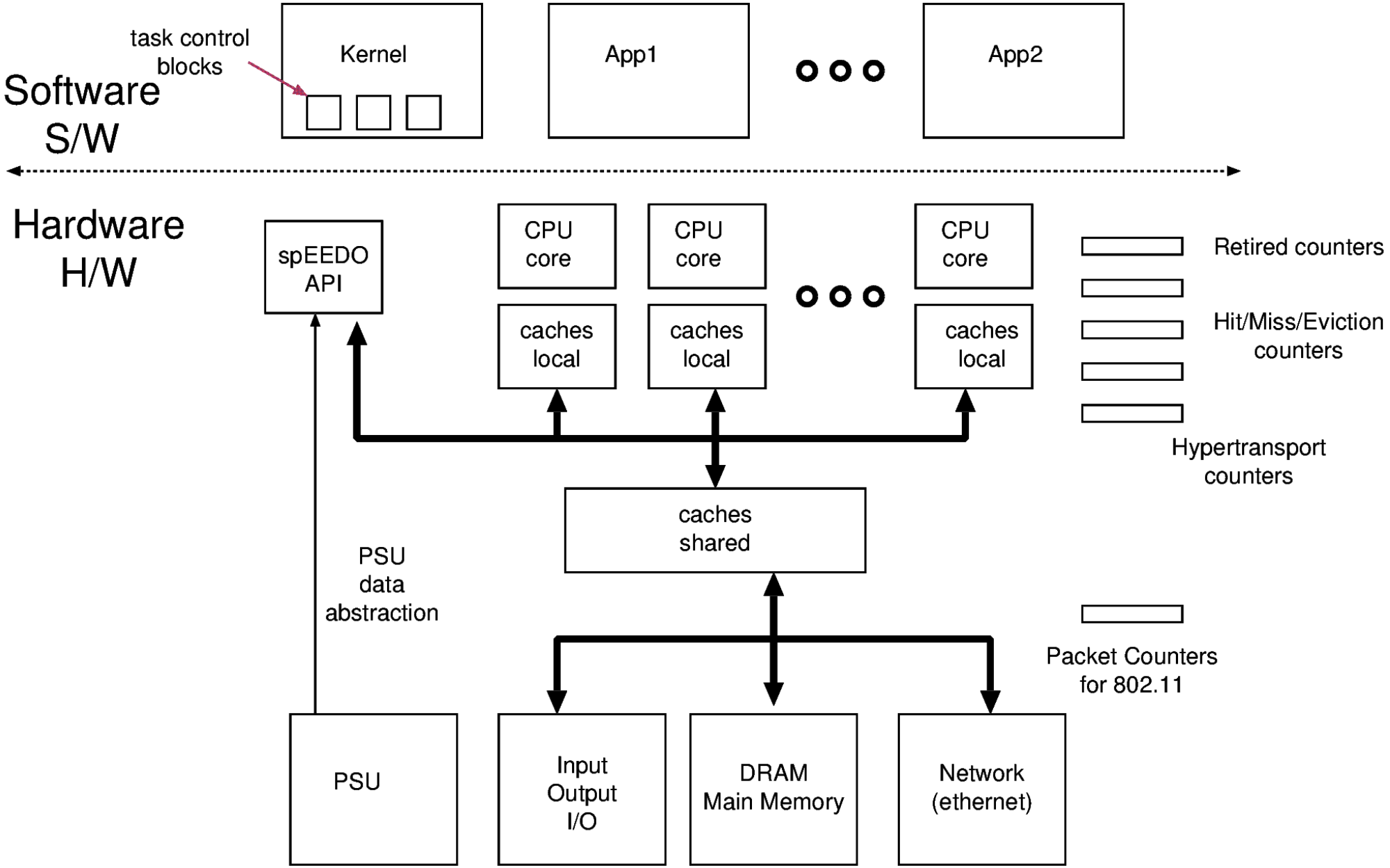
Measurement using switched-mode (SMPSU) duty cycle measurement

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.

Basic Blade/PC Architecture



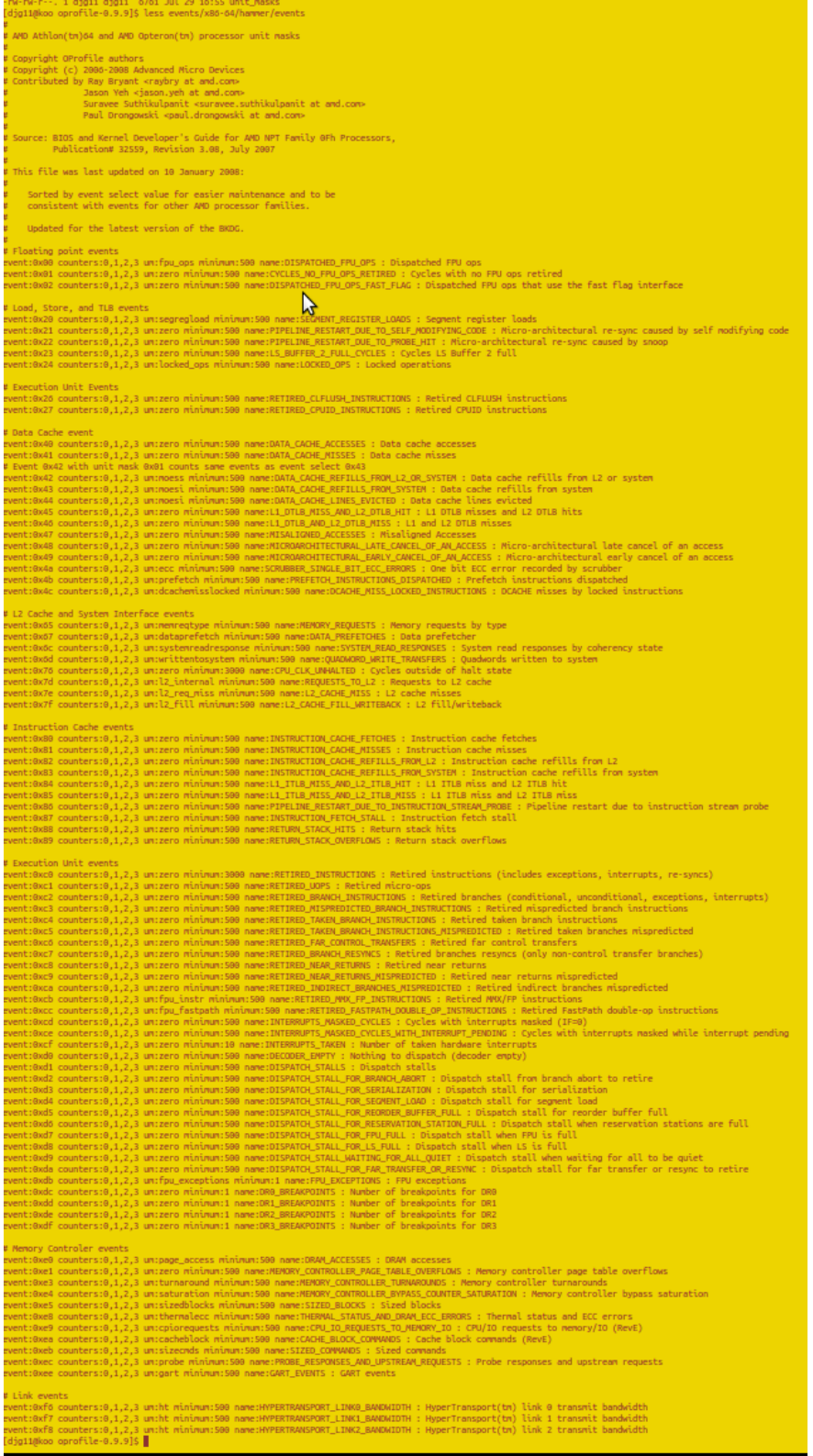
MSRs

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



David J Greaves et al

Intel's Power Gadget MSRs

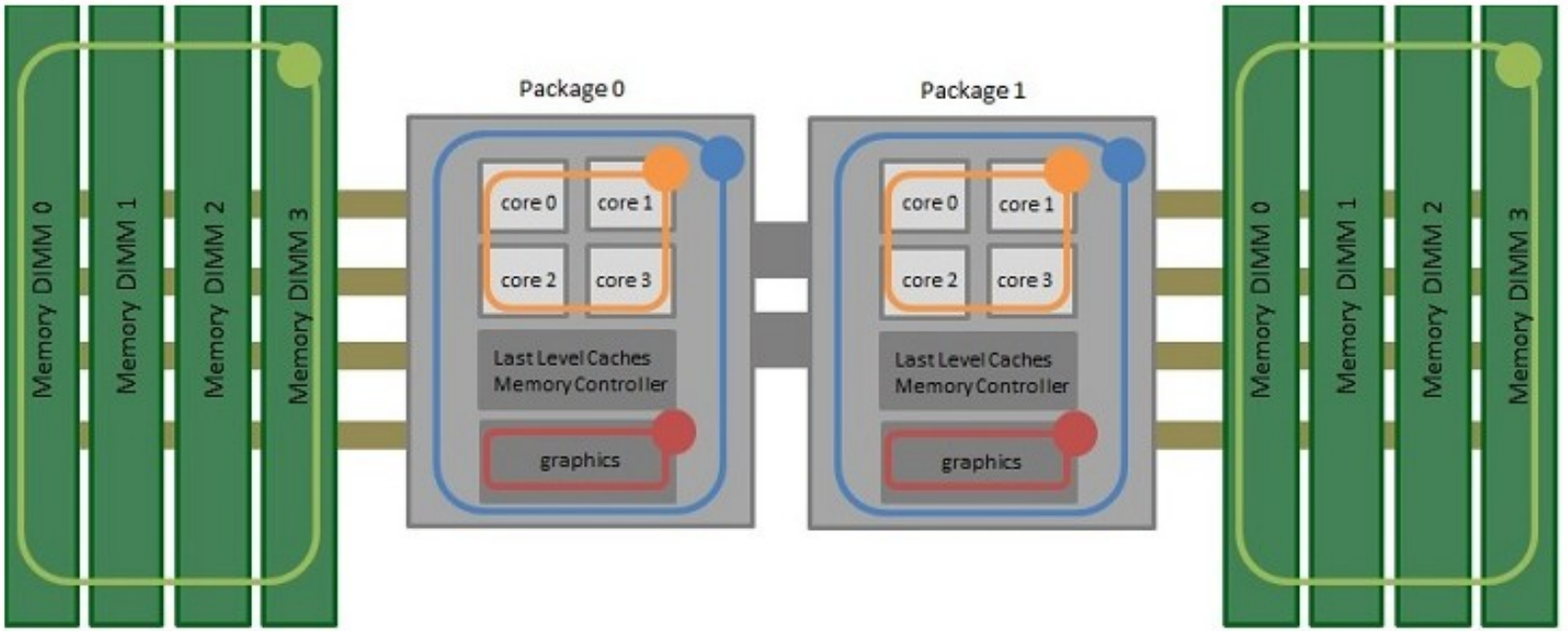
Intel has implemented a Running Average Power Limit (RAPL) on Sandybridge processors.

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

SandyBridge:

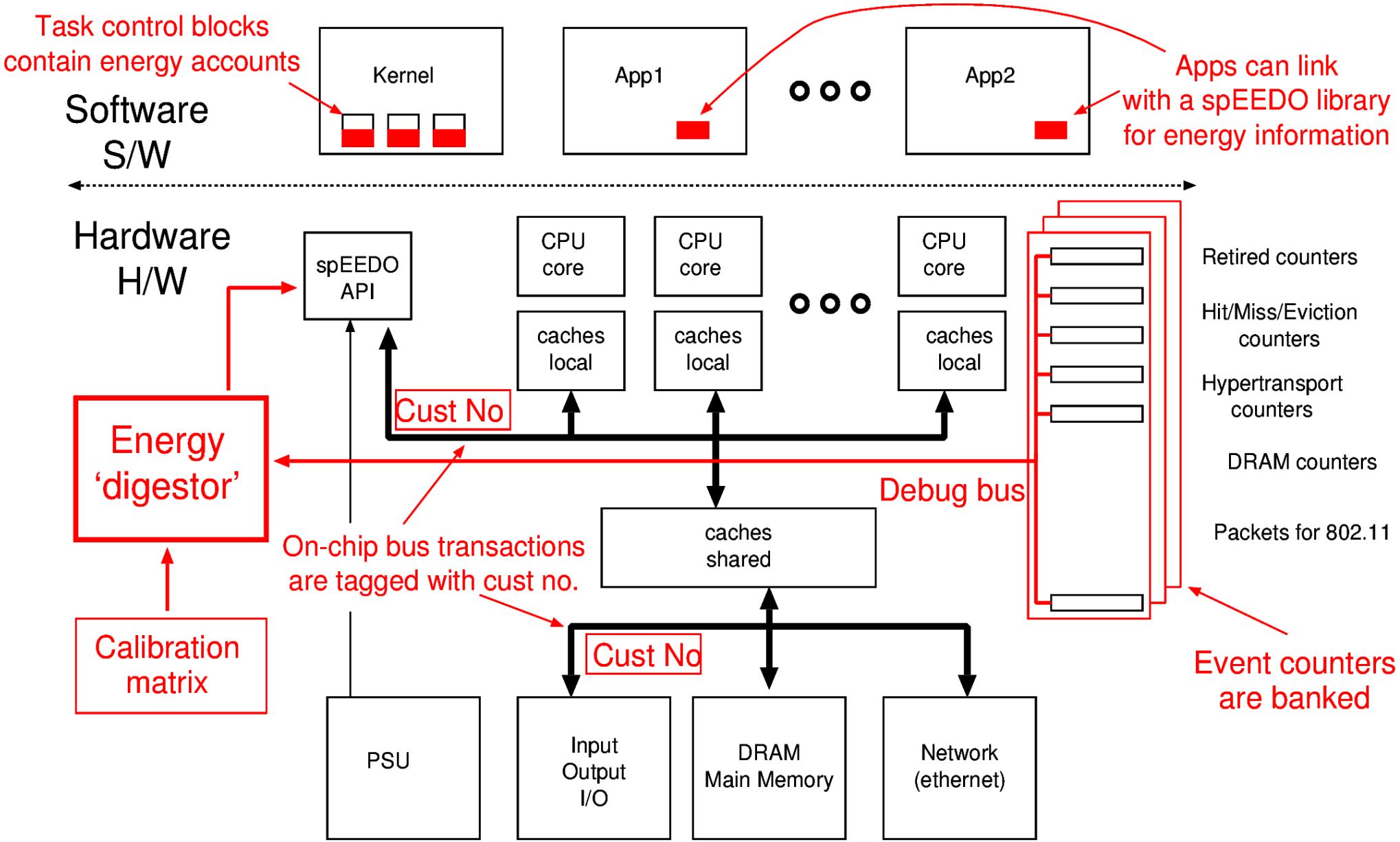
```
MSR_RAPL_POWER_UNIT   MSR_PKG_POWER_LIMIT    MSR_PKG_ENERGY_STATUS   MSR_PP0_POLICY
MSR_PP0_PERF_STATUS   MSR_PKG_POWER_INFO     MSR_PP0_POWER_LIMIT     MSR_PP0_ENERGY_STATUS
```

»Measuring Energy Consumption for Short Code Paths Using RAPL. Hähnel 2012



- package power plane
- pp0/core power plane (all cores on the package)
- pp1/graphics power plane (client only)
- DRAM power plane (server only)

Augmented Reference Architecture



Software Event Counts

Typical device driver stats:

```
eth0      Link encap:Ethernet  HWaddr 00:13:20:84:5d:81
          inet addr:128.232.9.140  Bcast:128.232.15.255  Mask:255.255.240.0
          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 spEEDO hardware API - unbanked version.

#define SPEEDO_REG_MONICA            0   // Contains an identifying constant
#define SPEEDO_REG_ABI               8   // Version number of the interface
#define SPEEDO_REG_ENERGY_UNITS      16  // Energy units for the following
#define SPEEDO_REG_CMD_STATUS        40  // Capability description and commands for res
#define SPEEDO_REG_GLOBAL_ENERGY     48  // Running total energy in the units given - i
#define SPEEDO_REG_TIME_UNITS        56  // Units for ticks in the time register.

#define SPEEDO_REG_CTX0_BASE     512
#define SPEEDO_REG_CTX1_BASE     (512+256)

#define SPEEDO_REFLECTION_URL0   1024 // First location of a canned URL giving further


// Each hardware context contains:

#define SPEEDO_CTX_REG_LOCAL_ENERGY  8  // Running local energy in the units given
#define SPEEDO_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:

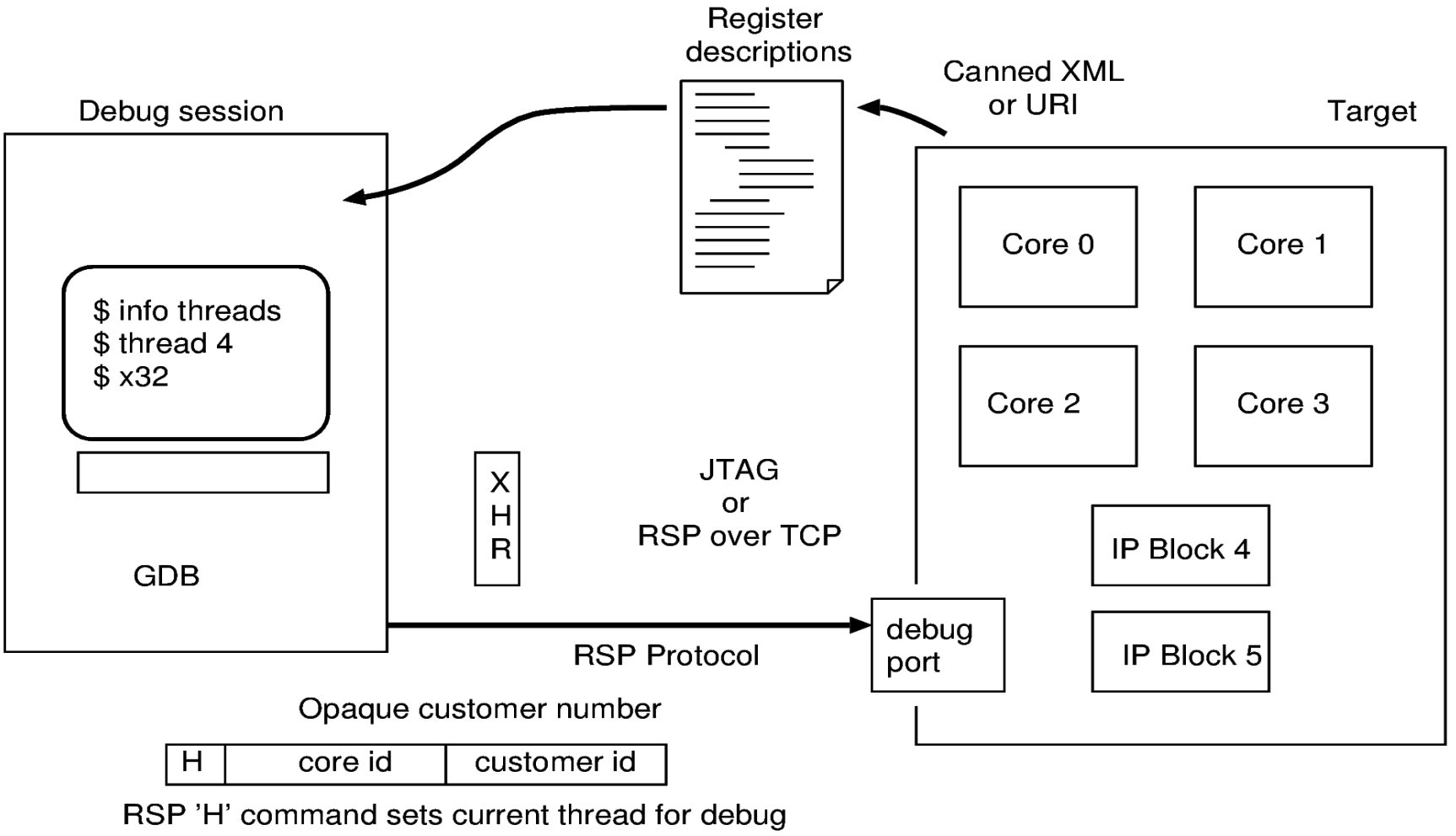
```
#define SOCDAM_SPEEDO_REGS_BASE   0xFFFD0000
#define READ_SPEEDO(X)    (((unsigned int  *)(SOCDAM_SPEEDO_REGS_BASE + X))[0])



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));
  _killsim(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.)

```
Hello World 45457073

Global energy units at start are 847327

Core 0: Energy units are 524070

Core 0: Energy units are 846693

Core 0: Energy units are 1171122

Core 0: Energy units are 1511514

Core 0: Energy units are 1852918

Core 0: Energy units are 2195073

Core 0: Energy units are 2537936

Core 0: Energy units are 2880756

Core 0: Energy units are 3224286

Core 0: Energy units are 3568353

Global energy units at end are 12006801
```

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.

Baseline GDB energy reporting ...

```
(gdb) info all-registers
r0             0x0        0
r1             0x0        0x0
r2             0x0        0x0
r3             0x0        0
r4             0x0        0
r5             0x0        0
r6             0x0        0
r7             0x0        0
r8             0x0        0
               0x0        0
               0x0        0
r30            0x0        0
r31            0x0        0
ppc            0x0        0
npc            0x100      0x100 <__reset>
sr             0x8001     32769
(gdb) gdbEPT
Energy = 256 j, Time = 0 ms, Power = 0 mW
(gdb)
```

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.

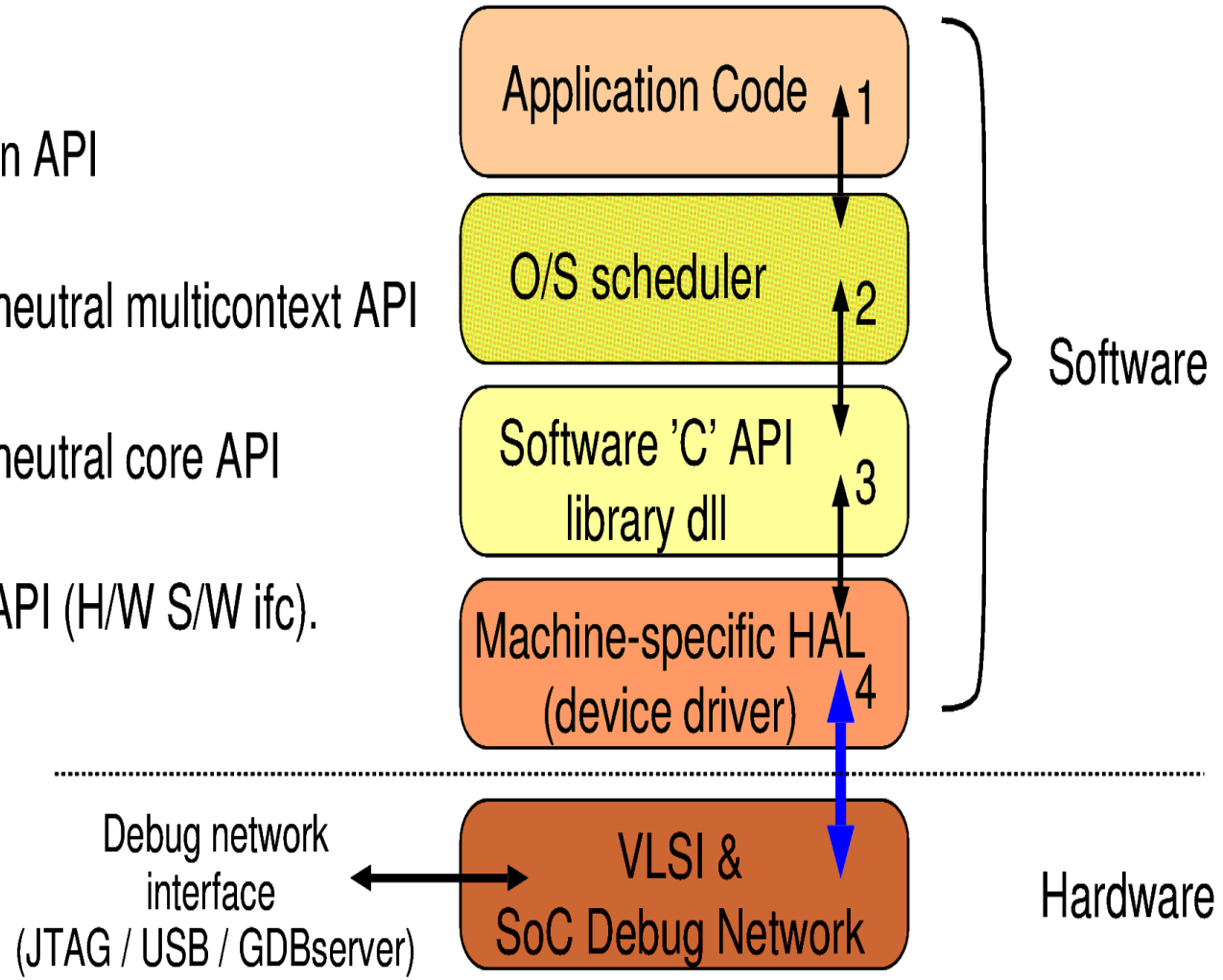
spEEDO API Stack

1) Application API

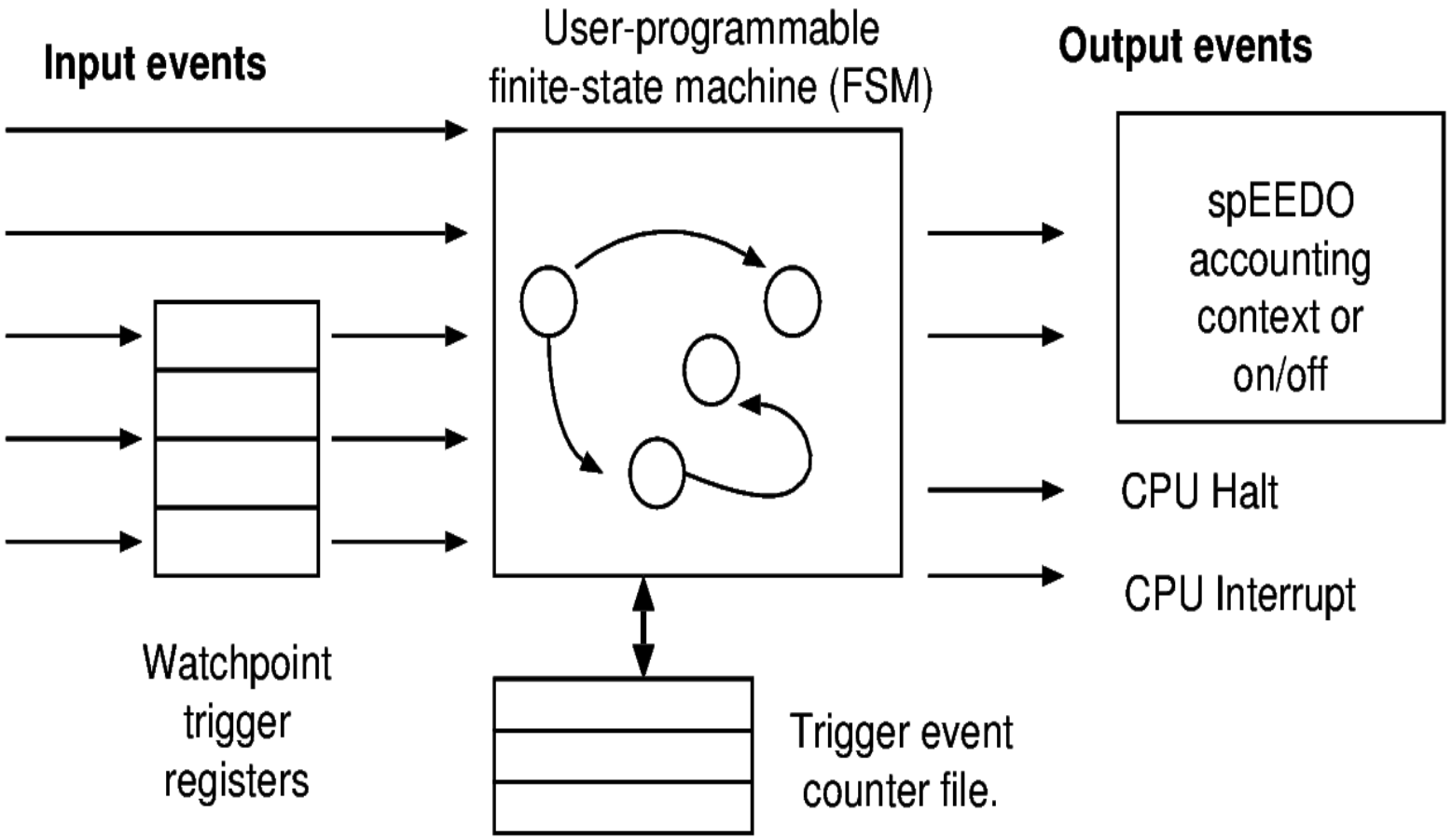
2) Machine-neutral multicontext API

3) Machine-neutral core API

4) Register API (H/W S/W ifc).



Programmable FSMs



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.

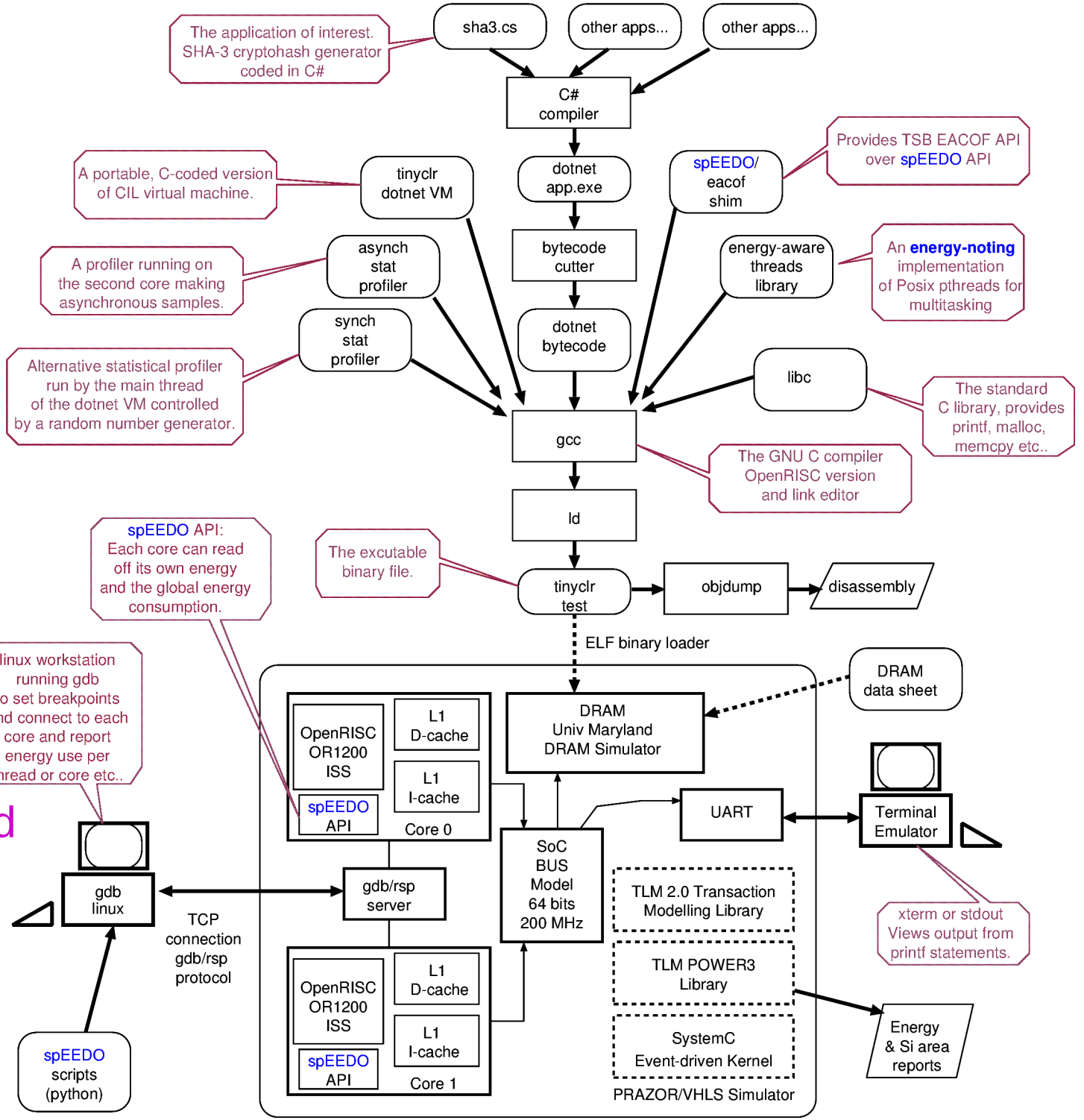
VM on virtual platform!

.net/mono runtime

Gives energy use per CLR thread.

Perform profile-directed energy optimisations?

Compare results between H/W and virtual plaform.



Energy Report With Customer Nos

MODULE NAME	STATIC0 ENERGY		DYNAMIC1 ENERGY		WIRING2 ENERGY	
Standalone modules:						
...top.coreunit_0.core_0	9.997983e-05J	0.77%	3.25128e-05J	0.25%	1.35116151e-07J	0.00%
Memory 0 (DRAM)	0.00866173075J	66.65%	0.00419979737J	32.32%	1.32334593e-07J	0.00%
the_top.uart0	0J	0.00%	8.84e-07J	0.01%	2.746e-12J	0.00%
Customer Accounts:						
anonymous	0.00866173075J	66.65%	3.25128e-05J	0.25%	2.6745349e-07J	0.00%
busaccess_0	0J	0.00%	0.00420136352J	32.33%	0J	0.00%
TOP LEVEL++	0.008761710 58J	67.42%	0.00423387632J	32.58%	2.6745349e-07J	0.00%

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)

MODULE NAME	STATIC0 POWER		DYNAMIC1 POWER		WIRING2 POWER	
Standalone modules:						
...top.coreunit_0.core_0	0.01W	75.38%	0.00325193592W	24.51%	1.35143409e-05W	0.10%
Memory 0 (DRAM)	0.866347818W	67.35%	0.420064464W	32.65%	1.3236129e-05W	0.00%
the_top.uart0	0W	0.00%	8.84178339e-05W	100.00%	2.74655e-10W	0.00%
Customer Accounts:						
anonymous	0.866347818W	99.62%	0.00325193592W	0.37%	2.67507446e-05W	0.00%
busaccess_0	0W	0.00%	0.420221111W	100.00%	0W	0.00%
TOP LEVEL++	0.876347818W	67.42%	0.423473047W	32.58%	2.67507446e-05W	0.00%

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...

MODULE NAME	STATIC0 ENERGY		DYNAMIC1 ENERGY		WIRING2 ENERGY	
Standalone modules:						
...top.coreunit_0.core_0	4.806e-08J	0.30%	1.3e-08J	0.08%	9.0815e-11J	0.00%
...top.coreunit_1.core_1	4.806e-08J	0.30%	1.46e-08J	0.09%	8.411e-11J	0.00%
Memory 0 (DRAM)	1.04443197e-05J	64.51%	5.6217599e-06J	34.72%	1.46992e-10J	0.00%
Customer Accounts:						
anonymous	1.04443197e-05J	64.51%	2.76e-08J	0.17%	3.21917e-10J	0.00%
busaccess_0	0J	0.00%	2.89187835e-06J	17.86%	0J	0.00%
busaccess_1	0J	0.00%	2.73060475e-06J	16.87%	0J	0.00%
TOP LEVEL++	1.05404397e-05J	65.10%	5.6500831e-06J	34.90%	3.21917e-10J	0.00%

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)

MODULE NAME	STATIC0 POWER		DYNAMIC1 POWER		WIRING2 POWER	
Standalone modules:						
...top.coreunit_0.core_0	0.01W	78.59%	0.00270495214W	21.26%	1.88961715e-05W	0.15%
...top.coreunit_1.core_1	0.01W	76.60%	0.00303786933W	23.27%	1.75010404e-05W	0.13%
Memory 0 (DRAM)	2.17318346W	65.01%	1.16973781W	34.99%	3.0585102e-05W	0.00%
Customer Accounts:						
anonymous	2.17318346W	99.73%	0.00574282147W	0.26%	6.69823138e-05W	0.00%
busaccess_0	0W	0.00%	0.601722503W	100.00%	0W	0.00%
busaccess_1	0W	0.00%	0.568165783W	100.00%	0W	0.00%
TOP LEVEL++	2.19318346W	65.10%	1.17563111W	34.90%	6.69823138e-05W	0.00%

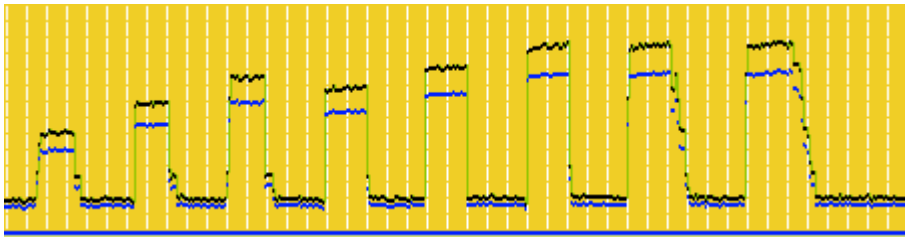
Future Work

- 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)

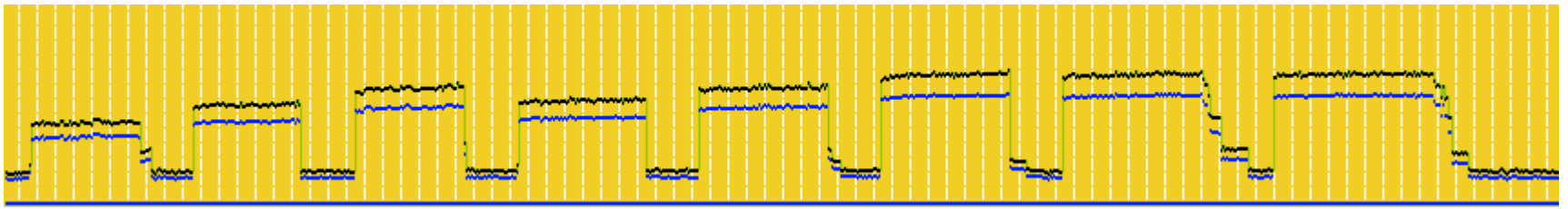
Integer ALU



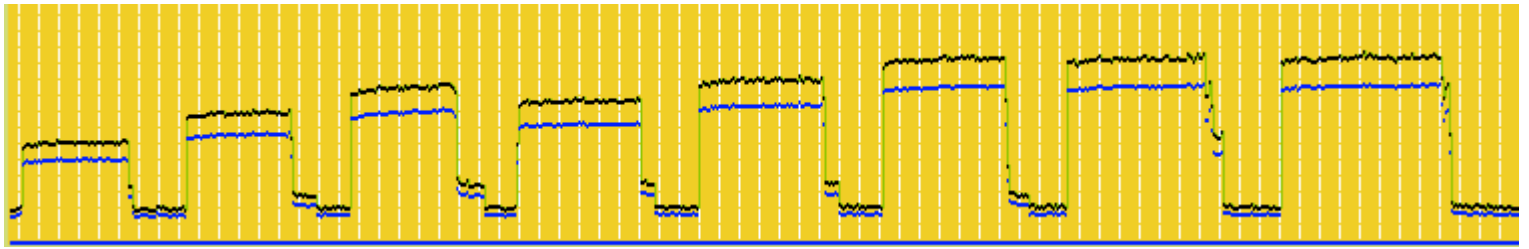
Vertical bar -> 1 second.
Horizontal scale -> 100 Watts.

System has 6 cores sharing one DRAM bank.

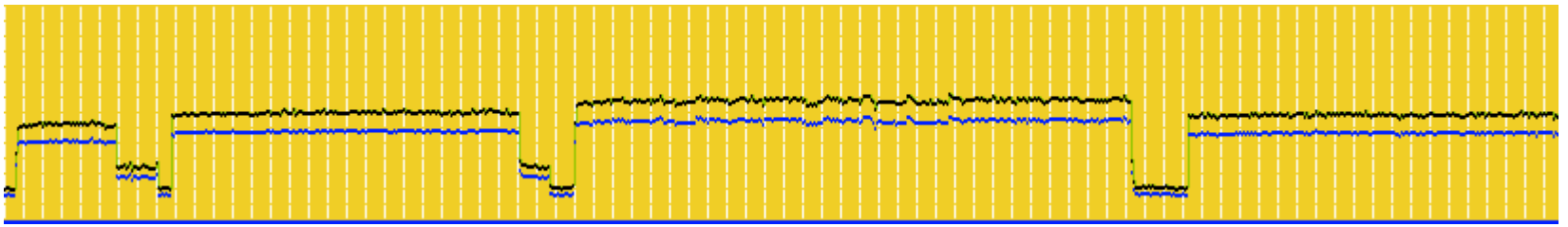
Floating Point ALU



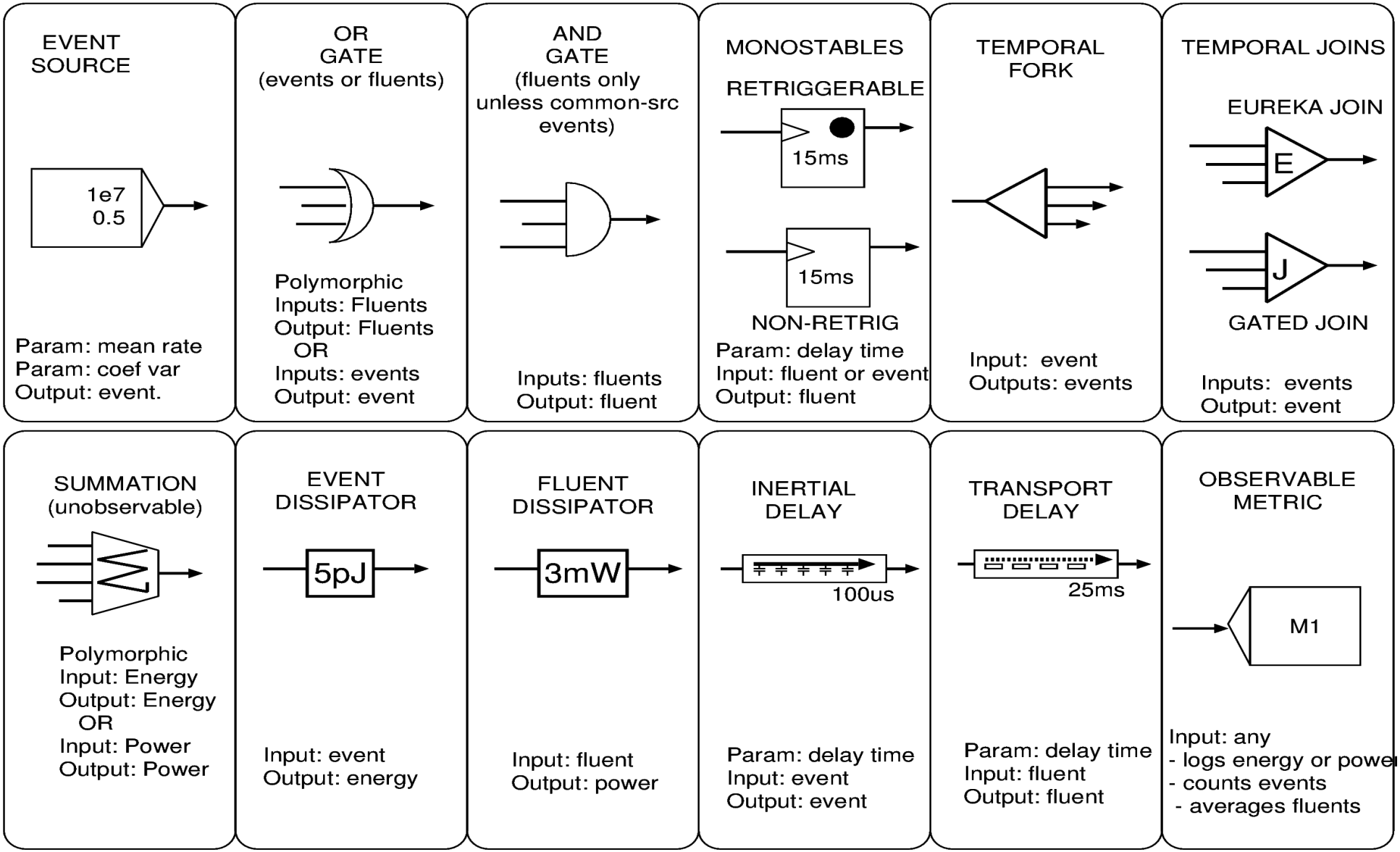
Memory Access: Disjoint Footprints



Memory Access: Overlapping Footprint



Event/Fluent/Energy/Power calculus+modelling language prims



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

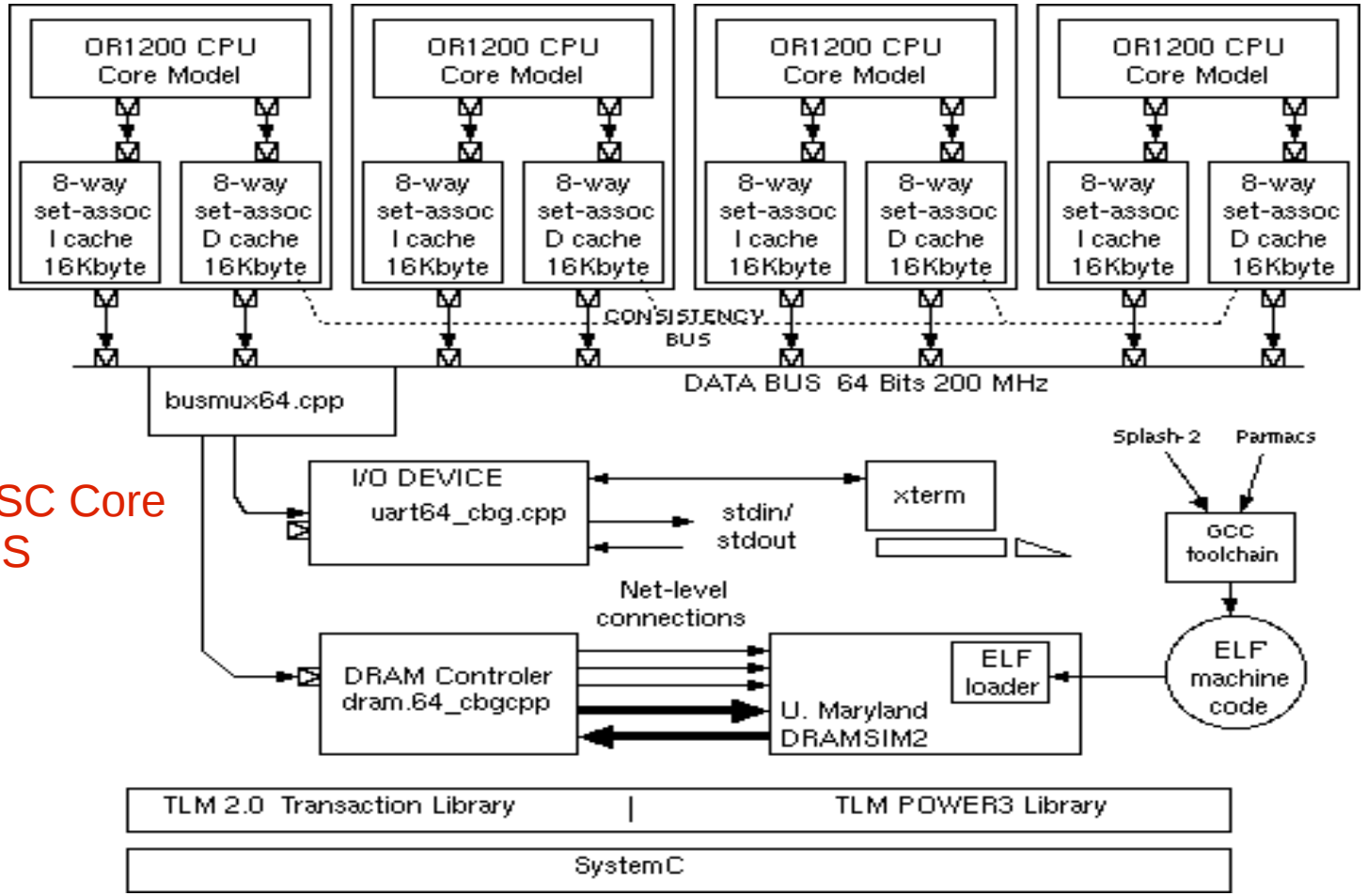
FDL'15
Power Aware Modelling + Design Session.

BACKUP SLIDES NOW FOLLOW

....

TLM Modelling
and TLM POWER 3

SMP OpenRISC Demo Platform



Verilated OpenRISC Core
Or fast ORSIM ISS
(Or MIPS64)

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

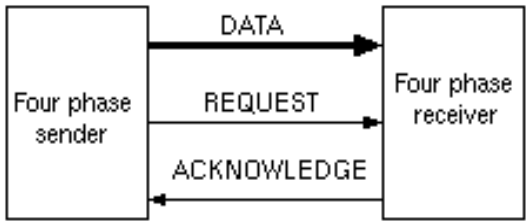
Each cache has power-annotated tag and data RAMs
SRAM parameters from CACTI
DRAM modelled by Univ Maryland DRAMSIM2

Customer Number

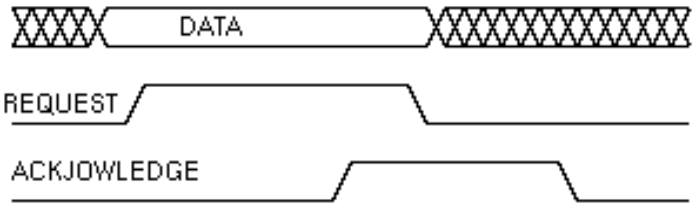
```
typedef unsigned int customer_t; // Value zero is reserved to denote the system global total.


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()); }
```

Transaction Level Modelling

Net-level (pin-level) interconnection.



Net-level protocol

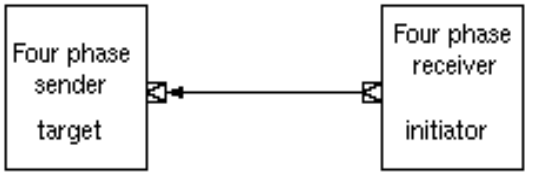


TLM push configuration.


```
char mydata = ... src ...;
target.putchar(mydata);
```

```
void target::putchar(char d)
{
  //.. do something with data
}
```

TLM pull configuration.




```
char target::getchar()
{
   char mydata = ... src ...;
   return mydata;
}
```

```
mydata = target.getchar();
```

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

Naive Coding Style

```
b_putbyte(char d)
{
  printf("Byte '%c'\n", d);
  wait(250, SC_NS);
}
```

Loosely-Timed Coding Style

Have a local variable 'delay' associated with each thread.

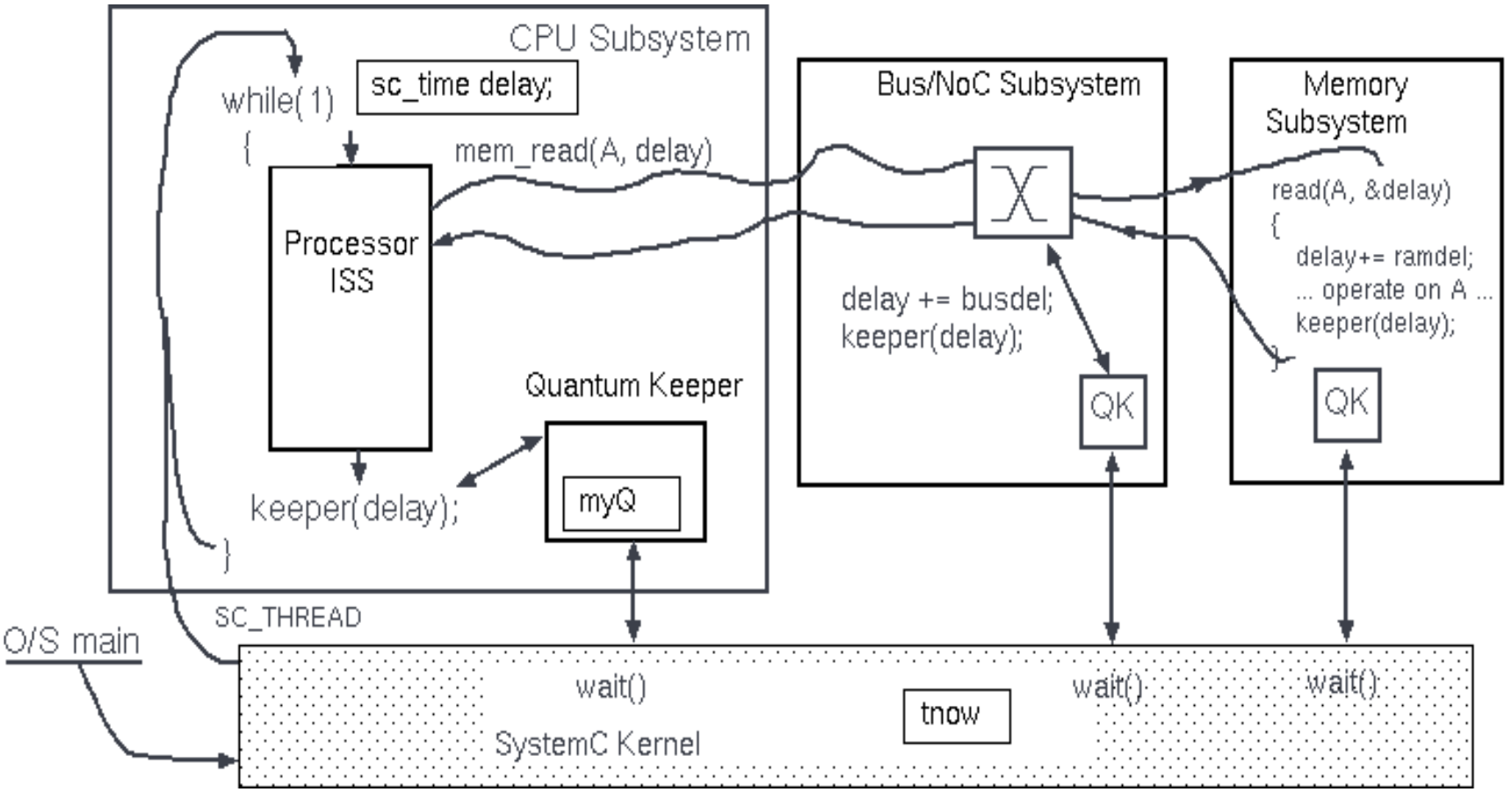
```
b_putbyte(char d, sc_time &delay)
{
  sc_time del(250, SC_NS);
  printf("Byte '%c'\n", d);
  delay += del;
}
```

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.

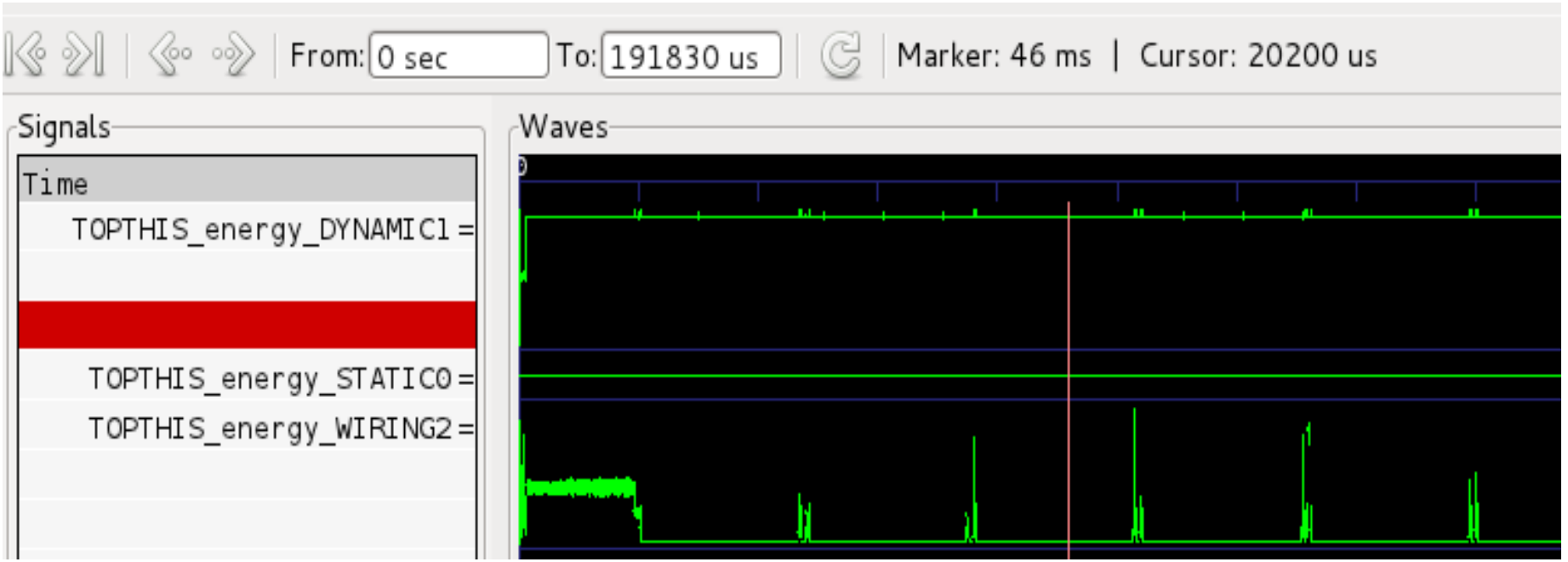
Loosely-timed TLM Modelling: General Structure



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 a placer.

Report Formats (3: VCD)

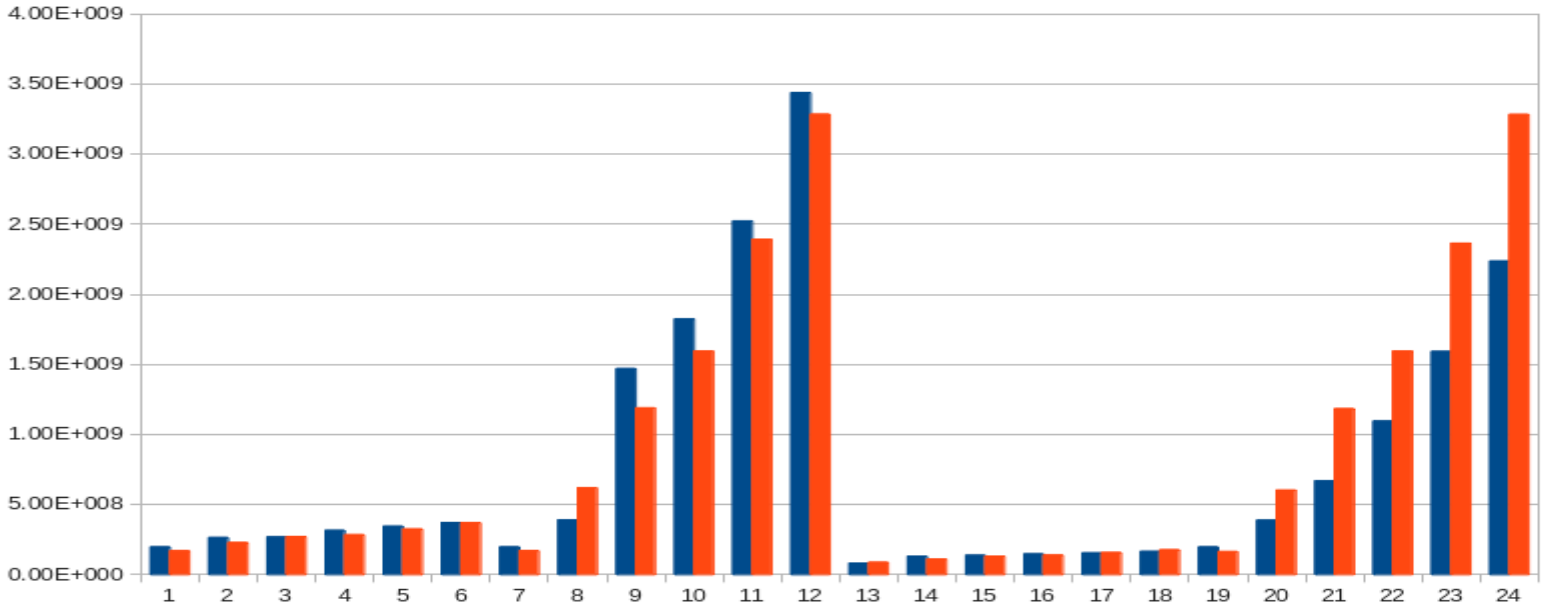


- Each account and their summations can be plotted in various forms
  - 1: Ascii-art table format
  - 2: SYLK or CSV spreadsheet format
  - 3: VCD temporal display (using dirac impulse response or average over interval)
- A physical layout file is also written.

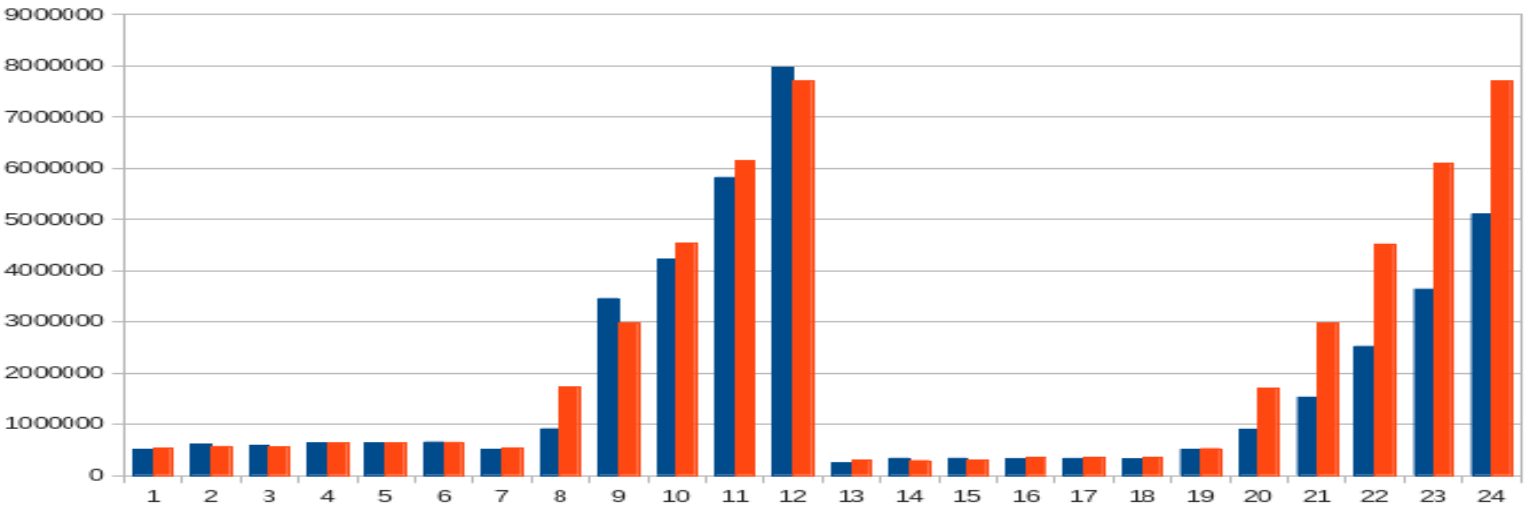
Report Formats (2: Ascii-art text file)

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


Title: privmem-c1n6000-dramsim-withcache-nile-gash-harvard
# Simulation duration: 24826590001096 ps
# Simulation duration: 24826590001096 ps

+--------------------------+------------------------------+------------------------------+------------------------------+
| MODULE  NAME             |             STATIC0 ENERGY   |            DYNAMIC1 ENERGY   |             WIRING2 ENERGY   |
+--------------------------+------------------------------+------------------------------+------------------------------+
Standalone modules:
| Memory 0 (DRAM)          |       0.173879501J   3.49%   |       0.0875462788J   1.76%  |     4.48687512e-07J   0.00%  |
| the_top.uart0            |                0J   0.00%    |         1.644e-06J    0.00%  |         6.7041e-11J   0.00%  |
| the_top.busmux0          |                0J   0.00%    |      1.1905216e-05J   0.00%  |                0J   0.00%    |
| the_top.dram=0           |       0.173879501J   3.49%   |       0.0875462788J   1.76%  |     4.48687512e-07J   0.00%  |
| ...top.coreunit_0.core_0 |         0.2482659J   4.99%   |       0.0044012626J   0.09%  |     1.34648772e-05J   0.00%  |
| ...reunit_0.l1_d_cache_0 |                0J   0.00%    |     0.000594064671J   0.01%  |     6.14810556e-06J   0.00%  |
| ...0.l1_d_cache_0.Data_0 |      0.0333542257J   0.67%   |     0.000107935695J   0.00%  |                0J   0.00%    |
| ...0.l1_d_cache_0.Tags_0 |      0.0317907464J   0.64%   |      4.18042825e-05J   0.00%  |                0J   0.00%    |
| ...0.l1_d_cache_0.Data_1 |      0.0333542257J   0.67%   |     0.000105833853J   0.00%  |                0J   0.00%    |
| ...0.l1_d_cache_0.Tags_1 |      0.0317907464J   0.64%   |      3.37903219e-05J   0.00%  |                0J   0.00%    |
| ...0.l1_d_cache_0.Data_2 |      0.0333542257J   0.67%   |     0.000105435493J   0.00%  |                0J   0.00%    |
| ...0.l1_d_cache_0.Tags_2 |      0.0317907464J   0.64%   |      2.60627187e-05J   0.00%  |                0J   0.00%    |
| ...0.l1_d_cache_0.Data_3 |      0.0333542257J   0.67%   |     0.000108887529J   0.00%  |                0J   0.00%    |
| ...0.l1_d_cache_0.Tags_3 |      0.0317907464J   0.64%   |      1.83743234e-05J   0.00%  |                0J   0.00%    |
```

Measured v Predicted: Runs 19-24 extrapolated from data fitting on 1-18.



Energy



Time

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 som
```

Power Estimation: Project Flow