Chapter 4
TLM POWER3: Power Estimation Methodology for SystemC TLM 2.0

David Greaves and Mehboob Yasin

Abstract We report on a SystemC add-on library which extends every SystemC module with non-functional data regarding power consumption and physical layout and which accumulates and estimates dynamic energy usage. It supports both phase/mode power modelling and energy-per-transaction logging for TLM (transactional-level modelling). Wiring energy is computed by counting bit-level activity within the TLM generic payload. Each leaf component can also register its physical dimensions to facilitate a wire length estimator that traverses the SystemC model hierarchy using either full placement or Rent's rule estimators. It also supports dynamic voltage islands and inter-chip wiring, where each transaction can consume energy according to the current supply voltage of the relevant islands and the nature of the interconnect. We report on basic performance from some SPLASH-2 benchmarks running on a modelled OpenRISC quad-core platform.

4.1 Introduction

With the current major emphasis on power consumption in electronic design it is important to be able to obtain power estimates during the architectural exploration phase. Power consumption is an emergent property arising once hardware and software have been selected. For results to be numerically accurate, a detailed, net-level layout of the design is required in the chosen target technology. This level of detail is inconsistent with rapid prototyping. However, with wiring power becoming

D. Greaves (✉)
Computer Laboratory, University of Cambridge, Cambridge CB3 0FD, UK
e-mail: David.Greaves@cl.cam.ac.uk

M. Yasin
Computer Laboratory, King Faisal University, Al-Ahasa, Saudi Arabia
e-mail: my294@cl.cam.ac.uk

J. Haase (ed.), *Models, Methods, and Tools for Complex Chip Design*, Lecture Notes in Electrical Engineering 265, DOI 10.1007/978-3-319-01418-0_4,

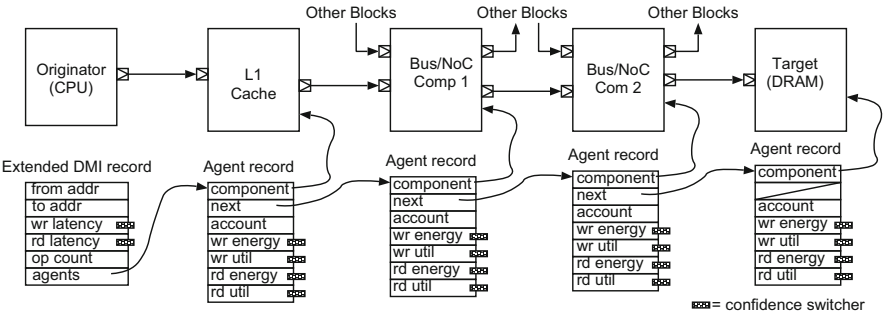

Fig. 4.1 An example extended DMI record and agent list. An initiator may typically have several of these active at once for different targets or addressable regions in a target. The TLM return path is always the same as the forward path and the agent records are incremented for utilisation and energy in (the active phase of) both directions

the dominant contributor in recent generations of VLSI technology, early indications of this aspect are becoming more essential. Indicators that are relatively accurate become useful. Relatively accurate indicators may have unknown linear error factors in the values they report, but they certainly have the correct polarity in their partial derivatives, thereby allowing the designer to tell whether a change is better or worse.

TLM modelling using SystemC permits high-performance models to be created. The greatest performance is facilitated by using the blocking transaction style with loose timing (L/T) and DMI (direct memory interaction). Using blocking transactions, interactions between a CPU and a cache, memory or I/O device are modelled as a simple method invocation with handshaking overheads being modelled simply by the call and return of the relevant subroutine [3]. The loose timing method allows a given initiator to hog the modelling workstation for an extended period of time, called its quantum, and thereby avoid the overhead of context switching needed to keep transactions and bus cycles strictly in the order they would really occur. DMI allows an initiator, such as a CPU, to make backdoor access to the workstation memory used to model the contents of RAMs and DRAMs, thereby avoiding the overheads of modelling caches and busses or NoCs (networks on chip). However, previous modelling systems have become highly inaccurate in terms of reported performance and (especially) power when these advanced modelling features are enabled.

Two previous libraries for SystemC power modelling are TLM POWER2 [6] and PKtool [13]. Our own library is called TLM POWER3 owing to its direct reuse of some infrastructure from TLM POWER2, but ideally one might merge it with PKtool so that the styles and approaches from both previous libraries are concurrently available. Higher-level approaches might also be included. For example, the Sesame approach to estimating power consumption uses an abstract model of execution, based on *computational event signatures* [8]. A similar higher-level approach was presented in [10], but built on SystemC.

The TLM POWER2 library for SystemC associated groups of SystemC modules with a pair of power account records called static and dynamic. The association was maintained either by inheriting a `pw_module` parent as well as the standard `sc_module` inheritance, or by setting a SystemC attribute to point to the appropriate set of accounts.[1] TLM POWER2 used the mode/phase approach to power modelling.

In the mode/phase approach, the consumption of an IP block is determined from its current state. The state of the block is characterized by both its phase and its mode. A mode is a particular DPM (Dynamic Power Management) mode (e.g. on, sleep, off). A phase, basically a functional phase, is characterized by its power and time duration (e.g. wait, read, compute). The available modes and phases are defined in a technology/instance file that is inspected by the constructor for the component. The constructor can nominate a specific file for a specific instance or the `kind()` of the component can be looked up and the details set for all instances of that kind of component. The behavioural model for the block must change mode and phase explicitly using calls such as

```
this->update_power(sc_pwr::PW_MODE_ON,
                   sc_pwr::PW_PHASE_IDLE);
```

Infact, in the mode/phase approach of TLM POWER2, there is no specific support for transactional modelling or loose timing. The TLM calls are unannotated and the SystemC kernel must be advanced for the appropriate period of time while a component is in a given power mode/phase for the correct energy accumulation to be logged.

PKtool is another SystemC library for power modelling, but its basic approach is to count transitions at the net level. Wrappers are provided for all of the common SystemC datatypes used for modelling wires, such as `sc_uint<7>`. When SystemC kernel time advances, the hamming distance of each wrapped type is computed and added to its transition count. The hamming distance is the number of bits that have changed value. For energy modelling, only the zero-to-one transition needs to be considered. A net will consume energy from the supply each time it rises, according to the standard $\frac{1}{2}CV^2$ formula, where C is the net capacitance, which is proportional to its length. (As explained below, we use the same approach for our TLM calls, but we then automatically disable it in favour of performance). However, PKtool library does not help estimate net length, and despite some recent extensions for TLM modelling, it has no support for the TLM generic payload. Directly relating the events in the model to the SystemC kernel timestamp cannot support loosely-timed models which locally run ahead of the kernel.

[1] We use the word *component* to denote an `sc_module` that is so associated.) SystemC augments every `sc_module` (or other entity that inherits `sc_object`) with a key/value space where the values are `void *` pointers.

4.2 Our Approach: TLM POWER3

TLM POWER2 defined physical units for power and energy in the same way as SystemC itself defines physical units for time. All of the standard arithmetic operators are overloaded to have the expected behaviour. For instance, a power multiplied by a time results in an energy. In TLM POWER3, we have added new physical units for voltage, distance and area, along with the appropriately overloaded operators. A component can describe its physical size in its constructor using one of the following TLM_POWER3 calls:

```
// Set actual dimensions of current component
void set_fixed_dimensions(pw_length x, pw_length y);

// Set additional area of current component
void set_excess_area(pw_area a, float max_aspect_ratio=2.5);

// Select chip/voltage island for current component
//  and its children.
void set_chip_name(string chipname, string island);
```

The former sets the actual dimensions of the current component, leading to a warning if this is smaller than the sum of its components. The `excess_area` call describes the additional area of the current component beyond that of the sum of its child components. The component is assumed to be flexible in shape from square up to an oblong of maximum aspect ratio specified. Aspect ratio is, however, ignored by our provided basic estimator that just sums areas within a component and does not attempt to give them co-ordinates within the component. Components can be specified to be placed on different chips or regions of chips but the default is to be on the same chip/region as their parent. This identifies which wiring crosses between chips and hence has different dimensions and technology. It can also be used to exclude logic from the current chip's dimensions, as is useful for example, when a DRAM bank model is instantiated inside the DRAM controller rather than exporting all of the connections (TLM or otherwise) up through the module hierarchy. The same partitioning approach defines dynamic supply voltage islands where voltage changes are applied to all members of a chip/region at once.

As well as supporting an external table of modes and phases for each instance/kind of component we enable the C model to contain explicit statements of power and energy. For instance, the constructor (or PVT callback, mentioned later) for an SRAM of `m_bits` might contain the following, where the first line creates a constant power value and the second logs this power in the static power account of the current component.

```
pw_power leakage = pw_power(82.0 * m_bits, PW_nW);
set_static_power(leakage);
```

Rather than just supporting a fixed pair of power accounts, as in TLM POWER2, our library supports any number of accounts per group of components with the

first three being nominally used for component static power, component dynamic power/energy and wiring dynamic energy. For full flexibility, each account can model both energy and power. Each account has energy as its primary accumulating representation and the power being a standing value that, form time to time, is converted to energy debits. Standing power is converted to energy when the standing power level is changed or at the end of simulation, the energy being the previous standing power level multiplied by the time since the last standing power change. An error is raised if the simulation exits at infinite time with a non-zero standing power level in any account. Energy figures are converted back to average power in some forms of report.

Our library also supports utilisation and transaction logging for visualisation purposes. Although this might seem orthogonal, there are some overlaps. One feature of the PKtool TLM modelling style is that idle power in a component is not accumulated while a transaction is active, and hence details of component utilisation are needed for this style of modelling. By recording utilisation we can apply this correction if desired: it might be very useful to model dynamic power and clock gating. In addition, when our library generates a VCD (Verilog change dump) report, it is convenient to have a graphical illustration of the transactions alongside the energy use bumps.

4.2.1 Extended Generic Payload: Distance + Hamming

Although our library can be used with the standard generic payload, more detail is captured using our extended version called PW_TLM_PAYTYPE. In TLM 2.0, sockets are templated types which default to use the standard generic payload, but we can instead use PW_TLM_PAYTYPE. Rather than explicitly extending the generic payload, we could have used the generic payload's own extensions (and these still work, as used for instance for extended commands such as load-linked/store conditional), but we chose not to for efficiency reasons. Socket definitions and calls now look like this (although CPP macros can tidy this up):

```
// Providing the third template argument to a socket:
tlm_utils::simple_initiator_socket
<mytype, 64, PW_TLM_TYPES> ifetch_socket, data_socket;

// Using the extended payload in the callbacks:
void b_access(PW_TLM_PAYTYPE &trans, sc_time &delay)
```

The extensions in PW_TLM_PAYTYPE assist with the following details:

1. Deciding which fields are active so that only the correct fields have their hamming distance processed for wiring power,
2. Establishing the trajectory of the transaction through the system so that traversed wire length is estimated,

3. Keeping note of the components encountered so that the correct power and utilisation accounts can be incremented under DMI,
4. Measuring the variance of metrics so that automated transition to DMI is enabled.

In a generic example, Fig. 4.1, the originator (CPU) will complete the address field of the payload and, for writes, also the data and byte-enabled fields. Generally, `multi_passthrough` TLM sockets are used in complex system models: these support forwarding the transaction onwards through bus, cache and NoC (network-on-chip) subsystems. The return path is always the reverse of the forward path owing to simple stack unwinding associated with method invocation. So intermediate components forward the payload, perhaps with minor changes (e.g. address space manipulations at bus bridges or VM units) to the target destination. This target will reply with a low-cost acknowledgement for a write and with the data for a read.

Our TLM payload offers an API with three library calls for bus energy modelling. These are `pw_set_origin`, `pw_log_hop` and `pw_terminus`. Currently models invoke these on a payload at the beginning, intermediate steps and end of a its payload trajectory. Rather than manual application, building these invocations into the TLM convenience sockets would be more convenient (sic).

```
void pw_set_origin(sc_module *where,
   uint flags=0,
   bit_transition_tracker *transition_reference=0);
pw_agent pw_log_hop(sc_module *where,
   uint flags=0,
  bit_transition_tracker *transition_reference=0);
void pw_terminus(sc_module *where);
```

The first argument `where` is the `this` pointer for the current component. This is used to track the path through the system.

The second argument is the flags that denote which fields in the payload are active. They can also encode bidirectional data busses and multiplexed address-data busses. When the physical nets of busses are re-used the transition count increases but there are fewer busses (e.g. the high order address bits might be mostly static on a dedicated address bus but are not if the same wires carry multiplexed address and data). Most flags are sticky and apply to subsequent hops that do not change that flag. In particular, if the flags argument is zero for the next hop then nothing has changed and the next hop has the same properties as the previous hop.

The third argument is a bus reference. Every transaction is considered to take place over a bus and a bus is a generic set of wires modelled with a `bit_transition_tracker`. Wires present (i.e. payload fields) that are not used consume no energy, so it is not important to customise the instance of a bus to its use (e.g. the bus from CPU to memory has address and write data whereas the return bus has just read data). The bus reference is needed so as to check which physical nets are transitioning with respect to their previous value.

We could integrate a layout package to estimate wiring lengths in detail. Currently we use the Rentian approach of [4] that provides a simple estimate

of average connection length in a well-placed implementation according to $\alpha.A^{\frac{1}{2}}$ where A is the area of the lowest common parent component to the source and destination of the signal and α is typically 0.3. The capacitance per unit length of nets for on-chip and off-chip wiring is read from a configuration file (we use 0.3 pF/mm).

TLM 2.0 defined a DMI record called `tlm::tlm_dmi` which stores the start and end addresses and access times for a region of memory that can be accessed via a fast backdoor mechanism: the client simply reads from the raw host memory that contains the memory contents. In addition, the target has a callback called `invalidate_direct_mem_ptr` whereby this record can be retracted. However, using this DMI mechanism bypasses the target model and also all intermediate bus components, including caches, so their utilisation and energy accounts cannot be updated directly during DMI accesses. In TLM POWER3, we make the forward trajectory of a TLM DMI-allowed call instantiate a chain of account records that contain the energy and utilisation and account number for each intermediate agent and the target. The energy and utilisation are also updated on the return transit of the thread. (For non-blocking calls, the updates are just made on the significant protocol phases.) At the initiator we augment the DMI record with a count of the number of DMI calls made (scaled by the relative size of the transaction if the payload burst size is varying). Aperiodically (e.g. at end of L/T quantum), and on DMI invalidate, the count field is reset with the appropriate credits being made to the utilisation and energy accounts of each referenced component. The invalidate DMI callback also performs such a flush and frees the agent list. Operations such as store conditionals must not use DMI, so can be used as flush points.

An alternative to building the agent records is to write DMI energy to a 'slush fund' account where it will appear (correctly) in the total for the system/subsystem but (incorrectly) in the slush fund of the originator of such transactions instead of the consuming component (which is ensured by our agent records). We would perhaps use account number 4 in each component for this purpose.

The energy and power figures in a call to the library can either be pre or post supply voltage scaling, where the former are multiplied by the supply voltage squared at the point of logging and the former are not scaled. Given that a component (`sc_module`) inherits our `pw_module_base`, transaction energy logging in a component can be as simple as:

```
m_read_energy_op = pw_energy(5.0 +
     1.5e-5 * m_bits, pw_energy_unit::PW_pJ);
m_read_energy_op *= get_current_vcc_squared();
this->record_energy_use(m_read_energy_op /*, 1*/);
this->record_utilisation(sc_time(1, sc_us), delay);
```

where the first line would typically be in the component's constructor, the second would be in the constructor or in the PVT (process/voltage/temperature) recalculate callback. The third line actually logs the energy and can specify an alternative account to the default of '1'. Multiplying by the supply voltage squared on every logging event is slow, and hence pre-computing this in the PVT method is preferred.

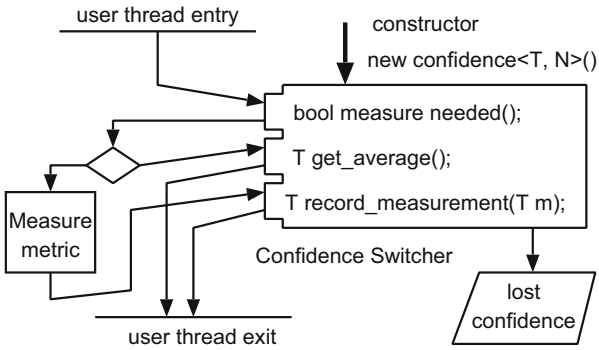

Fig. 4.2 General use pattern for the 'confidence switcher' component that first accumulates and then provides a mean value for a metric based on aperiodic measurements while raising an exception if accuracy is lost

Account one is the default intra-component dynamic energy account. The log of the utilisation itself takes the busy duration and an extra, optional second argument which is the advance over kernel simulation time needed for accurate rendering when loosely-timed. The '`this->`' prefix would either be missed out, but preferable is is to replace it with the agent handle returned from `log_hop` call. This applies the energy and utilisation debits to the current component but also inserts their values in the agent list (if one is being constructed) so that they are accounted when subsequent calls are replaced with DMI.

Using standard TLM 2.0, an initiator will start using DMI when calling `get_direct_mem_ptr` on the initiating socket after a transaction instantiates a valid, local DMI record. Typically the initiator has no knowledge of the accuracy of the latency figures in its DMI record: naively, these will just be the result of the first call (which could be much slower owing to cache misses etc.). We provide and use a '*confidence switcher*' to ensure DMI is employed with fairly accurate values for latency as well as energy and utilisation in an extended DMI record.

The confidence switcher (Fig. 4.2) is a simple library component designed to capture the value of a presumed-stationary statistic using a relatively small number of costly trials. It has internal state and three user methods and is parametrised by an integer N (default is 1,000) that averages a generic statistic over the second N measurements and then reports that average from then onwards while making pseudo-random occasional further measurements (with mean spacing every $1/N$) to check that the mean value has not significantly changed. The first N measurements are not included in the average to avoid start-up transients. This gives a performance boost of approximately N times in the overhead of this measurement. A change by more than 1% and more than $2/N$ is considered significant and this raises an `SC_ERROR` or `SC_WARNING` according to another construction parameter. Confidence switchers are used as much as possible, both in the POWER3 library and by the user models. They can record bit transition density counts, latency times and and power and energy units.

```
#############################################################################
#                    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 |     0.03335422257J  0.67% |    0.000107935695J  0.00% |              0J    0.00% |
| ...0.l1_d_cache_0.Tags_0 |     0.03179074643J  0.64% |     4.18042825e-05J  0.00% |              0J    0.00% |
| ...0.l1_d_cache_0.Data_1 |     0.03335422257J  0.67% |    0.000105833853J  0.00% |              0J    0.00% |
| ...0.l1_d_cache_0.Tags_1 |     0.03179074643J  0.64% |     3.37903219e-05J  0.00% |              0J    0.00% |
| ...0.l1_d_cache_0.Data_2 |     0.03335422257J  0.67% |    0.000105435493J  0.00% |              0J    0.00% |
| ...0.l1_d_cache_0.Tags_2 |     0.03179074643J  0.64% |     2.60627187e-05J  0.00% |              0J    0.00% |
| ...0.l1_d_cache_0.Data_3 |     0.03335422257J  0.67% |    0.000108887529J  0.00% |              0J    0.00% |
| ...0.l1_d_cache_0.Tags_3 |     0.03179074643J  0.64% |     1.83743234e-05J  0.00% |              0J    0.00% |
```

Fig. 4.3 Part of an example textual report file where a large number of separate components have been included. The grand total account is clipped off the *bottom* of this figure

4.2.2 Output Reports

Several kinds of report can be generated with the TLM_POWER3 library. These cover power consumption, utilisation and physical layout.

The library will automatically add up the power and energy used globally, but further detail on individual sub-systems can also be reported by selecting other points in the hierarchy to trace and passing the associated component as an argument to a power trace function. Each item traced generates either a fresh set of accounts that include that item and all of its children, or that item alone, or a fresh set of totals for each component (i.e. for that item alone and recursively for all its children separately).

Energy consumption and average power are primarily reported in a plain text file emitted at the end of simulation (or at other user request dump points). An example is shown in Fig. 4.3. This file can also be written in spreadsheet-friendly SYLK (SYmbolic Link) form. As mentioned, utilisation, energy and transaction activity reports are available in VCD form. The VCD generator can also output in a gnuplot-friendly multi-column file format. Figure 4.4 shows an example VCD plot. The L/T (loosely-timed) approach can upset the normal SystemC VCD report format owing to temporal decoupling (events are logged in their actual simulation order rather than their nominal correct order) but our VCD writer corrects this by writing the events to a circular RAM buffer whose temporal extent is greater than the L/T quantum. This also enables sensible energy plots to be made: energy events would be like Dirac pulses if rendered directly and cumulative energy plots are not especially informative, but our VCD writer implements a single-pole low-pass filter for the

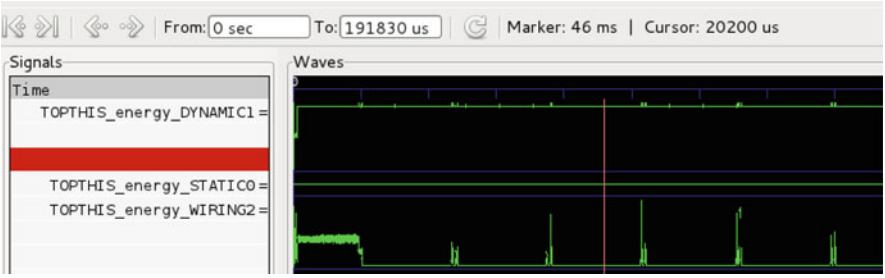

Fig. 4.4 An example VCD trace showing static, dynamic and wiring power consumption for a RADIX sort. The cores operate mainly from cache but exchange data between rounds

energy events so that they appear like exponentially-decaying pulses. Alternatively, in another mode, it reports the flat average power given by the last energy quantum divided by the time to the next-logged quantum on that account. Physical layout is currently printed as a text file which just reports which components are inside which others and the resulting area for each component. A graphical plot in `.svg` form is being implemented.

4.3 Performance

We examined the performance of the first two testbench programs in the Splash-2 suite [14]. These are a radix sort and a L/U matrix decomposition that can run on 1–16 cores. We compiled the Splash-2 programs to run bare metal supported by the standard linux `libc` and our own implementation of pthreads and a Simics/ANL (parmacs) shim layer [5]. Our testbench uses four OpenRISC processor cores [11] in verilated or fast ISS forms wrapped to use TLM 2.0 blocking calls served by an un-cached instance of DRAMSIM2 [9]. The cores log 250 pJ per instruction and run at 200 MHz unless paused waiting for other cores (50 mW core power). Each core has separate I and D L1 caches that included 17 RAMs each (tag and data for 8-way set associative and a write buffer). Each Core, Cache and each of the other components shown in Fig. 4.5 is a separately-annotated SystemC module that also inherits a TLM_POWER3 base and communication between them is completely with blocking TLM 2.0 calls. There are 16 SystemC modules in 3 levels of hierarchy. The individual RAMs had dimensions and power consumption computed according to the equations in Table 4.1 which were formed from our own regression of 45 nm CACTI runs [12].

Table 4.2 shows that simply taking the four-core ISS and putting it inside SystemC TLM degrades the performance by about ten-to-one owing to SystemC kernel overhead (gprof reveals major costs (more than 20% of execution time) are

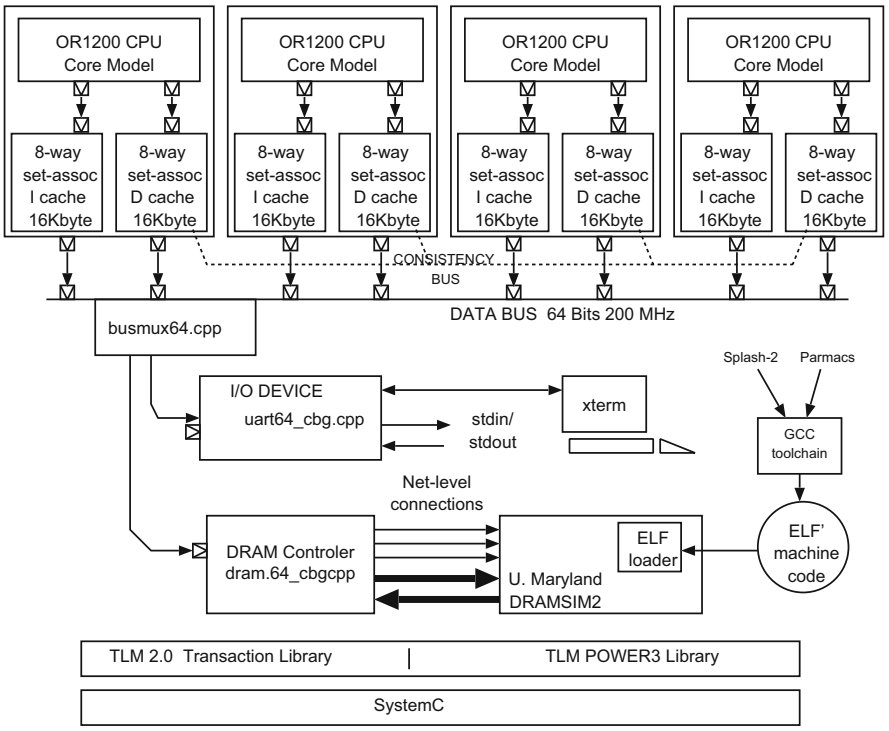

Fig. 4.5 Reference hardware platform for experiments (Quad-core OPENrisc with U. Maryland DRAM simulator)

Table 4.1 Interpolated CACTI 45 nm RAM parameter equations

Property	Model equation	Unit
Area	13,359 + 40*bits	squm
Read energy	$5 + 1.5E - 5$	pJ per bit
Write energy	$10 + 3.0E - 5$	pJ per bit
Access time	$0.21 + 3.E - 4 \times (\text{bits})^{0.5}$	ns
Leakage power	82	nW per bit

Table 4.2 Simulation performance using GCC 4.43 on Intel x86_64 3 GHz/6,000 BogoMips, 8 GB RAM (no paging) SystemC-2.2.0

Figure 4.6 name	Configuration	Instructions/s	Ratio
Not plotted	Fast ISS – No SystemC	11×10^6	1.0
Unannotated model	L/T = min, POWER3 = off	1×10^6	0.1
Unannotated model	L/T = max, POWER3 = off	4×10^6	0.4
DMI enabled	L/T = max, POWER3 = off	7×10^6	0.7
POWER3	L/T = max, POWER3 = on	0.5×10^6	0.05
POWER3 + HOPS	L/T = max, POWER3 = on + hops	0.3×10^6	0.03
POWER3 + HOPS+XITIONS	L/T = max, POWER3 = on + hops + hamming	0.2×10^6	0.02

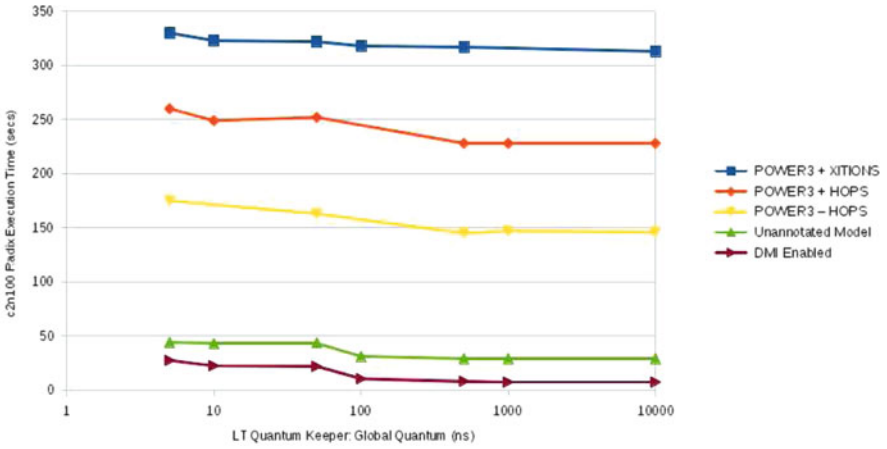

Fig. 4.6 Relative simulation performance of approximately-timed (*left-hand side*) and loose-timed (*right-hand side*) TLM Model (2 cores, running SPLASH-2 Radix Sort n = 100) with various configurations

in `sc_core::sc_simcontext::crunch(bool)` and `b_transport`. This degradation occurs with and without the inclusion of caches but the performance of the modelled system then changes as expected (i.e. program completes much faster with caches). The next lines in Table 4.2 are taken from Fig. 4.6 which plots the performance with and without DMI with respect to the L/T quantum keeper value. Using DMI and a maximal L/T quantum, so that the SystemC kernel is only entered during bus and mutex contention, restores some of the performance.

The effect of compiling with our power library with various configurations is reported at the bottom three lines of the table (and in further plots). It gives roughly a factor of two slow down and the logging of hops does not make it much worse (penultimate line). Implementing hamming distance computations under control of confidence switchers where with $N = 1{,}000$ causes a further rough factor of 2 performance degradation. Performance can perhaps be improved upon, but it is not overly bad. Interestingly, the degradation was much worse in an early version where the island voltage was squared at every use rather than recomputing the transaction energies just on each PVT change.

Figure 4.7 shows the measured power consumption of the processor (excluding DRAM) on a real Linux workstation as three identical runs of the RADIX benchmark were executed, the third one using only one CPU core. A 0.05 Ω resistor was put in series with the 12 V supply to the processor and its voltage drop and output voltage were logged at 60 Hz to record the energy consumption. Figure 4.8 shows the total power plot when the same C program was run on the SystemC model. Some differences in general shape are obvious and need investigation.

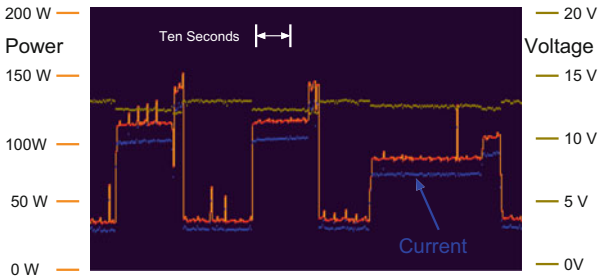

Fig. 4.7 Splash RADIX benchmark: probed processor power consumption. Two runs using two cores followed by one run on a single core. The end region of each run is the checking phase. Spikes are other unix processes on the dual-core workstation (Intel Pentium D 3 GHz)

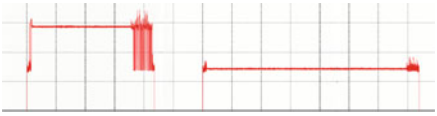

Fig. 4.8 Splash RADIX benchmark: TLM POWER3 total power consumption: we see one run using two cores followed by one run using a single core. No unix operating system was present

4.4 Accuracy

To explore the general accuracy of our library we used three simple test programs to generate calibration data and used this data to predict results for a fourth program. Each of the four programs could be run with one to six threads. We measured the CPU energy use and execution time on a 2.4 GHz AMD Phenom X6 1045T six-core chip, as plotted in Figs. 4.9 and 4.10 respectively. This chip has 64 KB I + D caches per core as well as a dedicated 512 MB L2 cache per core and a shared 6 MB L3 cache. The tests could all run within the L3 cache so DRAM power did not need to be included. The test programs were respectively a memory-bound program with disjoint regions that each fit within the dedicated caches of a core, a memory-bound program with a moderate amount of inter-core churn and a CPU-bound program. Using a multivariate regression spreadsheet within Libre Office the coefficients in Table 4.3 were determined. These were then used to calibrate the SystemC models using the POWER3 library to predict the power consumption and performance of other programs. Such a program was the SPLASH-2 RADIX benchmark, run with between one and six cores and plotted as the final six results.

In each test the total amount of work was increased linearly with the number of cores. The energy used can be seen to grow roughly linearly as well but the execution time only grows when the cores contend for cache lines. The final six results show good agreement between measured and predicted values (within 30%) which is acceptable for high-level architectural exploration. We cannot expect perfect prediction since the programs were compiled for OpenRISC during simulation and

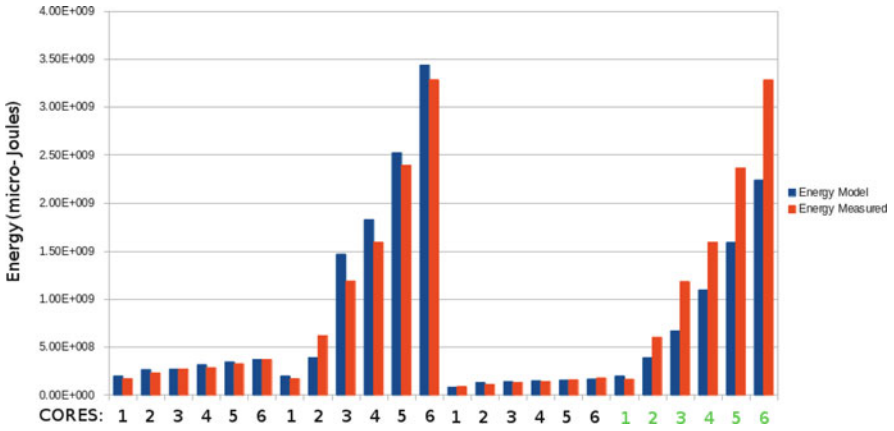

Fig. 4.9 Measured and modelled energy consumption for four tests each with one to six cores where coefficients from the first three tests were used to predict energy in the fourth

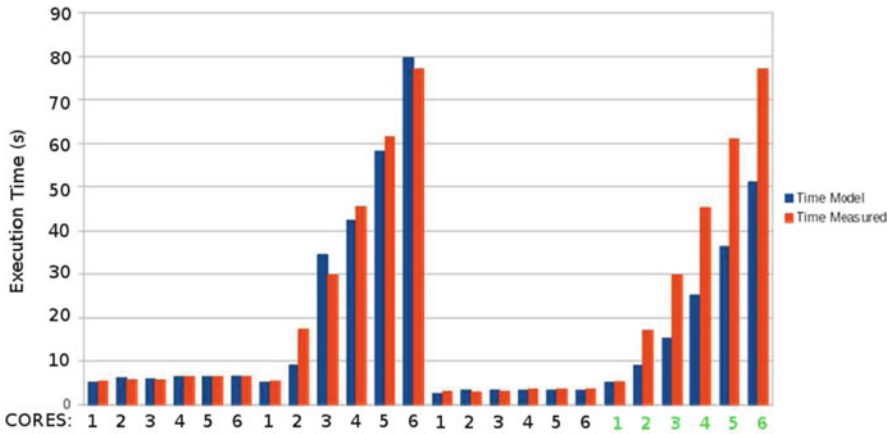

Fig. 4.10 Measured and modelled execution time for four tests each with one to six cores where coefficients from the first three tests were used to predict execution time for the fourth

Table 4.3 Energy debits obtained from curve fitting between simulation and measurement over 24 runs with 1–6 cores in use: CPU & Caches only (DRAM excluded)

Operation	Energy
Instruction	1 nJ
L1 + L2 I cache miss	50 nJ
L1 + L2 D cache miss	15 uJ
L2 cache snoop read	4 mJ
L2 cache consistency evict	7 mJ

for x86-64 during measurement. We have recently implemented a more-detailed model of AMD's hyper-transport system and Hammer cache protocols and will report more comprehensively in another paper.

4.5 Conclusion

Our framework provides an easy-to-use power estimation add on to SystemC TLM modelling. The use of the confidence switcher to dynamically disable detailed modelling is novel. The user may easily alter the system structure in radical ways, changing cache size, bus layout and so on. Standard ELF binaries can be easily generated with GCC/binutils tool chain. We also have a MIPS64 SMP system based around the same components. Because wiring power is becoming a dominant aspect it is important to include it in rapid exploration tools.

The benefit of rapidly exploring design options using SystemC was advocated in [1], but having performance predictions without power predictions is no longer acceptable. A fairly-detailed TLM model with power annotation was constructed by [2] for a PowerPC-based SoC. The activity for individual test transactions was extracted from VCD files and entered into a database. This approach can be applied in our framework to generate the individual transaction energies. Power estimation is also being performed for AMS (analog and mixed signal) subsystems within the SystemC framework [7].

We plan to further refine our API and library and release it for download. Including the `log_hop` operations inside the convenience sockets would be sensible. Also, further support for power islands might be needed, but currently we can use our chip number concept with zero volt supply setting to disable static power in regions. Further work will be to integrate back-annotation flows from real layouts and compare these with the Rentian approach. We would also like to extract net-level activity from the Verilated models to gain additional insight and confidence.

Acknowledgements We thank Matthieu Moy for providing the TLM POWER2 base platform.

References

1. Benini, L., Bertozzi, D., Bogliolo, A., Menichelli, F., Olivieri, M.: Mparm: exploring the multi-processor SoC design space with SystemC. J. VLSI Signal Process. Syst. **41**, 169–182 (2005)
2. Dhanwada, N.: A power estimation methodology for SystemC transaction level models. In: Proceedings of CODES-ISSS, Jersey City, pp. 142–147 (2005)
3. Ghenassia, F.: Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems. Springer, Secaucus (2006)
4. Greenfield, D., Moore, S.W.: Fractal communication in software data dependency graphs. In: SPAA'08: Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures, Munich, pp. 116–118. ACM, New York (2008)

5. Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., Werner, B.: Simics: a full system simulation platform. Computer **35**(2), 50–58 (2002)
6. Moy, M.: Mini power-aware TLM-platform. http://www-verimag.imag.fr/~moy/?Mini-Power-Aware-TLM-Platform (2010)
7. Pêcheux, F., El Abidine, K.Z., Greiner, A.: Early power estimation in heterogeneous designs using SoCLib and SystemC-AMS. In: Proceedings of the 20th International Conference on Integrated Circuit and System Design: Power and Timing Modeling, Optimization and Simulation, PATMOS'10, Grenoble, pp. 252–252. Springer, Berlin/Heidelberg (2011)
8. Piscitelli, R., Pimentel, A.D.: A signature-based power model for MPSoC on FPGA. VLSI Des. **2012**, 6:6–6:6 (2012)
9. Rosenfeld, P., Cooper-Balis, E., Jacob, B.: Dramsim2: a cycle accurate memory system simulator. Comput. Archit. Lett. **10**(1), 16–19 (2011)
10. Streubuhr, M., Rosales, R., Hasholzner, R., Haubelt, C., Teich, J.: ESL power and performance estimation for heterogeneous mpsocs using SystemC. In: Specification and Design Languages (FDL), 2011 Forum on, Oldenburg, pp. 1–8 (2011)
11. Tandon, J.: The openrisc processor: open hardware and linux. Linux J. **2011**(212) (2011)
12. Thoziyoor, S., Ahn, J.H., Monchiero, M., Brockman, J.B., Jouppi, N.P.: A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In: Proceedings of the 35th Annual International Symposium on Computer Architecture, ISCA'08, Beijing, pp. 51–62. IEEE Computer Society, Washington (2008)
13. Vece, G.B., Conti, M.: Power estimation in embedded systems within a SystemC-based design context: the PKtool environment. In: Seventh Workshop on Intelligent Solutions in Embedded Systems, Ancona, pp. 179–184 (2009)
14. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The splash-2 programs: characterization and methodological considerations. SIGARCH Comput. Archit. News **23**, 24–36 (1995)