KOCL: Power Self-awareness for Arbitrary FPGA-SoC-accelerated OpenCL Applications

James J. Davis, Joshua M. Levine, Edward A. Stott, Eddie Hung, *Member, IEEE,*
Peter Y. K. Cheung, and George. A. Constantinides, *Senior Member, IEEE*

***Abstract*—Given the need for developers to rapidly produce complex, high-performance and energy-efficient hardware systems, methods facilitating their intelligent runtime management are of ever-increasing importance. For energy optimisation, such control decisions require knowledge of power usage at subsystem granularity. This information must be made accessible to developers accustomed to creating systems from high-level descriptions, including those written in OpenCL. This article introduces KOCL: a tool allowing OpenCL developers targetting FPGA-SoC devices to query live kernel-level power consumption using function calls embedded in their host code. To maximise accessibility, KOCL is incredibly easy to use; crucially, it requires zero exposure to hardware.**

***Index Terms*—power estimation, OpenCL, FPGA-SoC, runtime measurement, online modelling.**

I. Introduction and Motivation

Three major factors motivated us to develop KOCL, short for KAPow for OpenCL:

1) the growing capabilities and popularity of high-level synthesis (HLS) tools for logic design,
2) the desire to monitor subsystem power consumption without its direct measurement and
3) the benefits yielded through measurement and modelling at runtime.

Systems-on-chip (SoCs) consisting of multi-core CPUs coupled with field-programmable gate arrays (FPGAs) are now commonplace. Their cost for low- to medium-volume applications makes them attractive for implementing systems featuring custom logic components. OpenCL is a software framework that enables developers to write applications targetting a range of heterogeneous platforms. In the context of FPGAs, it can be viewed as a means of specifying hardware systems at a high level of abstraction. *Kernel* functions, written in OpenCL's subset of C, are intended for execution on computing devices that offer higher degrees of parallelism than general-purpose processors, including FPGAs, in order to increase performance and/or efficiency. When targetted to FPGA-SoCs, kernel code is compiled offline into tailor-made hardware accelerators that reside on the FPGA. Work is then dispatched to these by the accompanying *host* code, which runs in software on the embedded CPU. FPGA-implemented applications are able to deliver substantial energy savings over their CPU and GPU equivalents, even when developed in OpenCL [1].

J. J. Davis, E. A. Stott, P. Y. K. Cheung, and G. A. Constantinides are with the Department of Electrical and Electronic Engineering, Imperial College London, London, SW7 2BT, UK (e-mail: james.davis06@imperial.ac.uk).
J. M. Levine is with Intel, London, UK.
E. Hung is with Invionics, Vancouver, BC, Canada.

Understanding where power is being consumed within a system is of paramount importance, a fact becoming increasingly clear as the number and complexity of subsystems within individual SoCs grows [2]. Optimising the execution of multiple tasks with differing priorities requires an understanding of the underlying hardware's power behaviour. Subsystems may exist in SoCs which, due to their design, variation at manufacture [3] or degradation over time [4], exhibit differing characteristics. Reliance on worst-case operating assumptions generally results in suboptimal performance and energy efficiency, but knowledge of runtime behaviour can enable power-aware adaption. Manufacturing SoCs with the many power islands necessary to permit direct measurement of subsystem power is usually impractical, particularly for FPGAs. This implies that a mechanism for relating a proxy—circuit switching activity—to power is needed instead.

Figure 1 demonstrates the benefit of using an online power model. For this experiment, two models were fed with the same switching activity measurements in order to estimate power consumption. The first was *offline*, trained before use, while the second was *online*, and so trained during use; our implementation of the latter is described in Section III-C. Voltage was stepped down over time after an initial training period of 1000 experimental iterations. Offline models are rarely trained to account for all possible sources of changes in power behaviour, including voltage, temperature, frequency, variation, degradation, input data patterns and noise. Because of this, they cannot always compensate for their effects, as reflected in Figure 1. The data shown was obtained from the system described in Section VII.

KOCL is the first tool capable of generating hardware systems from OpenCL descriptions that are able to report their own kernel-level power consumptions. The tool is general-purpose—it can be used with any OpenCL design that targets an Altera FPGA—and accessible thanks to its ease of application. While the principles of KOCL are generic, our implementation is Altera-specific; it relies upon that vendor's compilation tools and features optimisations for their FPGAs. Despite this, KOCL could be ported to alternative tools and devices, such as Xilinx's, if sufficient engineering effort were invested.

In the context of this article, the terms *subsystem*, *kernel*, *module* and *accelerator* are equivalent and refer to the hardware blocks that constitute a complete SoC.

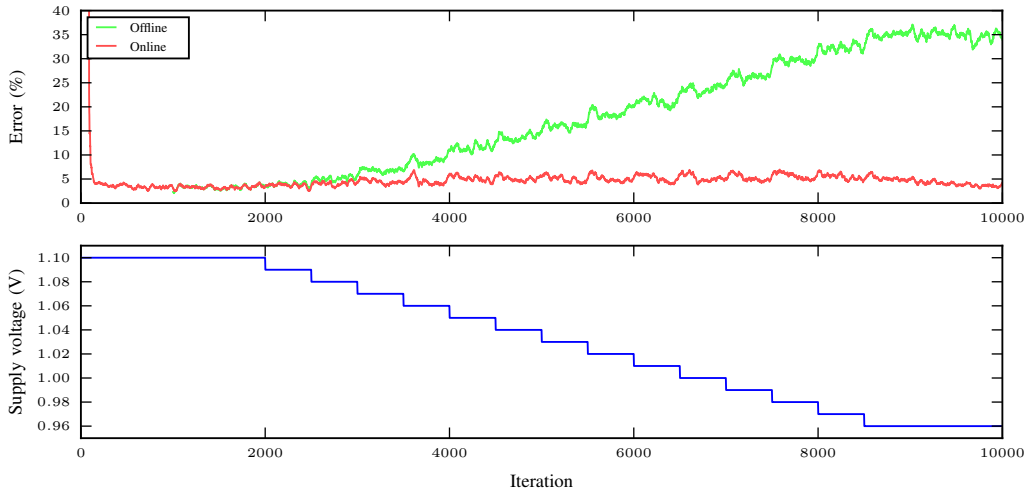

Fig. 1. Online vs offline modelling errors calculated from estimated and directly measured power consumptions under voltage scaling [5]. As the voltage is lowered to levels not seen during the offline model's training, its accuracy deteriorates while the online model compensates.

II. Background and Related Work

Power consumption can be separated into two constituents: dynamic and static. Dynamic power arises from the charging and discharging of circuit nets (signals) during switching, while static power is a property of the process technology and operating conditions. Both the dynamic and static components can vary during operation and, ideally, power models should account for this. Since switching activity influences dynamic power alone, the two components can be distinguished by measuring both activity and total power.

Offline power estimation tools are widely used to determine whether designs will meet their specifications and to inform packing, thermal management and power supply capacity decisions. Early implementations of these tools estimated power consumption by simulating circuits with typical test vectors. As design complexity increased, this became infeasible, causing power estimation tools to move towards probabilistic techniques [6] [7] instead. Further increases in complexity have driven the development of high-level power estimation models for modular systems [8].

System-wide dynamic power estimation achieved through the use of activity counters was proposed by Najem *et al.* [9]. Activity on automatically selected signals was used at runtime to estimate power. The use of an offline model led, in part, to errors of 15%. The indirect measurement of power consumption using ring oscillators woven through application circuitry has also been proposed [10]. However, since system components invariably merge during physical placement, such a technique is unlikely to be suitable for modular power estimation.

Recent works have explored the use of dynamic voltage and frequency scaling (DVFS) to save power in heterogeneous embedded systems. An infrastructure has been proposed for fine-grained DVFS of FPGA-implemented SoCs [11]. DVFS was combined with task mapping by Wu *et al.*, with tasks optimally executed on CPU, GPU or FPGA cores dependent on a realtime power budget [12]. The former requires infrastructural conformance, however, while both are subject to the limitation that FPGAs only have single user-accessible power islands. Information gleaned through the use of KOCL could facilitate adaptive strategies including DVFS, task mapping and power gating *at a subsystem level*.

III. KAPow: Module-level Power Estimation

KOCL relies upon our generic power estimation methodology, KAPow [5], to apportion power consumption within an arbitrary FPGA design to its constituent modules. It does this by combining realtime measurements of module-level switching activity with system-level power. These feed an adaptive, online model that produces module-level power breakdowns while the system operates. By using online modelling, KAPow forgoes distinct training periods; its models are built and continuously updated with real data at runtime. This section forms a 'crash course' on the fundamentals of KAPow.

A. Signal Selection

KAPow uses Altera's power analysis tool, PowerPlay, not to predict power consumption but rather to produce estimated switching activities for each of the modules' signals. Probabilistic activity estimation is used since it is fast, general-purpose and does not require test vectors. Signals with high predicted switching activity are likely to be indicative of high power consumption. From the list of signals outputted for each module, the N—a tunable parameter explored experimentally in Section VII-A—most frequently switching are selected for online monitoring.

B. Instrumentation

Figure 2 shows the high-speed and area-efficient instrumentation KAPow uses for counting rising edges on each monitored signal. Central to each is a W-bit (a further parameter analysed in Section VII-A) linear-feedback shift register (LFSR)-based counter. These are smaller and faster than their

arithmetic equivalents for FPGA implementation. Conversions from LFSR states to arithmetic values are performed in software. The instruments within a module are connected together to form a single-bit scan chain, enabling activity counts to be read out after a measurement period has elapsed.

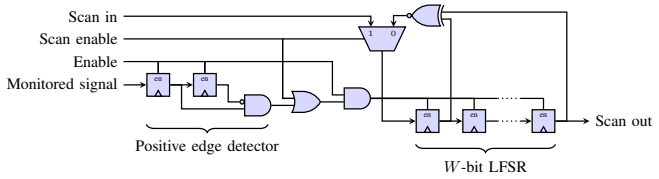

Fig. 2. KAPow activity counter [5]. When Enable $= 1$ and Scan enable $= 0$, the instrument behaves as an LFSR counter, advancing one state per positive transition on the monitored signal. When Enable $= 1$ and Scan enable $= 1$, however, it forms part of a scan chain, shifting its contents to the right once per clock cycle.

A first-in first-out (FIFO) buffer, instantiated as part of a template within the target module as shown in Figure 3, is used to enable activity count read-back. An activity measurement is triggered by loading the timer with the required period. All instruments count for this time, after which the scan chain's contents can be written into the FIFO. Registers are able to be queried in order to ascertain the module's instrumentation parameters, N and W.

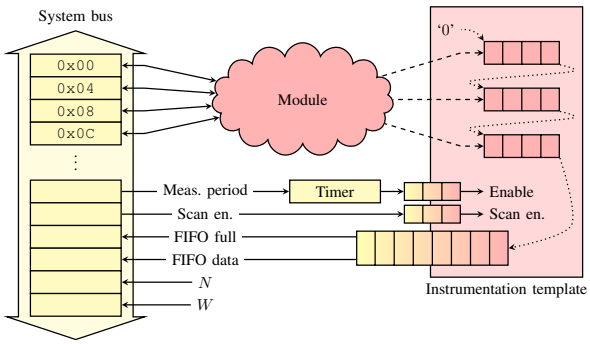

Fig. 3. KAPow-instrumented module [5], with colours indicating the division of bus (yellow) and module (red) clock domains. Counters are shown on the right, with a dotted line representing the scan chain; its head is grounded to reset the counters during read-back. Six registers are used per module for instrumentation control and data acquisition.

C. System Identification

KAPow assumes a linear relationship between signal activities $\boldsymbol{a}$ and measured system power y. Figure 4 illustrates the system identification methodology used. A black box representing the unknown behaviour of the system and its model are fed by $\boldsymbol{a}$. Vector $\hat{\boldsymbol{x}}$ is an estimate of $\boldsymbol{x}$, the 'true' coefficients of the system. y and estimated system power $\hat{y}$ form an error, e, and recursive least squares (RLS) filtering is used to tune $\hat{\boldsymbol{x}}$ in order to drive e towards zero.

Activity and coefficient vectors $\boldsymbol{a}$ and $\hat{\boldsymbol{x}}$ are composed of groups of elements in module order. Power $\hat{y}_m$ can therefore be estimated for each module m by partitioning these vectors. Two additional terms, a_s and $\hat{x}_\mathrm{s}$, are added to $\boldsymbol{a}$ and $\hat{\boldsymbol{x}}$ to allow the model to estimate system-wide static power. 'Leftover' power that cannot be attributed to any of the modules via switching activity is included in the static component.

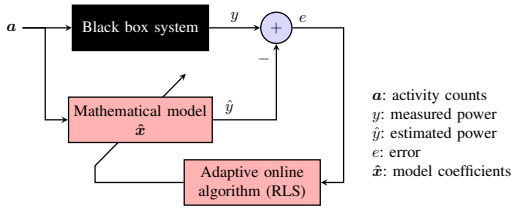

Fig. 4. System identification arrangement used by KAPow [5].

The model is a system of $MN + 1$ linear equations, where M is the number of modules, N is the number of activity counters per module and the $+1$ is representative of the static power component. As such, a single solution for $\hat{\boldsymbol{x}}$ can only be settled upon once $MN + 1$ updates to its elements have occurred. Consequently, power estimates produced before this time are unmeaningful.

IV. Usage

KOCL was designed to be user-friendly from the outset. Users are only exposed to a minimalist API; no hardware expertise is required in order to make full use of KOCL.

A. Kernel Compilation

Suppose a design with kernel source file `my_kernels.cl` were to be compiled for acceleration on an Altera Cyclone V SoC device. Ordinarily, when using the same vendor's OpenCL kernel-to-hardware tool, the Altera Offline Compiler (AOC), the command to be executed would be

```
aoc my_kernels.cl --board c5soc
```

Barring any errors, the output from this process is a configuration file (bitstream) ready to program the FPGA. All steps required to transform high-level kernel source code to a complete hardware system are wrapped within this single call.

To produce a power-instrumented system from the same kernel source, one would instead call

```
koc my_kernels.cl --board c5soc
```

where `koc` is the KOCL Offline Compiler. KOC serves as a drop-in replacement for AOC: any source AOC can process can also, assuming enough area headroom exists for the instrumentation to be added, be handled by KOC. No kernel source modifications are necessary. Additionally, as far as the Altera OpenCL runtime is concerned, systems produced by AOC and KOC are indistinguishable. Their kernel hardware functions identically and the instrumentation added by KOC is visible only to its controlling software.

Three additional, optional arguments exist to tailor KOC's output if required. The `--kernels` argument allows the user to specify a subset of kernels—by name—to be instrumented; the remainder will be left untouched. By default, all kernels are instrumented. `--kapow_n` and `--kapow_w` allow the

TABLE I
KOCL API FUNCTIONS DEFINED IN KOCL.H HEADER FILE

Function name	Parameters	Return type
KOCL_init()	float update_period	KOCL_t*
KOCL_get()	KOCL_t* KOCL, char* kernel_name	float
KOCL_built()	KOCL_t* KOCL	int
KOCL_del()	KOCL_t* KOCL	–

default values of 8 and 9 for KAPow parameters N and W, respectively, explained in Section III and justified empirically in Section VII-C, to be overridden.

B. Host Code Incorporation

Since KOC produces functionally identical hardware systems to AOC, they can be used without any modifications to their accompanying host code. Exposure of KOCL's functionality is made through a minimalist API containing just four functions, detailed in Table I. These are made visible to an OpenCL application through inclusion of the `KOCL.h` header file.

Constructor `KOCL_init()` starts KOCL's software—detailed in Section VI—whose primary function is to build and thereafter continuously update an online power model. The argument to this function, `update_period`, accepts a number, in seconds, used to set the frequency at which model updates occur. The period chosen will impact both the model's ability to compensate for fast-changing effects and the amount of time spent computing updated power estimates. Once initialised, `KOCL_get()` is the interface through which power estimates are obtained. By providing the KOCL data structure and the name of an instrumented kernel present in the design, a power consumption, in milliwatts, is returned. `KOCL_built()` returns true/false depending on whether or not the KAPow model is currently in its 'building' phase, mentioned in Section III-C. Power estimates returned by KOCL are only valid once this function returns `1`. Finally, `KOCL_del()` is KOCL's destructor.

V. Hardware Flow

Figure 5 presents a visualisation of the complete, fully automated KOC tool flow. KOC first calls AOC to assemble a system in QSys, Altera's system integration tool. This facilitates host communication with kernel accelerators, which are described in Verilog produced from the supplied OpenCL source via HLS. By using its hidden `-s` option, AOC terminates immediately after system generation.

KOC parses the Verilog, splitting it by kernel and creating a separate Quartus II, Altera's compilation tool chain, project for each. If the `--kernels` argument is used, only those kernels named are processed. The M kernels to be instrumented are then compiled, with a netlist extracted and signal activity file produced, as described in Section III-A, for each. The netlists describe the kernels' use of and connectivity between physical resources on the FPGA. Each is augmented with `--kapow_n` activity monitors, described in Section III-B, each `--kapow_w` bits wide, by a KAPow parser. The resultant, instrumented netlists are then substituted for the original Verilog.

KOC modifies the top-level QSys project, instantiating M KAPow controllers—one per instrumented kernel—and connecting them to their respective netlists through the kernel wrapper that contains them. Each controller is parameterised with an MD5 hash of the associated kernel's name. This allows the software, described in Section VI, to differentiate them at runtime. The controllers are then connected to the lower-level interface system via a bridge, allowing host communication.

At this point, AOC is called for a second time. Rather than generating a new system, however, it behaves as though it had generated the instrumented system itself, picking up from where it left off. After compilation, KOC's final task is to calculate and store the optimal measurement period, mentioned in Section III-B, for the KAPow controllers. The generated hardware's maximum operating frequency (f_{max}) is ascertained from the compilation report. Tiny ROMs in each controller are initialised with the measurement period calculated to maximise the counters' dynamic range without causing overflow. KOC's output is a bitstream packaged in the format used by Altera's OpenCL runtime.

VI. Supporting Software

KOCL's software is partitioned into three threads: `KOCL_iface`, `KOCL_pool` and `KOCL_model`. Of these, `KOCL_iface` is the master, spawning the remaining two when it is itself initialised by a `KOCL_init()` call. `KOCL_model` performs the majority of the work, interfacing with the hardware—the KAPow controllers and system power regulator—in order to update its online model. `KOCL_pool` serves to allow asynchronous operation of the other two threads. It accepts up-to-date power breakdowns from `KOCL_model` when they are available and provides the most recent kernel power estimates to `KOCL_iface` when requested via `KOCL_get()` calls.

On start-up, `KOCL_model` finds and reads the packaged bitstream produced by KOC to establish the names of the kernels present in the design. It then searches the hardware for KAPow controllers. This is facilitated by the first register in each controller's address space returning `0x7f323e2e`—the MD5 hash of 'KAPow'—when queried. `KOCL_model` is able to establish which kernel each controller it finds belongs to since, as stated in Section V, each is parameterised with the hash of the respective kernel name. The hardware also returns KAPow parameters N and W for each controller, thereby providing the software with all the information it needs without any user input.

Once this 'discovery' phase is complete, an RLS model—described in Section III-C—is constructed. Henceforth, activity and system power measurements are fetched from the hardware and used to update the model every `update_interval` seconds. Calls to `KOCL_get()` result in the most recently calculated power estimate for the given kernel being returned. Users can also pass `static` as the `kernel_name` argument to the same function to query

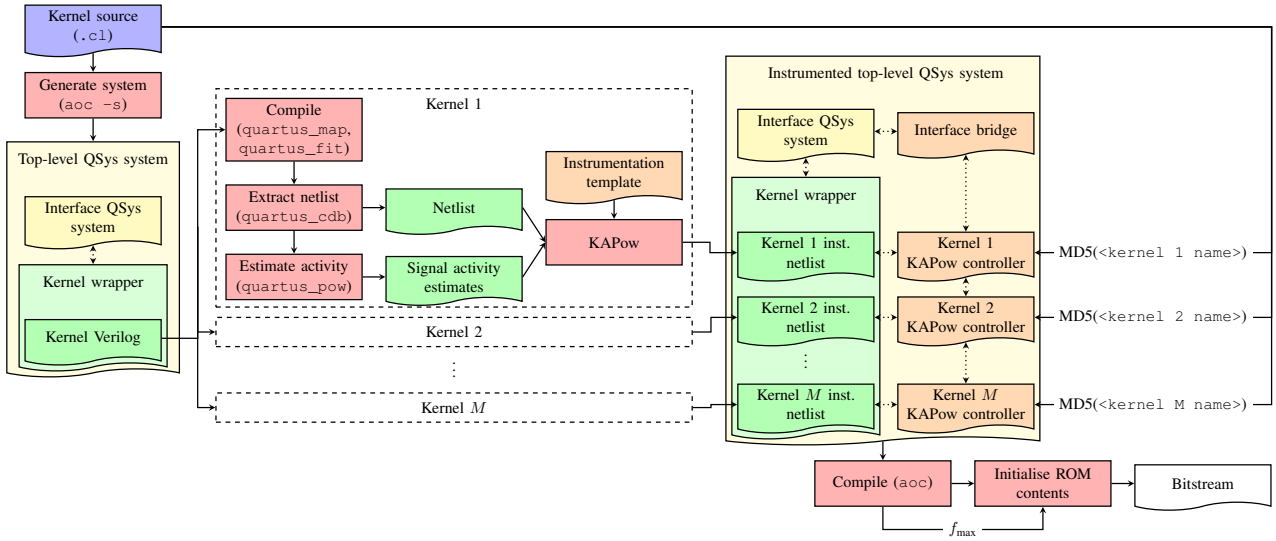

Fig. 5. KOC tool flow. KOC manipulates both Verilog and QSys-level intermediary files generated by AOC, using AOC once more to compile the complete system once those changes have been made.

the current static power. The software keeps track of the number of model updates it has performed, allowing calls to `KOCL_built()` to return accurate values.

VII. Accuracy and Overheads

In this section, experimental results are presented to highlight KAPow's—and, by extension, KOCL's—high estimation accuracy and low overheads. Further data can be found in our prior work [5].

The following results were obtained using a multi-module system comprised of seven FIR filter modules, described in Verilog, assembled in QSys and instrumented as per Sections III-A and III-B. All modules had identical latency and throughput, operating with 5×5 12-bit fixed-point convolution matrices on 240×160 8-bits-per-pixel greyscale images. Each was forced, however, to map its multipliers to different physical resources in order to exhibit differing power characteristics.

A Cyclone V SX FPGA-SoC development board was used for experimentation, with designs compiled using Quartus II 64-bit 15.0.0. The board's LTC2978 power supply regulator was used for taking system-wide runtime power measurements. The SoC's hard CPU cores, clocked at 925MHz and running Ubuntu 14.04, were used to communicate with the modules and to implement Section III-C's online modelling. Modules were each able to be clocked separately up to 200MHz, while the bus that connected them operated at a fixed 50MHz.

At runtime, each module could be independently exercised using a random combination of input data, filter coefficient sets and frequency. Input data was selectable from two checkerboard (alternating maximum-minimum values) patterns and two gradients (increasing values) with different periods, two uniform random data sets and values from red, green and blue frames of Lena. Available coefficients were all-zero, identity, two different 3×3 edge detection matrices, 3×3 and 5×5 box blurs, two Gaussian blurs (also 3×3 and 5×5), a 3×3 sharpen matrix and 5×5 unsharpen. The input data, coefficients and frequency selected for each differently implemented module allowed the power behaviour exhibited to be wide-ranging.

A. Dynamic Power Breakdown

Experiments were performed to explore how model accuracy changed as KAPow parameters N and W, described in Section III, were modified. These were conducted in iterations, with a different system workload—a random mapping of input data, filter coefficients and frequency to each module—applied each time. Activity and system-wide power measurements were taken to update the model following each application. During every tenth iteration, a true power breakdown was established by successively clock-gating modules and repeating power measurements. These were then used to form comparisons against the model's outputs.

In Figure 6, absolute error plots are included for various combinations of N and W. Each point is the mean error across the system's seven modules, with 50-point uniform moving-average windowing applied to reduce noise. The results show that larger values of N produced lower error but took longer to settle. Analysis across the various N suggests that the choice of W impacts significantly upon steady-state error but not necessarily on the model's convergence. Results for $N \geq 128$ revealed that 5mW accuracy was achievable when $W = 9$. Testing showed deviations in the system-level power measurements in the 5mW range, indicating that the model accuracy was able to meet the limits of our experimental setup.

B. Static Power Tracking

KAPow's ability to track changes in static power was verified by varying the FPGA-SoC package temperature while running the random task-mapping experiment described in Section VII-A. Here, N and W were fixed at 8 and 9, respectively. Temperature was initially held at 25°C. After 300 iterations, it was raised by 5°C every subsequent 150

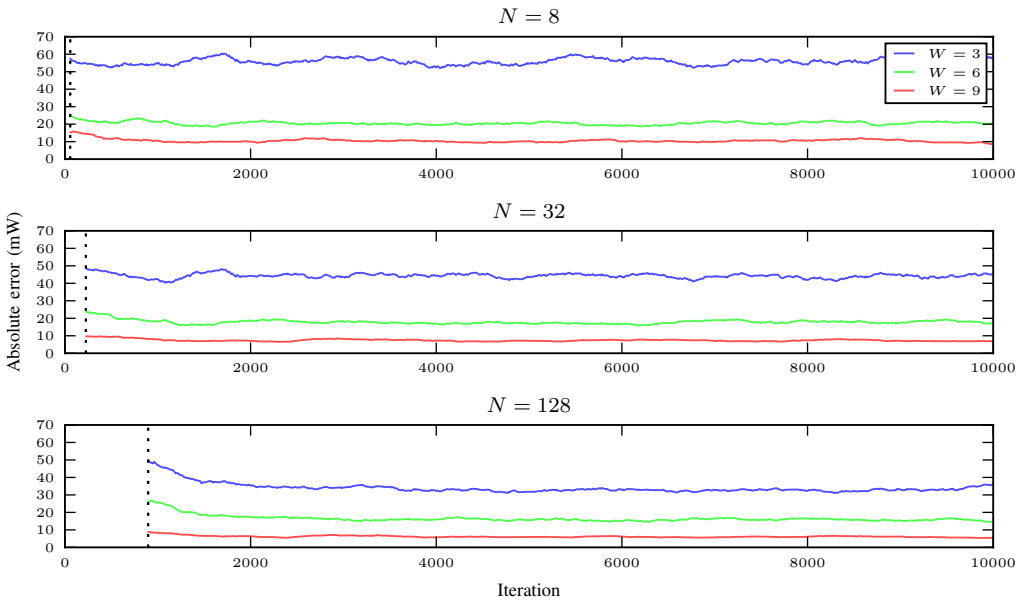

Fig. 6. Seven-FIR filter benchmark system mean per-module absolute error across various N and W [5]. Dotted lines mark the end of the model-building phase after $7N + 1$ iterations.

iterations until reaching the device's maximum rating of 85°C. Figure 7 shows the results of this experiment without averaging, displaying close correlation between modelled and measured static power.

C. Overheads

The uninstrumented reference system occupied 24% of the target device's adaptive logic modules (ALMs), spread over 62% of its logic array blocks (LABs), 94% of embedded memories and 80% of its digital signal processing elements. Figure 8 gives the FPGA resource overheads in terms of ALMs and LABs, as well as the compilation time and power penalties incurred by adding KAPow's instrumentation. These results, when considered alongside those in Section VII-A, suggest that $N = 8$ and $W = 9$ provide a reasonable tradeoff between absolute error, area (ALM) and power overheads: 10mW, 9.0% and 3.6%, respectively. It is for this reason that default values of 8 and 9 were chosen for KOC arguments `--kapow_n` and `--kapow_w`.

VIII. Conclusion and Future Work

In this article, we presented KOCL, a tool facilitating the provision of kernel-level power consumption breakdowns for OpenCL designs implemented using FPGA-SoCs. A combination of hardware instrumentation and system identification is used to achieve this. Instrumentation is added automatically to hardware systems described in kernel code by KOCL's offline compiler, while system identification runs in software alongside OpenCL host code. KOCL is general-purpose, applicable to any OpenCL design targetting an Altera FPGA, and its operation is transparent. Most importantly, KOCL necessitates no exposure to hardware whatsoever.

In the future, we will explore providing KOCL-derived power estimates to higher-level frameworks for runtime management, enabling those tools to make more intelligent scheduling decisions in order to save energy. Improvements to the underlying KAPow methodology, in particular its signal selection, will also be incorporated into KOCL. We will explore the combination of macro-modelling with our techniques to target signals associated with particular power modes. Finally, we hope to use similar methods to enable the monitoring of different system states, particularly with respect to reliability, in an equivalently unburdensome fashion.

Acknowledgements

This work was supported by the EPSRC-funded PRiME Project (grant numbers EP/I020357/1 and EP/K034448/1).
http://www.prime-project.org

The authors also acknowledge the support of Imagination Technologies and the Royal Academy of Engineering.

References

[1] D. Chen and D. Singh, "Fractal Video Compression in OpenCL: An Evaluation of CPUs, GPUs, and FPGAs as Acceleration Platforms," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2013, pp. 297–304.

[2] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *International Symposium on Computer Architecture (ISCA)*, 2011, pp. 365–376.

[3] E. Stott, Z. Guan, J. M. Levine, J. S. J. Wong, and P. Y. K. Cheung, "Variation and Reliability in FPGAs," *IEEE Design & Test*, vol. 30, no. 6, pp. 50–59, 2013.

[4] E. Stott, J. S. J. Wong, and P. Y. K. Cheung, "Degradation Analysis and Mitigation in FPGAs," in *International Conference on Field-programmable Logic and Applications (FPL)*, 2010, pp. 428–433.

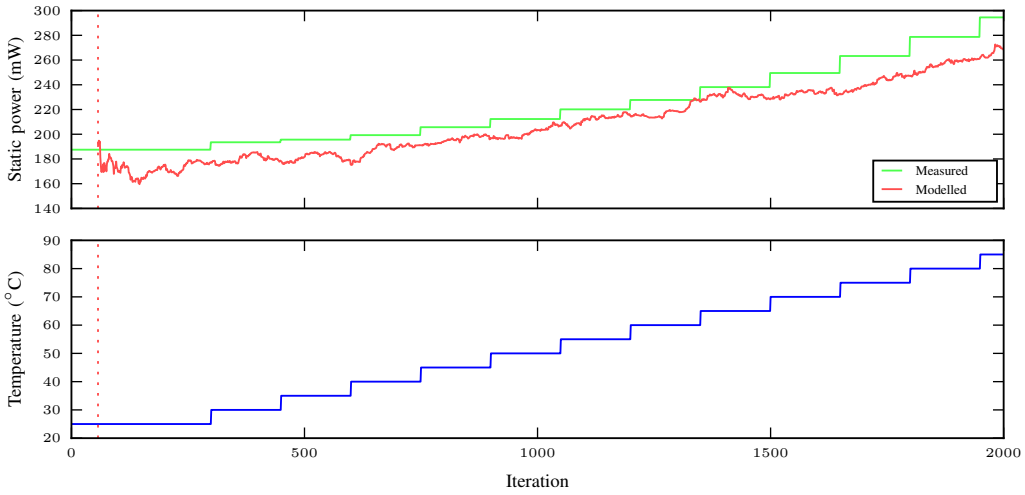

Fig. 7. Static power tracking during temperature sweep for seven-FIR filter benchmark system with $N = 8$ and $W = 9$ [5]. The dotted line marks the end of the model-building phase after 57 iterations.

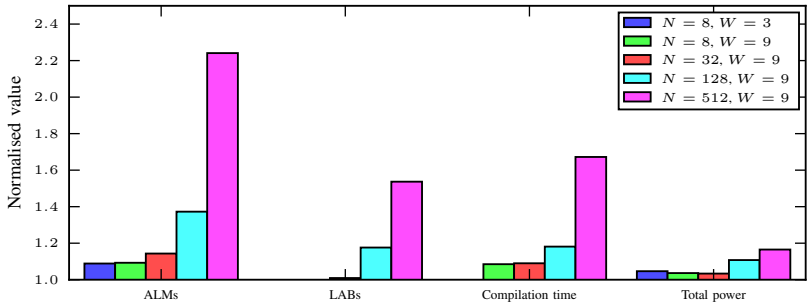

Fig. 8. Seven-FIR filter benchmark system area, compilation time and power overheads across various N and W [5]. Results are normalised to those for the equivalent uninstrumented ($N = 0$) design.

[5] E. Hung, J. J. Davis, J. M. Levine, E. A. Stott, P. Y. K. Cheung, and G. A. Constantinides, "KAPow: A System Identification Approach to Online Per-module Power Estimation in FPGA Designs," in *International Symposium on Field-programmable Custom Computing Machines (FCCM)*, 2016, pp. 56–63.

[6] F. N. Najm, "A Survey of Power Estimation Techniques in VLSI Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, pp. 446–455, 1994.

[7] J. Lamoureux and S. J. E. Wilton, "Activity Estimation for Field-programmable Gate Arrays," in *International Conference on Field-programmable Logic and Applications (FPL)*, 2006, pp. 1–8.

[8] A. Lakshminarayana, S. Ahuja, and S. Shukla, "High-level Power Estimation Models for FPGAs," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2011, pp. 7–12.

[9] M. Najem, P. Benoit, F. Bruguier, G. Sassatelli, and L. Torres, "Method for Dynamic Power Monitoring on FPGAs," in *International Conference on Field-programmable Logic and Applications (FPL)*, 2014, pp. 1–6.

[10] K. M. Zick and J. P. Hayes, "On-line Sensing for Healthier FPGA Systems," in *ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2010, pp. 239–248.

[11] P. Mantovani, E. G. Cota, K. Tien, C. Pilato, G. D. Guglielmo, K. Shepard, and L. P. Carlon, "An FPGA-based Infrastructure for Fine-grained DVFS Analysis in High-performance Embedded Systems," in *Design Automation Conference (DAC)*, 2016, pp. 1–6.

[12] Y. Wu, J. Nunez-Yanez, R. Woods, and D. S. Nikolopoulos, "Power Modelling and Capping for Heterogeneous ARM/FPGA SoCs," in *International Conference on Field-programmable Technology (FPT)*, 2014, pp. 231–234.

Dr James J. Davis is a Research Associate at Imperial College London. His research concerns the runtime monitoring and adaptation of digital electronic hardware for energy efficiency and reliability. James received his PhD in Electrical and Electronic Engineering from Imperial College London. He is a Member of the IEEE and ACM.

Dr Joshua M. Levine is a Hardware Engineer in Intel's Programmable Solutions Group. His research interests include FPGAs, reliability, resilience, ageing and the development and application of circuit-level knobs and monitors for runtime adaptation. Joshua received a PhD in Electrical and Electronic Engineering from Imperial College London. He is a Member of the IEEE.

Dr Edward A. Stott is a Teaching Fellow at Imperial College London. His research interests include FPGAs, CMOS reliability and ageing and dynamic voltage and frequency scaling. Edward received his PhD in Electrical and Electronic Engineering from Imperial College London. He is a Member of the IEEE.

Dr Eddie Hung is the Technical Lead at Invionics. His research interests include hardware debugging and runtime monitoring facilitating power and performance improvements. Eddie received his PhD in Electrical and Computer Engineering from the University of British Columbia. He is a Member of the IEEE.

Professor Peter Y. K. Cheung is the Vice Dean (Education) of Imperial College London's Faculty of Engineering. His research interests include VLSI architectures for signal processing, asynchronous systems, reconfigurable computing and architectural synthesis. Peter received his DSc from Imperial College London. He is a Senior Member of the IEEE and Fellow of the IET.

Professor George A. Constantinides is the Royal Academy of Engineering/Imagination Technologies Research Professor of Digital Computation and Head of Imperial College London's Circuits and Systems research group. George received his PhD in Electrical and Electronic Engineering from Imperial College London. He is a Senior Member of the IEEE and a Fellow of the BCS.