KAPow: High-accuracy, Low-overhead Online Per-module Power Estimation for FPGA Designs

JAMES J. DAVIS, Imperial College London
EDDIE HUNG, Imperial College London
JOSHUA M. LEVINE, Imperial College London
EDWARD A. STOTT, Imperial College London
PETER Y. K. CHEUNG, Imperial College London
GEORGE A. CONSTANTINIDES, Imperial College London

In an FPGA system-on-chip design, it is often insufficient to merely assess the power consumption of the entire circuit by compile-time estimation or runtime power measurement. Instead, to make better runtime decisions, one must understand the power consumed by each module in the system. In this work, we combine measurements of register-level switching activity and system-level power to build an adaptive online model that produces live breakdowns of power consumption within the design. Online model refinement avoids time-consuming characterisation while also allowing the model to track long-term operating condition changes. Central to our method is an automated flow that selects signals predicted to be indicative of high power consumption, instrumenting them for monitoring. We named this technique KAPow, for 'K'ounting Activity for Power estimation, which we show to be accurate and to have low overheads across a range of representative benchmarks. We also propose a strategy allowing for the identification and subsequent elimination of counters found to be of low significance at runtime, reducing algorithmic complexity without sacrificing significant accuracy. Finally, we demonstrate an application example in which a module-level power breakdown can be used to determine an efficient mapping of tasks to modules and reduce system-wide power consumption by up to 7%.

CCS Concepts: •**Computing methodologies** → **Learning linear models; Modeling methodologies;** •**Hardware** → **Design modules and hierarchy; Reconfigurable logic and FPGAs; System on a chip; On-chip resource management; On-chip sensors; Power estimation and optimization;**

ACM Reference Format:
James J. Davis, Eddie Hung, Joshua M. Levine, Edward A. Stott, Peter Y. K. Cheung, and George A. Constantinides, 2017. KAPow: High-accuracy, Low-overhead Online Per-module Power Estimation for FPGA Designs. *ACM Trans. Reconfig. Technol. Syst.* 0, 0, Article 00 (0000), 22 pages.
DOI: 0000000.0000000

1. INTRODUCTION

In a world increasingly dominated by systems-on-chip (SoCs), power efficiency is of ultimate concern due to the dark silicon effect [Esmaeilzadeh et al. 2011]: more transistors can be placed on a die than can be continuously switched. Designers put a large amount of effort into managing this challenge up-front, but many things can change once a system is manufactured and deployed; to simply assume worst-case behaviour incurs significant performance penalties under average conditions. For example, a sys-

This work was supported by the EPSRC-funded PRiME Project (grant numbers EP/I020357/1 and EP/K034448/1).
Authors' address: J. J. Davis, E. Hung, J. M. Levine, E. A. Stott, P. Y. K. Cheung and G. A. Constantinides, Department of Electrical and Electronic Engineering, Imperial College London, London, SW7 2AZ, United Kingdom. E-mail: {james.davis06, e.hung, josh.levine05, ed.stott, p.cheung, g.constantinides}@imperial.ac.uk.

DOI: 0000000.0000000

tem may be produced where, due to variation [Stott et al. 2013], some of its constituent components are more power-efficient than others. An intelligent, self-aware system might independently control the power consumption of some or each of them using dynamic frequency scaling. Tasks could then be mapped to these modules in a way that delivers the best overall performance given the constraints of the power budget, available hardware and work to be done.

Such runtime techniques would be particularly useful for FPGAs, where the shortened design cycles reduce the time available for offline analysis. FPGAs' reconfigurable hardware makes it more difficult to implement well established techniques, such as power gating, but also offers great opportunities for runtime adaptation. Unfortunately, the self-awareness necessary to deliver this vision is currently missing from the power consumption toolbox: we can measure system-wide power consumption at runtime and forecast per-module contributions at compile-time, but we cannot determine such a breakdown online.

The term *module* is used throughout this article. By module, we mean the hardware constituting an IP block, accelerator, kernel, etc. of granularity typically seen in FPGA-SoC designs. While designers have complete freedom to choose what constitutes a module, and thereby set the granularity of power estimation, in this work a module is always an individually addressable memory-mapped hardware accelerator.

1.1. Per-module Online Power Modelling

While power measurement at supply pins is common, manufacturing SoCs with per-module power domains is often impractical due to increased metal and pad costs. A more feasible approach is to instead monitor the switching activity within each module, since activity is a key indicator of dynamic power. Models that forecast power consumption based on predicted or simulated switching activity are well established for use at compile-time; however, inaccuracies inevitably arise from assumptions made regarding data patterns and operating conditions. Some of these assumptions can be avoided by training a model during commissioning but, unless the external conditions are static and all the possible system behaviour is captured by the training programme, such a model would be running blindly and errors will begin to accumulate. Instead, a means to calculate a runtime power breakdown without relying on an open-loop model susceptible to becoming stale is needed.

Figure 1 illustrates the benefits of an online, activity-based power model, described in this article, used to estimate power consumption. The plot shows the error between modelled and externally measured power for a system module as its supply voltage is lowered, comparing an online model to an offline version based on ordinary least squares, simulating such a model being set at commissioning. As the operating conditions deviate from their nominal values, the error in the offline power model's output increases, while the online model quickly adapts. Although voltage is controlled in this case, which, as with any other source of variation, could have been trained for offline, any offline model is necessarily incomplete in any real environment. The effects of different classes of input data, operating modes and exogenous conditions, such as temperature, voltage and degradation [Stott et al. 2010], can all be captured online, with the resulting model being far more useful for runtime control in a dynamic setting than an offline counterpart. Note that while Figure 1 is included here for motivational purposes, the system from which its data was obtained is that introduced in Section 5.1 using the online algorithm described in Section 4.

Using an adaptive online model in an embedded system is more practical than ever thanks to readily available general computing resources. FPGA-SoCs with hardened multicore CPUs are ideal platforms to use since the general-purpose processors can

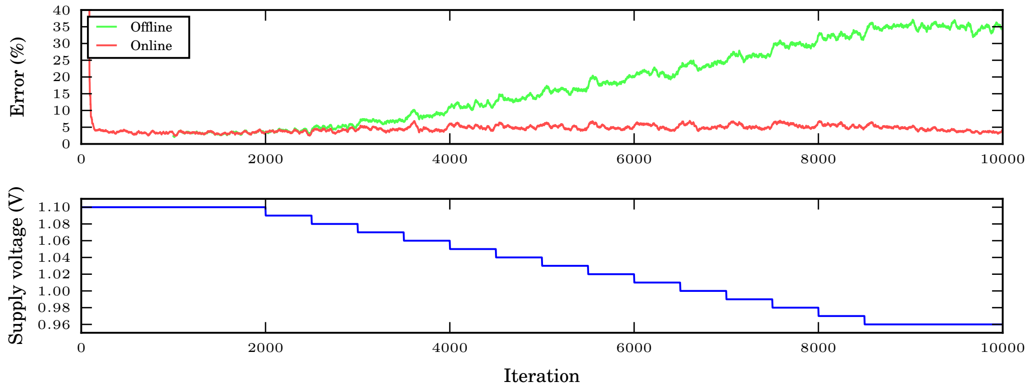

Fig. 1: Error accumulations in online- versus offline-generated signal activity-to-power models under voltage scaling. Here, a train-once use-forever model is shown to estimate inaccurately when faced with conditions for which it has not been trained.

carry out infrequent incremental model updates and optimise the computationally intensive tasks that run on the FPGA.

In this article, we describe the first runtime modelling framework capable of providing a per-module power breakdown for an FPGA-based system, making the following novel contributions:

— We describe an automated tool flow enabling the appropriate selection and instrumentation of signals within arbitrary RTL for runtime activity monitoring.
— We apply system identification to allow an activity-to-power model for any modular design to be trained and updated at runtime.
— We experimentally quantify the relationships between model accuracy and the incurred hardware and software overheads for 15 benchmark modules arranged across two multi-module systems.
— We demonstrate a simple yet effective signal pruning strategy capable of significantly reducing online modelling complexity without corresponding reductions in accuracy.
— We show how knowledge of per-module behaviour can facilitate the power-efficient mapping of tasks to hardware, leading to a power reduction of up to 7%.

An earlier version of this work appeared in the proceedings of the 24th IEEE International Symposium on Field-programmable Custom Computing Machines (FCCM) [Hung et al. 2016]. This article expands on the material presented therein by including results and analysis for a wider range of exemplary modules as well as a verified technique for reducing runtime complexity without significant loss of accuracy.

2. BACKGROUND

The power consumed by a CMOS integrated circuit can be split into static and dynamic components: static power consumption is a property of the process technology and operating conditions, while dynamic power originates from the charging and discharging of circuit nets due to switching activity. Both components can vary at runtime and need to be considered by a model. Equation 1 describes the relationship between power and its static and dynamic components, the latter being dependent upon switching activity a_i on net i, operating frequency f, effective capacitance C_i—determined by circuit topology and process technology—and supply voltage V.

$$\begin{aligned} P_{\text{total}} &= P_{\text{static}} + P_{\text{dynamic}} \\ &= I_{\text{leakage}}V + \sum_i a_i f C_i V^2 \end{aligned} \tag{1}$$

Power estimation tools are widely used at compile-time to ascertain whether designs will meet their power specifications and to inform decisions on packing, thermal management and power supply capacity. These tools initially estimated power consumption by simulating circuits with typical test vectors and later using vectorless, probabilistic techniques [Najm 1994]; the latter style has been further applied to FPGAs to facilitate power-aware compilation [Anderson and Najm 2004]. As the complexity of FPGA applications has increased, so too has the need for high-level power estimation models for modular systems [Lakshminarayana et al. 2011]. These statistical, learning-based models are trained using activity and power estimates as well as resource utilisation, allowing implementations to be compared without performing placement and routing.

Prior work [Najem et al. 2014] proposed the evaluation of system-wide power consumption through a dynamic power monitoring approach using activity counters. Therein, signals were automatically selected for monitoring and an offline model was developed through simulation to relate these counts to power; realtime activity measurements facilitated runtime power estimation. Because the model was trained offline, however, the results were found to have up to 15% error. Moreover, observation of only overall system power does not provide sufficient information to deploy adaptive strategies such as dynamic voltage and/or frequency scaling, task migration or power gating on a module-level basis.

The use of ring oscillators coupled to frequency counters woven through application circuitry to infer intra-die variations in phenomena including power consumption has also been proposed [Zick and Hayes 2010]. For the establishment of module-level power, however, such indirect measurement would suffer from differences in logical and physical mapping; while modules tend to be separated by design, in practice they merge, making the task of assigning power to each more complicated. The work also required that application circuitry be halted as part of the measurement procedure.

3. TOOL FLOW

Our automated tool flow comprises two main stages: signal selection, which identifies nets likely to be correlated with high dynamic power consumption, and instrumentation, which adds logic to them to allow their behaviour to be monitored online. In this work, we monitor only synchronous events on these signals, rather than those introduced asynchronously by glitches; the online model we use will, however, automatically compensate by adding more weight to signals that are prone to glitching. We target systems composed of multiple modules described in RTL and assembled in Altera's QSys system integration tool. These are transparently instrumented in order to report their own switching activity while remaining functionally identical. We assume a master CPU-slave accelerator model, corresponding to the typical use of newer FPGA-SoC devices: in this work, specifically the Altera Cyclone V SoC family.

The most straightforward approach to instrumenting a module would be to analyse (or simulate) its RTL in order to determine signals with high activity before augmenting it with counters. Three issues exist with this method: firstly, estimating the power consumption of individual signals based solely on behavioural RTL can be inaccurate since neither physical resources nor the data that exercise them are considered [Lakshminarayana et al. 2011]; secondly, signals specified in RTL invariably differ from those present in the resulting gate-level netlists; and, thirdly, augmentation of RTL with instruments would likely cause circuit mapping to be different, thereby altering

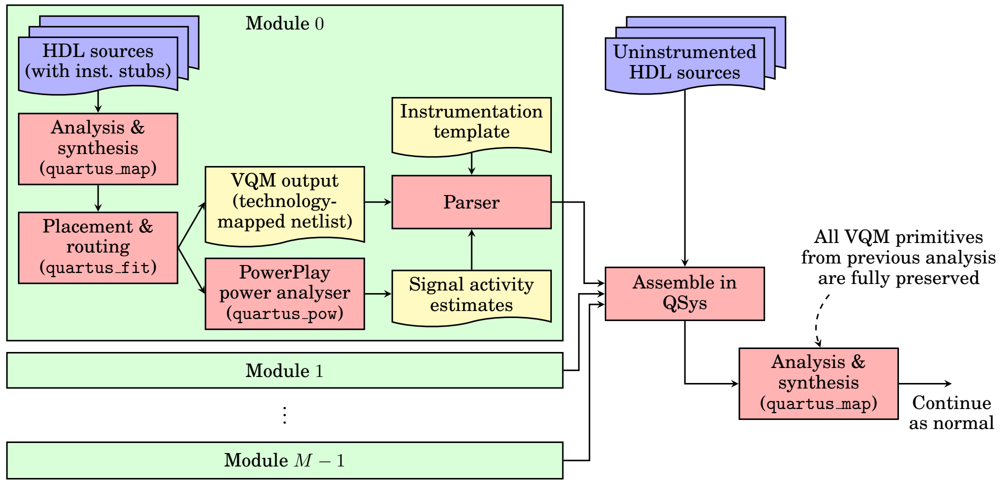
Module 0
HDL sources (with inst. stubs)
Analysis & synthesis (quartus_map)
Placement & routing (quartus_fit)
VQM output (technology-mapped netlist)
PowerPlay power analyser (quartus_pow)
Instrumentation template
Parser
Signal activity estimates
Module 1
Module $M-1$
Uninstrumented HDL sources
Assemble in QSys
All VQM primitives from previous analysis are fully preserved
Analysis & synthesis (quartus_map)
Continue as normal

Fig. 2: KAPow tool flow.

the very thing one wishes to observe. The latter problem is similar to that faced when inserting timing [Levine et al. 2012] or debugging infrastructure [Hung and Wilton 2015], both of which are even more sensitive to any circuit perturbations. Our flow, shown in Figure 2, overcomes these issues to some extent by performing signal selection and inserting instrumentation using placed and routed netlists.

Before compilation, some minor preprocessing must be performed on each of the M modules to be instrumented: each is modified to accommodate six additional words within its address space for instrumentation control. Following this, the modules' RTL is compiled and the resultant netlists extracted as Verilog Quartus Mapping (VQM) files [Murray et al. 2013]. In order to identify the signals that will best indicate dynamic power consumption, each module is fed through Altera's compile-time power analysis tool, PowerPlay, as described in Section 3.1. Selected signals are augmented with activity counters (detailed in Section 3.2) by modifying the VQMs, which are then substituted for the modules' original RTL within QSys. In order to minimise structural disturbances, some signals are excluded from monitoring: we do not instrument those added only to route through LUTs to access register inputs (so-called 'feeders'), nor signals that form carry chains. Finally, the system is compiled as normal: importantly, with VQM primitive preservation enabled.

3.1. Signal Selection

The purpose of signal selection is to identify nets in the design that are strongly indicative of their modules' power consumptions. We do this by using PowerPlay to report nets with high estimated switching activity. Unlike the authors of prior work [Lee et al. 2015], we do not restrict ourselves to selecting only modules' primary inputs or outputs for monitoring; hence, our selections lead to sets of internal nets with strong first-order relationships between activity and power. Signals may exist that exhibit more complex (potentially non-linear) switching-to-power relationships, such as control nets, and their specific exploitation is an area we leave to future work.

Switching forecasts come from one of two sources: in *vectored* mode, switching rates are derived from a simulation, while in *vectorless* mode, they are estimated using statistical methods instead [Lamoureux and Wilton 2006]. A vectorless estimation starts from primary inputs and propagates switching rates through all nets by considering the Boolean functions that associate them. Although vectorless estimation is known

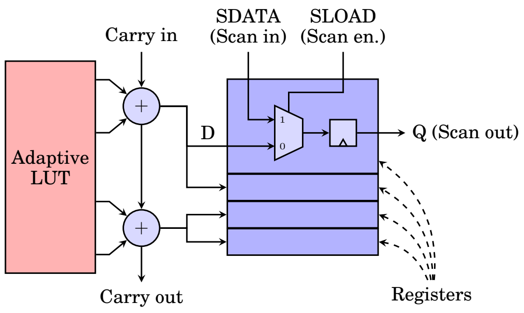

Fig. 3: Simplified view of a Cyclone V ALM. Its uppermost register has been exploded to demonstrate the use of its internal multiplexer in forming the first flip-flop within each instrument's LFSR.

to be less accurate than its vectored counterpart, we adopt this method for signal selection because it is general-purpose: it is applicable to any circuit, even those without source or testbench code or indicative test vectors. We leave the switching rate for primary inputs at its default value of 12.5%.

Since we only use PowerPlay to identify signals to monitor, the absolute accuracy of those signals' estimated activities is essentially irrelevant as they will be measured at runtime. What is of concern, however, are the relative accuracies of those estimates, since these determine the ordering. The tool produces a file containing switching rates for all signals, from which we select the N—a user-selectable parameter—most active. PowerPlay's estimates of power consumption are inherently inaccurate and do not capture any dynamic shifts brought about through changing input data or environmental conditions. However, for the purposes of identifying the 'hot' signals at compile-time this is unimportant since their coefficients will be determined and tuned at runtime.

3.2. Instrumentation

At the heart of any instrumentation that accumulates events is an efficient counter structure. We use linear-feedback shift register (LFSR)-based counters because they are smaller and faster than arithmetic counters [Xilinx 1996]: up to four LFSR bits can be placed in each adaptive logic module (ALM) of the Cyclone V architecture we target as opposed to two binary counter bits due to the presence of only two full adders. Output values are not consecutive; however, since each counter needs relatively few bits (justified in Section 6.1), the decoding is easy to perform in software via lookup.

A low-overhead single-bit scan chain is sufficient to read out the activity counts since the required sampling rate is low. Reading out the counters' contents also shifts zeroes into the head of the scan chain: an efficient reset method. An advantage of the Cyclone V architecture is that register resources each have two data input ports [Altera 2016], as shown in Figure 3: within the first register of each LFSR we use one to capture the next state of the counting logic while the second forms the scan chain input, avoiding the use of a soft multiplexer.

Figure 4 shows the full, scan-capable activity counter. In counting mode, when a positive edge is detected and the instrument is enabled (Enable $= 1$, Scan enable $= 0$), the LFSR's clock enable is driven high, causing it to advance to its next state. When the scan chain is in operation (Enable $= 1$, Scan enable $= 1$), the LFSR does not shift into its next state, but takes on the value at its scan input instead.

Scan chain read-back is accomplished through a FIFO instantiated as part of a template, shown in Figure 5, which augments each module. Clock domain crossing, if re-

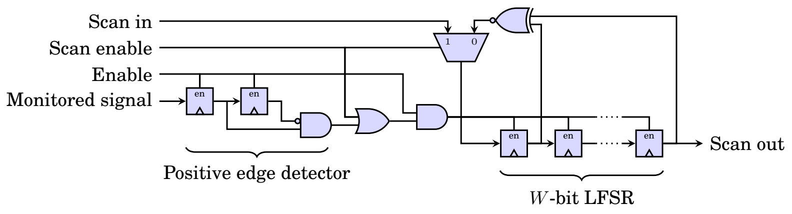

Fig. 4: LFSR-based activity counter with scan ports used to chain each module's instruments together for read-back.

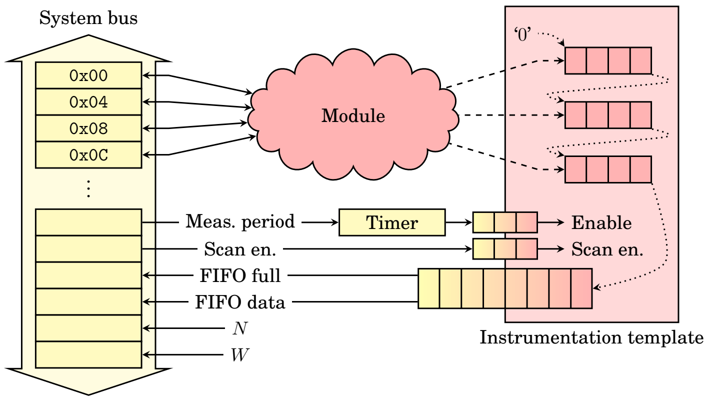

Fig. 5: Instrumented module, showing the division of clock domains used to allow the module and system bus to be independently clocked. Also shown is the scan chain, its head grounded to cause reset during read-back of counter values. Six registers within the module's address space must be reserved for instrumentation control.

quired, is handled automatically by the FIFO and handshaking for control signals. To take an activity measurement, the counters are enabled for a period dictated by an adjustable measurement timer, after which the contents of the scan chain can be read out across the system bus via the FIFO. To maximise dynamic range while guaranteeing no overflow, the timer's period should equal $2\left(2^W - 2\right)$ cycles of the module's clock, where W is the width of each activity counter in bits. Since the measurement timer operates on the system bus' clock domain rather than the module's, the period must be scaled by the ratio of bus-to-module clock frequencies.

While we have optimised our design for a particular device family, similar activity counters and read-back infrastructure could be implemented in alternative FPGAs. Similarly, although our flow was built upon Altera tools, a Vivado-based version targetting Xilinx devices would be largely equivalent in structure. While significant engineering effort would be required to port KAPow to alternative devices and tools, our principles of measurement (and modelling) would remain the same.

4. SYSTEM IDENTIFICATION

Signal activity can be translated into a power estimate using a weighted linear model, with partial system behaviour used to dynamically update the coefficients via methods of system identification [Ljung 1998]. Figure 6 shows a typical system identification setup in which an input vector, $\boldsymbol{a}$, is fed into both a black box—'black' since this technique has no, nor needs any, understanding of the system's internals—and its model. Coefficient vector $\hat{\boldsymbol{x}}$ is an estimate of $\boldsymbol{x}$, the latter containing the 'true' coefficients of

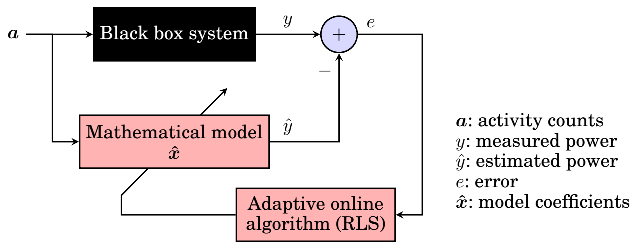
$\boldsymbol{a}$
Black box system
y
$+$
$-$
e
Mathematical model $\hat{\boldsymbol{x}}$
$\hat{y}$
Adaptive online algorithm (RLS)
$\boldsymbol{a}$: activity counts
y: measured power
$\hat{y}$: estimated power
e: error
$\hat{\boldsymbol{x}}$: model coefficients

Fig. 6: System identification overview.

the system. Outputs of the system and model, y and $\hat{y}$, respectively, are used to form an error, e; an adaptive algorithm seeks to tune $\hat{\boldsymbol{x}}$ in order to drive e towards zero for the latest observation as well as those that preceded it. We assume a linear relationship between signal activities $\boldsymbol{a}$ and measured system power y, as shown in Equation 2.

$$y = \boldsymbol{a}^{\mathrm{T}}\hat{\boldsymbol{x}} \tag{2}$$

In prior work [Najem et al. 2014], a non-adaptive offline model requiring multiple $(\boldsymbol{a},\ y)$ observation pairs for training was used. A vector of y and matrix of $\boldsymbol{a}$ were formed from these pairs, allowing Equation 2 to be solved in the least-squares sense to find $\hat{\boldsymbol{x}}$, which remained fixed during runtime. By contrast, the identification algorithm we use is recursive least squares (RLS) [Gauss 1821] [Plackett 1950], which iteratively updates $\hat{\boldsymbol{x}}$ as new measurements arrive. The RLS update function is of the form $\hat{\boldsymbol{x}}[t] = \hat{\boldsymbol{x}}[t-1] + f(e[t],\ \boldsymbol{k}[t])$, as defined in Equations 3, 4 and 5. $\boldsymbol{k}$ is the gain vector, containing factors determining the scaling of each coefficient, $\boldsymbol{P}$ the inverse covariance matrix, capturing the partial correlations between input variables, and λ the forgetting factor, a parameter determining the memory of the algorithm. $\lambda = 1$ means that all prior observations are given equal weighting (infinite memory) and is used for describing time-invariant systems, whereas lesser values allow prior samples to be assigned exponentially decaying contributions.

$$\boldsymbol{k}[t] = \frac{\lambda^{-1}\boldsymbol{P}[t-1]\boldsymbol{a}[t]}{1 + \lambda^{-1}\boldsymbol{a}[t]^{\mathrm{T}}\boldsymbol{P}[t-1]\boldsymbol{a}[t]} \tag{3}$$

$$\boldsymbol{P}[t] = \lambda^{-1}\boldsymbol{P}[t-1] - \lambda^{-1}\boldsymbol{k}[t]\boldsymbol{a}[t]^{\mathrm{T}}\boldsymbol{P}[t-1] \tag{4}$$

$$\hat{\boldsymbol{x}}[t] = \hat{\boldsymbol{x}}[t-1] + e[t]\boldsymbol{k}[t] \tag{5}$$

Note that choices made for the model update period and forgetting factor must be tailored on a per-application basis: model adaption that is too slow will prohibit compensation for fast-acting effects such as changes in operating modes, while overly frequent sampling could lead to overfitting. Changes to model update rates also impact upon the power overhead of our technique. As a result, we leave these settings as knobs to be tuned by developers to suit their application requirements.

4.1. Power Breakdown

With a linear model, it is straightforward to compute the power contribution $\hat{y}_m$ of each module m since the model's coefficients can be partitioned between modules and its individual power estimates computed as shown in Equations 6, 7, 8 and 9. Scalars a_{s} and $\hat{x}_{\mathrm{s}}$ represent the 'activity' and coefficient, respectively, for the device's static power, which, for us, also includes the dynamic power consumed by resources that are not correlated with module activity, such as the SoC bus. We keep a_{s} constant, allowing the model to tune $\hat{x}_{\mathrm{s}}$ over time. The dynamic power consumed by the activity counters themselves will be included within the respective modules' estimates since their own switching is dictated by the behaviour of the module to which each is connected.

$$\boldsymbol{a} = \begin{bmatrix} a_{\mathrm{s}} \ \boldsymbol{a}_0^{\mathrm{T}} \ \boldsymbol{a}_1^{\mathrm{T}} \ \ldots \ \boldsymbol{a}_{M-1}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \tag{6}$$

$$\hat{\boldsymbol{x}} = \begin{bmatrix} \hat{x}_{\mathrm{s}} \ \hat{\boldsymbol{x}}_0^{\mathrm{T}} \ \hat{\boldsymbol{x}}_1^{\mathrm{T}} \ \ldots \ \hat{\boldsymbol{x}}_{M-1}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \tag{7}$$

$$\hat{y}_m = \boldsymbol{a}_m^{\mathrm{T}} \hat{\boldsymbol{x}}_m \tag{8}$$

$$\hat{y} = a_{\mathrm{s}} \hat{x}_{\mathrm{s}} + \sum_{m=0}^{M-1} \hat{y}_m \tag{9}$$

5. BENCHMARK SYSTEMS

Two multi-module systems, each assembled in QSys and compiled as explained in Section 3, were designed to assess KAPow's effectiveness; these are described in detail in Sections 5.1 and 5.2. Each module was individually exercised with data fed from a coupled RAM. We targetted the Altera Cyclone V SX SoC development board [Altera 2015] for all experiments performed in this work, with Quartus II 64-bit 15.0.0 used for compilation. At the development board's core lies a 5CSXFC6D6F31C6 FPGA-SoC, consisting of two hard ARM Cortex-A9 cores tightly coupled to a 42k-ALM FPGA manufactured on a 28nm low-power process. The board also features two Linear Technology LTC2978 power supply regulators [Linear Technology 2009]—one each for the CPU cores and FPGA—the latter of which we used for taking runtime power measurements. The SoC bus and instrumentation controllers ran at 50MHz throughout all experiments conducted in this work and, aside from the experiment performed for Section 1.1, FPGA core voltage remained fixed at 1.1V.

System identification was implemented in software on the SoC's hard CPU cores, clocked at their default 925MHz and running Ubuntu 14.04. Communication with hardware modules and their instruments was accomplished through memory-mapped registers accessed from Linux using `mmap()`. Experimentation revealed that $\boldsymbol{P}[0] = 1000\boldsymbol{I}$ from starting coefficients $\hat{\boldsymbol{x}}[0] = \boldsymbol{0}$ gave good results and we found $\lambda = 0.999$ to work well in allowing the algorithm to adapt to changing operating conditions.

While versions of both systems were created with various combinations of N and W for area and power comparisons, $N = 512$ counters, each $W = 9$ bits wide, were embedded per module used during the experiments described in later sections. Uninstrumented systems, i.e. those with $N = 0$, were also created for baseline comparison. Where experiments demanded lower N and/or W, however, only the top (in terms of estimated switching activity) N counters were used, while the measurement period could be lowered to simulate reduced W. If $W = 3$ were required, for example, the period was reduced from $2(2^9 - 2) = 1020$ module clock cycles to $2(2^3 - 2) = 12$.

5.1. FIR Filters

The first SoC implementation developed consisted of seven functionally identical (in terms of throughput and latency) FIR filters, each operating on 240×160 8-bits-per-pixel greyscale images and with 5×5 Q4.8 fixed-point convolution kernels. Each filter formed an independently addressable module within the design. Heterogeneity was introduced by forcing each to map a different proportion of its multipliers, from all to none, to LUTs rather than DSP blocks; consequently, each module exhibited different power characteristics and, therefore, had instruments attached to different signals. The FIR modules had between 2384 and 2715 signals each, with a total of 17802 candidate signals in the whole system. The uninstrumented system occupied 24% of the available ALMs, spread out over 62% of the FPGA's logic array blocks (LABs), along with 94% of block RAMs and 80% of the available DSPs. All modules met timing at

200MHz, however each was independently clocked by a shared runtime-adjustable PLL, allowing frequency to be changed dynamically on a per-module basis.

At runtime, each filter could be exercised differently by selecting a random combination of input data sets, from a group of nine, and one of ten sets of coefficients. The former was achieved using a RAM that fed each filter cyclically after being filled with data from the host CPU, while the latter was facilitated by streaming coefficients into the filter modules: a facility provided by our design. Input data, both real and synthetic, consisted of two checkerboard (alternating minimal and maximal values) and two gradient (increasing values) patterns of differing periodicities, two sets of uniform random values and separate frames of red, green and blue colour values of Lena. The coefficient sets used were all-zero, identity, two 3×3 edge detection kernels, two box blurs (one 3×3 and the other 5×5), two Gaussian blur kernels (also in 3×3 and 5×5 variants), 3×3 sharpen and 5×5 unsharpen. The combination of input data, coefficients and clock frequency—selected at random between 10 and 200MHz—for each module therefore allowed each to exhibit a wide range of power behaviours.

5.2. Arithmetic Toolbox

The second system consisted of eight separate modules: single-precision floating-point exponential, logarithm, power and division blocks, 32-bit fixed-point CORDIC modules for calculating (co)sine and arctangent transforms and two 8-bit fixed-point FIR filters with 16- and 32-bit coefficients. As for the system described in Section 5.1, each was independently clocked and stimulated. Modules were based around arithmetic cores selected from Altera's IP library and were all able to be clocked at up to 100MHz. This arithmetic system had a total of 22215 candidate signals, spread much less evenly across its modules than for the FIR system: the largest (arctangent-calculating CORDIC) had 8080, while the smallest (16-bit-coefficient FIR) had just 214. The uninstrumented system occupied 64% of the device's ALMs across 96% of the LABs, 78% of its block RAM resources and 71% of the available DSP blocks.

Modules were stimulated using synthetic random data taken from a range of distributions—two uniform ranges (one wide and the other narrow), triangular, exponential, Gaussian and binomial Gaussian—and formatted appropriately for the module being targetted, written into a RAM connected to each module. For fixed-point data of width α bits, narrow uniform values were selected from $\left[0,\ 2^{\alpha/2} - 1\right]$ while wide values were chosen from the range $[0,\ 2^{\alpha} - 1]$. Uniform single-precision floating-point data was limited to either $[-1,\ 1]$ (narrow) or $\left[-2^{64},\ 2^{64}\right]$ (wide). Triangular values were bounded within $[0,\ 2^{\alpha} - 1]$ for fixed-point data and $\left[-2^{64},\ 2^{64}\right]$ for floating point, with symmetry around the midpoints of those ranges, while the exponentially distributed data had mean $\mu = 2^{\alpha-1}$ (for fixed point) and $\mu = 1000$ (for floating point). Finally, Gaussian distributions had $\mu = 2^{\alpha-1}$, standard deviation $\sigma = 2^{\alpha-3}$ and $\mu = 0$, $\sigma = 2^{16}$ for fixed and floating point, respectively, while binomial Gaussian values were selected from distributions with $\mu = 2^{\alpha-2}$ or $2^{\alpha-1} + 2^{\alpha-2}$ with $\sigma = 2^{\alpha-4}$ (for fixed point) and $\mu = 2^{-16}$ or 2^{16} with $\sigma = 2^{16}$ (for floating point) with 50:50 likelihood between the two μ. Clock frequencies (selectable between 10 and 100MHz) and stimuli could be changed at any time, with data freshly generated when required.

6. POWER BREAKDOWNS

We performed experiments to investigate how model accuracy—specifically, the absolute difference between modelled and measured module-level power consumption—changed as we modified parameters N, the number of activity counters per module, and W, the width of each counter in bits, for both benchmark systems.

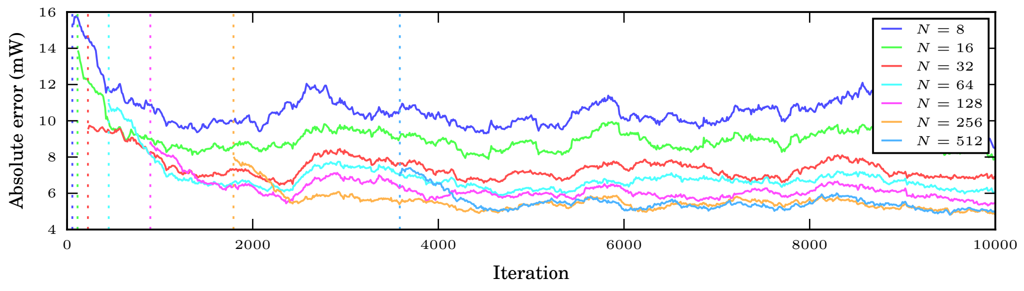

Fig. 7: Mean per-module absolute error for FIR filter benchmark system by counters per module N with counter width $W = 9$. Dotted lines represent the end of each model's building phase after $7N + 1$ coefficient updates.

Experiments were conducted in iterations, during each of which a different system workload—a random clock frequency and selection of input data (and coefficients, for the FIR filter system) for each module—was applied, following which activity and system-wide power measurements were taken and used to update the model. A true power breakdown was established on every tenth iteration by successively clock gating single modules and repeating power measurements, from which per-module power consumptions could be derived for comparison against the model's output.

The power estimation errors we report henceforth are always absolute. Beyond consistency, we opted for this presentation for two reasons. Firstly, as exhibited in Figures 9 and 12, our experiments caused rapid and wide variations in both intra- and inter-module power consumptions. Secondly, the accuracy of our reference power measurements, as touched upon in Section 6.1, was found to be magnitude-insensitive.

6.1. FIR Filters

Figure 7 shows how the absolute error between estimated and measured power consumption varies with counters per module N, with W fixed at 9. Each point is an average of errors across the system's seven modules, with 50-point uniform moving-average windowing applied in order to reduce noise and highlight the models' trends. In all cases in Figure 7 (and Figures 8 and 9 that follow), data collected for the first $7N + 1$ (the model's order) iterations are not shown since the model cannot converge on a single solution for $\hat{x}$ during this period. Larger values of N tended to result in lower relative error but took longer to converge, as one would expect since RLS adapts dynamically; absolute errors accurate to 10mW were seen with small N (8), while 5mW absolute accuracy was achievable with larger values (≥ 256).

Figure 8 shows how the absolute error varied with the counter width, W, with 50-point averaging used as in Figure 7. Analysis across the range of N tested suggests that the choice of counter width has a significant effect on steady-state error, but not necessarily the rate of convergence. These results indicate that reducing N had a smaller impact on error than decreasing W by the same factor to achieve a similar area gain for this benchmark system.

The scatter plots in Figure 9, in which each data point represents an experimental iteration with an associated power breakdown measurement, show the close correlations obtained between modelled and true per-module power consumptions for the system with $N = 8$ and $W = 9$. Once the model-building period had elapsed, the mean absolute error between modelled and measured per-module power consumption was found to be 9.8mW. There was some variation between the different filter implementa-

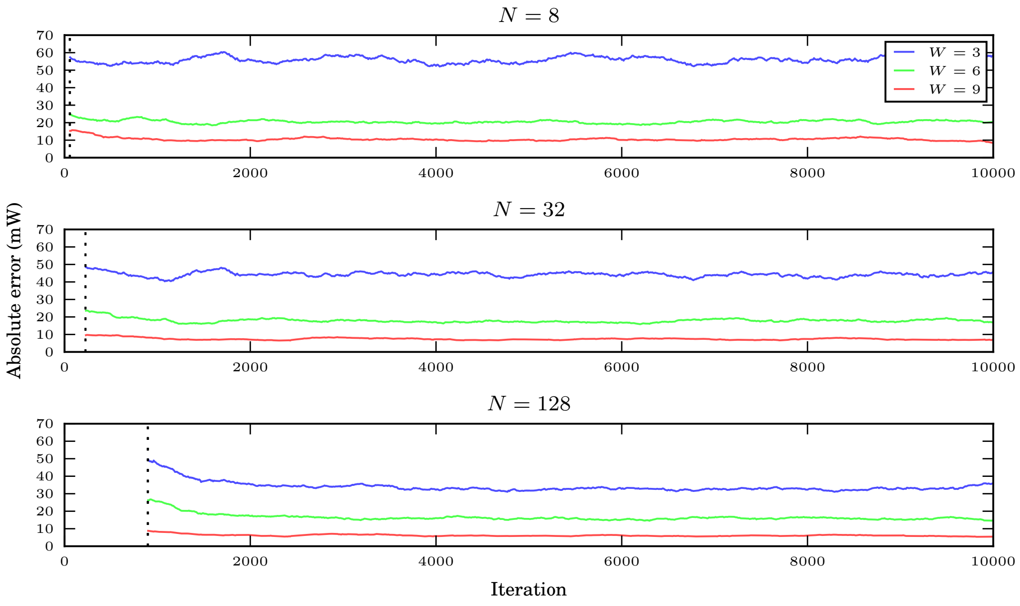

Fig. 8: Mean per-module absolute error for FIR filter benchmark system by W.

tions: the best (module 4) achieved a mean error of 8.3mW while the worst (module 6) was 13mW. For all of the modules, errors were small compared to the ranges in power consumed over their different operating modes. Static power tracking was particularly good, achieving mean absolute error of 4.0mW.

Repeatability experiments on the power breakdown measurements showed deviations in the 5mW range: the mean absolute measurement noise was 3.9mW, with outliers as large as 14mW. The observed model errors were therefore as low as the measurement noise, indicating that model accuracy was able to meet the limits of what could be measured with our test setup.

Figure 10 provides side-by-side comparisons of compile-time vectorless power forecasts from PowerPlay, true measurements and runtime estimates from KAPow. Measurements and runtime estimates were taken from a single, randomly chosen iteration during the experiment in which $N = 8$ and $W = 9$. Comparing the 'Vectorless' and 'Measured' bars, we can observe that vectorless estimation predicted approximately equal power behaviour across the modules, while measurement revealed significant variation. Looking now at the 'Modelled' bar, it can be seen that our online modelling was much closer to the measured data: KAPow successfully accounted for implementational and operational differences unforeseen by vectorless analysis.

6.2. Arithmetic Toolbox

Figures 11 and 12 are the equivalents of Figures 7 and 9 for the arithmetic-module system detailed in Section 5.2. 50-point averaging was again applied to the values in Figure 11, with data collected during the first $8N + 1$ (the model's order) iterations discarded. The scatter plots in Figure 12 used data from the system with $N = 32$, rather than 8 as in Figure 9, due to this system's relatively poor performance with only eight counters per module. This benchmark set reached an accuracy ceiling at $N = 32$, while the FIR filters showed (decreasing) benefits for every $N \geq 32$, suggesting

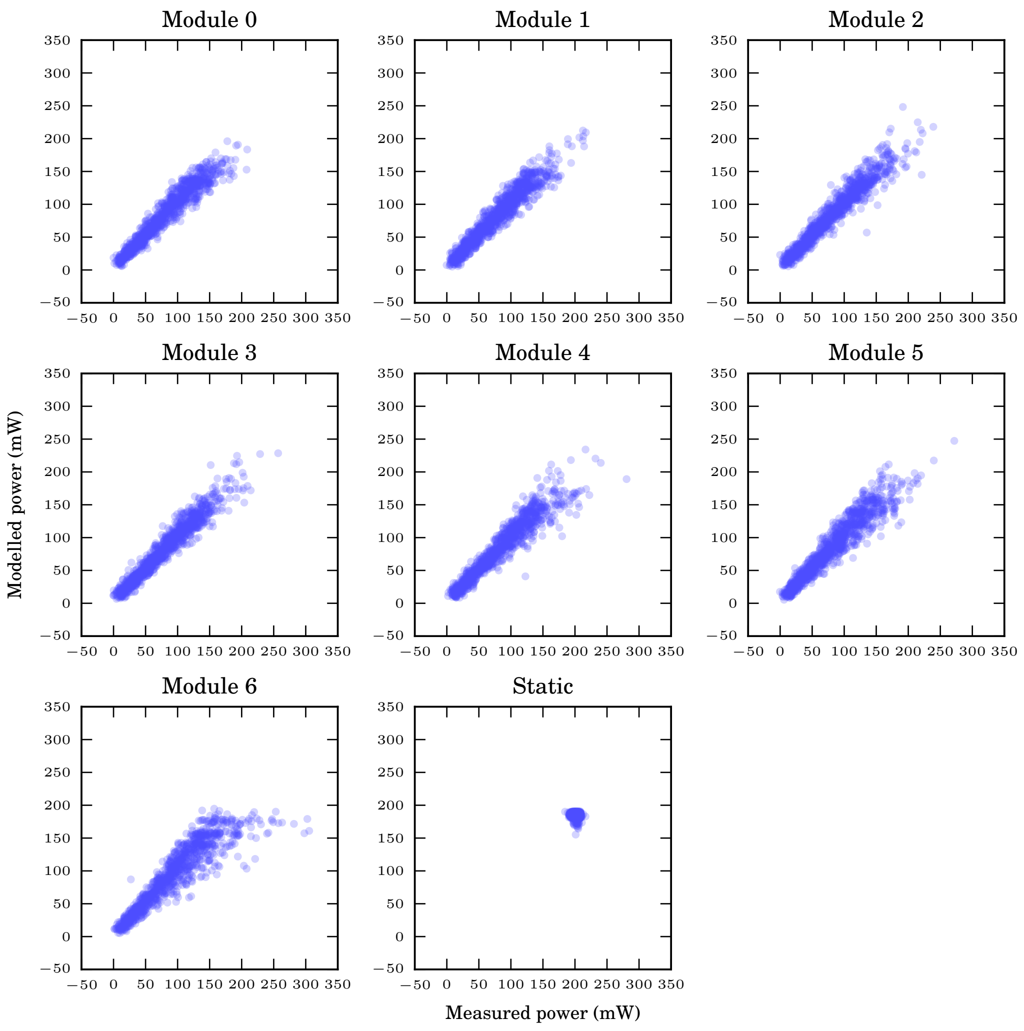

Fig. 9: Estimated versus true powers for FIR filter system with $N = 8$ and $W = 9$.

that fewer signals need to be monitored within this system to achieve comparable accuracy. Conversely, performance was found to be worse with $N < 32$ than for the multi-FIR system, indicating a wider spread of the most significant signals. These results highlight that the choice of an optimal N will depend on the specifics of the application being instrumented. While we cannot say that a particular choice of N will be ‘good enough’ for a given application, we can give the most counters, and hence the highest estimation accuracy, for given area and/or power constraints.

6.3. Static Power Compensation

An experiment was performed to verify KAPow’s ability to compensate for changes in static power consumption, a determining factor of which is temperature. We used the system described in Section 5.1 with $N = 8$ and $W = 9$ and a temperature control rig consisting of a thermoelectric effect heat pump, water cooler and resistance thermome-

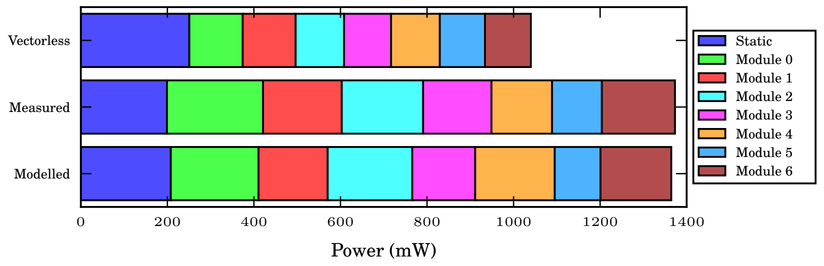

Fig. 10: Comparison of power breakdowns for FIR filter benchmark system.

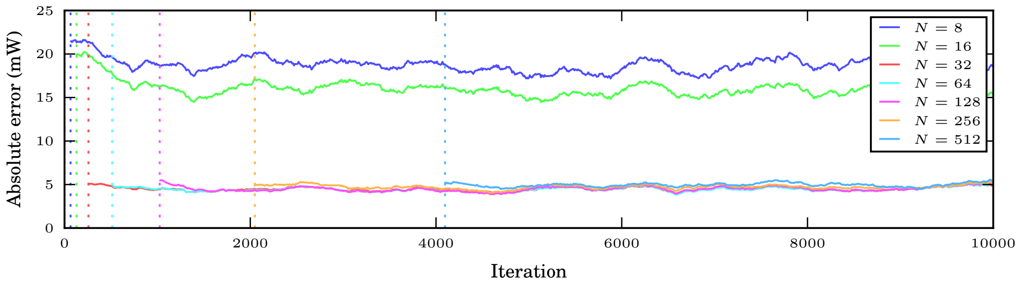

Fig. 11: Mean per-module absolute error for $W = 9$ arithmetic-module system by N.

ter, allowing the setting and maintenance of device package temperature over a wide range. Random workload changes were applied as described in Section 6.1, with static power estimated using our model and measured after clock gating all modules. Temperature was held at 25°C for the first 300 iterations, after which it was increased by 5°C every 150 iterations until reaching the device's upper temperature corner of 85°C. The results of the experiment, without averaging, are shown in Figure 13, demonstrating close tracking between model-predicted and measured static power consumption. No correlations were seen between dynamic power estimates and temperature. Notice that the static power was always underestimated: this was expected since large step changes in power occurred frequently. While such frequent temperature changes are unusual under normal operating conditions, RLS forgetting factor λ could be decreased to allow the model to adapt more quickly if such behaviour were anticipated.

7. OVERHEADS

We used the FIR benchmark system described in Section 5.1 as a representative example to demonstrate how KAPow's application and changes to its parameters, N and W, impact upon system overheads.

7.1. Hardware and Design Overheads

Figure 14 shows the overheads in area (ALMs and LABs), compilation time and power incurred through adding our instrumentation. As expected, the number of ALMs required increased with N, but the relationship between N and the number of LABs was less clear since the latter is determined by a packing heuristic. Compilation time appreciated with N, with $N = 512$ proving especially difficult to compile since it approached the limits of device capacity. The compilation time figures exclude that for

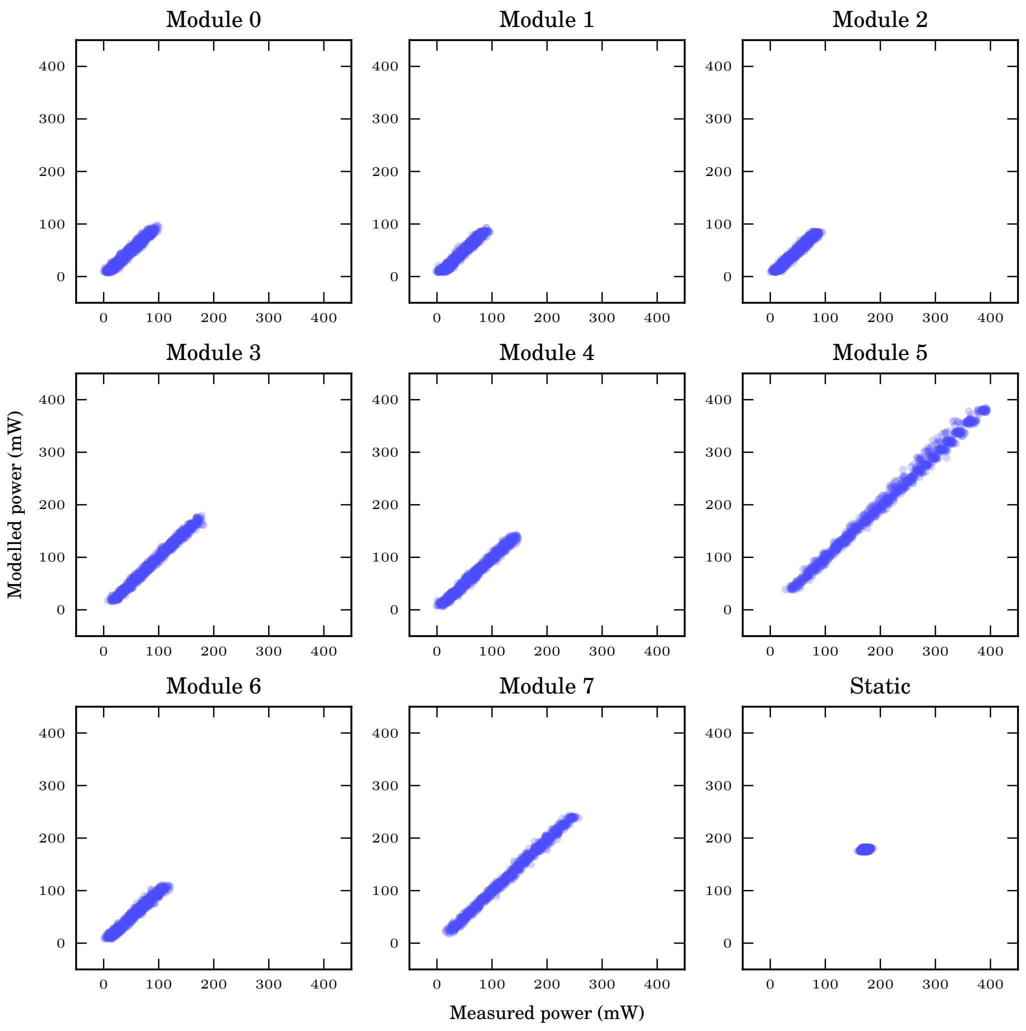

Fig. 12: Estimated versus true powers for arithmetic toolbox with $N = 32$ and $W = 9$.

initial VQM generation since, if incorporated with a production tool, this would be integrated into a single pass. System power consumption increased with N, reflecting the small structural disturbances and additional loading caused by adding activity counters. These results, coupled with those in Section 6.1, indicate that $N = 8$ and $W = 9$ can offer a reasonable tradeoff between absolute error (10mW) for area (ALM) and power overheads of 8.9% and 3.6% (53mW), respectively, for this system.

The total power overheads shown were taken with activity counters disabled. Experimentation revealed that system-wide measurements could be taken at up to 91 samples per second; with the counters operating at this rate, the average power for $N = 512$, $W = 9$ was found to increase by a further 3.7%, from 1.70W to 1.76W. With fewer counters (e.g. $N = 8$) and/or slower sampling (e.g. 1Hz)—more typical values—such additional increases were too small to measure. With modules operating at their

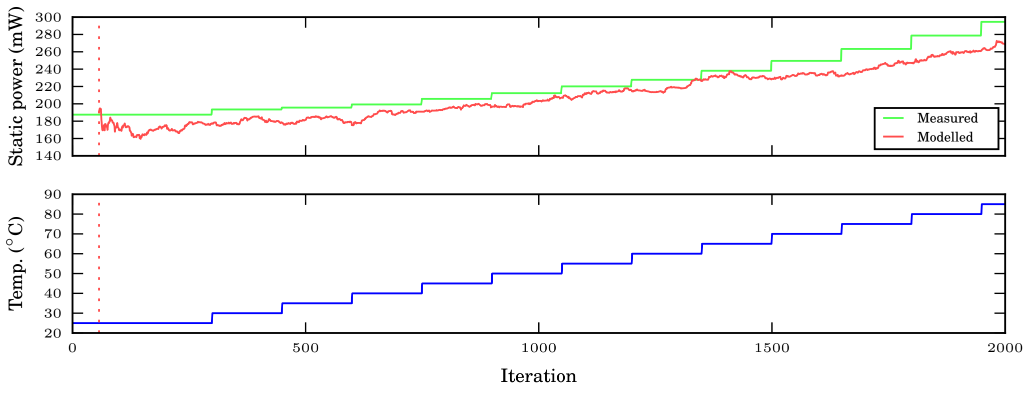

Fig. 13: Static power estimation within power breakdown during temperature sweep for FIR filter benchmark system with $N = 8$ and $W = 9$.

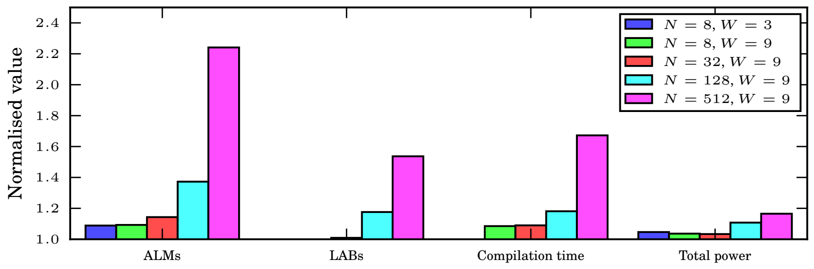

Fig. 14: Area, compilation time and power overheads for FIR filter benchmark system by N, W. All results are normalised to those for the uninstrumented design.

maximum frequency of 200MHz and 9-bit activity counters being sampled at 91Hz, instruments counted for approximately 0.05% of the time.

7.2. Software Overheads

The complexity of the RLS algorithm across various N is captured in Figure 15, the values of which represent the times needed to update the model with a new activity count and power measurement pair. With $N = 8$, each system-wide update required 690μs to complete, representing around 5% of total CPU time at the maximum sampling rate of 91Hz. Note, however, that sampling rates need not ordinarily be this high.

8. SIGNAL PRUNING

The differences in error observed across the range of counters per module N tested in Section 6 suggest that it is not necessary to monitor many of the signals selected via the rankings generated as described in Section 3.1. In order to verify this hypothesis, we modified the RLS algorithm described in Section 4 so that, after each model update, it identified signals with low contribution to the overall power estimate. The algorithm did this by selecting the coefficients in $\hat{x}$ whose absolute values were some factor—in our case, 10^4—smaller than the maximum, henceforth excluding those signals from modelling. We found starting this process after 1000 model updates to work well.

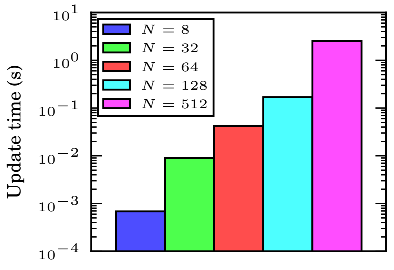

Fig. 15: System identification execution time for FIR filter benchmark system by N.

While the pruning proposed here does not directly result in area or FPGA power savings, it is nevertheless considered to be a worthwhile exercise since RLS update time—shown in Section 7.2 to be correlated with N—can be reduced by removing signals from consideration. While update periods are short for our benchmark systems when N is small, larger systems with tens or hundreds of modules may benefit dramatically from this reduction in complexity. Potentially, the instruments described in Section 3.2 could be modified to allow counter-level enabling and disabling, feasibly via the scan chain, thereby enabling power reductions as well. Information gleaned through pruning could also be used in a subsequent hardware revision, or following simulation with pruning, to enable N to be intelligently reduced.

8.1. FIR Filters

Figure 16's upper plots show the difference in results of the experiment described in Section 6.1 performed on the seven-FIR filter system detailed in Section 5.1 with and without signal pruning enabled, with positive values indicating deteriorations in modelling accuracy. 50-point averaging was applied to each plot, with each beginning after $7N + 1$ iterations, as before. Only minor increases in per-module error were observed for each N under pruning despite the low proportions of signals retained for monitoring, shown over time in the central plots of Figure 16. In the most extreme case, for $N = 512$, the total number of monitored signals was reduced from $7 \times 512 = 3584$ to just 44: a reduction nearing 99%. Even for $N = 8$, model accuracy was maintained despite half of the signals being excluded from modelling. Figure 16's lower plots show the corresponding reductions in modelling complexity, expressed as a proportion of the time required to update the model with pruning disabled. For $N = 512$, our model was able to update some $3700\times$ faster after pruning conducted until the end of the experiment.

It is interesting to note that $N = 512$ achieved lower post-pruning error than $N = 16$ despite a smaller set of signals—44 (down from 3584) versus 58 (from 112)—remaining monitored in the former case. This is an example in which the most power-indicative signals are too far down in the rankings derived from PowerPlay's activity estimates to be captured by embedding so few (in this case, 16) counters within each module.

8.2. Arithmetic Toolbox

The experiment detailed in Section 6.2 was also repeated with signal pruning enabled, the results of which are shown in Figure 17. Again, examination reveals no significant degradation in accuracy across the range of N tested, with even higher proportions of signals pruned as found for the FIR filter benchmark system and, thus, greater reductions in modelling complexity. These experiments validate the hypothesis regarding signal ranking—that it is not necessary to monitor many of the selected signals—and, consequently, motivate future work on improved design-time signal selection.

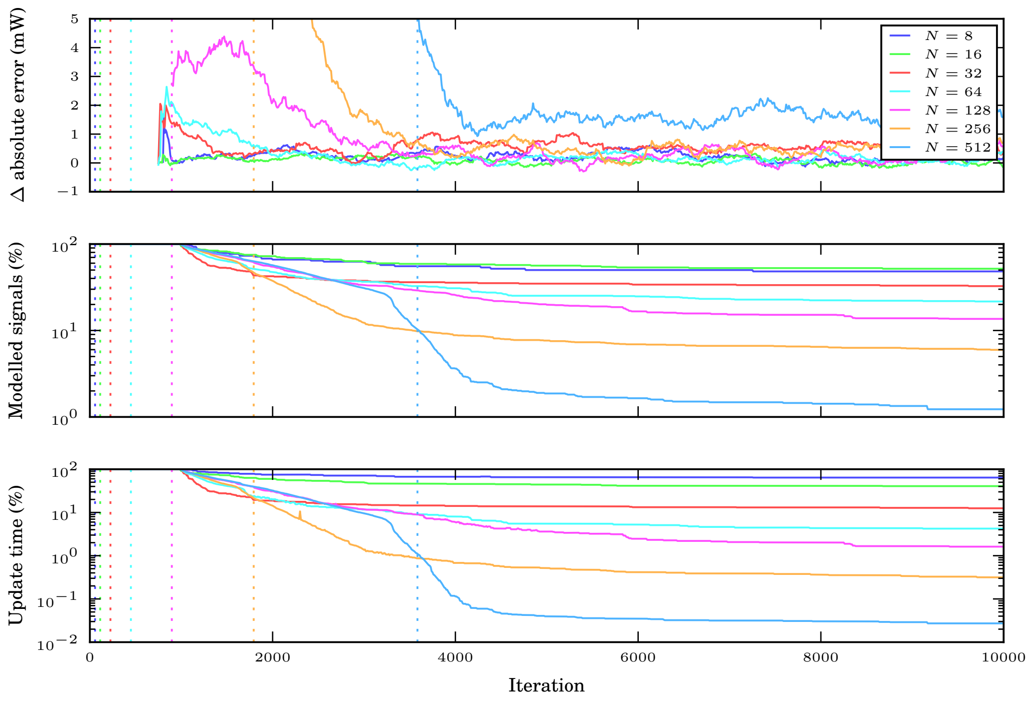

Fig. 16: Deteriorations in mean per-module absolute error by counters per module N with counter width $W = 9$ for FIR filter benchmark system under signal pruning. The central plots show, for each N, the proportion of the total signals still modelled after each update, with the corresponding proportions of the original (pruning disabled) model update times shown in the lower plots.

9. TASK MAPPING

We now demonstrate an application in which KAPow was employed to guide the mapping of tasks to modules for system-wide power optimisation. Since the device used for this experiment contained 112 DSP blocks, short of the 175 that would be needed to implement an all-DSP version of our FIR benchmark system, a case for task mapping existed; we wished to make best use of the available resources given the particular work to be done. Seven different filtering tasks (combinations of input data and coefficients), representing a range of complexities and, consequently, power behaviour, were specified to be executed simultaneously by the seven-module system described in Section 5.1, one per module. Recall that the modules were functionally identical but implementationally different, thus the task $\rightarrow$ module mapping chosen influenced the system-wide power consumption. Clock frequency remained fixed at 200MHz. The set of all possible mappings contained ${}^{7}P_{7} = 7! = 5040$ permutations; a histogram (with 2mW bins) of their power consumptions is given in Figure 18. This data suggested that an average mapping would result in system-wide power consumption of 1410mW, with best and worst cases of 1358 and 1471mW, respectively.

Figure 19 shows the results of an experiment run using a simple closed-loop controller that attempted to minimise total power consumption. Prior to RLS model initialisation, the tasks were rotated across the modules in seven steps in order to build a 7×7 activity table containing an activity vector, $\boldsymbol{a}_m$, for each task-module pair. This

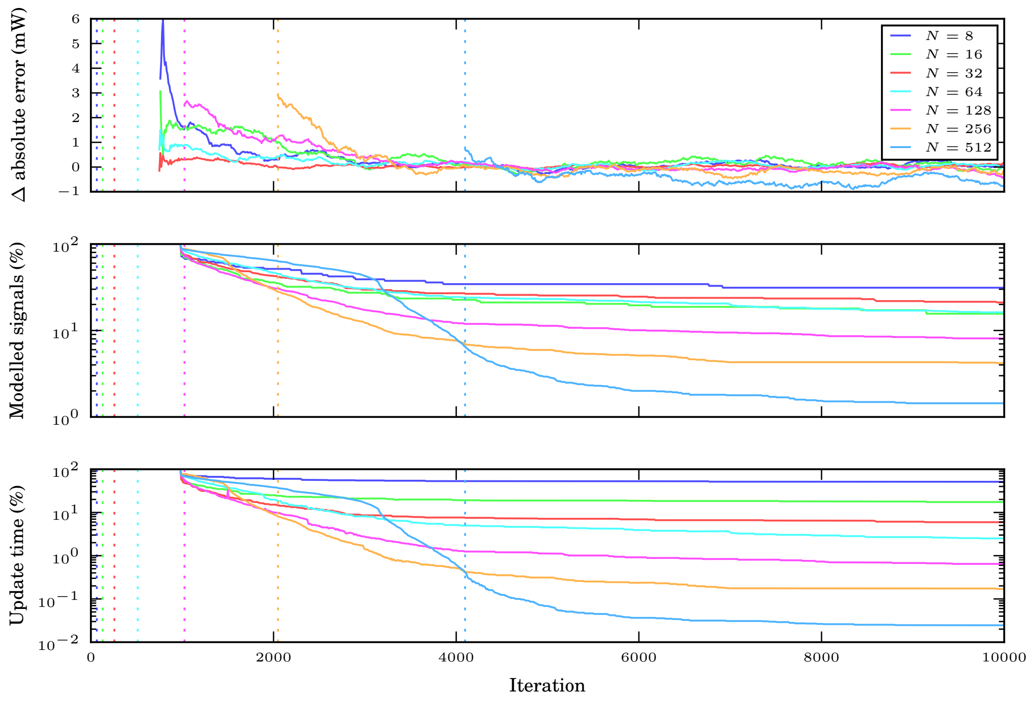

Fig. 17: Deteriorations in mean per-module absolute error by counters per module N with counter width $W = 9$ for arithmetic-module system under signal pruning.

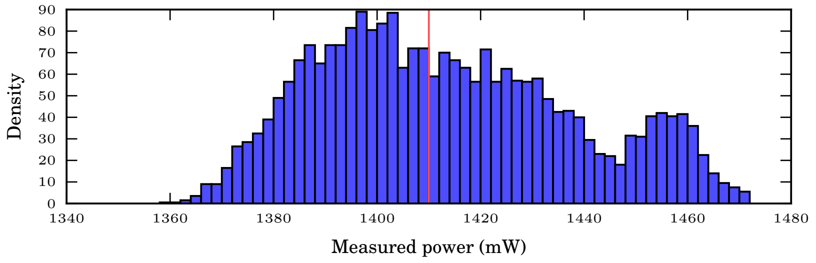

Fig. 18: System power for all task $\rightarrow$ hardware mappings, with 2mW bins, used in the task-mapping experiment. The red line marks the median.

was necessary since the modules were structurally different and, thus, had counters attached to different signals. The model-building phase—57 updates in this case, since $M = 7$ and $N = 8$—proceeded next, with a random task $\rightarrow$ module mapping applied during each iteration. Once building had completed, the controller used the RLS coefficients, $\hat{x}$, to exhaustively forecast the system-wide power consumption for all 7! mappings (340ms in software) using sets of $\boldsymbol{a}_m$ taken from the activity table in each subsequent iteration, the optimal of which was applied to the hardware. Activity counts and power measurements were taken as normal and used to update the model during the building and optimisation stages of the experiment.

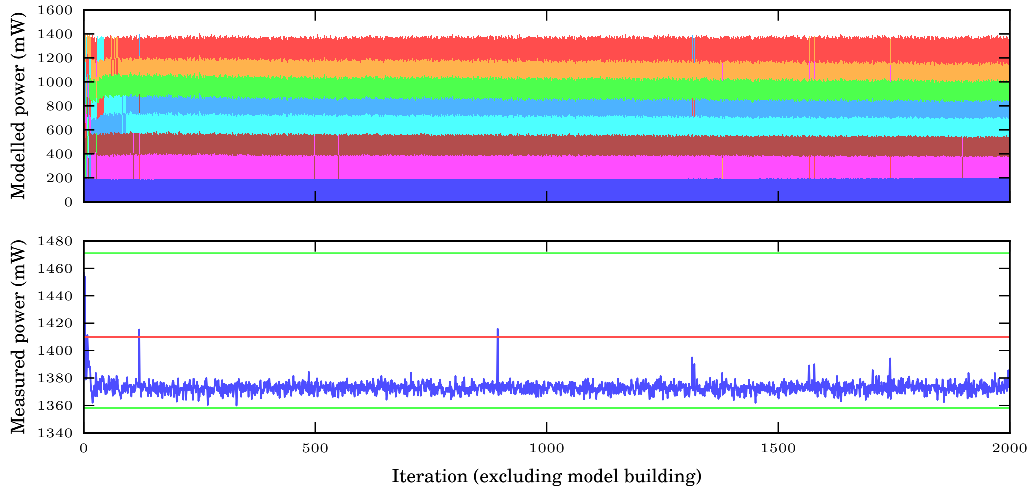

Fig. 19: Task-mapping experiment results for FIR benchmark system with $N = 8$, $W = 9$. The modelled power chart shows the estimated breakdown during each iteration, with per-module powers arranged in order above static power, shown in dark blue. Each additional colour represents the task mapped to the related module in each iteration. The measured power plot includes lines for the minimum, maximum (both green) and median (red) system powers.

Figure 19—a zoomed version of which can be found in Figure 20—shows that, for around the first 100 iterations, the model continued to adapt before converging on what it believed to be the optimal mapping. In this case, the controller was able to find a mapping that consumed 1373mW of power on average: 37mW (2.6%) lower than the median—whence the best possible improvement was 52mW (3.7%)—and 98mW (6.7%) lower than the worst case. Note that averaging was not applied to these results.

10. CONCLUSION

In this article, we presented KAPow, a technique that combines hardware instrumentation with software system identification to estimate per-module breakdowns of power consumption within FPGA designs. Our approach is based around the monitoring of influential signals within each module, determined at compile-time through vectorless power analysis. We demonstrated that low-overhead activity-counting logic can be mapped automatically, transparently and elegantly to FPGAs.

Rather than estimating system-wide power consumption, as in prior work, we measured it directly and applied system identification to train and continuously update a linear power model online. Once running, the model adapts to changes in operating conditions in ways that offline models cannot. Our approach facilitates runtime power optimisation without having to pre-characterise individual devices or applications and greatly improves upon the accuracy achievable with a one-size-fits-all model. Furthermore, the ability to establish power consumption breakdowns provides the foundation for a new field of runtime performance optimisation techniques, allowing challenges including process variation and dark silicon to be addressed.

We evaluated KAPow on two multi-module systems to establish its accuracy, adaptability and suitability to inform runtime task mapping for system-wide power optimisation. Absolute module-level power estimates were shown to be accurate, within

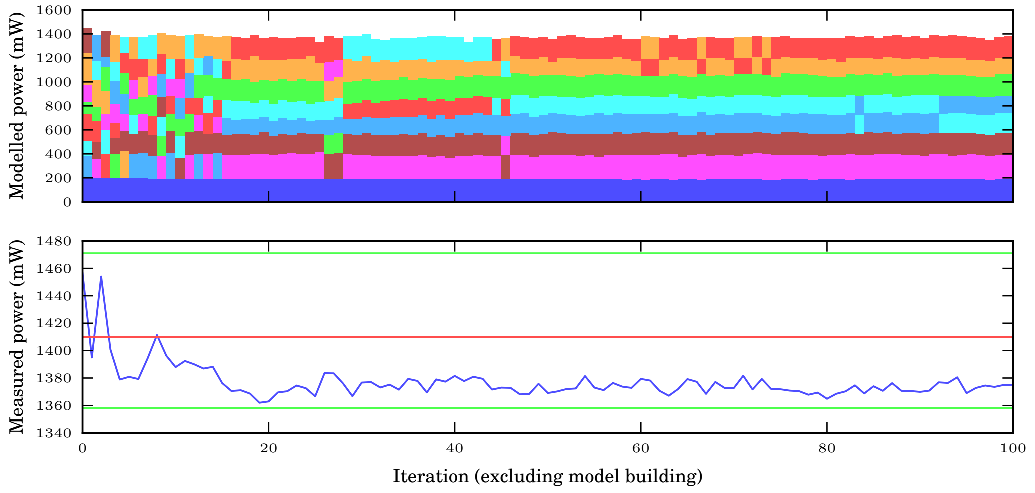

Fig. 20: Task-mapping experiment results, zoomed to show the first 100 iterations only.

5mW of true measurements and thus meeting the limits of measurement accuracy within our experimental setup. We proposed a simple signal pruning algorithm capable of dramatically reducing the number of signals modelled at runtime while having no discernible impact on accuracy. In our task-mapping experiment, power consumption improvement of up to 7% was achieved with incurred power and area overheads of sub-4% and 9% (or $\sim$2% of the device), respectively.

KAPow's largest limitation at present—and, we believe, the most promising avenue for future work—lies in signal selection. Choosing signals to monitor based solely upon vectorless analysis-predicted activity ignores the fact that some may not be indicative of high power consumption and that others may be highly correlated. As shown by our signal pruning experiments, the former is indeed a limitation with our methodology at present: many counters can often be eliminated with little effect on model accuracy.

10.1. Future Work

In the future, we would like to experiment with different signal selection methods, including vectored (simulation-based) approaches [Najm 1994] and by analysing circuit structure using centrality techniques [Hung and Wilton 2013]. We will also evaluate the use of asynchronous counters to explicitly capture glitches and explore methods enabling the reduction of instrumentation overhead. For example, it may be possible to specify that instrumentation be given lower priority over FPGA resources than the user circuit, perhaps by using incremental compilation techniques to constrain it to leftover resources alone [Levine et al. 2012]; alternatively, by building custom tools to employ latency-insensitive design techniques [Hung et al. 2014] that may allow activity counters to respond a few clock cycles late, for example, if they can improve overall performance. Finally, we envisage that a higher-level runtime management layer, in place of the simple controllers used in this work, may allow estimates from KAPow to be combined with application-specific parameters—kernel selection and achieved output accuracy for our FIR filter system, for instance—to enable finer-grained control.

ACKNOWLEDGMENTS

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

Altera. 2015. Cyclone V SoC Development Board – Reference Manual. `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/rm_cv_soc_dev_board.pdf`. (2015).

Altera. 2016. Stratix: High-performance ALM and Interconnect. `https://www.altera.com/products/fpga/features/stx-architecture.html`. (2016).

J. H. Anderson and F. N. Najm. 2004. Power Estimation Techniques for FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 12, 10 (2004), 1015–1027.

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

C. F. Gauss. 1821. *Theoria Combinationis Observationum Erroribus Minimis Obnoxiae*.

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

E. Hung, T. Todman, and W. Luk. 2014. Transparent Insertion of Latency-oblivious Logic onto FPGAs. In *International Conference on Field-programmable Logic and Applications (FPL)*. 1–8.

E. Hung and S. J. E. Wilton. 2013. Scalable Signal Selection for Post-silicon Debug. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21, 6 (2013), 1103–1115.

E. Hung and S. J. E. Wilton. 2015. Zero-overhead FPGA Debugging. *Reconfigurable Logic: Architecture, Tools, and Applications* 48 (2015), 71–96.

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

J. Lamoureux and S. J. E. Wilton. 2006. Activity Estimation for Field-programmable Gate Arrays. In *International Conference on Field-programmable Logic and Applications (FPL)*. 1–8.

D. Lee, T. Kim, K. Han, Y. Hoskote, L. K. John, and A. Gerstlauer. 2015. Learning-based Power Modeling of System-level Black-box IPs. In *International Conference on Computer-aided Design (ICCAD)*. 847–853.

J. M. Levine, E. Stott, G. A. Constantinides, and P. Y. K. Cheung. 2012. Online Measurement of Timing in Circuits: For Health Monitoring and Dynamic Voltage & Frequency Scaling. In *IEEE International Symposium on Field-programmable Custom Computing Machines (FCCM)*. 109–116.

Linear Technology. 2009. LTC2978: Octal Digital Power Supply Manager with EEPROM. `http://cds.linear.com/docs/en/datasheet/2978fd.pdf`. (2009).

L. Ljung. 1998. *System Identification*. Birkhäuser, 163–173.

K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz. 2013. Titan: Enabling Large and Complex Benchmarks in Academic CAD. In *International Conference on Field-programmable Logic and Applications (FPL)*. 1–8.

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

F. N. Najm. 1994. A Survey of Power Estimation Techniques in VLSI Circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 2, 4 (1994), 446–455.

R. L. Plackett. 1950. Some Theorems in Least Squares. *Biometrika* 37 (1950), 149–157.

E. Stott, Z. Guan, J. M. Levine, J. S. J. Wong, and P. Y. K. Cheung. 2013. Variation and Reliability in FPGAs. *IEEE Design Test* 30, 6 (2013), 50–59.

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

Xilinx. 1996. Efficient Shift Registers, LFSR Counters, and Long Pseudo-random Sequence Generators. `http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf`. (1996).

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

Received 0000; revised 0000; accepted 0000