KAPow: A System Identification Approach to Online Per-module Power Estimation in FPGA Designs

Eddie Hung, James J. Davis, Joshua M. Levine, Edward A. Stott,
Peter Y. K. Cheung and George A. Constantinides
Department of Electrical and Electronic Engineering
Imperial College London, London, SW7 2AZ, United Kingdom
E-mail: {e.hung, james.davis06, josh.levine05, ed.stott, p.cheung, g.constantinides}@imperial.ac.uk

***Abstract*—In a modern FPGA system-on-chip design, it is often insufficient to simply assess the total power consumption of the entire circuit by design-time estimation or runtime power rail measurement. Instead, to make better runtime decisions, it is desirable to understand the power consumed by each individual module in the system. In this work, we combine board-level power measurements with register-level activity counting to build an online model that produces a breakdown of power consumption within the design. Online model refinement avoids the need for a time-consuming characterisation stage and also allows the model to track long-term changes to operating conditions. Our flow is named KAPow, a (loose) acronym for 'K'ounting Activity for Power estimation, which we show to be accurate, with per-module power estimates as close to ±5mW of true measurements, and to have low overheads. We also demonstrate an application example in which a per-module power breakdown can be used to determine an efficient mapping of tasks to modules and reduce system-wide power consumption by over 8%.**

1. Introduction

In a world increasingly dominated by system-on-chip (SoC) designs, power efficiency is of ultimate concern due to the dark silicon effect: 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 system may be produced where, due to variation, some modules are more power-efficient than others. An intelligent, self-aware system might independently control the power consumption of each module 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 design-time, but we cannot determine such a breakdown online.

1.1. Per-module Online Power Modelling

While power measurement at V_{dd} pins is common, manufacturing SoCs with per-module power domains is usually impractical due to increased metal and pad costs, particularly for a configurable technology such as the FPGA. A more feasible approach is to instead monitor the switching activity within each module, since switching is a key indicator of dynamic power. Models that forecast power consumption based on predicted switching activity are well established for use at design-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, what is needed is a means to calculate a runtime power breakdown without relying on a stale model.

Figure 1 illustrates the benefits of an online, activity-based power model—described in this paper—used to estimate power consumption. The plot shows the error between

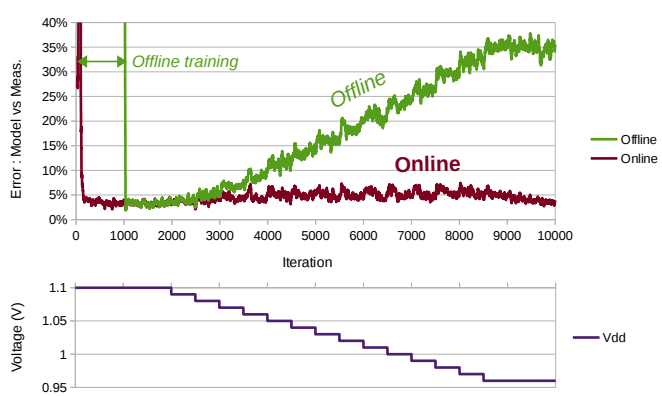


Figure 1: Error accumulations in online- vs offline-generated signal activity-to-power models under voltage scaling

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 (OLS). As the operating conditions deviate from their nominal values, the error in the offline power model increases, while the online model quickly adapts. The effects of different classes of input data, operating modes and exogenous conditions, such as temperature, voltage and degradation, can all be captured online, with the resulting model being far more useful for runtime control in a dynamic environment than its offline counterpart. Note that while Figure 1 is included here for illustrative purposes, the system its data was obtained from is that described in Section 6, run on the platform explained in Section 5 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 carry out infrequent incremental model updates and optimise the computationally intensive tasks that run in soft logic.

In this paper, we describe the first runtime modelling framework providing a per-module power breakdown in an FPGA system, making the following novel contributions:

- We describe an automated tool flow capable of instrumenting arbitrary HDL for runtime monitoring of signal activity.
- We apply system identification techniques to allow an activity-to-power model for any SoC design to be trained and updated at runtime.
- We experimentally quantify the relationships between model accuracy and the incurred hardware and software overheads.
- We show how knowledge of per-module behaviour can facilitate the power-efficient mapping of tasks to hardware, leading to an over-8% power reduction.

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, however dynamic power is the only consideration required for a per-module breakdown as it is relevant to runtime decisions regarding clock frequency and scheduling. Equation 1 describes the relationship between dynamic power P_{dynamic}, switching activity α_i on a net i, operating frequency f, intrinsic capacitance C_{i} and supply voltage V_{dd}.

$$P_{\text{dynamic}} = \sum_i \alpha_i f C_i V_{\text{dd}}^2 \tag{1}$$

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 [1]; the latter style has been further applied to FPGAs to facilitate power-aware compilation [2]. As the complexity of FPGA applications has increased, so too has the need for high-level power estimation models for modular systems [3]. 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 [4] proposed the evaluation of system-wide power consumption through a dynamic power monitoring approach using activity counters. Therein, signals were automatically selected for activity counting and an offline model was developed through simulation to relate these activity counts to power: realtime activity measurements facilitated runtime power estimation. Because the model was statically trained, however, the results were found to have $\pm 15\%$ error. Moreover, observation only of 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.

3. Tool Flow

Our automated tool flow comprises two main stages: signal selection, which identifies nets likely to be correlated with high 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 at rising clock edges on these signals, rather than those introduced asynchronously by glitches; the online model we use will, however, automatically tune itself to add more weight to signals that are prone to glitching. We target systems composed of multiple modules (IP blocks, typically hardware accelerators) described in HDL 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 HDL in order to determine signals of interest before augmenting it with activity counters. Two issues exist with this method: firstly, estimating the power consumption of individual signals based solely on behavioural HDL can be inaccurate since neither physical resources nor the data that exercise them are considered [3] and, secondly, augmentation of the HDL with instruments would likely cause circuit mapping to be different, thereby altering the very thing one wishes to observe. The latter problem is similar to that faced when inserting timing [5] or debugging infrastructure [6], both of which are even more sensitive to any circuit perturbations. Our flow, shown in Figure 2, overcomes these issues to

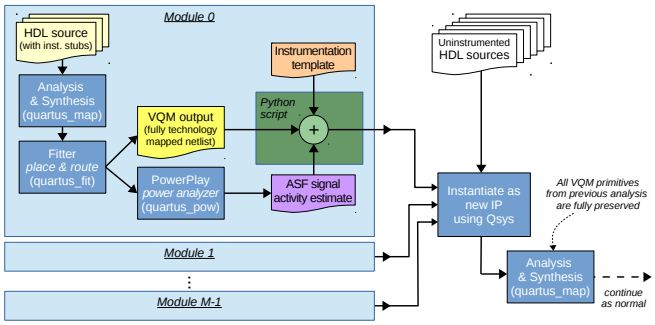


Figure 2: KAPow tool flow

some extent by performing signal selection and inserting instrumentation using placed and routed netlists.

Some minor preprocessing must be performed on each of the M modules to be instrumented before compilation: each is modified to accommodate eight additional words within its address space for instrumentation control. Following this, the modules' HDL is compiled and the resultant netlists extracted as Verilog Quartus Mapping (VQM) files [7]. In order to identify the signals that best indicate power consumption, each module is fed through Altera's design-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 HDL within QSys. Finally, the system is compiled as normal: importantly, all VQM primitives are fully preserved.

3.1. Signal Selection

The purpose of signal selection is to identify nets in the design that are strongly indicative of their modules' power consumption. We do this by using PowerPlay to report nets with high estimated switching activity, since these should consume the most power. Altera report that PowerPlay's power estimates are accurate to within $\pm 20\%$ of true measurements when provided with realistic activity data [8]. The tool makes power consumption estimates by summing the static and dynamic power components contributed by all logic and routing resources based on a detailed database of device characteristics. Dynamic power components are scaled using relevant clock frequencies and switching forecasts 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 [9]. A vectorless estimation starts from primary inputs and propagates switching rates through all nets by considering the Boolean functions that connect them.

Although vectorless estimation is known to be less accurate than its vectored counterpart, this is the method we adopt 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 accuracy of those signals' estimated activities

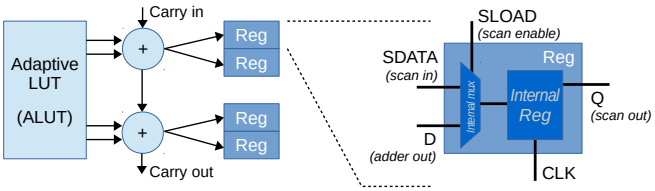


Figure 3: Simplified view of a Cyclone V ALM

is essentially irrelevant as they will be measured at runtime. Thus, the only important factor at this stage is whether they are likely to have a high impact on power consumption. The tool produces a file containing switching rates for all signals, from which we select the N—a user-selectable parameter—most frequently toggling.

We note that 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 design-time this is not important since the coefficients associated with these signals will be determined and tuned at runtime.

3.2. Instrumentation

At the heart of any instrumentation that counts synchronous events is an efficient counter structure. We use linear feedback shift register (LFSR)-based counters because they are smaller and faster than arithmetic counters: 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. Outputs are not sequential; however, since each counter has relatively few bits (justified in Section 5.2), the decoding is easy to perform in software with a lookup table.

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, as shown in Figure 3: 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. When a positive edge is detected and the instrument is enabled (`EN = 1`), the LFSR's clock enable (`CE`) is driven high, causing it to advance to its next state since `SLOAD` is held permanently high. When the scan chain is in operation (`SE = 1`), the LFSR does not shift into its next state, taking 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 wraps each instrumented module. Clock domain crossing, if required, is handled automatically. To take an activity measurement, the counters are enabled for a period of time dictated by an adjustable 'instrumentation lifetime' counter,

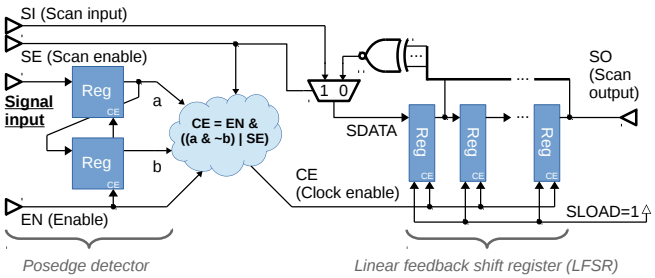


Figure 4: LFSR-based activity counter

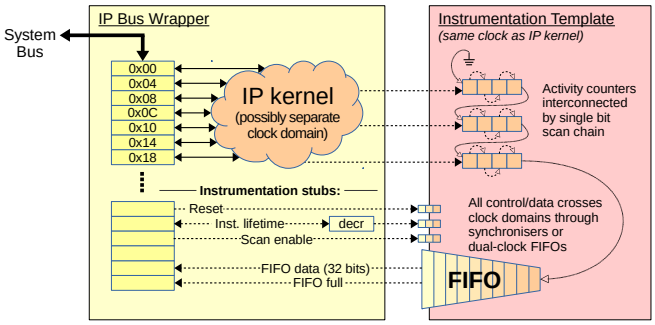


Figure 5: Instrumentation template

after which the contents of the scan chain can be read out across the system bus via the FIFO.

While we have optimised our design for a particular device family, similar activity counters and read-back infrastructure could easily 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.

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 [10]. 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 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 activities $\boldsymbol{a}$ and measured system power y, as shown in Equation 2.

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

In prior work [4], 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

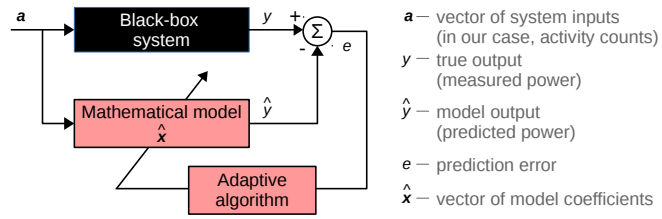


Figure 6: System identification overview

sense to find $\hat{\boldsymbol{x}}$, which remained fixed during runtime. By contrast, the identification algorithm we use is recursive least squares (RLS) [11], which iteratively updates $\hat{\boldsymbol{x}}$ as new measurements arrive. The RLS update function is of the form $\hat{\boldsymbol{x}}[t] = \hat{\boldsymbol{x}}[t-1] + \mathrm{f}(e[t],\ \boldsymbol{k}[t])$, 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 weights.

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

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

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

With a linear model, it is straightforward to compute the power contribution of each module m, $\hat{y}_m$, since the model's coefficients can be partitioned 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 'input' 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 is included within the respective modules' estimates since their own switching is dictated by the behaviour of the module to which each is connected.

$$\boldsymbol{a} = [a_{\mathrm{s}}\ \boldsymbol{a}_0\ \boldsymbol{a}_1\ \ldots\ \boldsymbol{a}_{M-1}] \tag{6}$$

$$\hat{\boldsymbol{x}} = [\hat{x}_{\mathrm{s}}\ \hat{\boldsymbol{x}}_0\ \hat{\boldsymbol{x}}_1\ \ldots\ \hat{\boldsymbol{x}}_{M-1}] \tag{7}$$

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

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

5. Errors and Overheads

Experiments were performed to investigate how model accuracy and overheads change as we modify parameters N, the number of activity counters per module, and W,

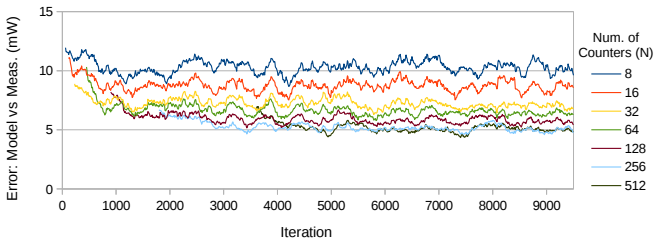


Figure 7: Relative error by counters per module N

the width of each counter in bits. The system used to obtain the results in this section was the seven-module design described in detail in Section 6. We targetted the Altera Cyclone V SX SoC development board 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—one each for the CPU cores and FPGA—which we used for taking runtime power measurements.

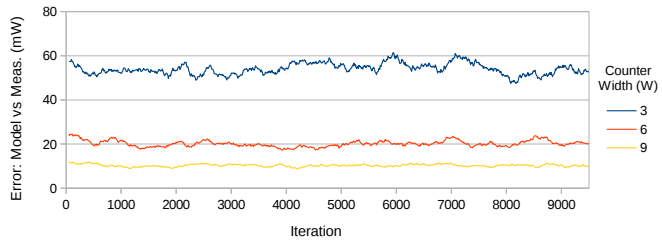
System identification was implemented entirely in software on the SoC's hard CPU cores, clocked at their default 925MHz and running Ubuntu 14.04. Communication with hardware modules and their activity counters was accomplished through memory-mapped registers accessed from Linux using `mmap()`. Experimentation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

5.1. Error by Number of Activity Counters

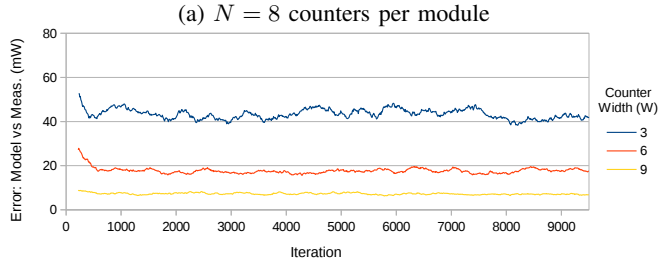
Figure 7 shows how the absolute error between estimated and measured power consumption varies with the number of activity counters, N, used per module with counters each $W = 9$ bits wide. Each point is an average of errors across the system's seven modules, with 32-point windowing applied in order to reduce noise and highlight the models' trends. In all cases in Figure 7 (and Figures 8 and 11), 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 $\hat{\boldsymbol{x}}$ 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: errors accurate to $\pm$10mW were seen with small N (8), while $\pm$5mW was achievable with larger values ($\geq$ 256).

5.2. Error by Counter Width

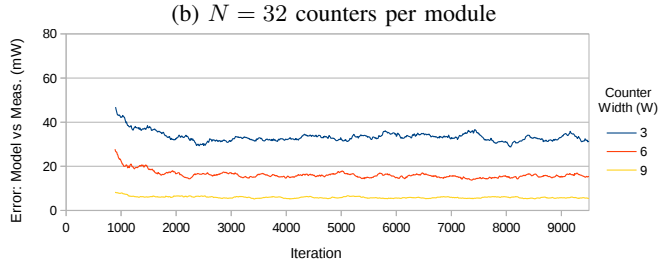
Figure 8 shows how the relative error varies with the counter width, W, with instrumentation lifetime equal to the number of cycles, $2(2^W - 2)$, needed to maximise dynamic range while guaranteeing no overflow. Analysis across the range of N tested shows that the choice of counter width has a significant effect on steady-state error, but not necessarily



(a) $N = 8$ counters per module



(b) $N = 32$ counters per module



(c) $N = 128$ counters per module

Figure 8: Relative error by counter width W

the rate of convergence. These results indicate that reducing N has a smaller impact on error than decreasing W by the same factor to achieve a similar area gain.

5.3. Hardware and Design Overheads

Figure 9 shows the overheads in area (ALMs and logic array blocks (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 appreciates with N, with $N = 512$ proving especially difficult to compile since it approached the limits of device capacity. The compilation time figure excludes the initial VQM generation because this would be integrated into a single pass in a fully developed tool flow. System power consumption increased with N, reflecting the disturbances and wire length increases created by intrusively attaching activity counters inside each module. These results indicate that $N = 8$ and $W = 9$ offer a reasonable tradeoff between relative error ($\pm$10mW) for area (ALM) and power overheads of 9% and 3.6% (53mW), respectively.

The total power overhead measurements quoted 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 $W = 9$ was found to increase by

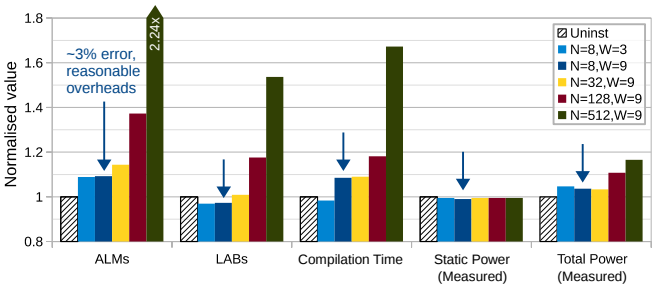


Figure 9: Area, compilation time and power overheads

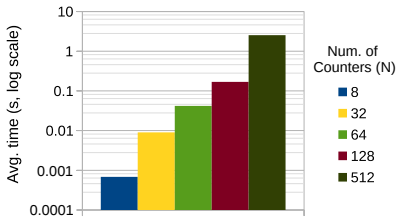


Figure 10: System identification execution time

3.7%, from 1.70W to 1.76W. This measurement was found to be unaffected by changes in N.

5.4. Software Overhead

The complexity of the RLS algorithm across various N is captured in Figure 10, the values of which represent the times needed to update the model during each iteration. With $N = 8$, each system-wide update required 0.6ms to complete, representing around 5% of total CPU time at the maximum sampling rate of 91Hz.

6. Power Breakdown

A system was developed upon which KAPow's estimation of module-wise power breakdowns could be evaluated. A SoC implementation containing seven functionally identical (in terms of throughput and latency) FIR filter modules was devised, with each module instrumented as described in Section 3. Heterogeneity was introduced by forcing each module 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 uninstrumented system occupied 24% of the available ALMs, spread out over 62% of the FPGA's LABs, along with 94% of block RAM and all DSPs. All modules met timing at 200MHz, however each was independently clocked by a runtime-adjustable PLL, allowing frequency to be changed dynamically on a per-module basis. 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. $N = 8$ activity counters were used per module, each $W = 9$ bits wide; this remained true for all later-described experiments as well.

During each experimental iteration, a different system workload was applied: a clock frequency, one of nine input data sets and one of ten groups of coefficients were selected at random for each filter module. Activity and system-wide power measurements were also taken in each iteration and used to update the model. A true power breakdown was established for comparison on every fifth iteration by taking a system-wide power measurement and then successively clock-gating single modules and repeating power measurements, from which per-module dynamic and system static power consumptions were derived.

The scatter plots in Figure 11 show the close correlations obtained between modelled and true per-module power consumptions. Each data point represents an experimental iteration with an associated power breakdown measurement. Once sufficient training time (750 iterations) 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 implementations: the best (module 4) achieved a mean error of 8.3mW while the worst (module 6) was 12.9mW. For all of the modules, the mean error was small compared to the range in power consumed over their different operating modes. Static power tracking was particularly good, achieving mean absolute error of 4.0mW.

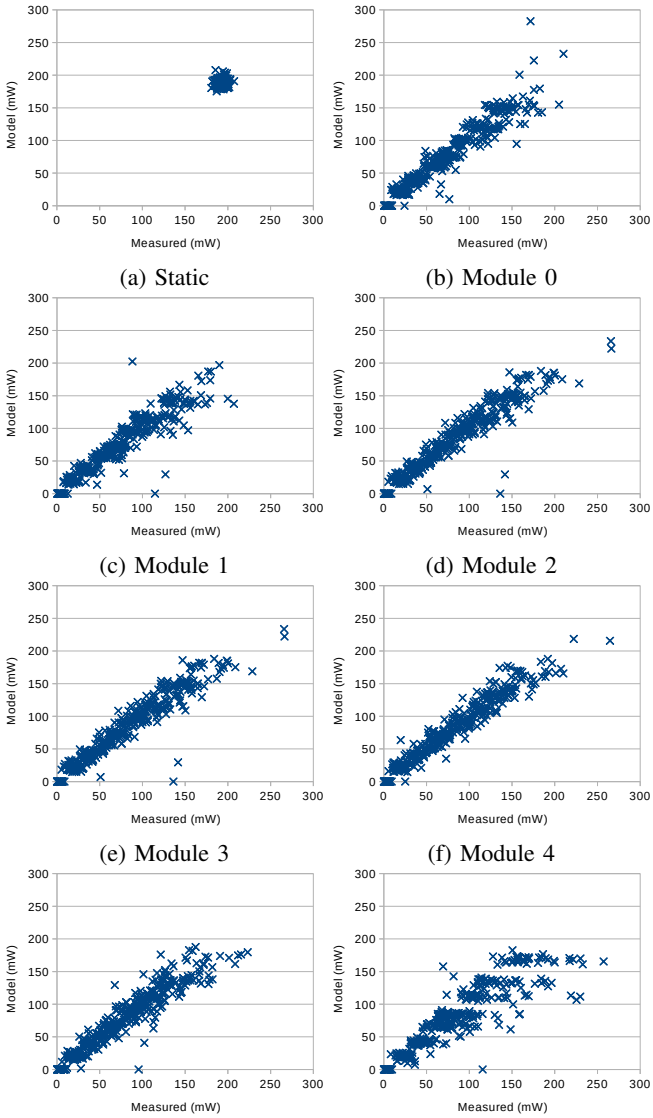
Repeatability experiments on the power breakdown measurements showed deviations in the $\pm$5mW range: the mean absolute measurement noise was 3.9mW, with outliers as large as 14.0mW. The observed model errors were therefore not significantly greater than measurement noise, indicating that the model accuracy approached the limits of what could be measured with our test setup.

7. Static Power Compensation

An experiment was performed to verify KAPow's ability to compensate for changes in static power consumption, the key determiner of which is temperature. We used the system described in Section 6 and a temperature control rig consisting of a thermoelectric effect heat pump, water cooler and resistance thermometer, allowing the setting and maintaining of device temperature over a wide range. Random workload changes were applied as described in Section 6, with static power estimated using our model and measured after clock-gating all modules. Initial training lasted for 300 iterations, with temperature held at 25°C. Following this, the temperature was increased by 5°C every 150 iterations until it reached the device's upper temperature corner of 85°C. The results of the experiment are shown in Figure 12, which demonstrates close tracking between model-predicted and measured static power consumption. No correlations were seen between dynamic power estimates and temperature.

8. 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. Seven different filtering tasks (input data and coefficients), representing a range



(a) Static (b) Module 0 (c) Module 1 (d) Module 2 (e) Module 3 (f) Module 4 (g) Module 5 (h) Module 6

Figure 11: Per-module power estimates versus true values

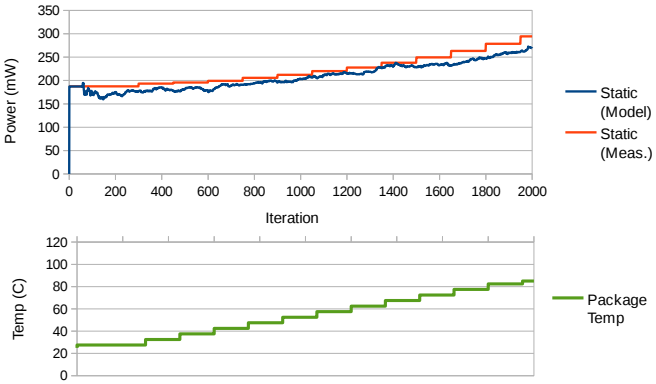


Figure 12: Static power tracking within power breakdown during temperature sweep

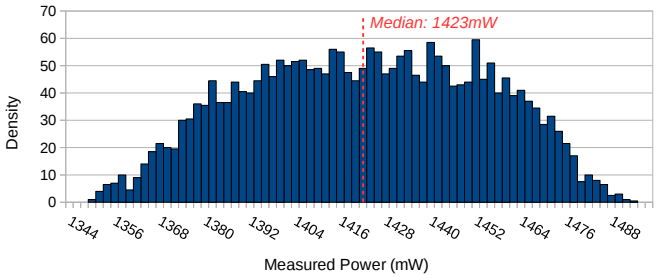


Figure 13: System power for all task → hardware mappings

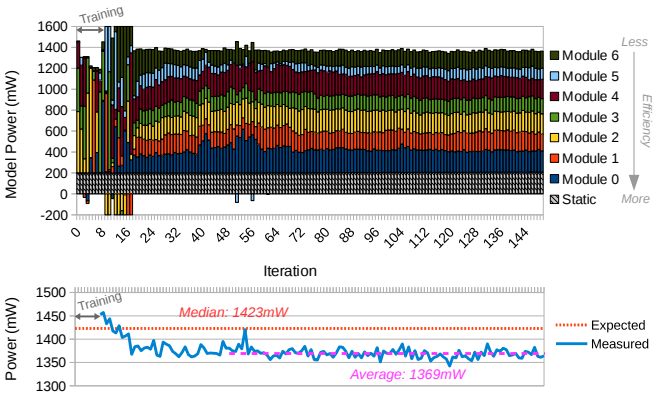


Figure 14: Task mapping power breakdown

of complexities and, consequently, power behaviour, were specified to be executed simultaneously by the seven-module system described in Section 6. Recall that the modules are functionally identical but implementationally different, thus the task → module mapping chosen will influence the system-wide power consumption. Clock frequency remained fixed at 200MHz. The set of all possible mappings contained ${}^7P_7 = 7! = 5040$ permutations; a histogram (with 2mW bins) of their power consumptions is given in Figure 13. This data suggests that an arbitrary mapping will result in system-wide power consumption of 1420mW, with best and worst cases of 1350 and 1500mW, respectively.

Figure 14 shows the results of an experiment run using a simple closed-loop controller that attempted to minimise total power consumption. Training completed during the first seven iterations consisted of the rotation of tasks across the seven modules, initialising a 7×7 activity table. In each subsequent iteration, activity counts and power measurements were taken as normal to update the model. The controller then used its new coefficients, $\hat{x}$, to exhaustively forecast the system-wide power consumption for all 7! mappings (340ms in software), the optimal of which was applied to the hardware. Figure 14 shows that, for around the first 50 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 1369mW of power on average: 54mW (3.8%) lower than the median—whence the best possible improvement is 75mW (5.3%)—and 126mW (8.4%) lower than the worst case, after accounting for the overhead of instrumentation.

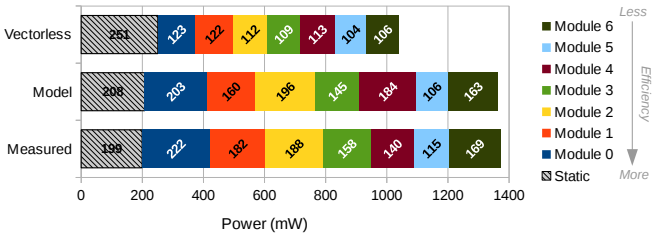


Figure 15: Vectorless, modelled and true power breakdowns

Finally, Figure 15 provides side-by-side comparisons of compile-time vectorless power forecasts from PowerPlay, true measurements and runtime estimates from KAPow. Measurements and runtime estimates are a snapshot taken during the experiment's 150th iteration. Comparing the 'Vectorless' and 'Measured' bars, we observe that vectorless estimation predicts approximately equal power behaviour across the modules, while measurement reveals significant variation. Looking now at the 'Model' bar, it can be seen that our online modelling is much closer to the measured data: KAPow successfully accounts for implementational and operational differences not foreseen by vectorless analysis.

9. Conclusion

In this paper we presented KAPow, a flow that combines hardware instrumentation with software system identification techniques to estimate per-module breakdowns of power consumption within FPGA-SoC applications. 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 can unburden vendors from the costly expense of having to pre-characterise a custom power model to fit individual devices and applications, and greatly improves on the accuracy achievable with a one-size-fits-all model. Furthermore, the ability to establish power consumption breakdowns unlocks a whole field of runtime performance optimisation techniques, allowing challenges including process variation and dark silicon to be addressed.

We evaluated KAPow on a multi-module system to establish its accuracy, adaptability and suitability to inform runtime task mapping for system-wide power optimisation. Module-level power estimates were shown to be accurate to as low as $\pm$5mW of true measurements: of the order of measurement noise within our experimental setup. In our task mapping experiment, power consumption improvement of over 8% was achieved for a 9% area overhead (or $\sim$2% of the device) and worst-case power overhead of sub-4%.

Early indications show that the presented accuracy holds for arbitrary circuits. The largest limitation at present—and, we believe, most promising avenue for future work—lies in signal selection. Selecting signals based solely upon vectorless analysis-predicted activity ignores the fact that some may be highly correlated. This is indeed a limitation with our current flow: often, several counters can be eliminated with no effect on the accuracy of the model.

In the future, we would like to experiment with different signal selection methods, including vectored (simulation-based) approaches [1] and by analysing circuit structure using centrality techniques [12]. We will also evaluate the use of asynchronous counters to explicitly capture glitches. 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-level parameters 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

[1] 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, 1994.

[2] J. H. Anderson and F. N. Najm, "Power Estimation Techniques for FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 10, 2004.

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

[4] 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.

[5] J. M. Levine, E. Stott, G. A. Constantinides, and P. Y. K. Cheung, "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)*, 2012.

[6] E. Hung and S. J. E. Wilton, "Zero-overhead FPGA Debugging," *Reconfigurable Logic: Architecture, Tools, and Applications*, vol. 48, 2015.

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

[8] Altera, "Quartus II Handbook Volume 3: Verification," http://www.altera.com/literature/hb/qts/qts_qii5v3.pdf.

[9] J. Lamoureux and S. J. E. Wilton, "Activity Estimation for Field-programmable Gate Arrays," in *International Conference on Field-programmable Logic and Applications*, 2006.

[10] L. Ljung, "System Identification," in *Signal Analysis and Prediction*, ser. Applied and Numerical Harmonic Analysis. Birkhäuser, 1998.

[11] MathWorks, "Adaptive Filters – MATLAB & Simulink," http://uk.mathworks.com/help/dsp/adaptive-filters.html.

[12] E. Hung and S. J. E. Wilton, "Scalable Signal Selection for Post-silicon Debug," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, 2013.