Adaptive Energy Minimization of Embedded Heterogeneous Systems using Regression-based Learning

Sheng Yang[1], Rishad A. Shafik[1], Geoff V. Merrett[1] , Edward Stott[2], Joshua M. Levine[2], James Davis[2]& Bashir M. Al-Hashimi[1]
[1]School of ECS, University of Southampton, UK [2]Department of EEE, Imperial College, UK
E-mail:[1]{sy2u12,ras1n09,gvm,bmah}@ecs.soton.ac.uk, [2]{edward.stott07,josh.levine,james.davis06}@imperial.ac.uk.

***Abstract*—Modern embedded systems consist of heterogeneous computing resources with diverse energy and performance trade-offs. This is because the computing resources exercise the application tasks differently, generating varying workloads and energy consumption. As a result, minimizing energy consumption in these systems is challenging as it requires continuous adaptation of application task mapping (i.e. allocating tasks among the computing resources) and dynamic voltage/frequency scaling (DVFS). Existing approaches lack such adaptation with practical validation (Table I).**

This paper proposes a novel adaptive energy minimization approach for embedded heterogeneous systems. Fundamental to this approach is a runtime model, generated through regression-based learning of energy/performance trade-offs between different computing resources in the system. Using this model, an application task is suitably mapped on a computing resource during runtime, ensuring minimum energy consumption for a given application performance requirement. Such mapping is also coupled with a DVFS control to adapt to performance and workload variations. The proposed approach is designed, engineered and validated on a Zynq-ZC702 platform, consisting of CPU, DSP and FPGA cores. Using several image processing applications as case studies, our proposed approach can achieve significant energy savings (70% in some cases, i.e. from 43mJ per frame to 13 mJ per frame), when compared to existing approaches.

***Index Terms*—Energy efficiency; dynamic voltage/frequency scaling; runtime optimization; linear regression.**

I. Introduction

Energy efficiency continues to be a prime design objective in embedded systems [1]. To facilitate energy reduction these systems are equipped with dynamic voltage/frequency scaling (DVFS) capabilities, examples include ARM's VFS firmware [2] and Linux's power governors [3]. With such capabilities the operating voltage/frequency (V/F) can be suitably chosen at runtime to reduce energy [4]–[6].

To accommodate a broader range of applications, modern embedded systems consist of diverse computing resources on a single chip. These include, for example, CPUs for general purpose computing, DSPs for signal processing tasks and GPUs/FPGAs for hardware acceleration of compute-intensive algorithms. Since these resources are architecturally distinct from each other, they exhibit different performance and energy trade-offs for a given task [5], [7]. Hence, to minimize the energy effectively, the energy/performance trade-offs between these resources must be learned to suitably control the application task mapping (i.e. allocating a task to a computing resource) [7]. Moreover, to adapt to dynamic workload and application performance variations, such control will need to be coupled with DVFS [8].

Over the years, researchers have proposed various approaches for energy-efficiency in embedded systems, as shown in Table I. Many of these approaches have employed DVFS controls alone, using feedback from performance counters during runtime, or the task information known apriori. For example, Pallipadi [3] proposed a DVFS control approach for maximizing CPU usage. When the CPU usage decreases, the operating frequency (and also the associated operating voltage) is lowered in steps. Conversely, when the CPU usage increases, the operating frequency is increased to the maximum point. Nabina and Nunez-Yanez [8] showed a DVFS approach for FPGA-based video motion compensation engines. Their approach uses application-specific requirements to monitor the available timing margins per motion compensation task and control the operating voltage/frequency accordingly. Luo and Jha [9] showed another DVFS (and task scheduling) approach for energy-efficiency in heterogeneous real-time systems. The DVFS and task scheduling controls are established through task graph representations of the target applications with known task execution times. Using these task graphs, slack times are minimized through simulating annealing in a multiprocessor system with variable supply voltages.

TABLE I
Existing design approaches and the proposed approach

Approach	*Adaptive?*	*Validation*	*Key Method*
[3]	Partially	Practical implementation	Runtime optimization + CPU only DVFS
[4]	No	Simulations	Offline optimization + task mapping and DVFS
[5]	No	Simulations	Runtime optimization + task mapping and DVFS
[7]	No	Practical implementation	Offline optimization + task mapping and DVFS
[8]	Partially	Practical implementation	Runtime optimization + FPGA only DVFS
[9]	No	Simulations	Offline optimization + CPU only scheduling and DVFS
[10]	No	Simulations	Offline optimization + task mapping and DVFS
[11]	No	Simulations	Offline optimization
[12]	No	Simulations	Offline optimization + task mapping and DVFS
Proposed	Yes	Practical implementation	Runtime model based task mapping and DVFS optimization

To minimize energy effectively in a heterogeneous system, researchers have also proposed approaches considering joint optimization of task mapping and DVFS. Qiu and Sha [10] presented one such approach for heterogeneous real-time systems. The approach formulates the energy minimization problem using task graph representations of the given applications with given timing constraints. This is then followed by optimization of probabilistic task mapping and DVFS controls through offline heuristics. Goh *et al.* [4] presented another approach using heuristic algorithm-based application task mapping. For each task mapping control, an energy-gradient-based algorithm selects the DVFS controls with an aim to achieve energy-efficiency. Their approach is validated through synthetic application task graphs in a simulated heterogeneous system. Goraczko *et al.* [5] proposed a software task

partitioning and mapping approach using a linear programming (LP)-based runtime optimization. For each task mapping option, an operating point (i.e. VFS) is pre-determined to reduce the overall energy consumption. Among others, the authors in [7], [11], [12] showed system-level approaches for application task mapping and DVFS, considering the offline profile-driven trade-offs between performance and energy consumption.

Existing approaches (Table I) have the following limitations. Firstly, the approaches [4], [5], [7], [10]–[12] use offline heuristics, which cannot minimize the energy consumption effectively due to lack of runtime adaptation to workload and application performance variations. Secondly, the other approaches [4], [5], [7], [11] lack practical implementations, which are much needed to advance the research in energy-efficient heterogeneous systems.

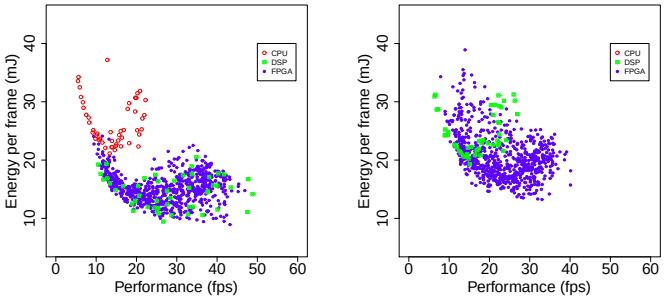
To address the limitations of the existing approaches, this paper makes the following ***contributions***:

- an adaptive energy minimization approach for embedded heterogeneous systems through runtime application task mapping and DVFS controls,
- an energy/performance model, generated through regression-based learning and runtime measurements, and
- a prototype runtime implementation, engineered and validated on a Zynq-ZC702 platform.

To the best of our knowledge, this is the first adaptive approach for energy-efficient embedded heterogeneous systems, demonstrating implementation and validations using real applications.

II. Motivation

To motivate adaptive energy minimization in heterogeneous systems, Fig. 1 shows scatter plots of measured energy (in mJ per frame) and performance (in frames per second, fps) values obtained from two image processing applications on a Zynq-ZC702 platform. Fig. 1(a) shows the energy/performance trade-offs for an image edge detection application, while Fig. 1(b) shows the same for a multi-pass image blurring application. The image processing tasks were executed separately on the platform's CPU, DSP and FPGA with available V/F operating points (see Section IV). Energy and performance values were measured directly from the on-board performance counters and power sensors. The following two *observations* can be made:



(a) Edge detection Filter (b) Multi-pass blurring Filter

Fig. 1. Scatter plot of measured energy (in mJ per frame) and performance (in fps) for the CPU (red), FPGA (blue) and DSP (green)

Observation 1: For a given implementation, the most energy-efficient operating point (i.e. minimum energy per frame) depends on the performance requirement and the chosen VFS. Referring to Fig. 1(a), both the DSP and FPGA can provide a lower lower energy consumptions than the CPU for the edge detection application when the performance requirement is between 5 fps to 35 fps. At higher performance requirements (beyond 35 fps), the DSP provides with a lower energy consumption than the FPGA. *Observation 2*: The energy/performance trade-offs vary depending on the image processing workload. Referring to Fig. 1(b), with higher image processing workloads in the blurring application, the DSP can no longer provide the most energy-efficient operating point for similar performance requirements to the image edge detection application (Fig. 1(a)). In fact, the FPGA invariably provides the lowest energy consumption for all performance requirements for the given workloads per image processing task. Note that the energy and performance values of the CPU are not shown in Fig. 1(b) due to being out-of-range.

From the above two observations, it is evident that, to achieve energy efficiency, the application task must be suitably mapped on a computing resource with careful selection of its VFS considering the energy/performance trade-offs (observation 1). Such mapping and VFS decisions should also be adapted to workload and application performance variations (observations 1 and 2). However, such adaptation requires runtime optimization over a large decision space of $\sum_{c=1}^{C} D_c$ for each task, where C is the number of computing resources and D is the number of V/F operating points available for the c-th computing resource ($1 \leq c \leq C$). Existing offline approaches, such as [4], [5], [7], [11], are infeasible for such optimization as these are highly computationally intensive. To reduce the optimization space substantially ensuring low control overheads, an accurate runtime energy/performance model is much needed, which is one of the aims of this paper. In the following section, an adaptive energy minimization approach is proposed highlighting a regression-based learning of runtime energy/performance model.

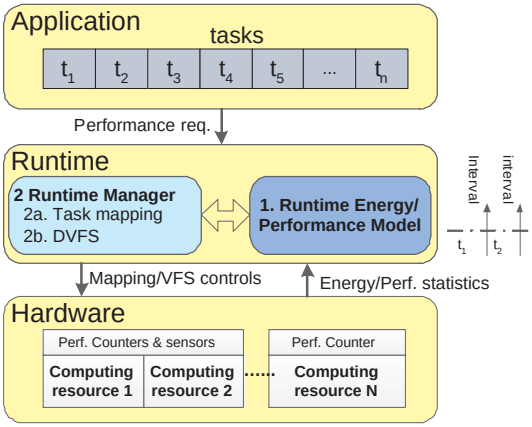


Fig. 2. Block diagram of the proposed energy minimization approach

III. Proposed Energy Minimization Approach

Fig. 2 shows a block diagram of the proposed approach, highlighting the interactions between application, runtime and hardware. The application consists of a number of computation tasks defining a specific functionality. The performance requirement of the application is communicated to the runtime layer through an application programming interface (API) as below

```
rts.set_perf(25);
```

where *rts* is a runtime variable within the API, *set_perf* sets the performance requirement as 25 frames per second, as an example. The runtime layer consists of two components: a learning-based energy/performance model and a runtime manager.

The energy/performance model is derived and learned through runtime measurements from the different computing resources in the system. This model then guides the runtime manager to carry out optimized application task mapping and DVFS controls, while meeting the application-specified performance requirements. These two components are further detailed below.

A. Learning-based Runtime Energy/Performance Model

A runtime model provides the flexibility in terms of system design, as applications' behavior can be learnt without expensive engineering efforts associated with offline profiling. Through careful design of the runtime models, high accuracy can also be achieved with little overhead (Section. IV-B and Section. IV-D). Hence, our proposed approach uses runtime model as a critical component for energy-efficient adaptation. Such a runtime model enables the prediction of the trade-offs under different operating conditions (application task mapping and VFS) and their variations. For a given computing resource, the model is learned using runtime measurements from hardware performance counters and sensors. Fig. 3 presents flowchart of learning the model using regression in 5 steps. The modeling starts by varying the operating frequencies of the computing resources (step 1). For every frame, current and latency measurements are read from the power sensors and performance counters (step 2). If the number of frames is equal to the sampling interval, the measured current and latency values are averaged over the period to produce a stable learning sample. The sample is then used to test the hypotheses in the regression process (step 3). Such sample collection and hypotheses testings are continued until the learning interval (which is a multiple of sampling interval) is reached (Section IV justifies the choice of the learning interval). After this interval, the current and latency models are generated for the given computing resource (step 4). Using these models as components the energy/performance model is then constructed (step 5).

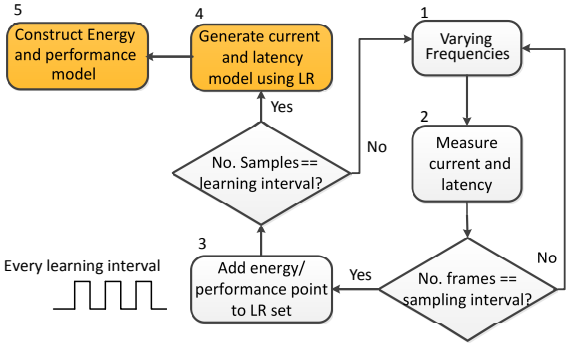


Fig. 3. Flowchart of the runtime energy/performance model generation

To set up the model, linear regression is used with an aim of establishing the relationship between the dependent variables (i.e. energy and performance) and their associated independent predictor variables (e.g. task mapping, VFS, etc.) [13]. Rigid linear regression based model is used as it provides with two advantages. First, such model requires much less training samples, and hence, results in a low runtime overhead. Second, it provides with better inference of the dependent variables, when the hypothesis is based on a reasonable assumption of the underlying property of the modeled system. Section. IV-B shows the accuracy and cross-validation results of the hypotheses of the proposed runtime model, despite being a rigid model.

The relationship is defined by a hypothesis function as:

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \Theta^T X \quad . \tag{1}$$

where x_i is a predictor, n is the number of predictors, θ_i is a fitting coefficient, X and $\Theta^T X$ are the matrix representations of x_i and θ_i. The Θ values need to be carefully chosen to minimize the mean-squared prediction error ($J(\theta)$) of the hypothesis in (1), which is given by

$$\begin{aligned} J(\theta) &= \sum_{j=1}^{m} (h_\theta(x^{(j)}) - y^{(j)})^2 \\ &= (\Theta^T X - \overrightarrow{y})^T (\Theta^T X - \overrightarrow{y}), \end{aligned} \tag{2}$$

where y is the measured value, m is number of learning samples. $J(\theta)$ is minimum when its gradient becomes 0. Hence, from (2) the gradient of $J(\theta)$ can be defined as

$$\begin{aligned} \nabla J(\theta) &= \nabla (\Theta^T X - \overrightarrow{y})^T (\Theta^T X - \overrightarrow{y}) \\ &= X^T X \Theta - X^T \overrightarrow{y}. \end{aligned} \tag{3}$$

From (3), the fitting coefficients Θ of the hypothesis in (1) can then be computed as:

$$\Theta = (X^T X)^{-1} X^T \overrightarrow{y} \quad . \tag{4}$$

From (4), it is evident that the computation complexity of the regression-based modeling is $O(n^2 \times m)$, where n is the number of predictors and m is the number of learning samples. Hence, to achieve a fast runtime model both n and m need to be small. In this work, two predictors are used: CPU/DSP frequency (f_{cpu}) and FPGA frequency (f_{fpga}); hence, n=2. Section IV demonstrates the impact of the number of learning samples on prediction accuracy and the associated runtime overheads.

TABLE II
MODELING HYPOTHESIS

Model	Hypothesis $h_\theta(x)$
Latency (T_{cpu}/T_{dsp})	$\theta_0 + \theta_1 \cdot \frac{1}{f_{cpu}}$
Current (I_{cpu}/I_{dsp})	$\theta_0 + \theta_1 \cdot V + \theta_2 \cdot V \cdot f_{cpu}$
Latency (T_{fpga})	$\theta_0 + \theta_1 \cdot \frac{1}{f_{cpu}} + \theta_2 \cdot \frac{1}{f_{fpga}}$
Current (I_{fpga})	$\theta_0 + \theta_1 \cdot V + \theta_2 \cdot V \cdot f_{fpga}$
Performance	$P(f) = 1/T$
Energy	$E(f) = V \cdot I \cdot T$

As energy and performance are not linear functions of frequency, constructing their hypotheses as direct functions of these will not produce a good energy/performance model. Hence, we first generate models for output current (I) and latency (T). These models can then be used to derive the energy/performance trade-off model. Table II shows the different hypotheses used to generate such model. Column 1 shows the target model and column 2 shows the hypothesis used. These models and their hypotheses are explained further as follows:

1) CPU/DSP latency (T_{cpu}/T_{dsp}) per frame is expressed as a sum of a constant term (θ_0) meaning delay contributed by frequency-independent factors (such as memory contention, I/O setup, etc.) and also a term proportional to the CPU clock period (row 2).
2) CPU/DSP current (I_{cpu}/I_{dsp}) is expressed as a sum of three terms: the first two terms ($\theta_0 + \theta_1 \cdot V$) approximate the leakage current, while the last term ($\theta_2 \cdot V \cdot f_{cpu}$) signifies the dynamic current (row 3).
3) FPGA latency (T_{fpga}) per frame is expressed as a sum of three terms: the first term (θ_0) means the delay contributed by frequency-independent factors (such as memory contention, I/O set up, etc.), the second term ($\theta_1 \cdot \frac{1}{f_{cpu}}$) is

a function of the CPU clock period (as CPU carries out some computations before offloading), and the third term ($\theta_2 \cdot \frac{1}{f_{fpga}}$) is proportional to the FPGA clock period (row 4).

4) FPGA current (I_{fpga}) is expressed as a sum of three terms: the first two terms ($\theta_0 + \theta_1 \cdot V$) define the leakage current contribution, while the third term ($\theta_2 \cdot V \cdot f_{fpga}$) estimate the dynamic current, similar to CPU (row 5).
5) Performance (of CPU, DSP or FPGA) in terms of frames per second (fps) is expressed as an inversely proportional function to the latency of the corresponding computing resource (row 6).
6) Energy consumption (of CPU, DSP or FPGA) per frame is expressed as a product of the current (I), supply voltage (V) and latency (T) of the computing resource (row 7).

The supply voltage used in these hypotheses (Table II) is derived as a direct function of the operating frequency (see Section IV-A for details). The regression-based learning of the energy/performance trade-offs and their validations are further detailed in Section IV-B.

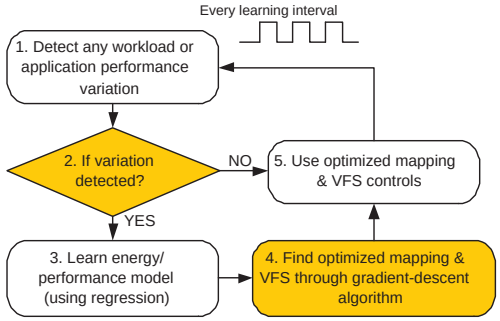


Fig. 4. Block diagram of the proposed adaptive runtime manager

B. Runtime Manager

The runtime model (Section III-A) enables the runtime optimization of energy and performance of the system for various task mapping and VFS controls in the presence of workload and application performance variations. Fig. 4 shows the flowchart of the adaptation steps in the runtime manager. At each learning interval, any workload or application performance variation is detected (step 1); the workload variation is detected through feedback from the performance counters, while application performance variations are detected through the API. If any variation is detected, the runtime learns (or re-learns) the energy/performance model through regression as discussed in Section III-A (step 2 and 3). Using this model, the runtime uses gradient-descent based search in the optimization space to quickly predict the required mapping and VFS controls. However, when no variation is detected, the runtime continues to use the previously learned runtime model-driven controls.

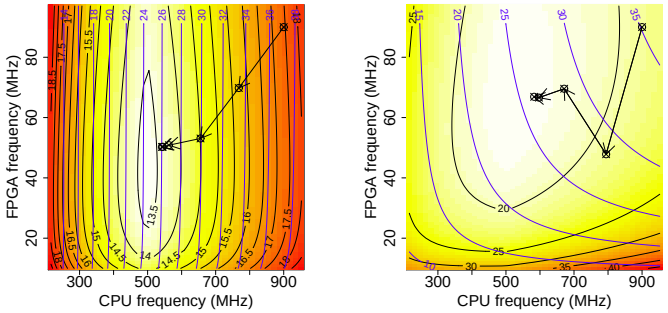
Algorithm 1 shows the pseudo-code of gradient-descent based search for optimized application task mapping and DVFS controls. The algorithm requires a performance requirement as an input and generates the minimum energy operating point (i.e. task mapping and VFS controls) as the output. The operating frequencies (f_i) of the computing resource (CPU, DSP and FPGA) are initialized (line 1); note that the DSP shares the CPU frequency as they are tightly coupled (Section IV-A). The learning rate (α_i) in the algorithm is also initialized to a high value to ensure a fast convergence (line 1). The search is initiated through pre-

Algorithm 1 Gradient descent-based task mapping and DVFS control

Input: Performance Requirement: P_{req}
Output: Minimum energy point: E_{min}, and Operating freq.: f_i, $i \in (CPU, FPGA)$

1: Initialize:f_i and learning rate: α_i
2: **repeat**
3: Predict energy: E^n=$F_1(f_i^n)$
4: **for** $\forall i$ **do**
5: Set $f_i^{n+1} := f_i^n - \alpha_i \frac{\partial}{\partial f_i} E^n$
6: Predict performance: P^{n+1}=$F_2(f_i^{n+1})$
7: **while** $P^{n+1} < P_{req}$ **do**
8: Set $\alpha_i := 0.5 \times \alpha_i$
9: Set $f_i^{n+1} := f_i^n - \alpha_i \frac{\partial}{\partial f_i} E^n$
10: Predict performance: P^{n+1}=$F_2(f_i^{n+1})$
11: **end while**
12: **if** $\left(\frac{\partial}{\partial f_i} E^{n-1} \times \frac{\partial}{\partial f_i} E^n\right) < 0$ **then**
13: $\alpha_i := 0.5 \times \alpha_i$
14: **end if**
15: **end for**
16: **until** $\Delta E \approx 0$
17: **return** $E = F_1(f_i^n)$ as $J_{E_{min}}$

diction of the energy consumption (E), which are functions of f_i [$i \in (CPU, FPGA)$] (line 3). For each computing resource f_i is updated by a gradient descent (line 5). If the updated f_i^{n+1} does not meet the specified performance requirement, the learning rate is reduced (line 8). At this time, the f_i^{n+1} is further updated by another gradient descent (line 9). Such predictions and updates are continued until the performance requirement is met (lines 7-10). The gradient of energy/performance predictions is not continuous due to the clipping of supply voltages. Hence, it is likely that normal gradient-descent will oscillate between the slopes of opposite gradients. To avoid such oscillation the learning rate is reduced (lines 12-13). The task mapping (to a computing resource) that provides the minimum energy consumption (E) while meeting the specified performance requirement (P) is returned as the optimized task mapping with its chosen operating frequency (f_i^n).



(a) Edge detection (b) Multi-pass blurring

Fig. 5. Contour maps of the energy/performance trade-offs between CPU and FPGA for different operating frequencies; yellow represents lower energy consumptions, red represents higher energy consumptions

As an example illustration of DVFS controls through Algorithm 1, Fig. 5(a) shows the energy/performance trade-offs between the CPU and FPGA for different operating frequencies. For demonstration purposes, an image edge detection application is used, which uses the CPU for video decoding and output display, and implements the image filtering task in the FPGA. To determine the optimal VFS controls of CPU and FPGA for such mapping, the energy and performance are predicted using high operating frequencies (f_i) initially, which give a higher performance (37 fps) than the required performance (26 fps). To

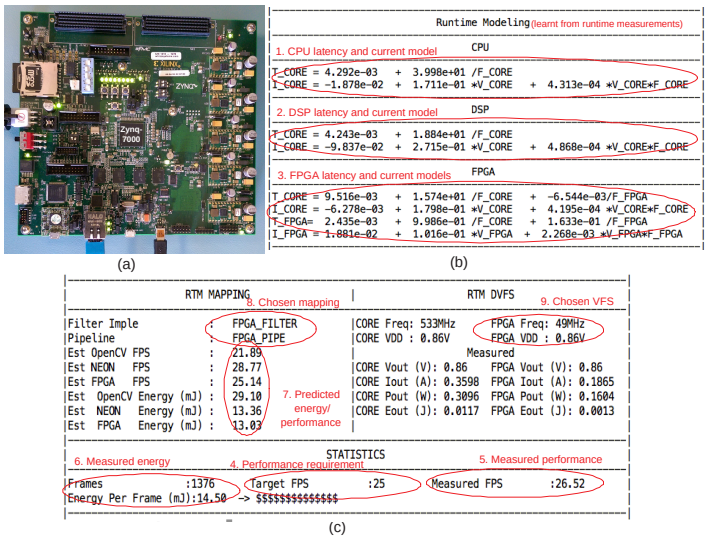


Fig. 6. (a) Zynq-ZC702 implementation platform, (b) snapshot of runtime energy/ performance model, and (c) runtime measurements and runtime adaptation

expedite the heuristic search, initially the learning rate is set to an arbitrarily high value for a larger gradient descent. For each next operating frequency during the search, the step size is reduced with the gradient. The frequencies at which the performance requirement is met and the predicted energy is the minimum, is chosen as the operating frequencies of the CPU and FPGA. As can be seen, the gradient-descent heuristic search results in intermediate operating frequencies: 670 MHz for the CPUs and 70MHz for the FPGA giving a 33 fps, followed by 650 MHz for the CPUs and 55MHz for the FPGA giving a 30 fps, etc., until it converges at the operating frequencies: 520 MHz for the CPUs and 50MHz for the FPGA giving a 26 fps.

Fig. 5(b) demonstrates another example of the same algorithm applied in a multi-pass image blurring application. Due to higher workload detected in such application, the runtime re-learns the energy/performance model. Underpinning this model, a gradient-descent based heuristic search (Algorithm 1) is carried out to determine the mapping and VFS controls for a given application performance requirement. As can be seen, for the same application performance requirement (i.e. 26 fps), the algorithm requires only five steps to determine the optimized controls. The effectiveness of the runtime management, including application task mapping and DVFS controls, is further validated in Section IV-C.

IV. Experimental Results

To validate the effectiveness of the proposed approach, a number of experiments are carried out through practical implementation. The implementation details are given below, followed by extensive validations of the runtime model and management.

A. Experimental Setup and Implementation

The proposed approach is engineered as a prototype runtime framework (implementing application, runtime and hardware) on a Zynq-ZC702 development board (shown in Fig. 6(a)), consisting of a Zynq-7000 chipset with two ARM Cortex-A9 CPU cores, two DSP cores (Neon SIMD) and an FPGA fabric with 85K logic cells [14]. The board uses a UCD9248 digital power controller, with the same voltage/frequency island for the CPU/ DSP cores, and a separate power controller for FPGA. The CPU/ DSP frequency can be varied from 216MHz to 1000MHz in

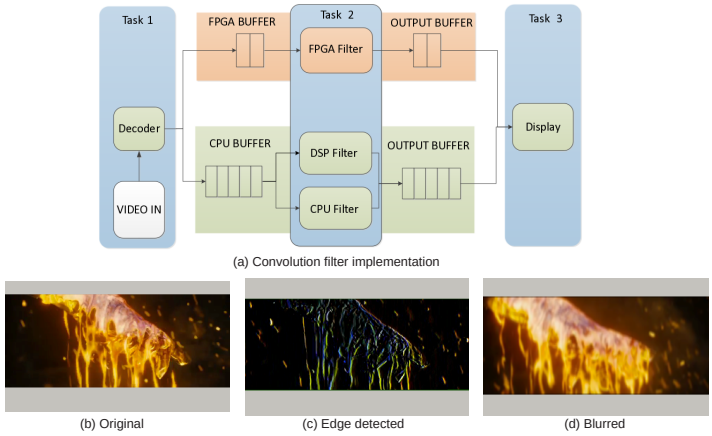


Fig. 7. (a) Image processing application implementation, (b)-(d) input and outputs of the image processing applications

16MHz steps, and the corresponding voltage varies from 0.86V to 1.10V, derived empirically as

$$Vdd_{cpu} = 4.21 \times 10^{-4} \times Freq_{cpu} + 0.647 \quad . \qquad (5)$$

Similar to the CPU/DSP, the FPGA operating voltage is also characterized as a function of its operating frequency as

$$Vdd_{fpga} = 4.53 \times 10^{-3} \times Freq_{fpga} + 0.554 \quad . \qquad (6)$$

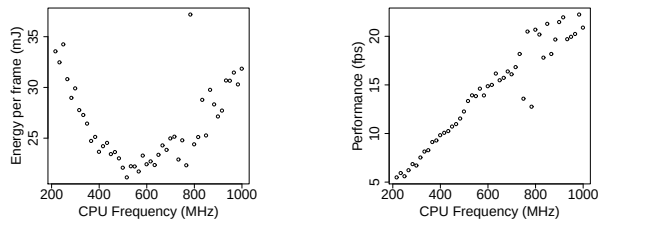
The FPGA operating frequency varies from 10MHz to 100MHz, while the supply voltage varies from 0.86V to 1.10V.

For demonstration purposes, two image processing applications were used. Fig. 7(a) shows the block diagram of the application implementation using three major tasks. Task 1 decodes images from a video file (360p resolution in this case study), task 2 carries out intended image processing operations, and finally, task 3 buffers and displays the output images. The CPU image processing was implemented using OpenCV [15], DSP processing was implemented using NEON assembly instructions [14] and that in the FPGA was implemented using synthesized RTL. For overall system and I/O management, tasks 1, 3 and the runtime are pre-mapped and implemented in the CPU. Task 2 mapping is controlled through the runtime manager as it is the most compute-intensive task. The operating VFS points for all tasks are also controlled by the runtime manager using the runtime energy/ performance model. Fig. 7(b)-(d) show an example input image and the corresponding output images of the applications.

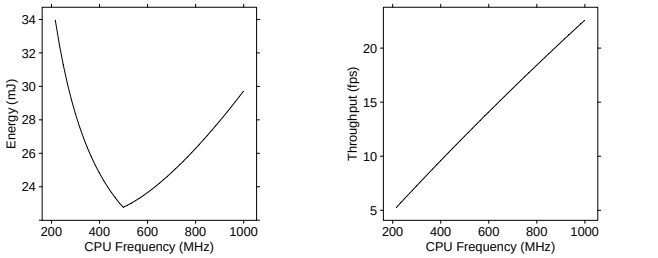
Fig. 6(b) shows a snapshot of the runtime energy/performance model (Section III-A). The energy model shows the current and latency models for CPU (highlighted, red circle 1), DSP (highlighted, red circle 2) and FPGA (highlighted, red circle 3). These models are learned through regression using real measurements from power monitors and performance counters. Fig. 6(c) shows example demonstrations of the runtime measurements and management (Section III-B). With a given performance requirement (highlighted circle 4), the energy and performance measurements are carried out at each frame interval (highlighted, red circles 5 & 6). These measurements set up the runtime energy/performance model (Fig. 6(c)), to enable prediction during runtime (highlighted, red circle 7). Based on the predicted operating point, a suitable task mapping (highlighted, red circle 8) and appropriate VFS control are chosen (highlighted, red circle 9).

B. Runtime Model Validation

The runtime model validation is carried out in two stages. In the first stage, to establish the hypotheses (Table II) used in the

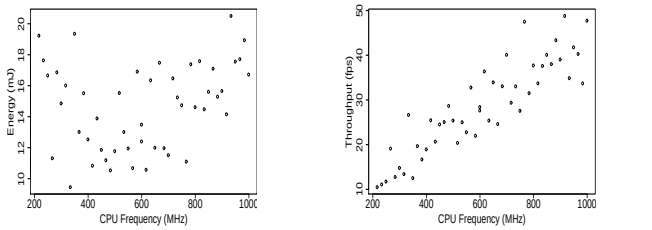


(a) CPU measured energy (b) CPU measured performance

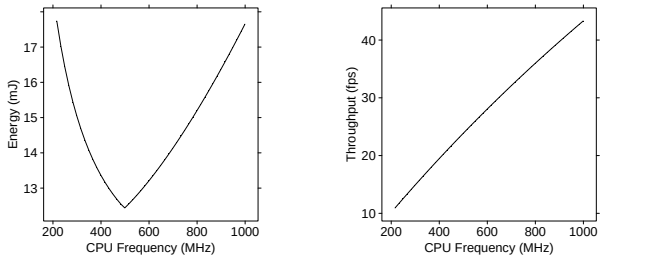


(c) CPU modeled energy (d) CPU modeled performance

Fig. 8. (a) Measured energy consumption (in mJ per frame) (b) measured performance (fps), (c) Modeled energy (in mJ per frame) and (d) modeled performance (fps) for CPU



(a) DSP measured energy (b) DSP measured performance

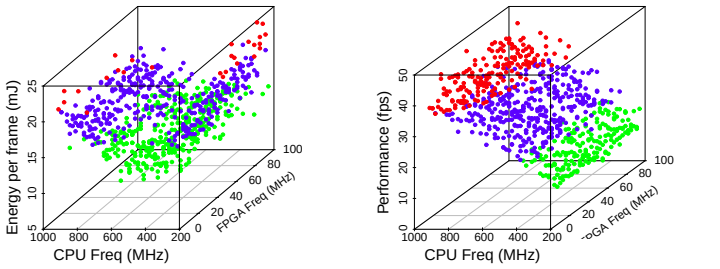


(c) DSP modeled energy (d) DSP modeled performance

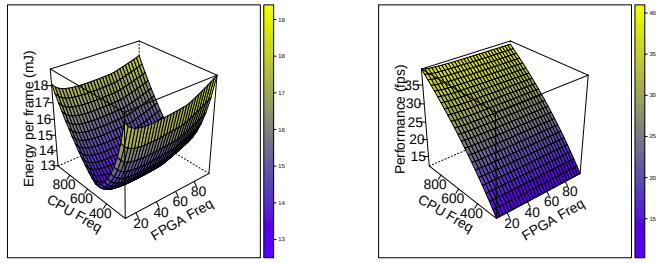
Fig. 9. (a) Measured energy consumption (in mJ per frame) (b) measured performance (fps), (c) Modeled energy (in mJ per frame) and (d) modeled performance (fps) for DSP

energy/performance model, a training model is generated using an extensive samples. The model generated using these hypotheses is then used to generate the runtime model in the second stage.

Fig. 8 shows the measured and modeled energy consumptions (mJ per frame) and performances (fps) using these hypotheses for the CPU. The measured energy (Fig. 8(a)) and performance (Fig. 8(b)) values are generated with varying operating frequencies, while the modeled values (Fig. 8(c) and Fig. 8(d)) consist of predictive energy and performance values generated using the training model. As can be seen, the modeled values exhibit a high degree of correlation with the measured values (with an average modeling error of 4.6% and 5.9% for energy and performance, Table III). Fig. 9 shows the same for the DSP. Similar to the CPU, there is also a high degree of correlation between the measured



(a) FPGA measured energy (b) FPGA measured performance

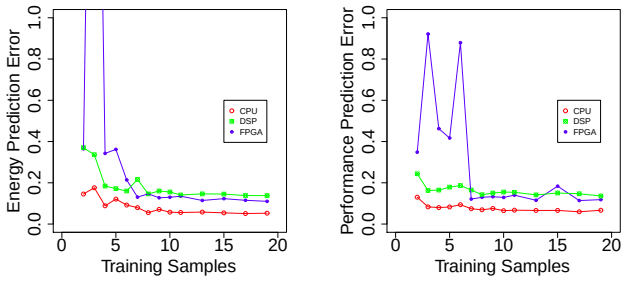


(c) FPGA modeled energy (d) FPGA modeled performance

Fig. 10. (a) Measured energy consumption (in mJ per frame) (b) measured performance (fps), (c) modeled energy (in mJ per frame) and (d) modeled performance (fps) for FPGA

values and the modeled values (with an average modeling error of 12.5% and 12.4% for energy and performance, Table III). The higher modeling error in DSP is due to uncertainty caused by instruction cache sharing between the CPU and DSP.

Fig. 10 shows the measured and modeled energy consumptions (mJ per frame) and performances (fps) of the FPGA implementation for different operating frequencies. Unlike CPU or DSP, FPGA implementation has two predictors: frequency of CPU (f_{cpu}) and frequency of FPGA (f_{fpga}). However, despite higher number of predictors, the measured energy (Fig. 10(a)) values also show a high degree of correlation with the modeled energy (Fig. 10(c)) values (with an average modeling error of 9.8%, Table III). This is because the runtime model (Section III-A) is generated using realistic component models: current (I) and latency (T). For the same reason, the measured and modeled performances (Fig. 10(b) and (d)) also show high correlations between them (average modeling error of 9.9%, Table III).



(a) Energy (b) Performance

Fig. 11. Energy and performance prediction errors vs learning samples

Using the validated hypotheses, a runtime energy/performance model is learned using a number of measured samples of current and latency values. The accuracy of this model depends on a number of factors: the number of samples acquired, the number of predictors for a given computing resource, and the underlying

TABLE III
MODELING ERROR INCURRED FOR THE GIVEN HYPOTHESES (TABLE II)

Computing resource	Energy (in %)	Performance (in %)
CPU	4.6	5.9
DSP	12.5	12.4
FPGA	9.8	9.9

relationships between current, latency, performance and energy. Fig. 11 shows this learning trade-offs for all computing resources (CPU, DSP and FPGA). As can be seen, the model's error in terms of misprediction is initially $> 80\%$ for both energy and performance models. After 5 learning samples the prediction errors of CPU/DSP reduces and converges close to ($\approx$6% for CPU and $\approx$15% for DSP). The FPGA takes about 8 learning samples until it converges to $\approx$13% prediction error for energy and performance (Fig. 11(a) and (b)). All computing resources require low convergence time as regression with such small number of predictors is rigid and the variance in the model is usually small [16]. The higher convergence time for FPGA is incurred due to usage of two predictors (f_{cpu} and f_{fpga}), compared to only one predictor in the case of CPU and DSP (f_{cpu}).

TABLE IV
AVERAGE PREDICTION ERRORS FOR IMAGE PROCESSING WORKLOADS

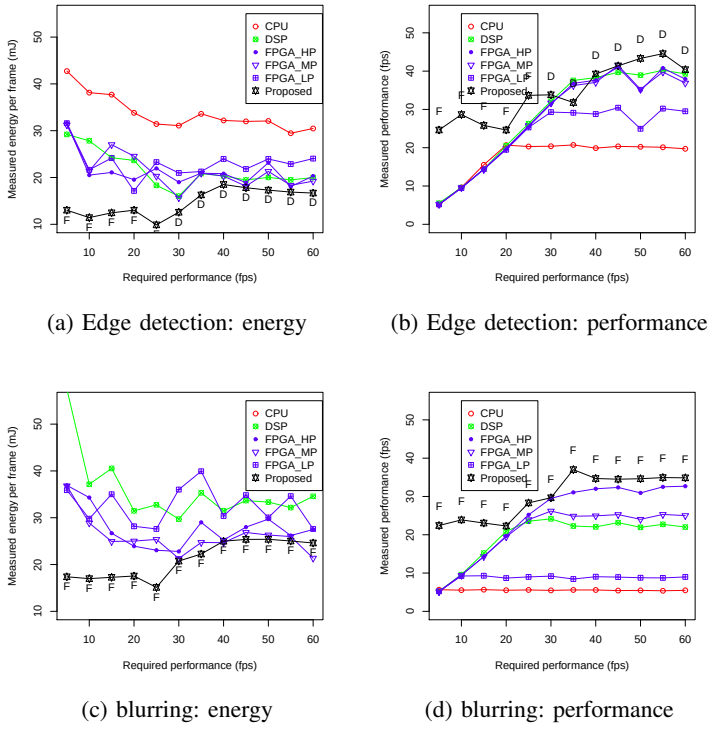
Workload	Models	CPU (%)	DSP (%)	FPGA (%)
Edge det.	Energy	5.8	15.1	13.0
Edge det.	Performance	6.4	15.0	12.1
Edge det.$\rightarrow$Blurring	Energy	73	40	27
Edge det.$\rightarrow$Blurring	Performance	317	62	26

The energy/performance model generated through regression remains valid for a given workload, based on which the model is learned. The validity of the model is characterized by low prediction errors ($<\approx 15\%$). Table IV shows the average energy and performance prediction errors of different computing resources (in rows 2 and 3) of image edge detection application. As can be seen, the highest prediction error is incurred by the DSP (15.1% for energy model and 15% for performance model). This is because both the DSP and CPU share the same instruction queue, which introduces uncertainty in its performance. The CPU models give the lowest prediction errors for both energy and performance.

When the workload per image processing increases, the model will incur larger prediction errors as the previously learned model does not represent the trade-offs between energy/performance any more (Section 4). As can be seen in 4th and 5th rows (Table IV), when the image processing workload increases from edge detection to multi-pass blurring, the prediction error increases. The highest mispredictions are incurred by the CPU, with up to 317% error. This is because the increased workload has the highest impact on CPU energy and performance. To ensure that the model always follows the energy/performance trade-offs, re-learning is initiated when the average prediction errors goes beyond 20%. With the re-learned model, the runtime manager can adapt to the changing workloads. See Fig. 12 for experimental results demonstrating such adaptation.

C. Comparative Evaluation of Runtime Adaptation

Fig. 12 shows the comparative evaluations of the proposed approach. Fig. 12(a) and (b) show the energy (mJ, per frame) and performance (fps) comparisons using the edge detection application (Fig. 7(a)). For comparison, the following different image edge detection implementations were carried out: CPU implementation with Linux ondemand governor (red), DSP implementation with Linux ondemand governor (green), FPGA implementation with 10 MHz frequency (purple square, FPGA_LP), FPGA implementation with 50 MHz frequency (purple triangle, FPGA_MP), FPGA implementation with 100 MHz frequency (purple dot, FPGA_HP) and finally, the filter implementation using our adaptive approach (black star). The Linux ondemand governor was chosen as it features practical implementations and partial runtime adaptation based on core usage (Table I). The chosen application task mappings are indicated alongside the plotted points with the first letter of the computing resource (C: CPU, D: DSP and F: FPGA). Fig. 5 shows example illustrations of how optimal frequency is chosen for a given task mapping.



(a) Edge detection: energy

(b) Edge detection: performance

(c) blurring: energy

(d) blurring: performance

Fig. 12. Comparative evaluations of different image processing applications

As expected, the CPU implementation gives the highest energy consumption per frame for a given performance requirement (Fig. 12(a)). The performance of such an implementation is, however, constrained by the highest permissible operating frequency as it cannot provide more than 22 fps (Fig. 12(b)). The DSP implementation, on the other hand, gives better energy efficiency and performance than the CPU. As expected, the DSP implementation favors higher operating frequencies when the performance requirement increases due to core usage-based operating frequency scaling [3]. The FPGA implementations do not vary the operating frequencies based on the requirement, which results in non-adaptive energy and performance variations for all three frequencies chosen. The energy efficiency of FPGA implementation follows similar trends as the DSP filter for FPGA_MP and FPGA_HP. The proposed approach can detect the change in application performance requirement and suitably learn the energy/performance trade-offs through a runtime model (Section III-A). Based on this model, the approach can adapt controls and choose the most energy-efficient computing resource through optimized task mapping (among CPU, DSP and FPGA)

and VFS controls. As a result, such adaptation can achieve high energy reduction for the varying performance requirements (up to 70% energy reduction compared to the CPU-only implementation for a given performance requirement of 3 fps). At lower performance requirements the approach favors the mapping of the task on the FPGA and at higher performance requirements it favors the mapping of the task on the DSP for the edge detection application. Note that beyond 45 fps, none of the implementation can meet the performance requirement.

To show the comparative effectiveness of the proposed approach with increased image processing workloads, another set of experiments were carried out using the multi-pass image blurring application. Fig. 12(c) and (d) show the energy (mJ, per frame) and performance (fps) comparisons using this application. As can be seen, due to such a challenging workload, the CPU, DSP, FPGA_LP and FPGA_MP filters fail to adapt to performance requirement (beyond 26 fps) and provide energy efficiency (CPU energy figures not shown in Fig. 12(c) as it is out of range). FPGA_HP provides better energy efficiency but does not adapt to the required performance at lower performance requirements. Our proposed approach continues to adapt to the required performance and workload per filter task, and provides the most energy-efficient task mapping and DVFS controls. In fact, compared to the CPU-only implementation of the application (the CPU-only energy and performances are not shown due to being out of axes limits), our approach achieves significantly higher energy savings (i.e. more than an order of magnitude savings).

TABLE V
RUNTIME MANAGEMENT OVERHEADS FOR COMPUTING RESOURCES

Control/ Monitor	Occurrence	CPU/DSP (in ms)	FPGA (in ms)
Modeling	Workload change	0.054	0.18
Management	Performance/workload change	0.03	0.06
Perf. monitor	Sampling period	0.77	0.98
VFS transition	Performance/workload change	1.50	2.60

D. Runtime Overheads

The proposed approach incurs runtime overheads due to various adaptation steps, including optimization, control and monitor operations. Table V shows the runtime overheads for different operations during runtime. Column 1 lists the runtime control/ monitor operations, column 2 indicates their occurrences (i.e. how often these operations are used), column 3 gives the runtime overheads for the CPU/DSP, while column 4 gives the same for the FPGA. As can be seen, the runtime model learning, which is carried out when any workload change is detected, incurs a small overhead (0.054 ms) for the CPU/DSP. The overhead for the FPGA is slightly higher (0.18 ms) due to a higher number of learning samples (Section III-A). The runtime adaptation (and optimization) exhibits the minimum overhead of all (0.03 ms for CPU/DSP and 0.06 ms for FPGA) due to simple gradient-descent based search operation for a given runtime model. Such adaptation takes place every time a performance or workload change is detected. The performance monitor results in the second largest runtime overhead, which depends on the type of computing resource used; generally the CPU/DSP incurs less than the FPGA as it involves monitor interaction via the CPU. Such overhead takes place at every sampling period (50 used in our work for denoising the measurement errors). The largest single overhead is generated by the VFS transition, which takes place every time the performance or workload changes. As expected, as FPGA transition is actuated via the CPU and gives higher VFS transition overheads than the CPU.

V. CONCLUSIONS

This paper has presented an adaptive energy minimization approach for embedded heterogeneous systems. The approach is based on an energy/performance model learned through regression using runtime measurements from the performance counters and sensors. Using this model application task mapping and DVFS controls are suitably optimized during runtime to adapt to workload and performance variations. Through extensive validations on a Zynq-ZC702 platform, the approach demonstrated significant energy reductions (more than 70% in some cases) compared to the existing approaches, while meeting a given application performance requirement. The work is expected to be particularly useful for applications that require computation on a number of different computing resource in a heterogeneous system. Our future work is to extend this technique to deal with concurrent applications, competing for the computing resources.

VI. ACKNOWLEDGMENTS

The authors would like to thank the EPSRC-UK for funding this work under PRiME project with grant number EP/K034448/1.

REFERENCES

[1] R. A. Shafik *et al.* System-level design optimization of reliable and low power multiprocessor system-on-chip. *Microelectronics Reliability*, 52(8):1735 – 1748, 2012.

[2] D. Flynn. An ARM perspective on addressing low-power energy-efficient SoC designs. In *Proc. of ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, pp. 73–78, NY, USA, 2012. ACM.

[3] V. Pallipadi. Enhanced Intel SpeedStep technology and demand-based switching on Linux. *Intel Developer Service*, 2009.

[4] L. K. Goh *et al.* Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems. *Parallel and Distributed Systems, IEEE Trans. on*, 20(1):1–12, Jan 2009.

[5] M. Goraczko *et al.* Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. In *Procc of the 45th Annual Design Automation Conference*, DAC'08, pp. 191–196, NY, USA, 2008. ACM.

[6] M.T. Schmitz *et al.* Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems. In *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proc.*, pp. 514–521, 2002.

[7] K. Gruttner *et al.* An ESL timing, power estimation and simulation framework for heterogeneous SoCs. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on*, pp. 181–190, July 2014.

[8] A. Nabina and J. Luis Nunez-Yanez. Adaptive voltage scaling in a dynamically reconfigurable FPGA-based platform. *ACM Trans. Reconfigurable Technol. Syst.*, 5(4):20:1–20:22, December 2012.

[9] J. Luo and N.K. Jha. Power-efficient scheduling for heterogeneous distributed real-time embedded systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, 26(6):1161–1170, June 2007.

[10] M. Qiu and E. H. M. Sha. Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 14(2):25:1–25:30, April 2009.

[11] S. Mohanty and V. K. Prasanna. A hierarchical approach for energy efficient application design using heterogeneous embedded systems. In *Proc. of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '03, pp. 243–254, NY, USA, 2003. ACM.

[12] A. Schranzhofer *et al.* Dynamic power-aware mapping of applications onto heterogeneous MPSoC platforms. *Industrial Informatics, IEEE Trans. on*, 6(4):692–707, Nov 2010.

[13] J. Cohen *et al. Applied Multiple Regression/Correlation Analysis For The Behavioral Sciences*. Routledge, 2013.

[14] Xilinix. Zynq-7000 all programmable SoC: ZC702 evaluation kit and video and imaging kit (ISE design suite 14.2). 2012.

[15] G. Bradski. The OpenCV library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.

[16] N. R. Draper *et al. Applied Regression Analysis*, volume 3. Wiley New York, 1966.