Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices

Domenico Balsamo, Alex S. Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M. Al-Hashimi, Geoff V. Merrett and Luca Benini

***Abstract*—Energy harvesters are being used to power autonomous systems, but their output power is variable and intermittent. To sustain computation, these systems integrate batteries or supercapacitors to smooth out rapid changes in harvester output. Energy storage devices require time for charging and increase the size, mass and cost of systems. The field of transient computing moves away from this approach, by powering the system directly from the harvester output. To prevent an application from having to restart computation after a power outage, approaches such as *Hibernus* allow these systems to hibernate when supply failure is imminent. When the supply reaches the operating threshold, the last saved state is restored and the operation is continued from the point it was interrupted. This work proposes *Hibernus++* to intelligently adapt the hibernate and restore thresholds in response to source dynamics and system load properties. Specifically, capabilities are built into the system to autonomously characterize the hardware platform and its performance during hibernation in order to set the hibernation threshold at a point which minimizes wasted energy and maximizes computation time. Similarly, the system auto-calibrates the restore threshold depending on the balance of energy supply and consumption in order to maximize computation time. *Hibernus++* is validated both theoretically and experimentally on microcontroller hardware using both synthesized and real energy harvesters. Results show that *Hibernus++* provides an average 16% reduction in energy consumption and an improvement of 17% in application execution time over state-of-the-art approaches.**

***Index Terms*—IEEEtran, journal, LaTeX, paper, template.**

I. Introduction

RECENT momentum of the Internet-of-Things (IoTs) is driving the need for embedded systems comprising of one or more ultra low-power and resource constrained sensors [1]. Power management of these devices is emerging as a primary challenge for system designers as they typically have to last for many years, without intervention to charge or replace batteries [2]–[4]. Energy harvesting (EH) offers the potential for low-power systems to operate without batteries, by generating electrical power from environmental sources including light, vibration, motion or temperature differences [5]–[7].

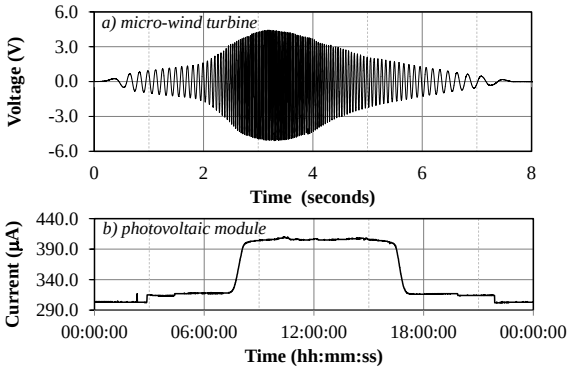


Fig. 1. Example outputs from energy harvesters: voltage of a wind harvester over one 'gust', and current from an indoor photovoltaic module over a day

A primary challenge in developing IoT systems with micro-power environmental energy harvesters is the unpredictable nature of the sources. The power obtained from energy harvesters is dependent on the harvester, deployment location, and often on other factors such as weather, time of day, or machine activity. Kinetic or wind energy harvesters typically give an AC output relative to the frequency of vibration or rotation, while photovoltaic modules or thermoelectric generators typically give a more slowly-varying DC output. To highlight this transient nature, Figure 1 plots the output of (a) a micro wind turbine, and (b) a photo-voltaic cell. As can be seen from this figure, the output from the micro wind turbine has a very high power-cycle frequency (supply falling below 0 V at intervals of the order of milliseconds). On the other hand, the output current from the photovoltaic cell is slowly varying, with a low power-cycle frequency.

The load profile of embedded computing systems is typically bursty. These systems remain in a low power mode, waking up to take measurements or perform calculations or communication. The variability and typically low level of power output from energy harvesters (Figure 1) implies that systems powered directly from their output would result in repeated power-cycling, restarting program execution from the beginning. To address this, storage devices are typically used to buffer energy so that systems can operate continuously and avoid unstable operation. However, energy storage devices require time to charge up to a usable voltage, and increase the size, mass and cost of the system. As an example, the two

Manuscript received Mmmmmm dd, yyyy. This work was supported by EPSRC Grants EP/L000563/1 and EP/K034448/1 (the PRiME Programme www.prime-project.org). It was also supported by a Telecom Italia s.p.a. PhD grant and PHIDIAS (EU 7th Framework Programme CA 318013).

Experimental data used in this paper can be found at DOI:10.5258/SOTON/389749 (http://dx.doi.org/10.5258/SOTON/389749).

D. Balsamo, A. Das, A. S. Weddell, A. R. Arreola, G. V. Merrett and B. M. Al-Hashimi are with the Pervasive Systems Centre, Electronics and Computer Science, University of Southampton, UK. D. Brunelli is with the Department of Industrial Engineering, University of Trento, Italy. L. Benini is with the Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi" (DEI), University of Bologna, Italy.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.201x.xxxxxxx

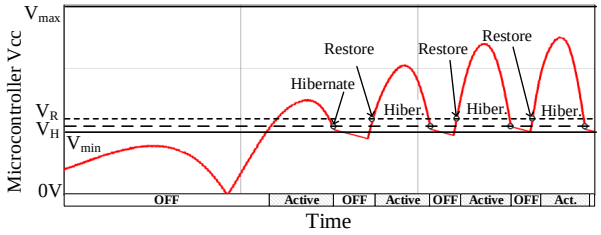


Fig. 2. Operation of a transiently-powered system

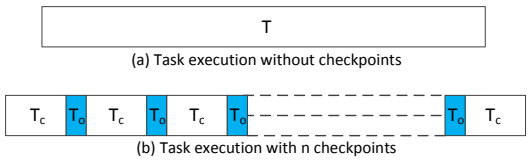


Fig. 3. Task execution with and without checkpoints

AA-sized batteries on a Crossbow Telos mote [8] occupy over half of its overall volume.

For IoT devices that have constrained dimensions, such as implantable wearable bio-sensors [9], personalized healthcare [10], home and building automation [11] and RFID devices [12], it is desirable to power systems directly from the energy harvester without using any energy storage other than decoupling and parasitic capacitance. However, this makes systems susceptible to frequent power interruptions and resets caused by the transient supply. Clearly, these two cases (conventional energy storage to buffer continued operation, vs zero energy storage) represent two extremes of a continuum, where an intermediate solution incorporating some capacitance may provide improved behaviour. However, our research focuses on the extreme case where systems have no additional energy storage. To address this, we developed *Hibernus* [13]: an approach to enable computation to be sustained in systems powered directly from energy harvesters. The principle behind *Hibernus* is to save a system snapshot (RAM and CPU registers) to non-volatile memory and suspend operation when power supply failure is imminent, i.e. when the supply voltage falls below a predefined threshold. Similarly, when the supply voltage increases above a restore threshold, *Hibernus* restores the last snapshot to continue operation from the point it was suspended. Figure 2 shows the generic operation of *Hibernus*; the voltage across the microcontroller is plotted in the figure. When the voltage falls below V_H, the system stores a snapshot and hibernates. When the voltage rises above V_R, the system restores a snapshot and continues operation. A further extension to *Hibernus* considered the addition of Dynamic Frequency Scaling (DFS) to modulate consumed power in response to available power [29]. The main limitation of *Hibernus* is that it requires an off-line characterization to fix V_H and V_R, specific to the platform being used.

In this paper we develop *Hibernus++* an adaptive version of *Hibernus*, which adjusts the hibernate and restore thresholds dynamically in response to the system power consumption, the on-board decoupling capacitance and the dynamics of the energy harvester (Figure 1). The objective is to sustain operation for a longer duration within the constrained power budget. Following are the novel contributions of this work:

- the ability to self-configure the hibernate and the restore thresholds on-the-fly, depending on the dynamics of the power source and system power consumption;
- theoretical formulation of the approach, including characterizing the hibernate and restore thresholds for a specific energy harvester source; and
- a thorough practical validation using both synthesized and real EH sources on an FRAM-based microcontroller with a range of applications.

We discuss the current state of research in this field and the dynamics of energy harvesters in Section II. We then develop a model to represent the operation of transiently-powered systems in Section III. The approach is explored mathematically in Section IV and then practically validated in Section V: firstly using synthesized energy harvesting signals (to enable comparison between methods), and then through the use of real energy harvesters. Finally, in Section VI a cold-start circuit is proposed which enables operation of the system with ultra-low input currents.

II. Background and Related Work

A new paradigm, which addresses computing challenges with transient power sources such as energy harvesting, is of 'transiently-powered computing' [14]. This typically borrows from the concept of checkpointing, which has been used in large-scale computing for decades to provide robustness against errors or hardware failure [15]. This technique involves systematically saving data to non-volatile memory (NVM). State-of-the-art embedded systems use a variety of classic and advanced NVM structures to save their state. Examples of memories used for state retention are flash [16] or battery-backed SRAM memories [17].

To recover from a failure, systems roll back to the previous valid checkpoint, before continuing operation. Figure 3 shows task execution (a) without and (b) with n checkpoints. The task's execution time T is divided into $(n+1)$ intervals. At each interval, the task is executed for a duration $T_c = \frac{T}{(n+1)}$. In the figure, T_o represents the time overhead of checkpointing, i.e., saving the system state. T_o does not include the restore time, which needs to be considered as it introduces a significant overhead that depends on the chosen checkpointing policy. Thus, the task execution time with n checkpoints is $(n+1)(T_c + T_o) = (T + (n+1)T_o)$. However, a drawback of checkpointing is that it is impossible to predict the exact time of failures, so computation time will be wasted by (1) taking unnecessary checkpoints, and (2) rolling back to the last checkpoint if power failure occurs towards the end of a checkpoint interval.

Attempts have been made to optimize systems to address these problems, for example by assuming different failure distributions. Recently, the checkpointing concept has been applied to embedded devices with unstable power supplies, to avoid power-cycling causing loss of computation. Checkpointing enables systems to save their state so that, when their power supply resumes, they can continue operation from

the last valid checkpoint. As shown in Figure 2, this allows computation to continue across several power-cycles, which would conventionally have caused a system to reset repeatedly. Prominent works in this area include *Mementos* [18], *QuickRecall* [19] and *Hypnos* [20]. *Mementos* saves the system state to non-volatile memory (NVM) periodically, which enable it to return to a previous checkpoint after a power failure. A number of compile-time checkpoint placement heuristics are proposed, including at the beginning of every function-call or before any loop. Disadvantages of this approach include the use of flash memory (which is slow and power-hungry); the fact that many checkpoints will be taken (most of which will be redundant); and that space must be reserved in non-volatile memory for two complete snapshots in case a power interruption occurs whilst a snapshot is being taken. *Mementos* is also selective about the data it saves. A similar selective snapshot approach, where the snapshot contains only data which has changed since the last snapshot, has also been proposed in other work [14]. This offers behaviour complementary to the approaches presented in this paper, permitting a reduction in the time (and hence energy) required to hibernate. The complete system state (e.g. the peripherals) cannot be restored as it takes too much time with flash memory. This limits its applicability to purely computational applications, rather than embedded systems which may need to interface with other devices. A few recently published papers show that the time and energy cost of distributed state-retentive logic elements can be lowered by orders of magnitude with respect to traditional flash-based approaches using alternative technology such as FRAM [21] and ReRAM [22].

QuickRecall [19] proposed a refinement to the *Mementos* technique, involving the use of a microcontroller with FRAM non-volatile memory, hence reducing the overheads of saving a snapshot. Their approach also used FRAM as unified memory, so that the system's RAM was not used for storing variables. *QuickRecall* ensures checkpointing by setting an appropriate trigger voltage to interrupt the normal program execution and save a snapshot. Furthermore, it exploits an external power management unit that uses hysteresis to turn on and off the supply when the operating voltage is above or below two fixed thresholds. A disadvantage of *QuickRecall* is that it uses an inflexible fixed voltage threshold to prompt an interrupt to take a snapshot. It also relies on the use of a processor with unified FRAM. However, by only utilizing this type of memory the system consumes more energy than with SRAM, especially for write operations, introducing a significant overhead in active mode. Finally, there is an overhead due to the initialization of the microcontroller and the peripherals, which varies depending on the application, meaning that it is not application agnostic. *Hypnos* [20] proposed an ultra-low power sleep mode for micro-controllers that overcomes the limitations of both these approaches. This technique is based on the observation that the on-chip SRAM in a microcontroller exhibits data retention even at a much lower supply voltage (as much as 10x lower) than the typical operating voltage of the microcontroller. Hypnos exploits this observation by performing extreme voltage scaling when the microcontroller is in sleep mode. However, this solution suffers from a data retention problem in cases where the power source is unavailable for a prolonged period of time.

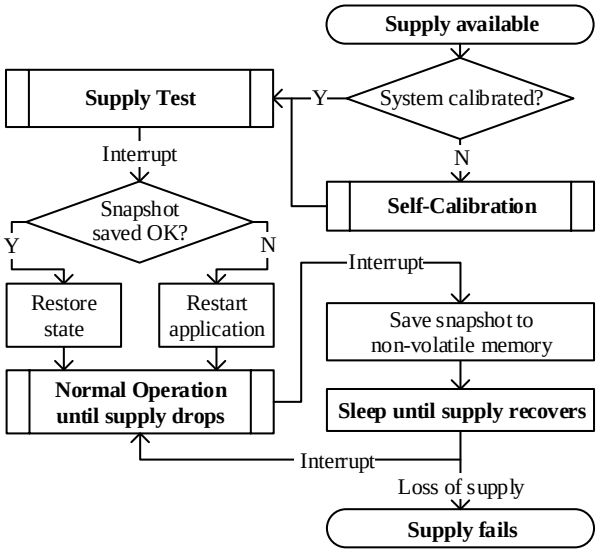


Fig. 4. Operation of *Hibernus++*

Recently, a number of hardware approaches to transient computing have been proposed, which explore Non-Volatile Processor (NVP) architectures [25] to optimise behaviour. Wang *et al.* [26] proposed a NVP using ferroelectric flip-flops that incorporate both volatile and non-volatile elements for checkpointing; Bartling *et al.* [27] presented an ARM-based NVP, exploiting SoC FRAM-based logic arrays for state retention, and Sakimura *et al.* [28] presented a 16-bit RISC CPU based on MeRAM. The major advantage of customized NVPs such as these is the significant reduction in time and energy for data retention. However, these solutions are currently still experimental research platforms based on technologies and NVMs that are not available on the market; hence their power consumption, performance and cost are not well understood. In contrast, *Hibernus++* can be applied to conventional off-the-shelf NVM processors, and use software approaches to create NVPs.

III. Hibernus++: Enabling Computation with Intermittent Power Supplies

An ideal transiently-powered system hibernates at the last possible moment before supply failure, and resumes at the earliest optimal point so that the maximum amount of computation can be carried out before the next power failure. The aim of our approach is to maximize the useful computation that can be carried out by transiently-powered systems with a given power source, without the need to add energy storage. We aim for systems to be able to use power whenever it is available; this may be for periods as short as a single cycle from EH with an alternating current (AC) output, or continuously with minimal overheads if the output of the EH is sufficient to continuously power the system.

A. Principle of Operation

The following terms are introduced here to aid understanding of the operating principles:

- **Snapshot:** copying the system state (RAM, processor and peripheral registers) into non-volatile memory.
- **Checkpoint:** a point in the application execution when the supply voltage is polled to decide whether a snapshot should be taken.
- **Hibernate:** save a snapshot and enter in low-power mode.
- **Restore:** restore a system state from non-volatile memory and continue operation.

Figure 4 outlines the operation of *Hibernus++*. When power is first applied to the system (in other words, the supply voltage rises above the minimum operating voltage), the system checks whether it has been calibrated: if not, it runs the calibration routine, which sets the voltage threshold for hibernation (V_H) by evaluating the rate of voltage drop in the case of a sudden loss of supply. Next, the system tests its supply and (1) continues if the supply provides sufficient power to sustain the system's operation in active mode, or (2) sleeps, for lower-power supplies, until the supply voltage reaches a sustainable value. If a snapshot was attempted but failed, V_H is increased by 0.1V and the application is restarted from the beginning. If a valid snapshot has previously been stored in memory, the system restores the snapshot and continues operation. Otherwise, execution is started from the beginning. Following these routines, normal operation of the system continues until the supply voltage drops below V_H, at which point the system hibernates. If the supply voltage recovers without dropping below the microcontroller's minimum operating voltage, V_{min}, the system resumes operation without the need to restore its state. Otherwise, if the supply voltage has dropped below V_{min} causing the volatile memory contents to be lost, the system restores its state, provided that it was saved successfully.

B. Hibernation Strategy and Calibration Routine

As seen in Figure 3, checkpointing involves a timing overhead (T_o). It is also associated with an energy cost (represented as E_o) for storing a snapshot in the non-volatile memory. With n checkpoints, the total execution time and energy overheads are $(n+1)\cdot T_o$ and $(n+1)\cdot E_o$, respectively. *Clearly, the fewer checkpoints there are, the lower the time and energy overheads.* Conversely, when a power outage occurs, the checkpointing system rolls back to the last valid checkpoint. In the worst case, the loss in useful computation is $T_c = \frac{T}{(n+1)}$ (corresponding to the case where the outage occurs at the end of a checkpointing segment, during or just before the process of saving the snapshot). *Clearly, the fewer checkpoints there are, the higher the loss of computation during outage.*

For systems with checkpointing operating from transient sources, an error condition refers to the state where the power supply drops below the minimum operating voltage of the microcontroller. The power failure probability can be very high for some energy harvesting sources (with the frequency being many Hz), so a trade-off exists for selecting the number of checkpoints. To address this, *Hibernus++* uses an adaptive approach, where the system saves a snapshot and hibernates only when a power failure is imminent.

Operating in an 'ideal' manner, i.e. hibernating at the last possible moment, is the most efficient strategy but it is risky. The consequences of a untimely hibernation may be severe: it may mean that the system is unable to restore its state, losing valuable data that was stored in volatile memory, and has to restart its computation from the very beginning.

In general, the remaining operational time T_χ (before power loss) for a given system can be expressed by:

$$T_\chi = \frac{(V - V_{min})\,C}{I_l - I_h} - T_h \tag{1}$$

with initial supply voltage V, minimum operating voltage of the microcontroller V_{min}, supply capacitance C, and time for hibernation T_h. In order to compute this accurately, the harvested current I_h and load current I_l must be known. This may be used to identify the optimal time for a hibernation operation to be triggered, but assumes that the load and hibernation currents are constant. Moreover, if only one non-volatile memory block is used for snapshot, a power loss during a snapshots is likely to result in the loss of all data up to that point. If two memory blocks are used (and the system alternates between saving to each of them), see *Mementos* [18], or if the power loss occurs before a snapshot starts, all data since the last successful snapshot will be lost. If there is a "repetitive" power failure caused by the dynamics of the power supply (i.e. a sinusoidal waveform input) and the load behavior, the system could repeatedly fail to save state and get 'stuck'. This means that a substantial amount of computation time can be lost from an incomplete or missed snapshot. To minimize the loss of computation, it is necessary for the system to operate conservatively or adaptively in response to the power supply dynamics.

The dynamics of a power source can be learned to predict the moment at which power is likely to be lost. However, given the natural variability of energy harvesting this would be imperfect, resource-intensive, and would incur significant energy and computation cost. We address this by designing *Hibernus++* to operate with the conservative assumption that the incoming power may drop to zero at any time, and so the system should be able to hibernate using only the energy stored in the system's internal capacitance. To enable this, a calibration routine is used to determine the hibernation threshold, V_H. An interrupt is configured to cause a snapshot to be saved when the supply voltage drops below this threshold, i.e., saving state once per power interruption. Moreover, this calibration strategy makes the hibernate transparent and portable across multiple systems by adapting V_H at run-time considering the decoupling capacitance.

Figure 5 shows the self-calibration routine, which is used to determine V_H. It waits for the supply voltage to reach the calibration start voltage (V_{cal}). Once this voltage is reached, the harvesting source is disconnected or short-circuited by closing the switch in Figure 6, and a complete snapshot is saved to non-volatile memory. At this stage, in the worst case scenario every significant peripheral should be enabled to take into account their power consumption.

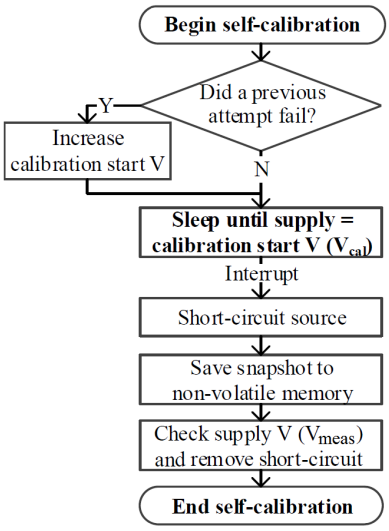


Fig. 5. Self-calibration of the hibernation voltage threshold, V_H

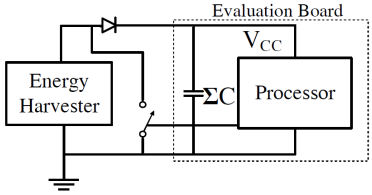


Fig. 6. Hibernation self-calibration circuit

The drop in supply voltage due to hibernation (the process of storing the snapshot) is given by $V_{cal} - V_{meas}$, where V_{meas} is the voltage measured at the end of the hibernation process. To ensure that the microcontroller has sufficient time for hibernation before the voltage drops below the minimum operating voltage V_{min}, the hibernation threshold is set as

$$V_H = V_{min} + (V_{cal} - V_{meas}) \tag{2}$$

If selective snapshots were also utilized [14], simple modulation of V_H by the fraction of state changed could be performed, hence minimizing the time required to hibernate.

If the calibration does not succeed, the capacitance C is not large enough to allow hibernation (equation 3). Therefore, extra capacitance needs to be added to the system.

$$C_{min} \geq \frac{I_h \cdot T_h}{(V_{max} - V_{min})} \tag{3}$$

Equations 2 and 3 are derived based on the assumption that the current drawn is approximately constant across the range of supply voltages from V_{min} to the microcontroller's maximum voltage V_{max}. To investigate the validity of this assumption, the current draw of a TI MSP430FR5739 microcontroller between 2.0 and 3.6 V was measured, and found to vary by less than 10%. In our experimental setup, V_{cal} is initially set to V_{MCUon} (microcontroller on), while the microcontroller is switched off once V_{dd} drops below V_{MCUoff} (microcontroller off). For the MSP430FR5739, the typical values for V_{MCUon} and V_{MCUoff} are 1.94V and 1.88V, respectively.

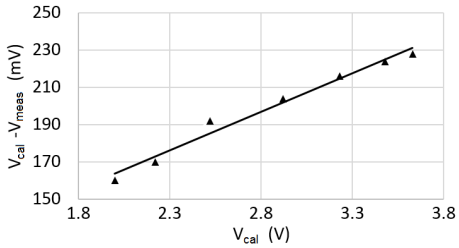


Fig. 7. Calculated voltage drop (V_{cal} - V_{meas}) with different V_{cal}

If the calibration routine fails, V_{cal} has to be increased (see Fig. 5). V_{cal} is first set as low as possible ($V_{cal} = V_{MCUon}$) to determine the lowest value of V_H. Therefore, an increased V_{cal} results in a higher V_H than necessary (see Figure 7).

C. Restore/Wake-up Strategy and Triggering

As with the calibration routine, the restore strategy is also important in order to adapt the system to the dynamics of the energy harvesting source. We present a restore strategy that classifies the type of input source at run-time and enables the system to adapt, taking full advantage of the available source. To determine this optimal restore point, we first consider the system as a transducer, which extracts energy from the environment and uses this energy to directly power the load without any energy storage. This system can operate at time t when

$$P_s(t) \geq P_c(t) \tag{4}$$

where $P_s(t)$ is the power output from the energy source at time t and $P_c(t)$ is the power consumed at that time. However, with small energy storage (e.g. on-board decoupling capacitance), the above equation expands to

$$\int_0^T P_s(t)dt \geq \int_0^T P_c(t)dt + Eo \qquad \forall\ T \geq 0 \tag{5}$$

where Eo is the initial energy stored in the on-board decoupling capacitance. The system tests the supply, using a short segment of code, and classifies it as either:

- *High-power* ($P_s(t) \geq P_c(t)$): the energy harvester is able to supply enough power to sustain the operation of the microcontroller in active mode (high $P_c(t)$). An example of this is an AC-output harvester such as a wind energy harvester; these usually supply large amounts of power but in short bursts.
- *Low-power* ($P_s(t) < P_c(t)$): the energy harvester is unable to supply enough power to directly run the microcontroller in active mode. Small photovoltaic cells operating from indoor light belong to this category.

If the source is classified as 'high-power', the system will restore immediately to take advantage of the abundant power. Conversely, if the supply is classified as 'low-power', and the system tries to restore immediately after the supply crosses the minimum voltage, the power drawn by the microcontroller will result in the supply dropping below this minimum value causing it to hibernate. This will cause repeated cycling between

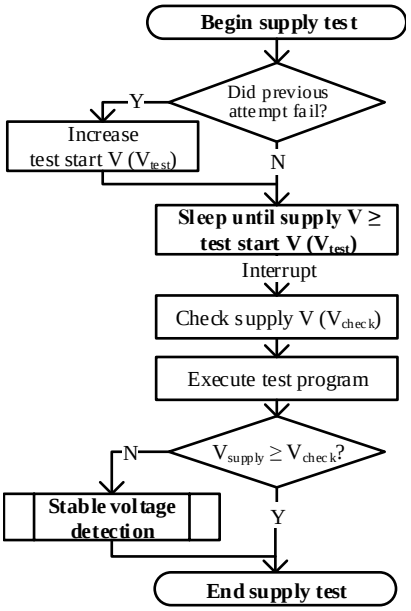


Fig. 8. The *Hibernus++* supply classification procedure

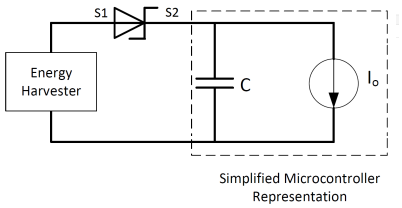


Fig. 9. Schematic of the system architecture for our simplified model

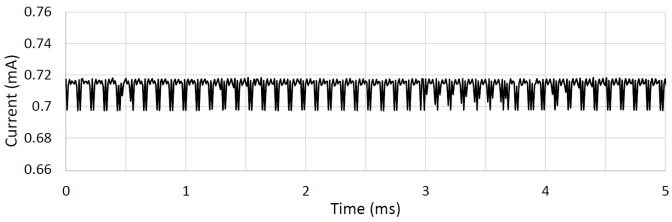


Fig. 10. Current consumption during execution of FFT algorithm at 2.5V

hibernate and restore operations, wasting useful operating time. To avoid this, in the 'low-power' state, we allow the voltage across the decoupling capacitance to charge to a higher voltage before restoring.

The supply test process illustrated in Figure 8 highlights the classification process. The system sleeps until an interrupt is triggered when the supply voltage rises above the classification start voltage, V_{class}. After this, the system logs the supply voltage (V_{check}) and executes a short reference segment of code[1] The voltage is checked again on completion. If $V_{supply} \geq V_{check}$, the harvester is supplying at least as much power as is being consumed by the microcontroller in active mode; the source is classified as a 'high-power' source, and the microcontroller is allowed to restore. Alternatively, if $V_{supply} < V_{check}$ at the end of the test, then the system classifies the source as 'low-power', and enters the 'stable voltage detection' process. During this process, the system waits until the rate of increase of the supply is below a given threshold, implying that there is no benefit in waiting longer to charge the capacitance any further. To do this, the system sets up two separate interrupts: (1) to detect increasing voltage, and (2) to act as a time-out. As the supply voltage continues to increase, the system resets the timer; when the voltage stops increasing, the timer interrupt intervenes. This allows the system to detect when the capacitance has stopped charging. The system will then restore.

IV. Mathematical Analysis

In this section, we model a microcontroller's behavior to estimate the energy overheads associated with hibernation and restore operations. Figure 9 represents the example system architecture, where the EH output is half-wave rectified and used to power an autonomous device, represented as a constant current sink. This architecture is implemented later in this paper (Sec. V), and parameters from this hardware platform are used in this analysis. In Figure 9, the microcontroller input (S2) is connected to the output of the energy harvester (S1) through a diode, which prevents back-flow of charge to the harvester. Figure 10 plots the current drawn by the microcontroller during the execution of a test case, which represents a common long-running task for energy harvesting systems: a Fast Fourier Transform (FFT) analysis of three arrays, each holding 128 8-bit samples of accelerometer data. The maximum current variation is 20 μA which is less than 3% of the mean current of 0.715 mA. For all our analysis in this section, this variation is ignored and the microcontroller is represented using a constant current load for a given application, as shown in Figure 9. This may be considered an artificial workload and, in practice, real workloads can exhibit greater variation. However, this model represents a simplified transient computing system, which allows the exploration of our approach under controlled conditions. In Section V we explore our approach with both synthesized and real sources and compare results with this model.

A. Energy Overhead of Hibernation and Restore

The *Hibernus++* algorithm is intended to sustain computation despite power interruption. Initial modelling and controlled experiments at the start of Section V use sinusoidal signals as a proxy for a transient source with controllable ON-time and regularity of interruption, before subsequently validating using real synthesized harvester performance. To estimate the energy overhead, f_{source} denotes the frequency of a sinusoidal signal powering the microcontroller. The time for which the microcontroller is active is the sum of the time to charge the capacitance to its peak value and the time to discharge. The charging time is approximately

$$t_{charge} = t_2 - t_1 = \frac{1}{4 \cdot f_{source}} - \frac{\sin^{-1}\left(\frac{V_R}{V_{max}}\right)}{2\pi \cdot f_{source}} \quad (6)$$

[1]This segment of code consists of a mock restore, which copies 100 bytes from FRAM to RAM. The number of bytes is determined by two factors: time and ADC resolution to capture correctly a voltage increment or decrement.

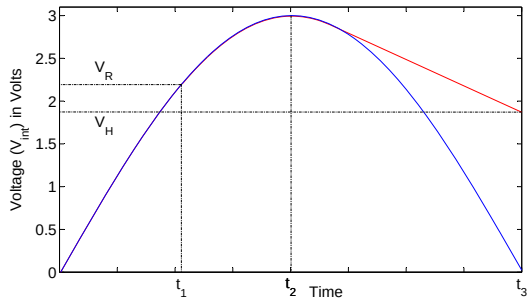


Fig. 11. Behavior with a transient input (in blue). The response is shown in red, which is the voltage measured after the schottky diode in Figure 9

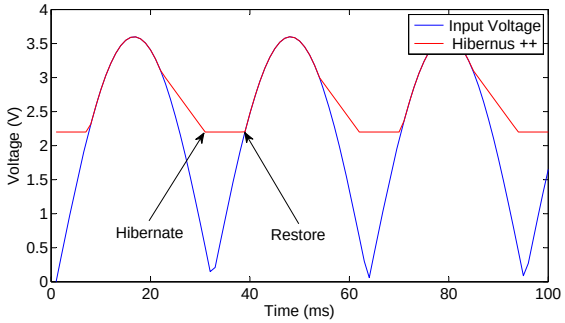


Fig. 12. Microcontroller response to a fully rectified sinusoidal signal

where the first term in the above equation is the rise time of the input source to its peak value V_{max}. The second term, which represents the time the input signal takes to reach the restore voltage V_R, is discounted from the first term to highlight the fact that the microcontroller starts its operation when the supply voltage reaches the threshold V_R. Assuming a constant current sink model, the discharge time is given by

$$t_{discharge} = t_3 - t_2 = \frac{C\left(V_{max} - V_H\right)}{I_O} \tag{7}$$

where V_H is the hibernate threshold. The total on-time of the microcontroller using this input source of frequency f_{source} is

$$t_{ON} = t_3 - t_1 = \frac{1}{4 \cdot f_{source}} - \frac{\sin^{-1}\left(\frac{V_R}{V_{max}}\right)}{2\pi \cdot f_{source}} + \frac{C\left(V_{max} - V_H\right)}{I_O} \tag{8}$$

Let T_{app} denote the uninterrupted execution time of an application powered by a constant voltage source. If this application is executed by a microcontroller powered using a full-wave rectified source of frequency f_{source}, the execution time is extended by an amount Δt, which depends on the number of times the microcontroller hibernates and restores in this interval. The microcontroller's response to this full-wave rectified signal is shown in red in Figure 12. The flat line in Figure 12 is due to our model assuming zero power consumption and capacitor leakage in low power mode.

As seen from Equation 8, during the application execution of duration t_{ON}, the microcontroller restores and hibernates once. The number of restores and hibernates in the entire application execution duration is given by

$$N_r = N_h = \lceil \frac{T_{app}}{t_{ON}} \rceil \tag{9}$$

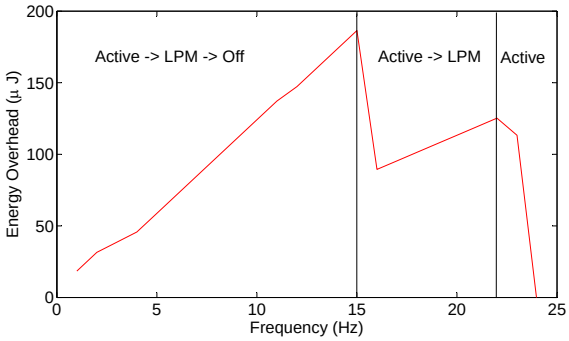


Fig. 13. Energy overhead of hibernate/restore, at various input frequencies

The energy overhead is given by

$$E_{overhead} = N_r \cdot t_r \cdot P_r + N_h \cdot t_h \cdot P_h \tag{10}$$

where t_r and P_r represent the time taken and power consumption for restoring a snapshot, respectively. Similarly, t_h and P_h represent the time taken and power consumption for storing a snapshot, respectively. From equations 8-10 it can be seen that, as the frequency of the input source increases, there is an increase in the number of restores and hibernates leading to an increase in the energy overhead. However, with an increase in frequency, the time period of the input source decreases. After a certain frequency, the capacitance starts charging before the voltage across it drops below V_H. This causes the microcontroller to be continually on, reducing the energy overhead of restore and hibernate to zero. To find this break-even frequency beyond which the microcontroller is continually on, we consider the time from the peak of one half-wave pulse to the restore threshold of the next pulse. This interval is given by

$$t_{interval} = \frac{1}{4 \cdot f_{source}} + \frac{\sin^{-1}\left(\frac{V_R}{V_{max}}\right)}{2\pi \cdot f_{source}} \tag{11}$$

where the first term is the time for the input source to drop from V_{max} to 0 and the second term is the time for the input source to rise from 0 to V_R. The microcontroller will be always on when the discharge voltage during this interval is greater than the restore threshold V_R i.e.,

$$V_{max} - \frac{I_O \cdot t_{interval}}{C} \geq V_R \tag{12}$$

Figure 13 plots the energy overhead for the microcontroller as the frequency of the input source is varied from 1 Hz to 25 Hz, covering the range of source frequencies typically generated using a micro wind turbine. As can be seen, the energy overhead first increases with the frequency (as discussed before). The model indicates that, at frequencies above 15 Hz, the microcontroller never drops below V_{min} after hibernation, thus doesn't need to restore. This results in a sudden drop in the energy overhead. It is important to note that, when the microcontroller is alternating between active and low-power modes, the energy overhead due to hibernation is significant and therefore rises with increasing input frequency. Above 23 Hz, the microcontroller voltage never drops below V_H, meaning that the system is permanently powered-on, and reduces the energy overhead of hibernation/restore to zero.

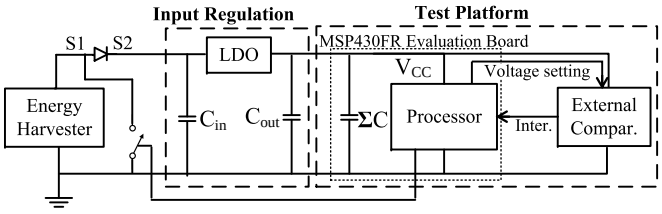


Fig. 14. Schematic of the test platform and input regulation circuitry

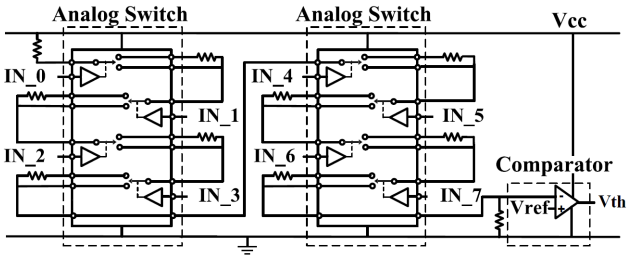


Fig. 15. Schematic of the external comparator circuitry

V. Practical Validation

The system has been validated with both synthesized and real EH sources, allowing the overheads of the scheme to be verified and compared against the state-of-the-art techniques, *Mementos*, *Hibernus* and *QuickRecall* [23].

Mementos places static checkpoints after function calls or before loops, referred to as "function" and "loop". For a fair comparison, our implementation of *Mementos* saves the complete system state (rather than a limited subset) to non-volatile memory. *QuickRecall* uses a lightweight, in-situ checkpointing technique, exploiting FRAM as unified memory.

The experimental set-up is shown in Figure 14. The test platform uses a Texas Instrument MSP-EXP430FR5739 microcontroller [24], which is chosen because of its low-power features as well as its built-in FRAM memory. The EH output is passed through a Schottky diode: for DC-output EHs, this acts to prevent the back-flow of current; for AC-output harvesters, this acts to rectify their output. This is then passed through a low drop-out (LDO) voltage regulator, which limits the maximum supply voltage to protect the microcontroller.

The system depends on an external comparator circuit (Figure 15) which enables interrupts to be triggered when the supply voltages surpasses threshold set by the microcontroller. This additional circuit has 8 digital inputs to set the voltage threshold, and one digital output. It is based on a comparator (with built-in 1.18 V reference), two analog switches, and a bank of resistors. This additional circuit is powered by the energy harvester, and the external comparator has a propagation delay of 5μs. For this platform, this provides correct behaviour for source currents $<$ 2.4A; currents of this magnitude and greater are not practical in energy harvesting systems of this scale. It draws 1.0 μA at 2.0 V; the power consumption is over an order of magnitude lower than the microcontroller's built-in comparator and reference circuits.

Mementos uses an ADC to measure voltage when making checkpointing decisions, comparing V_{cc} to a threshold voltage (V_m). *Mementos* is not disadvantaged through the use of the ADC rather than our external comparator circuit (we chose it to replicate their approach [18]): FFT execution at 2.8V with the ADC circuitry consumes 0.996 mA, while with the external comparator consumes 0.997 mA. Above V_m, *Mementos* assumes that it does not need to save a snapshot; a voltage lower than V_m is an indicator that a power failure is imminent and a snapshot needs to be saved. For *Mementos*, the checkpoint threshold can be calculated considering a constant current draw I so that the time Δt between two voltage levels V and V_{min} is $\Delta t = C(V - V_{min})/I$. In this specific case, an MSP430 draws $\sim$ 0.8mA in active mode, fails to write a snapshot to FRAM below 1.9V, and needs 1.4ms to write a snapshot, so that *Mementos* should start check-pointing at latest when supply falls to 2.1V. However, *Mementos*' ability to precisely save a checkpoint depends on the distribution of the trigger points, which estimate available energy. So we set V_m higher than $2.1V$ i.e., at $V_m = 2.4V$, assuming that no energy will be harvested between a trigger point and a power failure. However, *Mementos* is more stable with higher V_m, but the performance decreases due to the large number of snapshots, as shown in Table I.

TABLE I
Mementos performance with two different values of V_m

FFT with Mmtos. (function)	Vm=2.4V		Vm=2.8V	
Sine wave frequency (Hz)	N° C'point	N° Restore	N° C'point	N° Restore
2	27	0	27	0
4	27	1	28	1
6	29	2	34	3

For a fair comparison, our implementation of *QuickRecall* uses the same external comparator utilized for *Hibernus++* (Figure 15), configured with a trigger voltage V_{trig} of 2.03V and a hysteresis of 100mV for restoring.

A. Comparing Hibernus++ with Mementos for ideal sources

Figure 16(a) compares the number of hibernations/checkpoints executed by *Hibernus++* and *Mementos* during the execution of the case study FFT algorithm. A range of supply frequencies (2-20 Hz, and DC) were chosen to represent the intermittent power output that may be expected from a high-power EH source. We did not consider frequencies higher than this because the decoupling capacitance of our system meant that V_{CC} never decays below V_{min}, and hence transient operation is redundant.

As can be seen, *Hibernus++* modulates the number of times that snapshots are executed as a function of the supply interruption frequency and hence does not perform any redundant checkpoints (i.e. exhibits ideal behaviour). *Mementos* executes a static number of checkpoints (12 and 27 times), although some are repeated when $V_{cc} < V_{min}$ during a snapshot. It is important to note that *Mementos* operates unstably at frequencies higher than 6 Hz due to the static and uneven placement of checkpoints at compile time: checkpoints are only inserted at function calls or loops. In cases where the supply is interrupted in the period between a restore and the next snapshot being saved, the system can get 'stuck',

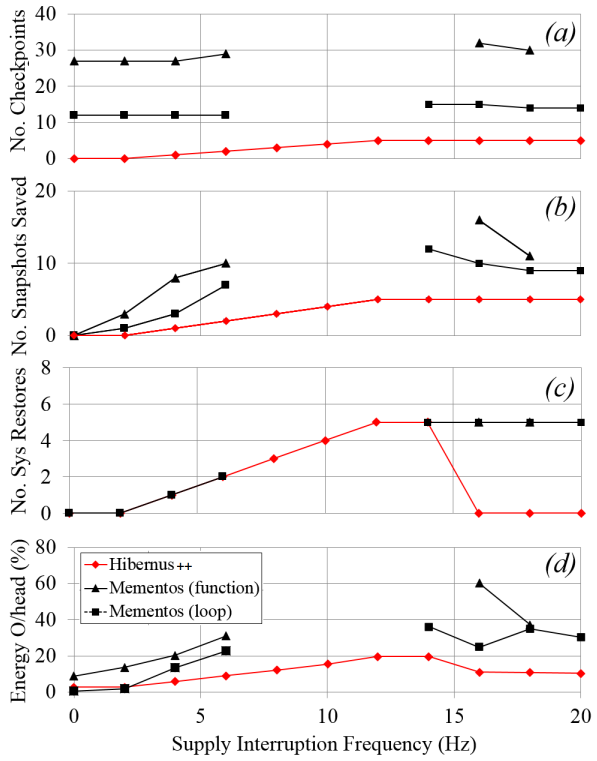


Fig. 16. Comparison of *Hibernus++* with *Mementos* and *QuickRecall*, showing performance when executing an FFT (averaged over 3 executions): (a) number of checkpoints made, (b) number of snapshots saved, (c) number of snapshots restored, (d) energy overhead

i.e. executes the same portion of code from the last saved checkpoint before $V_{cc} < V_{min}$ without reaching or being able to save a snapshot at the next checkpoint. As a result, the data for these frequencies are missing for the *Mementos* approach in the above Figure 16.

Figure 16(b) compares the number of snapshots that are saved by *Hibernus++* and *Mementos*. *Hibernus++* saves a snapshot every time the hibernate routine is executed without making any redundant checkpoints, while *Mementos* saves a snapshot only when $V_{cc} < V_{min}$. The number of snapshots with *Mementos* depends on the checkpoint placement, the value of V_{min} and the supply interruption frequency; while for *Hibernus++* it only depends on the supply interruption frequency. Figure 16(c) shows that *Hibernus++* and *Mementos* complete execution of the FFT over the same number of power interruptions at frequencies lower than 14 Hz. However, frequencies higher than 14Hz, *Hibernus++* will always stay in ON mode (MCU always ON), alternating between active and low-power states, so that the number of restores will always be zero and hence does not perform any redundant restores. This is because of the on-board decoupling capacitance which is big enough to maintain the system ON after saving a snapshot. This can also be seen from Figure 16(d), which compares the energy overheads of running *Hibernus++* and *Mementos*. The energy overhead of *Hibernus++* is always lower than *Mementos* (both function and loop mode) by average 16%. With frequencies higher than 14Hz, *Hibernus++* significantly outperforms *Mementos* by achieving an average 26% energy saving.

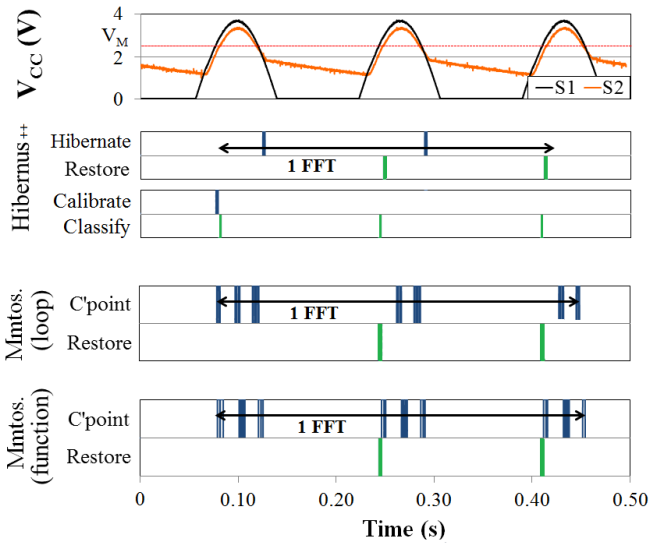


Fig. 17. Experimental results showing basic hibernate/restore operation

TABLE II
COMPARISON OF EXECUTION TIME FOR THE FFT APPLICATION

Input Frequency (Hz)	Mementos (loop) Active Time (ms)	Mementos (loop) Time O/head (%)	Mementos (function) Active Time (ms)	Mementos (function) Time O/head (%)	Hibernus++ Active Time (ms)	Hibernus++ Time O/head (%)
0 (DC)	104.4	4.4	108.8	8.8	103.2	3.2
2	106.0	6.0	114.4	14.4	103.2	3.2
4	118.3	18.3	122.4	22.4	107.0	7.0
6	128.8	28.8	133.8	33.8	110.7	10.7
8	-	-	-	-	114.5	14.5
10	-	-	-	-	118.2	18.2
12	-	-	-	-	122.0	22.0
14	145.2	45.2	-	-	122.0	22.0
16	133.2	33.2	165.4	65.4	110.2	10.2
18	142.8	42.8	141.1	41.1	110.2	10.2
20	138.2	38.2	-	-	110.2	10.2

Figure 17 illustrates the system behaviour with a 6Hz supply interruption frequency. Signals `S1` and `S2` on this figure refer to the unrectified and rectified supply inputs, respectively. The other parts of the figure compare the operation of *Hibernus++* against *Mementos* (loop and function). For *Hibernus++*, the figure demonstrates hibernate, restore, calibration and classification times. For *Mementos*, the figure shows checkpoints and restores only. It is to be noted from the figure that *Hibernus++* self-calibrates only once, and classifies the source as either low- or high-power after each interruption.

Table II reports the execution time of *Hibernus++* in comparison with *Mementos* for the FFT application. The FFT execution itself takes 100 ms using a DC source. The table reports the execution time for a range of input supply frequencies, including the result corresponding to a DC source. As can be seen from the table, the execution time for the FFT increases with the input frequency for *Mementos*. The increase in execution time ranges from 5-45% for *Mementos* (loop), and 9-65% for *Mementos* (function). In comparison to these, *Hibernus++* increases execution time by only 3-22% . The overall improvement of this approach with respect

TABLE III
COMPARISON BETWEEN HIBERNUS (V_H=2.17V AND V_R=2.27), QUICKRECALL (V_{trig}=2.03V), AND HIBERNUS++ WITH DYNAMIC V_H

Decoupling capacitance ΣC (µF)	Hibernus			QuickRecall			Hibernus++			
	N. Restore	N. Hibern.	Total Time (ms)	N. Restore	N. C'point	Total Time (ms)	N. Restore	N. Hibern.	Total Time (ms)	V_H (V)
10	-	-	-	-	-	-	2	2	395.2	2.03
20	2	2	376.3	2	2	370.1	2	2	389.4	1.97
30	2	2	376.1	2	2	370.0	1	1	243.7	1.93
40	2	2	376.0	2	2	369.9	1	1	238.9	1.91

Using a voltage input of 3V @ 6Hz

TABLE IV
EXPERIMENTALLY MEASURED PARAMETERS

Synthesized Sources	MCU ON (ms)	FFT (ms)	Low power (ms)	∑ Test (ms)	Calibration (ms)	∑ Restore (ms)	∑ Hibernate (ms)	Time O/head (%)	Energy O/head (%)
Wind Turbine	173	100	40.3	9.0	2.2	10.8	11.2	73.5	28.0
Kinetic	156	100	45.3	3.0	2.2	2.7	2.8	56.0	9.4
PV	511	100	401	1.0	2.2	0.0	7.0	411.0	13.4
Input Current 200uA	548	100	428	1.0	2.2	0.0	16.8	448.0	21.1

to *Mementos* is on average 13% (1-23%) and 17% (5-33%) compared to the loop and function modes respectively.

B. Comparing Hibernus++ with Hibernus and QuickRecall for ideal sources

Table III shows the total time taken by our earlier proposed approach, *Hibernus*, *QuickRecall*, and our current approach, *Hibernus++*, executing the FFT application (using an external sinusoidal signal of 3V input at 6Hz frequency). Results are reported for four different values of decoupling capacitance. The hibernate and restore thresholds for *Hibernus* are manually characterized for a constant decoupling capacitance of 20μF, and therefore the execution time of *Hibernus* is lower than *Hibernus++* for this capacitance due to the overhead of the automatic calibration (2.2ms) and classification (1ms). Similarly, *QuickRecall* sets the trigger voltage for a constant decoupling capacitance. *Hibernus++* self calibrates the hibernate and restore thresholds dynamically, resulting in a lower FFT execution time than *Hibernus* and *QuickRecall* for other capacitance values. *Hibernus++*'s efficiency improvements outweigh the overheads. It is important to note that *Hibernus* and *QuickRecall* do not work for capacitance values lower than the one it is designed for (20μF in this case). So they are not able to execute the FFT application with less than 20μF. *Hibernus++* calibrates the hibernate and restore thresholds dynamically based on the value of decoupling capacitance, and therefore is able to execute the FFT application for all capacitance values (subject to meeting Equation 3).

QuickRecall only relies on the use of a processor with unified FRAM memory. However, by utilizing this type of memory, the system introduces a significant energy overhead. Figure 19 shows the energy overhead for *QuickRecall* and *Hibernus++*, with the FFT application, as a function of the supply interruption frequency. *QuickRecall* has a higher energy overhead due to the higher current consumption in active mode. However, the energy overhead of *Hibernus++* due to the restore/hibernate strategy has a greater impact as the frequency increases. At frequencies higher than 14Hz, *Hibernus++* significantly outperforms *QuickRecall*.

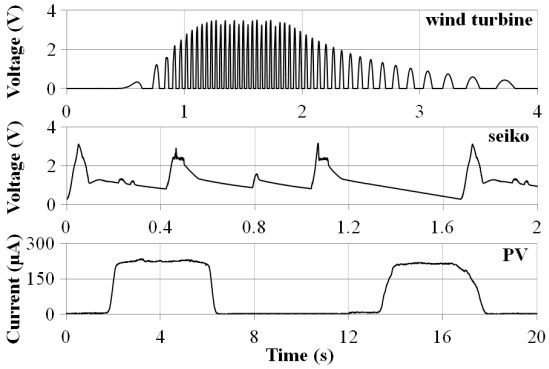


Fig. 18. Three synthetic traces of real harvesters used in Section V-B

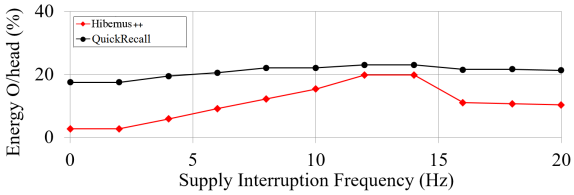


Fig. 19. Energy overhead comparison between *Hibernus++* and *QuickRecall*

C. Results with Synthesized Energy Harvesters

Table IV shows experimentally obtained values for synthesized EHs (Figure 18): a wind turbine, a wearable kinetic watch (Seiko watch), a micro PV and a constant current source. Due to limitations of the power analyser (which captures power traces and allows them to be replayed as a synthesized source), we could only collect 20 s of indoor PV behaviour during which lights are turned on and off twice. These traces were obtained from real EHs and replayed via a source-measurement unit. It shows the time and energy overhead with each scheme powered by these sources, confirming that *Hibernus++* modulates its behavior dependent on the dynamics of the EH. In particular, the wind turbine and the kinetic harvesters have been classified as high-power sources while the micro PV and the constant current source have been classified as low-power sources. In the first case (wind-turbine and kinetic) the system behaves as already shown with the sinusoidal sources (see Figure 17), while in the second case (micro PV and constant current source) the system V_{cc} never drops below V_{min}. This means that it only needs to classify the source once, and never needs to restore its state, despite the increase in the time spent in low-power mode.

Table V reports measured data to compare the earlier proposed *Hibernus* and the current work *Hibernus++* for the three synthetic traces of real harvesters in Figure 18. Results are compared in terms of the number of hibernations, number of restores and the total execution time. As can be seen from this table, *Hibernus++* results in fewer restores and hibernations than *Hibernus*, resulting in a reduction of execution time of

TABLE V
COMPARISON BETWEEN HIBERNUS AND HIBERNUS++ USING SYNTHETIC HARVESTERS

Synthesized source	Hibernus			Hibernus++		
	N. Restore	N. Hibernate	Total Time (ms)	N. Restore	N. Hibernate	Total Time (ms)
Wind Turbine	7	7	**584.9**	8	8	**512.6**
Kinetic	6	6	**3399.0**	2	2	**2145.0**
PV	-	-	**-**	0	5	**582.4**
Input Current 200uA	-	-	**-**	0	12	**806.8**

36%. On the other hand, for the wind turbine the number of restores and hibernations using *Hibernus++* are higher than the *Hibernus* approach; however, there is still an improvement of 12% in execution time. This is due to the execution time improvement using dynamic thresholds in *Hibernus++* as compared to fixed thresholds in *Hibernus*. It is also important to note that *Hibernus* is not able to sustain operation with sources having very high internal resistance (e.g. PV and constant current source), which causes fast voltage drops when a load is applied. This is because, every time the voltage across the decoupling capacitance increases above the micocontroller minimum voltage, the microcontroller is turned on to restore a snapshot, draining more current and bringing the voltage below the minimum. *Hibernus++*, on the other hand, first checks if the power source is high enough to sustain a full restore before actually restoring it. By waiting for the supply to reach a safe voltage level before continuing operation, *Hibernus++* is able to execute the FFT with different energy harvesting sources.

D. Results with Real Energy Harvesters

Finally, *Hibernus++* has been verified operating directly from real EHs: a micro-wind turbine (high-power source) and a micro photovoltaic module (low-power source). Figure 20 shows the activity of the system when powered by a real wind harvester. The operating parameters of the system are shown: hibernate and restore operations, the calibrate and classify operations, and the time for the FFT execution and the system in ON mode. As already shown in Figure 17, the system saves and restores a snapshot once per interruption. Moreover, it classifies the source (as low-power or high-power) once per interruption, while it only self-calibrates once (at the beginning). The total time for executing the FFT is approximately 440ms while the system is on for 225ms. During this time, the system saves and restores 10 snapshots.

Figure 21 shows the activity of the system when powered by a real PV module. This illustrates the behavior of *Hibernus++* with a current source. In this case, the system is always on, alternating between low-power and active modes. The system calibrates and classifies the source only once (at the beginning) and it never restores. The total FFT execution time is 670ms, and it saves five unused snapshots (although it never needs to restore). This is the first case considered where *Hibernus++* makes unnecessary snapshots, and therefore does not exhibit

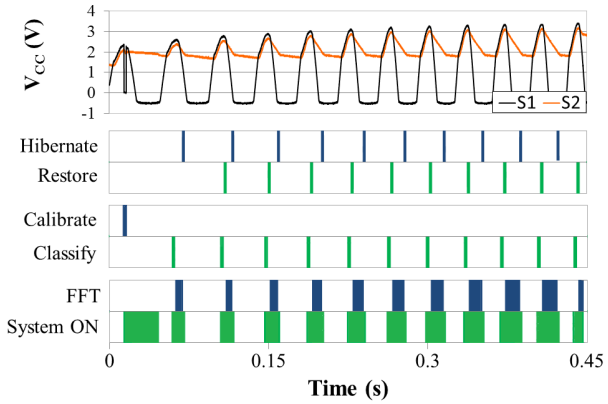


Fig. 20. System activity powered by a real wind harvester

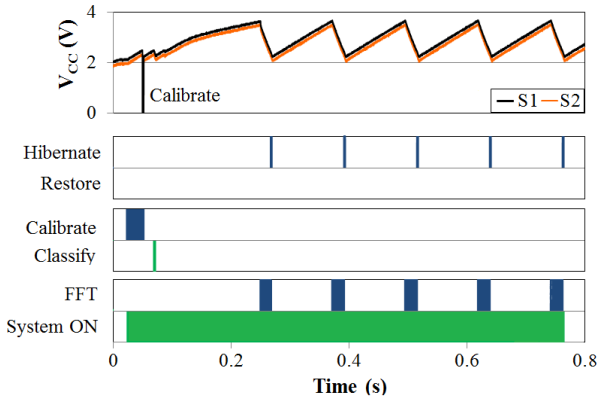


Fig. 21. System activity with high-current input from real PV module

improved behaviour. This has occurred because, after hibernating, the power consumed is less than that harvested, and V_{CC} recovers without state being lost (i.e. a restore required).

VI. ENABLING ULTRA-LOW CURRENT OPERATION

The system cannot start up reliably with supply currents below 100 μA. This is because the microcontroller draws high levels of current when its supply voltage is below its V_{min} and slowly ramping up. Several techniques were explored to mitigate this effect. External supervisory circuits to hold the microcontroller in reset until $V > V_{min}$ were ineffective, as the current draw in reset was found to be substantial. Instead, a cold-start circuit (Figure 22) has been developed which can reliably start the system with lower current levels. This cold-start circuit guarantees a reliable start by detecting the input voltage and only turning on the supply to the microcontroller when its input voltage is above a threshold (V_{in-H}), and switching it off when the voltage drops below a minimum voltage (V_{in-L}). In practice, for reliable operation, V_{in-L} must be slightly higher than V_{min}. This is enabled by a pair of microcurrent voltage monitors, which are configured in a MOSFET latch arrangement. As with the schematic in Figure 14, it incorporates an LDO voltage regulator to limit the supply voltage to the microcontroller, and the harvester short-circuit arrangement for the self-calibration routine. These extra components draw 2 μA at 2V.

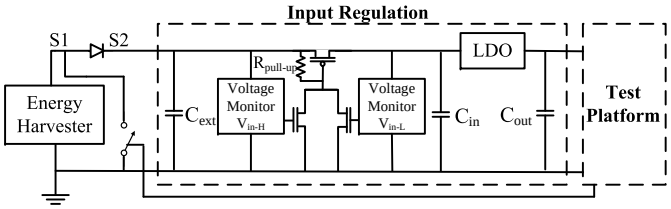


Fig. 22. Schematic of the cold-start arrangement, to allow ultra low-current start-up. The 'Test Platform' is as shown in Figure 14

TABLE VI
EXPERIMENTALLY MEASURED PARAMETERS

Input Current (uA)	MCU ON (ms)	FFT (ms)	Low power (ms)	∑ Test (ms)	Calibration (ms)	∑ Restore (ms)	∑ Hibernate (ms)	Time O/head (%)	Energy O/head (%)
70 (*)	3430	100	3323	1.0	2.2	0.0	4.2	3330.0	46.8
200 (*)	850	100	744	1.0	2.2	0.0	2.8	750.0	14.1

(*) Using the external start-up circuit

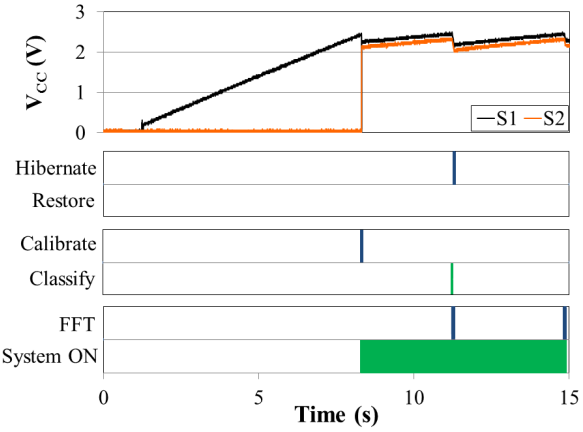


Fig. 23. Result: Activity of system with 30μA input from real PV module

A complication of this scheme is that the microcontroller platform used for our validation has a substantial level of decoupling capacitance (approximately 16 μF). If there is insufficient capacitance on the input, or hysteresis between V_{in-H} and V_{in-L}, the system will oscillate as it will not be able to power the microcontroller for long enough to allow it to initialize and enter a low-power mode. Therefore, the cold start circuit incorporates additional capacitance and two voltage detectors set to provide hysteresis between V_{in-H} and V_{in-L}. Because of this additional capacitor, the time-overhead will be higher compared to the system without any cold-start circuitry and it depends on the input current. This can be seen in Figure 23, which shows the system working with a very-low input current (30 μA). However, the energy overhead will be improved. Table VI shows experimentally obtained values for a constant current source. In particular, it shows that the system can start reliably also with currents below 100 μA (in this case 70 μA). Moreove, it shows that, with a current input of 200 μA (see Table II), the energy overhead has improved using the cold-start circuit.

VII. CONCLUSION

A new approach for sustaining computation during intermittent supply, *Hibernus++*, has been proposed. This manages transient computation dynamically, using different energy harvesting sources and intelligently adapts the hibernate and restore thresholds in response to system properties and dynamics. This allows a new class of embedded systems - "transient computing systems" - to sustain computation through power outages, which are common in energy-harvesting systems, and to adapt their behaviour. This allows operation without using any external energy buffer. The system has been validated with both synthesized and real EH sources, demonstrating experimentally that it has a lower energy and time overhead than recently proposed approaches. This contributes to the development of future energy harvesting transient systems.

REFERENCES

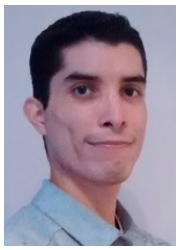
[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
[2] A. Sinha and A. Chandrakasan, "Dynamic power management in wireless sensor networks," *IEEE Design & Test of Computers*, 2001.
[3] H. Jayakumar, K. Lee, W. S. Lee, A. Raha, Y. Kim, and V. Raghunathan, "Powering the internet of things," in *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*. ACM, 2014, pp. 375–380.
[4] J. A. Khan, H. K. Qureshi, and A. Iqbal, "Energy management in wireless sensor networks: A survey," *Computers & Electrical Engineering*, vol. 41, pp. 159–176, 2015.
[5] S. Beeby and N. White, *Energy harvesting for autonomous systems*. Artech House, 2014.
[6] S. Roundy and J. Tola, "Energy harvester for rotating environments using offset pendulum and nonlinear dynamics," *Smart Materials and Structures*, vol. 23, no. 10, p. 105004, 2014.
[7] K. Ylli, D. Hoffmann, A. Willmann, P. Becker, B. Folkmer, and Y. Manoli, "Energy harvesting from human motion: exploiting swing and shock excitations," *Smart Materials and Structures*, vol. 24, no. 2, p. 025029, 2015.
[8] "Crossbow telos mote datasheet:." [Online]. Available: http://www.willow.co.uk/TelosB_Datasheet.pdf
[9] P. D. Mitcheson, "Energy harvesting for human wearable and implantable bio-sensors," in *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*. IEEE, 2010, pp. 3432–3436.
[10] G. Acampora, D. J. Cook, P. Rashidi, and A. V. Vasilakos, "A survey on ambient intelligence in healthcare," *Proceedings of the IEEE*, vol. 101, no. 12, pp. 2470–2494, 2013.
[11] D. Balsamo, G. Paci, L. Benini, and B. Davide, "Long term, low cost, passive environmental monitoring of heritage buildings for energy efficiency retrofitting," in *Environmental Energy and Structural Monitoring Systems (EESMS), 2013 IEEE Workshop on*. IEEE, 2013, pp. 1–6.
[12] S. Naderiparizi, A. N. Parks, Z. Kapetanovic, B. Ransford, and J. R. Smith, "Wispcam: A battery-free rfid camera," in *RFID (RFID), 2015 IEEE International Conference on. IEEE*, 2015.
[13] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini, "Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems," *Embedded Systems Letters, IEEE*, vol. 7, no. 1, pp. 15–18, 2015.
[14] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Architecture exploration for ambient energy harvesting nonvolatile processors," in *High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 526–537.
[15] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *Software Engineering, IEEE Transactions on*, no. 1, pp. 23–31, 1987.
[16] H. Asano, "Flash non-volatile memory," Apr. 11 1995, US Patent 5,406,529.
[17] H. Kim, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Building fully functional instant on/off systems by making use of non-volatile ram," in *Consumer Electronics (ICCE), 2011 IEEE International Conference on*. IEEE, 2011, pp. 675–676.
[18] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on rfid-scale devices," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 159–170, 2012.
[19] H. Jayakumar, A. Raha, W. S. Lee and V. Raghunathan, "QuickRecall: A HW/SW Approach for Computing across Power Cycles in Transiently Powered Computers," in *J. Emerg. Technol. Comput. Syst.*, August 2015.
[20] H. Jayakumar, A. Raha and V. Raghunathan, "Hypnos: An ultra-low power sleep mode with SRAM data retention for embedded microcontrollers," in *Int'l Conf. Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 1-10, 12-17 Oct. 2014.

[21] M. Qazi, A. Amerasekera, and A. P. Chandrakasan, "A 3.4-pj feram-enabled d flip-flop in 0.13-cmos for nonvolatile processing in digital systems," *IEEE Journal of Solid-State Circuits*, 2014.
[22] S. Onkaraiah, M. Reyboz, F. Clermidy, J.-M. Portal, M. Bocquet, C. Muller, C. Anghel, A. Amara *et al.*, "Bipolar reram based non-volatile flip-flops for low-power architectures," in *New Circuits and Systems Conference (NEWCAS)*. IEEE, 2012, pp. 417–420.
[23] A. Rodriguez, D. Balsamo, A. Das, A. Weddell, D. Brunelli, B. Al-Hashimi, and G. Merrett, "Approaches to transient computing for energy harvesting systems: A quantitative evaluation," in *Int'l Workshop Energy Harvesting and Energy Neutral Sensing Systems (ENSsys)*, 2015.
[24] TI, "Msp430fr5739 fram experimenter board," 2013. [Online]. Available: http://www.ti.com/lit/ug/slau343b/slau343b.pdf
[25] K. Ma, X. Li, S Li, Y. Liu, J. J. Sampson, Y. Xie and V. Narayanan, "Nonvolatile Processor Architecture Exploration for Energy-Harvesting Applications," *in Micro, IEEE*, vol. 35, no. 5, pp. 32-40, Sept.-Oct. 2015.
[26] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, M. Chiang, Y. Yan, B. Sai and Huazhong Yang, "A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops," *in ESSCIRC (ESSCIRC), 2012 Proceedings of the*, pp.149-152, 17-21 Sept. 2012.
[27] S. C. Bartling, S. Khanna, M. P. Clinton, S. R. Summerfelt, J. A. Rodriguez and H. P. McAdams, "An 8MHz 75A/MHz Zero-Leakage Non-Volatile Logic-Based Cortex-M0 MCU SoC Exhibiting 100% Digital State Retention at VDD=0V with <400ns Wakeup and Sleep Transitions," *ISSCC*, pp. 432-433, 2013.
[28] N. Sakimura, Y. Tsuji, R. Nebashi, H. Honjo, A. Morioka, K. Ishihara, K. Kinoshita, S. Fukami, S. Miura, N. Kasai, T. Endoh, H. Ohno, T. Hanyu and T. Sugibayashi, "10.5 A 90nm 20MHz fully nonvolatile microcontroller for standby-power-critical applications," *in Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp.184-185, 9-13 Feb. 2014.
[29] D. Balsamo, A. Das, G. V. Merrett, A. S. Weddell, D. Brunelli, L. Benini and B. M. Al-Hashimi, "Graceful Performance Modulation for Power-Neutral Transient Computing Systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, to be published.

Domenico Balsamo received the Master degree in Electronics Engineering from University of Modena and Reggio Emilia, Italy, in 2008. He received his Ph.D. degree in computer engineering in the area of embedded systems from the University of Bologna, Italy, in 2015. He is currently a post-doctoral research fellow at the University of Southampton. His research interests are in high-performance and ultra-low power embedded systems.

Alex S. Weddell received the MEng (Hons) degree in Electronic Engineering (2005) followed by a PhD (2010) from the University of Southampton, where he is now Lecturer in Electronic and Electrical Engineering. He has authored over 30 papers, and has special interests in the areas of energy-aware systems and energy harvesting.

Anup Das received the B.Eng. degree in Electronics and Telecommunication Engineering from Jadavpur University, India, in 2004. He received the Ph.D. degree in computer engineering in the area of embedded systems from the National University of Singapore, in 2014. He is currently a post-doctoral research fellow at the University of Southampton. His research interests include reliability and energy-aware system architecture.

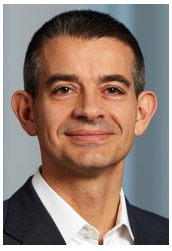


Alberto Rodriguez Arreola received his MSc. degree in System on Chip from the University of Southampton, in 2015. He is currently a PhD student in Electronics and Computer Science, University of Southampton. His research interests include energy harvesting and transient computing embedded systems.

Davide Brunelli (M'10) received the M.S. (cum laude) and Ph.D. degrees in electrical engineering from the University of Bologna, Italy, in 2002 and 2007, respectively. He has been an Assistant Professor with the University of Trento, Italy, since 2010. His research interests include smart grids and the development of new techniques of energy scavenging for wireless sensor networks and embedded systems.

Bashir M. Al-Hashimi (M'99-SM'01-F'09) is a Professor of Computer Engineering and Dean of the Faculty of Physical Sciences and Engineering at the University of Southampton, UK. He is ARM Professor of Computer Engineering and Co-Director of the ARM-ECS research centre. His research interests include methods and tools for low-power design and test of embedded computing systems. He has published over 300 technical papers, authored or co-authored 5 books and has graduated 33 PhD students.

Geoff V. Merrett (GSM06-M09) received the BEng degree (Hons) in Electronic Engineering and the PhD degree from the University of Southampton, UK, in 2004 and 2009 respectively. He was appointed as a Lecturer in energy-efficient electronic systems at the University of Southampton in 2008, and was promoted to Associate Professor in 2014. He has research interests in energy-efficient embedded systems and low-power pervasive computing, and has published over 100 scientific papers in journals and refereed conference proceedings.



Luca Benini is the chair of digital circuits and systems at ETHZ and a Full Professor at the University of Bologna. He has served as Chief Architect for the Platform2012/STHORM project in STmicroelectronics, Grenoble. Dr. Benini's research interests are in energy-efficient system design and Multi-Core SoC design. He is also active in the area of energy-efficient smart sensors and sensor networks for biomedical and ambient intelligence applications. He has published more than 700 papers in peer reviewed international journals and conferences, four books and several book chapters.