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

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

***Abstract*—Energy harvesters are increasingly being used to power autonomous systems, but their output power is typically variable and intermittent. To sustain computation, these systems typically 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. Modern systems are increasingly being powered directly from transient output from energy harvesters. To prevent an application from having to restart computation due to power cycles, we have previously developed *Hibernus*, an approach which allows these systems to hibernate when supply failure is imminent. Later, 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, *Hibernus++*, extends our earlier work to adapt the hibernate and restore thresholds in response to the dynamics of the energy harvesting source. Specifically, capabilities are build in the system to autonomously reconfigure the hibernation threshold to balance the energy consumption with the energy stored in the on-board decoupling capacitor. Similarly, the system auto-calibrates the restore threshold depending on the balance of the supply voltage and the consumption. *Hibernus++* is validated both theoretically and experimentally on a microcontroller hardware with a mix of synthesized and real energy harvesters. Results show that *Hibernus++* results in 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

THE RECENT momentum of 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 few 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, human motion or temperature differences [5]–[7].

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

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. D. Balsamo and L. Benini are 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

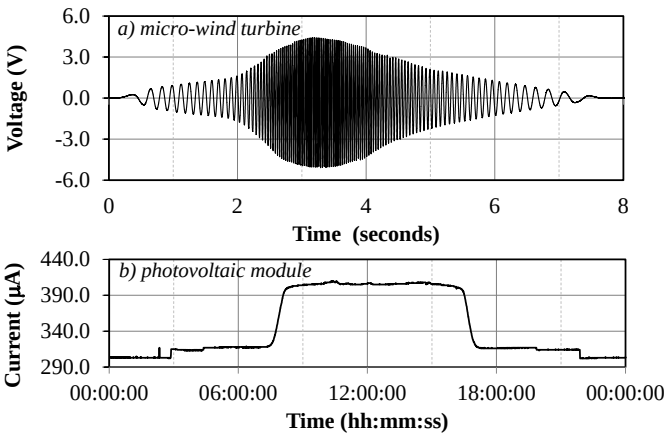

Fig. 1. Example outputs from energy harvesters: voltage of a wind energy harvester over one 8-second 'gust', and current from an indoor photovoltaic module over the same period.

The 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 (e.g. vibration or movement) or wind energy harvesters normally give an AC output relative to their 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 photovoltaic 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 seconds). On the other end, 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 the output of energy harvesters would result in repeated power-cycling, restarting program execution from the beginning. To address this, storage devices are used to buffer energy so that systems can operate continuously and avoid unstable operation. However, energy storage devices require time to charge up, and increases the size, mass and cost of the system. As an example, the two AA-sized batteries on a Crossbow Telos mote [8] occupy over

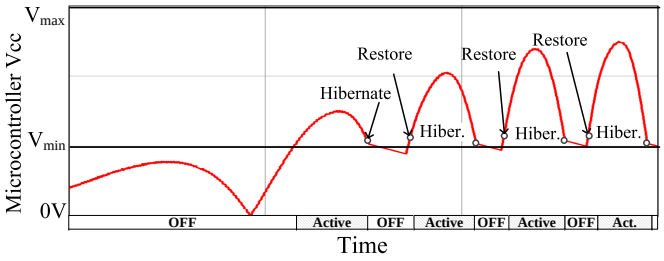

Fig. 2. Operation of transiently-powered system, continuing computation across several power cycles, with the first activation at $V_{min} = 2$V.

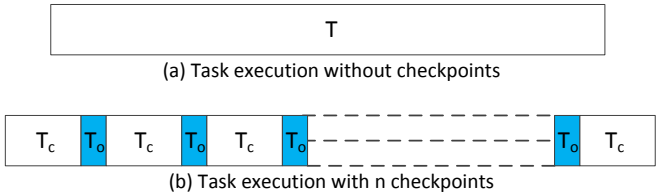

Fig. 3. Task execution with and without checkpoints.

half of its overall volume. Additionally, energy storage such as batteries pose pollution and sustainability issues.

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, the drawback being that the supply can be transient causing computation to be frequently interrupted and reset. To address this limitation, we developed *Hibernus* [13], an approach to enable computation to be sustained with systems powered directly from energy harvesters. The principle behind *Hibernus* is to save 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 continue operation.

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 energy harvester (Figure 1). The objective is to sustain operation for a longer duration within the constrained power availability. Following are the novel contributions of the *Hibernus++* as compared to the previously proposed *Hibernus*.

- the ability to self-configure the hibernate threshold;
- the ability to determine the restore threshold 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 to typical energy harvester source; and
- practical validation of the approach on 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 through synthesized energy harvesting signals (to enable comparison between methods), and then through the use of real energy harvesting inputs. Finally, a cold-start circuit is proposed which enables operation of the system with ultra-low input currents in Section VI.

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 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 structure to save their state. Examples of memories used for state retention are Flash [16], battery-backed SRAM memories [17].

To recover from a failure, systems roll back to the previous valid checkpoint, before continuing operation. Figure 3 shows the 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. Thus, the task execution time with n checkpoints is $(n+1) \cdot (T_c + T_o) = (T + (n+1) \cdot T_o)$. 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. Moreover, systems shut-down and wake-up have significant time and energy cost, and they must be minimized.

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, these systems 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], which uses checkpoints placed at compile-time. *Mementos* saves periodic snapshots of system state to non-volatile memory (NVM), which enable it to return to a previous checkpoint after a power failure. A number of 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 checkpoints in case a power interruption occurs whilst one checkpoint is being taken. *Mementos* is also selective about the data it saves. 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 nvm technology such as FRAM [19]. Some more advanced technologies are under development, such ReRAM [20], which could further reduce non-volatile storage energy and cost.

QuickRecall [21] proposed a refinement to *Mementos* technique, involving the use of a microcontroller with FRAM non-volatile memory, hence reducing the overheads of checkpointing. Their approach also used FRAM as unified memory, so that the system's RAM was not used for storing variables. A disadvantage of QuickRecall is that it uses an inflexible fixed voltage threshold to prompt an interrupt to take a checkpoint. It also relies on the use of a processor with a unified FRAM memory. Finally, Hypnos [22] 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 data retention problem in cases where the power source is unavailable for a prolonged period of time.

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

An ideal transiently-powered system would hibernate at the last possible moment before supply failure, and resume 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 an energy harvester (EH) with an AC output, or continuously with minimal overheads if the output of the EH is sufficient to continuously power the system.

This section introduces the operation of *Hibernus++* (as outlined in Figure 4). An important assumption is that the output voltage of the EH source has to exceed the minimum operating voltage of the system.

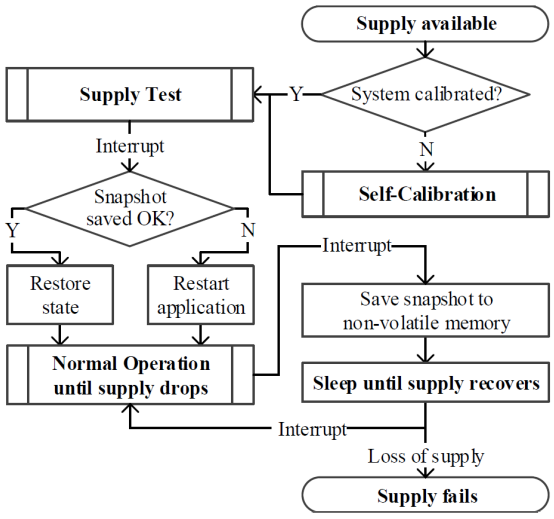

Fig. 4. Outline of the operation of *Hibernus++*

A. Principle of Operation

The following terms are introduced here for the easy understanding of the operating principles:

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

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 stable value. If a valid snapshot has previously been stored in memory, the system restores the snapshot and continue operation; if not, the application starts from scratch.

Following these routines, normal operation of the system continues until the supply voltage drops below V_H, at which point the system stores a snapshot to non-volatile memory, then sleeps. 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 the 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 lower the number of checkpoints, the lower is the time and energy overheads.* On the other hand, when error 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 error occurs at the end of checkpointing segment, during or just before the process saving of saving the snapshot). *Clearly, lower the number of checkpoints, higher is the loss of computation during error.*

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. Power failure probability can be very high for some energy harvesting sources (with the frequency being one in less than a second), a trade-off exists for selecting the number of checkpoints. To address this, *Hibernus++* uses adaptive checkpointing, 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 state-saving, a power loss during a state-save 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 state-save starts, all data since the last successful state-save will be lost. If there is a "repetitive" power failure caused by the dynamics of the power supply (i.e. 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 state-saving. To minimize the loss of computation, it is necessary for the system to operate conservatively or adaptively in response to the power supply dynamics.

Dynamics of a power source can be learned in order 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 on-board decoupling capacitance.

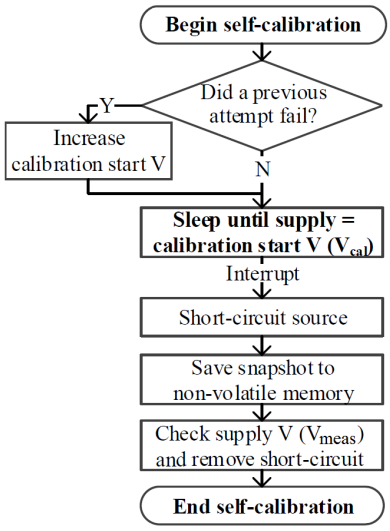

Fig. 5. Self-calibration of hibernation voltage threshold, V_H.

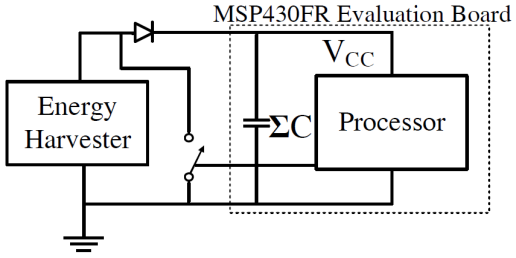

Fig. 6. Energy harvesting source short-circuit.

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. 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

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

This equation is derived based on the assumption that the current drawn is approximately constant across the range of

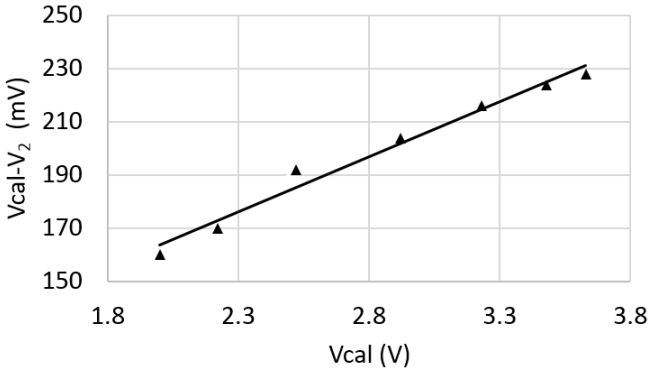

Fig. 7. The calculated voltage drop (V_{cal} - V_2) with different values of calibration start voltage (V_{cal}).

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 equal 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. If the calibration routine fails, V_{cal} has to be increased (see Fig. 5). V_{cal} is set as minimum as possible ($V_{cal} = V_{MCUon}$) in order to have the lowest value of V_H. Therefore, increased V_{cal} will result in V_H higher 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 with respect to the dynamics of the energy harvesting source. We present a restore strategy that classifies the type of input sources 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 exacts energy from the environment and use it directly by the load without an energy storage. This system can operate at time t when

$$P_s(t) \geq P_c(t) \tag{3}$$

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

$$\int_0^T P_s(t)dt \geq \int_0^T P_c(t)dt + Eo \qquad \forall\ T \geq 0 \tag{4}$$

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 the AC-output harvesters such as wind energy harvesters; these harvesters 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 belong to this category.

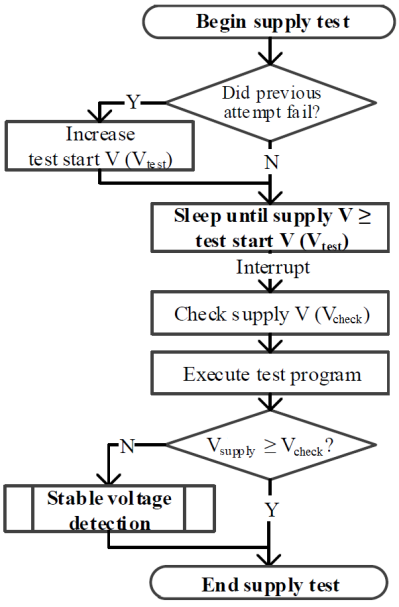

Fig. 8. Supply test flow chart.

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 system tries to restore immediately after the supply crosses the minimum voltage, the power drawn by the microcontroller will result in the supply dropping again below this minimum value causing the microcontroller to hibernate. This will cause repeated cycling between 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, where the system sleeps until the supply voltage reaches a stable voltage.

[1]This short segment of code consist of writing a few bits of the RAM to the FRAM. This code is for checking if the power source is sufficient to sustain the complete restore operation and therefore, does not include saving the entire snapshot.

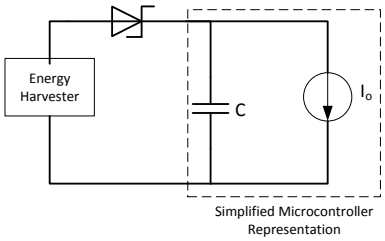

Fig. 9. Example system architecture.

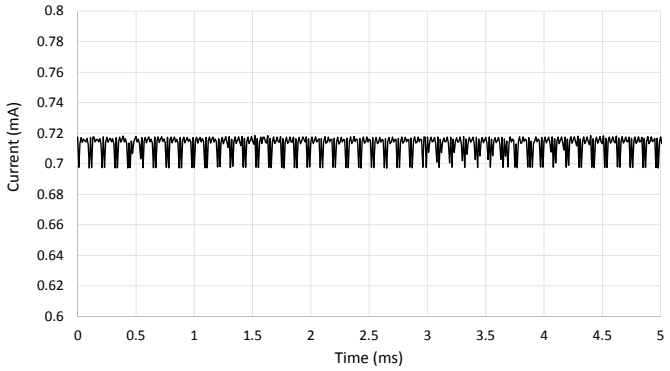

Fig. 10. Current consumption during execution of example FFT algorithm.

In the stable voltage detection process, 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 (implying that there is no benefit in waiting longer for it to charge any further). 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. 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 tri-axial 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.

To evaluate the overheads, we consider a scenario in which the EH input is first applied and then discontinued, demonstrating how the microcontroller responds to this transient source. This behavior is shown in Figure 11, which plots the input voltage (in blue) and the microcontroller supply voltage (in red). As discussed in Section III, as soon as the microcontroller supply voltage crosses the hibernation threshold (V_H), the microcontroller saves its state and suspends operation.

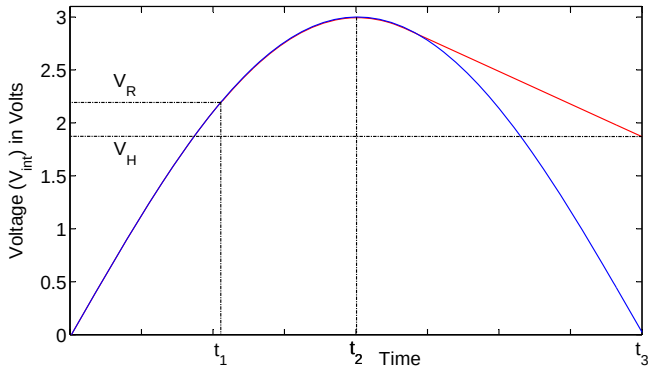

Fig. 11. Behavior with a transient input (in blue). The response to this input is shown in red, which is the voltage measured after the schottky diode in Figure 9.

A. Energy Overhead of Hibernation and Restore

To estimate the energy overhead, we let f_{source} denote the frequency of a sinusoidal signal powering the 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

$$t_{charge} = t_2 - t_1 = \frac{1}{4 \cdot f_{source}} - \frac{\sin^{-1}\left(\frac{V_R}{V_{max}}\right)}{2\pi \cdot f_{source}} \tag{5}$$

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 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

$$t_{discharge} = t_3 - t_2 = \frac{C\left(V_{max} - V_H\right)}{I_O} \tag{6}$$

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

$$t_{ON} = t_3 - t_1 = \frac{1}{4 \cdot f_{source}} - \frac{\sin^{-1}\left(\frac{V_R}{V_{max}}\right)}{2\pi \cdot f_{source}} + \frac{C\left(V_{max} - V_H\right)}{I_O} \tag{7}$$

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.

As seen from Equation 7, 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

$$N_r = N_h = \lceil \frac{T_{app}}{t_{ON}} \rceil \tag{8}$$

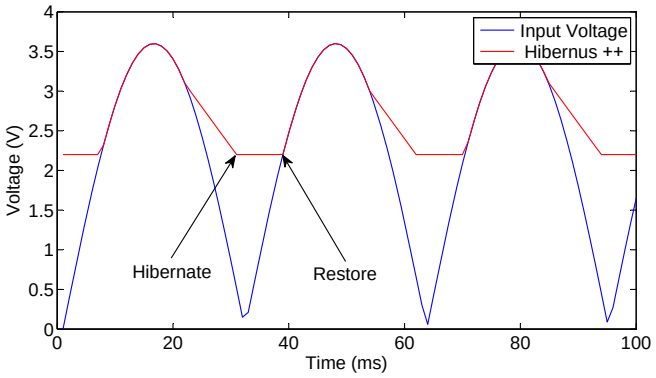

Fig. 12. Dynamic behavior of microcontroller in response to a fully rectified sinusoidal signal.

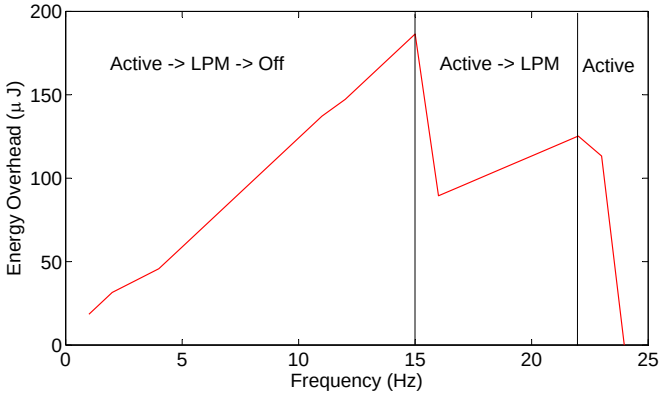

Fig. 13. Energy overhead of hibernate/restore, at various input frequencies.

The energy overhead is given by

$$E_{overhead} = N_r \cdot t_r \cdot P_r + N_h \cdot t_h \cdot P_h \tag{9}$$

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 7-9, 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 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 between the peak of one half-wave pulse to the time of the restore threshold of the next pulse. This interval is given by

$$t_{interval} = \frac{1}{4 \cdot f_{source}} + \frac{\sin^{-1}\left(\frac{V_R}{V_{max}}\right)}{2\pi \cdot f_{source}} \tag{10}$$

where the first term is the time for the input source to reach 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.,

$$V_{max} - \frac{I_O \cdot t_{interval}}{C} \geq V_R \tag{11}$$

Figure 13 plots that 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 simulations indicated 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 mode, the energy overhead due to hibernate is significant and therefore, there is a rise in energy with increase in 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.

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 technique, *Mementos*. *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 the limited subset of registers and global variables) to non-volatile memory.

The experimental set-up is shown in Figure 14. The test platform uses a Texas Instrument MSP-EXP430FR5739 microcontroller [23], 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.

The system depends on an external comparator circuit (Figure 15) which enables interrupts to be triggered when the supply voltages surpass thresholds 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. 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 ADCs voltage measurement when making checkpointing decisions: it compares the V_{cc} to a threshold voltage (V_m). 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

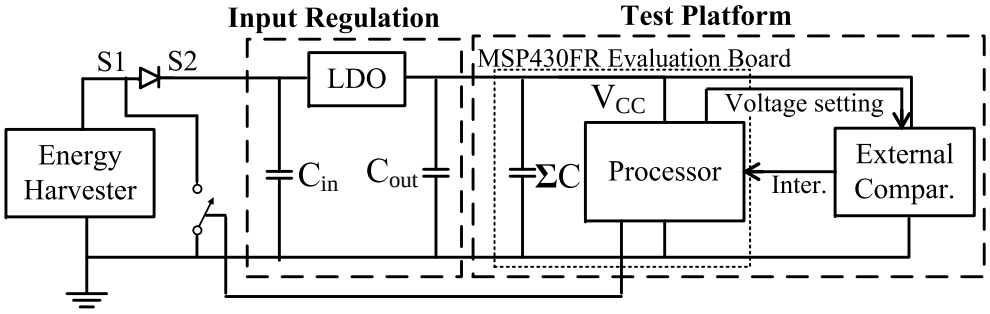

Fig. 14. Schematic: system connections

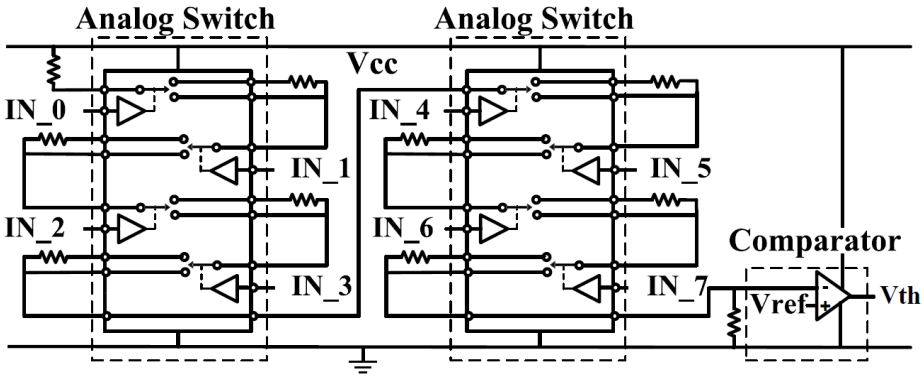

Fig. 15. Schematic of 'external comparator' circuit.

TABLE I
Mementos PERFORMANCE WITH TWO DIFFERENT VALUES OF *Vm*

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

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*'s ability to precisely time a checkpoint depends on the frequency of trigger points. 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 saved snapshots, as shown in Table I.

Mementos operates unstably with 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 become 'stuck', 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.

A. Results using Ideal Sources

Figure 16(a) compares the number of 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. As can be seen, *Hibernus++* modulates the number of times that snapshots are executed as a function of the supply interruption frequency, while *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 with 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', 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 results for these frequencies are missing for the *Mementos* approach in the above figure.

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, while *Mementos* saves a snapshot only when $V_{cc} < V_{min}$. The number of snapshots with *Mementos* depends on the check-point placement, the value of V_{min} and the supply interruption frequency; while for *Hibernus++* it only depends on the

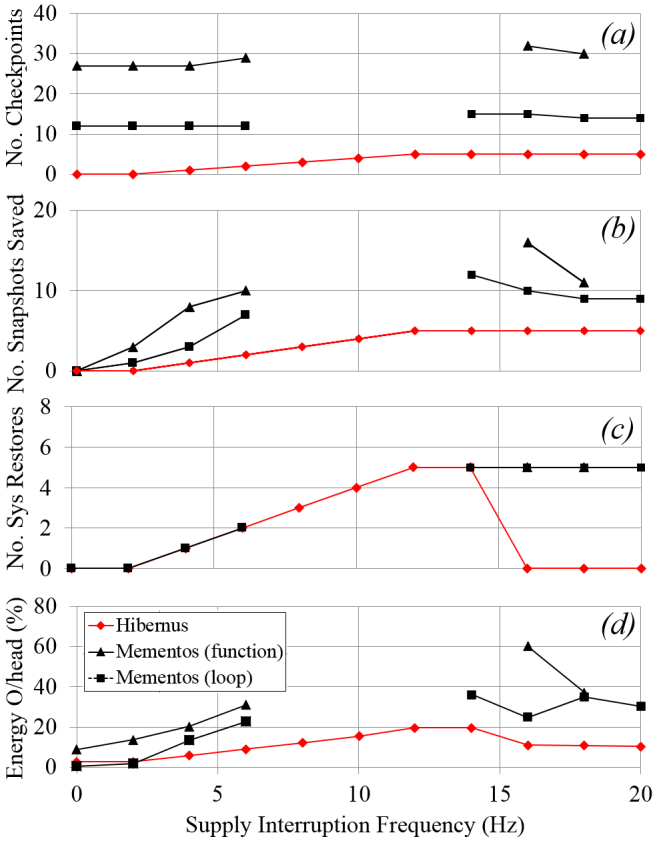

Fig. 16. Comparison of *Hibernus++* against *Mementos*, showing performance when running the FFT text program (averaged over 3 executions): (a) number of checkpoints saves, (b) number of snapshots saves, (c) number of times snapshots were restored, (d) energy overhead.

supply interruption frequency. Figure 16(c) shows that *Hibernus++* and *Mementos* complete execution of the FFT over the same number of power interruptions with frequencies lower than 14 Hz. However, with frequencies higher than *14Hz*, *Hibernus++* will always stay in ON mode (MCU always ON), alternating active and low-power states, so that the number of restores will always be zero. 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 savings.

Figure 18 illustrates the detailed behavior of the system under test, 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

TABLE II
EXECUTION TIME COMPARISON 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)	104.40	4.40	108.80	8.80	103.20	3.20
2	106.00	6.00	114.40	14.40	103.20	3.20
4	118.30	18.30	122.40	22.40	106.95	6.95
6	128.80	28.80	133.80	33.80	110.70	10.70
8	-	-	-	-	114.45	14.45
10	-	-	-	-	118.20	18.20
12	-	-	-	-	121.95	21.95
14	145.20	45.20	-	-	121.95	21.95
16	133.20	33.20	165.40	65.40	110.20	10.20
18	142.80	42.80	141.10	41.10	110.20	10.20
20	138.20	38.20	-	-	110.20	10.20

TABLE III
COMPARISON BETWEEN *Hibernus* AND *Hibernus++*.

Decoupling capacitance ΣC (μF)	Hibernus N. Restore	Hibernus N. Checkpoint	Hibernus Total Time (ms)	Hibernus++ N. Restore	Hibernus++ N. Checkpoint	Hibernus++ Total Time (ms)
10	-	-	-	2	2	**395.2**
20	2	2	**376.3**	2	2	**389.4**
30	2	2	**376.10**	1	1	**243.7**
40	2	2	**376.00**	1	1	**238.9**

Using a voltage input of 3V @ 6Hz

each interruption.

Table II reports the execution time of the *Hibernus++* approach in comparison with the *Mementos* for the FFT application. Without supply interruption, the FFT execution takes 100 ms to execute. The table reports the execution time for a range of input supply frequency, including the result corresponding to DC source. As can be seen from the table, the execution time for the FFT execution increases with an increase in the input frequency for the *Mementos* approach. The increase in execution time ranges from 5-45% for the *Mementos* with loop mode, while 9-65% for the *Mementos* with the function mode. In comparison to these, the *Hibernus++* approach increases execution time by 3-22% only. The overall improvement of this approach with respect to *Mementos* is on average 13% (1-23%) and 17% (5-33%) with respect to the loop mode and the function mode respectively.

Table III reports the total time taken by our earlier proposed approach *Hibernus* and our current *Hibernus++*, while 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 determined for a constant decoupling capacitance of 20μF and therefore, the execution time of *Hibernus* is lower than that of *Hibernus++* at this value of decoupling capacitance. *Hibernus++* self calibrates the hibernate and restore thresholds dynamically, resulting in lowest FFT execution time than *Hibernus* for other capacitance values. It is important to note that *Hibernus* does not work for capacitance values lower than the one it is designed for (20μF in this case). So *Hibernus* is not able to execute the FFT application for 10μF. *Hibernus++* calibrates the hibernate and restore thresholds dynamically based on the value of decoupling capacitance, and therefore is able to execute the

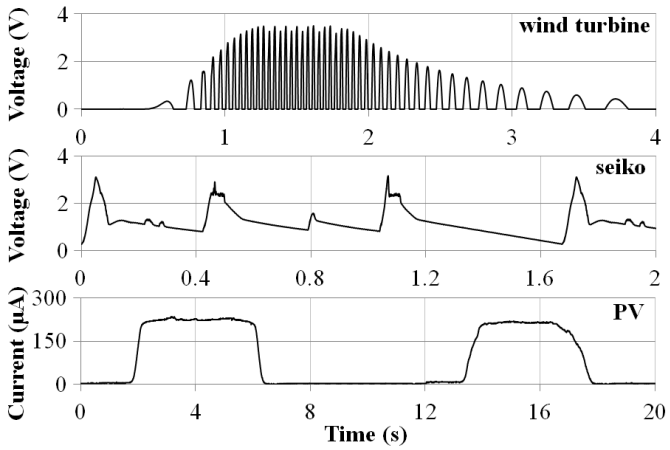

Fig. 17. Three synthetic traces of real harvesters.

TABLE IV
COMPARISON BETWEEN *Hibernus* AND *Hibernus++* USING SYNTHETIC HARVESTERS.

Synthesized source	Hibernus			Hibernus++		
	N. Restore	N. Checkpoint	Total Time (ms)	N. Restore	N. Checkpoint	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	-	-	-	0.00	12.00	**806.8**

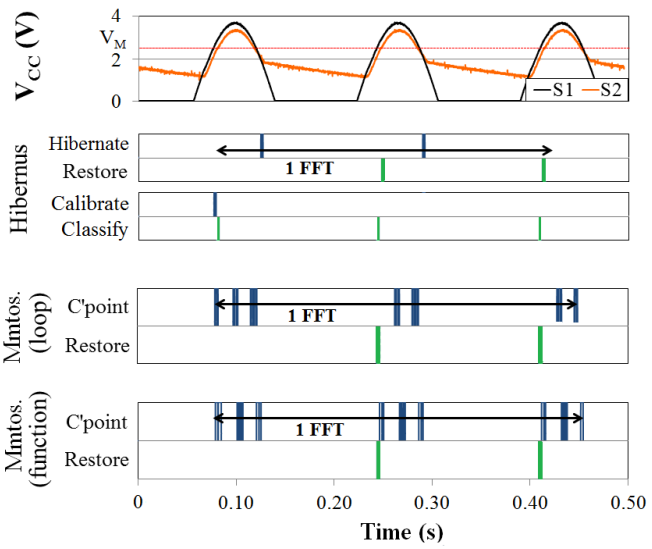

Fig. 18. Graph: basic hibernate/restore operation.

TABLE V
EXPERIMENTALLY MEASURED PARAMETERS.

Synthesized Sources	MCU ON (ms)	FFT (ms)	Low power (ms)	∑ Test (ms)	Calibration (ms)	∑ Restore (ms)	∑ Hibernate (ms)	Energy O/head (%)
Wind Turbine	173.45	100.00	40.25	9.00	2.20	10.80	11.20	28.00
Kinetic	156.00	100.00	45.30	3.00	2.20	2.70	2.80	9.42
PV	511.00	100.00	400.80	1.00	2.20	0.00	7.00	13.36
Input Current 200uA	548.00	100.00	428.00	1.00	2.20	0.00	16.80	21.08

FFT application for all capacitance values.

B. Results with Synthesized Energy Harvesters

Table V shows experimentally obtained values for synthesized EHs: a wind turbine, a wearable kinetic seiko system, a micro PV (Figure 17) and a constant current source. 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 18), while in the second case (micro photovoltaic 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 IV reports the comparison between the earlier proposed *Hibernus* and the current work *Hibernus++* for the three synthetic traces of real harvesters of Figure 17. Results are compared in terms of the number of checkpoints, number of restores and the total execution time. As can be seen from this table, the *Hibernus++* results in lower number of restores and checkpoints than *Hibernus*, resulting in a reduction of execution time by 36%. On the other hand for wind turbine, the number of restores and checkpoints using the *Hibernus++* are higher than the *Hibernus* approach. There is still an improvement of 12% in execution time. This is due to the execution time improvement using a 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 current sources (PV and the constant current source). This is because, every time the voltage across the decoupling capacitance increases above the micocontroller minimum voltage, the microcontroller is turned on to restore snapshot draining more current and bringing the voltage below the minimum. *Hibernus++* on the other hand, first checks if the 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 application with different kinds of energy harvesting sources.

C. Results with real energy harvesters

Finally, *Hibernus++* has been verified with real EHs: a micro-wind turbine (high-power source) and a micro photovoltaic module (low-power source). Figure 19 shows the activity of the system when powered by a real wind harvester. The operating parameters of the system are shown: hibernate and

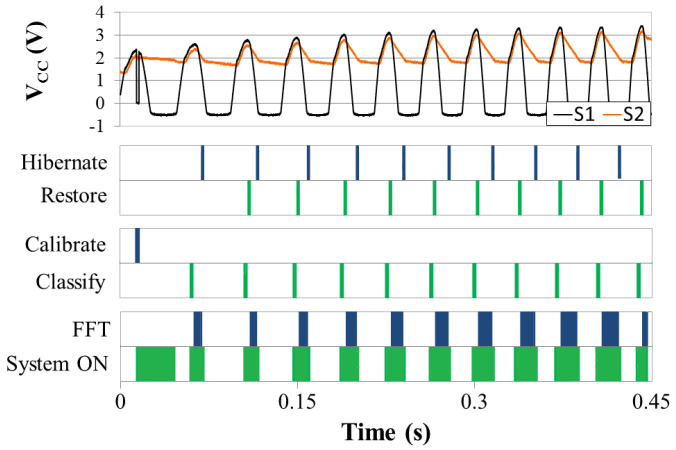

Fig. 19. Result: Activity of system with input from real wind harvester.

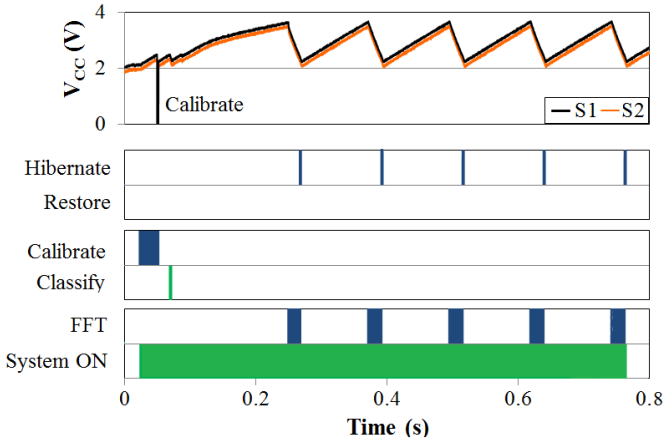

Fig. 20. Result: Activity of system with high-current input from real PV module.

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 18, 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 20 shows the activity of the system when powered by a real PV module. This illustrates the behavior of *Hibernus++* with a low-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 time for executing the FFT is *670ms*, and it saves five snapshots (although it never needs to restore).

VI. Enabling Ultra-Low Current Operation

The system cannot start up reliably with supply currents below 80 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 21) has been developed which can reliably start the system with lower current levels. It does this 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.

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 against the system without any cold-start circuit and it depends on the input current (see Figure 22). However, the energy overhead will be improved. Table VI shows experimentally obtained values for a constant current sources. In particular, it shows that the system can start reliably also with currents below 80 μA (in this case 70 μA). Moreover, it shows that with a current input of 200 μA (see Table II), the energy overhead has improved using the cold-start circuit.

TABLE VI
Experimentally measured parameters.

Input Current (uA)	MCU ON (ms)	FFT (ms)	Low power (ms)	Σ Test (ms)	Calibration (ms)	Σ Restore (ms)	Σ Hibernate (ms)	Energy O/head (%)
70 (*)	3430.00	100.00	3322.60	1.00	2.20	0.00	4.20	46.79
200 (*)	850.00	100.00	744.00	1.00	2.20	0.00	2.80	14.07

(*) Using the external start-up circuit

VII. Conclusion

A new approach for sustaining computation during intermittent supply, named *Hibernus++* has been proposed, which manages dynamically different energy harvesting sources, by increasing the amount of useful computation. This allows a new class of embedded systems, named "transient computing systems", to sustain computation through power outages which are common in energy-harvesting systems and to adapt their behavior as a function of the input source, allowing "energy-neutral" operation without using any external energy buffer. The system has been validated with both synthesized and real

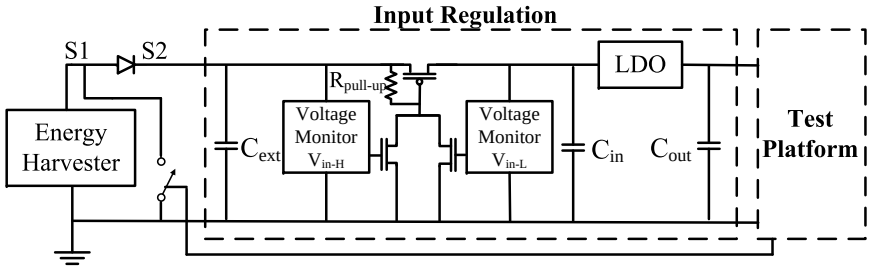

Fig. 21. Schematic: validation circuit with cold-start arrangement, to allow ultra low-current start-up. The 'Test Platform' is as shown in Figure 14.

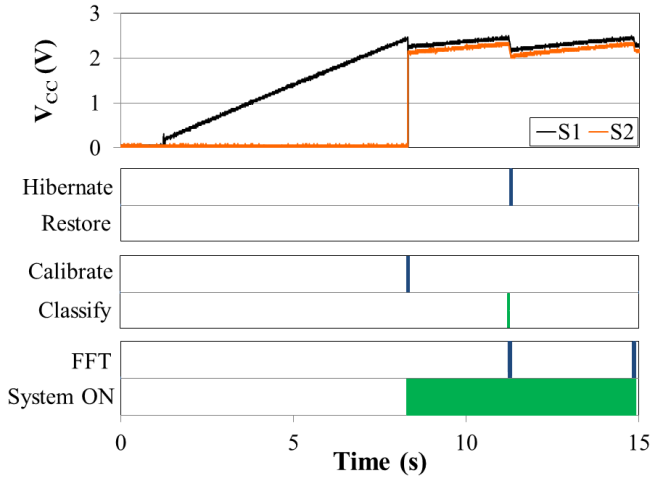

Fig. 22. Result: Activity of system with 30μA input from real PV module.

EH sources, demonstrating experimentally that it has a lower energy and time overhead than a recently proposed scheme. This contributes to the development of future energy harvesting transient systems. Our continuing work is investigating performance with other real micro-energy harvesters.

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," *Design & Test of Computers, IEEE*, vol. 18, no. 2, pp. 62–74, 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), 2015 IEEE 21st International Symposium on*. 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] 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," *Solid-State Circuits, IEEE Journal of*, vol. 49, no. 1, pp. 202–211, 2014.

[20] 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), 2012 IEEE 10th International*. IEEE, 2012, pp. 417–420.

[21] H. Jayakumar, A. Raha, and V. Raghunathan, "Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers," in *VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on*. IEEE, 2014, pp. 330–335.

[22] A. Jindal, A. Pathak, Y. C. Hu, and S. Midkiff, "Hypnos: understanding and treating sleep conflicts in smartphones," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 253–266.

[23] TI, "Msp430fr5739 fram experimenter board," 2013. [Online]. Available: http://www.ti.com/lit/ug/slau343b/slau343b.pdf