Hibernus: Sustaining Computation during Intermittent Supply for Energy-Harvesting Systems

Domenico Balsamo, Alex S. Weddell, *Member, IEEE,* Geoff V. Merrett, *Member, IEEE,*
Bashir M. Al-Hashimi, *Fellow, IEEE,* Davide Brunelli, *Member, IEEE,* and Luca Benini, *Fellow, IEEE*

***Abstract*—A key challenge to the future of energy-harvesting systems is the discontinuous power supply that is often generated. We propose a new approach, Hibernus, which enables computation to be sustained during intermittent supply. The approach has a low energy and time overhead which is achieved by reactively hibernating: saving system state only once, when power is about to be lost, and then sleeping until the supply recovers. We validate the approach experimentally on a processor with FRAM non-volatile memory, allowing it to reactively hibernate using only energy stored in its decoupling capacitance. When compared to a recently proposed technique, the approach reduces processor time and energy overheads by 76-100% and 49-79% respectively.**

***Index Terms*—energy harvesting, checkpointing, embedded software**

I. Introduction

Energy-harvesting systems power themselves by extracting energy from the environment [1]. However, the energy provided is often highly temporally dynamic, providing an intermittent supply that is incapable of sustaining computation. This is because processors switch off when the supply drops below their minimum operating voltage and, when power is available again, restart computation from the beginning.

To manage an intermittent supply, one approach is to use a battery or supercapacitor to buffer energy. However, the level of miniaturisation required to realise medical implants [2] or visions of 'smart dust' [3] causes energy storage to be minimised, constraining the computational ability of systems. Recently, a different approach (Mementos [4]) was proposed, which uses the well-known concept of checkpoints [5] placed at compile-time. Mementos saves periodic snapshots of system state to non-volatile memory, 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. At run-time, when these checkpoints are reached, the supply voltage (V_{CC}) of the processor is inspected using the an analog-to-digital converter (ADC). If it is deemed to be failing ($V_{CC} <$ a threshold V_M), a snapshot of the system state is saved to non-volatile memory. This requires regular polling of the supply voltage, and can result in multiple snapshots being saved when the supply voltage is close to the threshold; both introduce time and energy overheads.

D. Balsamo, A. S. Weddell, 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.

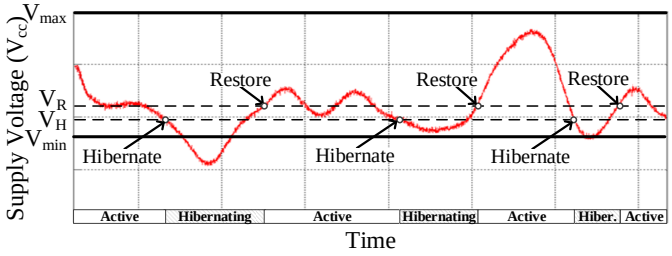

Fig. 1. Operation of *Hibernus* in response to intermittent supply voltage.

This brief proposes *Hibernus*[1], a new approach which automatically saves a snapshot only once (without the need for checkpoint placement heuristics), immediately before power failure, then sleeps. *Hibernus* saves the system's complete volatile memory; this is enabled in part by developments in Ferroelectric RAM (FRAM), a non-volatile memory technology that is more efficient than flash, and is now being monolithically integrated into low-power microcontrollers [6]. The speed and efficiency of integrated FRAM means we can react to power loss and save a snapshot using only the energy stored in a system's decoupling capacitance.

II. Hibernus

The *Hibernus* approach has two states: active and hibernating. It moves between these states when the supply voltage (V_{CC}) passes thresholds (Fig. 1). It uses a hardware interrupt to detect when V_{CC} drops below V_H, then prompts a reactive hibernation – saving an immediate snapshot of volatile memory, then entering deep sleep. The snapshot is restored by another interrupt, when the supply voltage rises above V_R. The approach is illustrated in Fig. 2 and differs from Mementos, whose checkpoint locations are set in advance. Due to this, our approach is more energy- and time-efficient than existing approaches (experimentally demonstrated in Sec. III), and does not depend on checkpoint placement heuristics.

Hibernus is application-agnostic and transparent to the programmer, because it can reactively hibernate at any time during the execution of an application. Therefore, to save a snapshot of system state, it copies all registers and volatile memory to non-volatile memory. The energy consumed by

[1]In computing, 'hibernation', from the Latin *hībernus*, is the process of saving state to allow power to be removed.

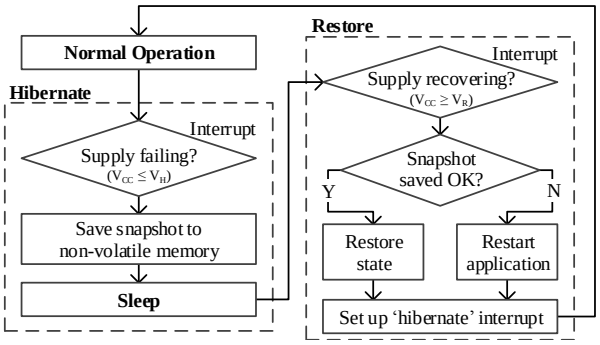

Fig. 2. Flow-chart illustrating the *Hibernus* approach.

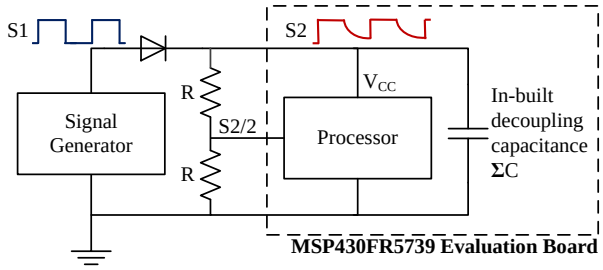

Fig. 3. The test platform used to experimentally validate *Hibernus*.

this process, E_σ, depends on the size of the volatile memory and the energy consumption for copying each byte.

$$E_\sigma = n_\alpha E_\alpha + n_\beta E_\beta \tag{1}$$

Here, n_α and n_β are the sizes of the RAM and registers (in bytes). E_α and E_β are the energy required to copy each RAM and register byte to non-volatile memory (J/byte).

Hibernus requires sufficient non-volatile memory to save the contents of all processor registers and RAM. This is the case with modern microcontrollers, e.g. [6]. It also requires enough energy to be stored in the capacitance between the supply rails to save a full snapshot. Energy harvesting systems normally operate across a range of voltages, from V_{min} to V_{max}. Below V_{min}, processors may operate unpredictably (brown-out), or shut down completely. Given the total capacitance ($\sum C$), the energy E_δ stored between a given voltage V and V_{min} is:

$$E_\delta = \frac{V^2 - V_{\text{min}}^2}{2} \cdot \sum C \tag{2}$$

V is used to define the threshold V_{H}, and V_{R} is set higher to add hysteresis, allowing the system to restore without taking the V_{cc} below V_{H}. For small embedded microcontrollers (with relatively small n_α) using fast-write non-volatile memory (therefore relatively low E_α), it is possible to save a snapshot without additional C (using only the system's decoupling capacitance); this is explored in Sec. III. However, if $E_\delta < E_\sigma$ with $V = V_{\text{max}}$, it will not be possible to guarantee that snapshots can be taken reliably, and extra C must be added.

The total time, T_{hibernus}, to execute a test algorithm with *Hibernus* is given by (3), where T_a is the CPU time required to execute the algorithm, n_ι is number of power interruptions (where $V_{\text{CC}} < V_{\text{min}}$) per algorithm execution, T_s is the time required to save a snapshot to non-volatile memory, T_r is the time required to restore from non-volatile memory, and $\overline{T_\lambda}$ is the average time spent sleeping (after a snapshot has been saved but before $V_{\text{cc}} = V_{\text{min}}$, and on power-up when $V_{\text{min}} < V_{\text{CC}} < V_{\text{R}}$).

$$\underbrace{T_{\text{hibernus}}}_{\text{Total execution}} = \overbrace{T_a}^{\text{Algorithm}} + \underbrace{n_\iota}_{\text{No. interruptions}} (\overbrace{T_s}^{\text{Save snapshot}} + \underbrace{T_r}_{\text{Restore snapshot}} + \overbrace{\overline{T_\lambda}}^{\text{Sleep}}) \tag{3}$$

The total time, T_{mementos}, to execute an algorithm with Mementos is given by (4), where n_m is the number of checkpoints per complete execution of the algorithm, T_m is the time taken for an ADC reading of V_{CC}, and P_s is the proportion of checkpoints resulting in a snapshot, taking T_s.

$$\underbrace{T_{\text{mementos}}}_{\text{Total execution}} = \overbrace{T_a}^{\text{Algorithm}} + \underbrace{n_\iota}_{\text{No. interruptions}} (\overbrace{T_r}^{\text{Restore snapshot}} + \underbrace{\frac{T_a}{2n_m}}_{\text{Backtrack}}) + \overbrace{n_m(T_m + P_s T_s)}^{\text{Monitoring and save snapshot}} \tag{4}$$

Hence, $T_{\text{hibernus}} < T_{\text{mementos}}$ provided $n_\iota(T_a/2n_m) + n_m T_m + (n_m P_s - n_\iota)T_s > n_\iota \overline{T_\lambda}$; that is, *Hibernus* spends less time sleeping than Mementos spends on backtracks (re-running code that was executed between a snapshot and a power interruption), sampling V_{cc}, and redundant snapshot saves. This is evaluated experimentally in the next section.

III. Experimental Validation

Hibernus has been validated with an intermittent power supply and representative workload. Its energy and time overheads have been evaluated, and compared against Mementos.

A. Implementing Hibernus

While most microcontrollers have flash non-volatile memory (and consequently a high E_σ), processors are emerging that incorporate fast, low-power FRAM rather than flash (and hence have a low E_σ). The test platform (Fig. 3) uses a development board combining a Texas Instruments MSP430 processor [6] with FRAM non-volatile memory. This means that its decoupling capacitance alone allows $E_\delta >> E_\sigma$ when $V = V_{\text{max}}$, requiring no additional energy storage (battery or large capacitor) to support *Hibernus*.

The platform's datasheet parameters were inspected, and identified E_α as 4.2 nJ/byte and E_β as 2.7 nJ/byte, with a total RAM size of 1024 bytes and register size of 524 bytes. The platform operates with a $V_{\text{max}} = 3.6\ V$ and $V_{\text{min}} = 2.0\ V$. Using (1), a complete operation copying all registers and RAM to FRAM consumes 5.7 μJ. The decoupling capacitance on the board totals $\sum C$ = 16 μF. Using (2), it was found that this alone is sufficient for *Hibernus* and V_{H} was set to 2.17 V. It was found empirically that $V_{\text{R}} = 2.27$ V was sufficient for reliable operation. The test platform's V_{CC} input (*S2*) is connected to the output of a signal generator (*S1*) through a diode, which prevents back-flow of charge to the harvester (Fig. 3). Square and sinusoidal traces (Fig. 4) with a peak amplitude of 3.6V are presented as examples. The slower decay of *S2* compared to *S1* is due to the input diode; the slow decay on the negative

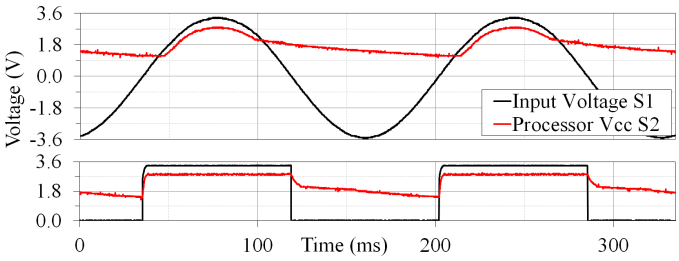

Fig. 4. Measured behavior of signals *S1* and *S2* (Fig. 3) with (a) 6 Hz square-wave input; (b) 6 Hz sinusoidal input.

```
#include "hibernus.h"

int main (void) {
  if (flag) restore(); //restore system state
    else initialise(); //initialise hibernus
  // application code goes here
}

__interrupt void COMP_D_ISR(void) {
  hibernate(); //save system state & sleep
}
```

Fig. 5. Example code used for evaluation of *Hibernus*.

edge illustrates the discharge of the decoupling capacitance by the current drawn by the processor.

Hibernus functionality is contained within the `hibernus.h` library file; application developers need only include this library and call the `initialise()`, `hibernate()` and `restore()` routines, as illustrated in Fig. 5. As shown in Fig. 2, the algorithm requires that interrupts are generated when V_{cc} passes V_H and V_R; this is facilitated by comparators and voltage references. The test platform has an on-chip comparator configured with an on-chip variable reference voltage generator, and an external voltage divider ($R = 200$ kΩ) giving $V_{cc}/2$, as inputs. This is set up in the `initialise()` routine. Dependent on whether the system is hibernating or active, the interrupt is set to trigger off either $V_{cc} \leq V_H$ or $V_{cc} \geq V_R$. The handler then calls `hibernate()` or `restore()`.

When `hibernate()` (Fig. 2) is called, it first pushes the core registers onto the FRAM memory. It then copies the entire RAM contents (stack segment, local and global variables) into the FRAM, followed by the general registers, and finally the Stack Pointer (SP) and Program Counter (PC). It then sleeps in a low-power mode. The system remains in sleep mode until $V_{CC} > V_R$. The `restore()` routine is then called and the complete previous system state is restored. The system phases the restore of the memory locations to reinstate its operating state reliably. The general registers are restored first, followed by the RAM, and lastly the core registers including the SP and PC. When the PC is restored from the snapshot, the system implicitly transfers to the application and resumes operation.

B. Experimental Setup

The evaluation test case 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 FFT algorithm was chosen as an illustration: *Hibernus* is application-agnostic and will provide the same functionality to any embedded program, with minimal impact on the application developer (see Fig. 5). Supply interruption frequencies f_ι (of 2, 4, 6, 8, 10 Hz, and DC) were chosen to represent the types of intermittent power output that may be expected from an energy harvester (e.g. micro wind turbine or inductive power transfer to a rotating object). They allow the overheads of the *Hibernus* approach to be compared against Mementos.

Our implementation of Mementos places static checkpoints after function calls or before loops, referred to as 'function' and 'loop'. ADC (V_{cc}) measurements are taken and compared to a threshold ($V_m = 2.5V$), chosen for each scheme to ensure that a snapshot can be saved at least once before power failure. At each checkpoint, $V_{cc} < V_m$ indicates imminent power failure, and a snapshot is saved. Mementos consumes energy for multiple checkpoints, both for ADC readings and saving snapshots. In contrast, *Hibernus* consumes energy for a single hibernation per power-outage, plus the quiescent consumption of the voltage reference and comparator.

The power consumption at mid-range between V_{max} and V_{min} of the FFT algorithm (without *Hibernus* or Mementos running), ADC, voltage reference, and comparator were measured as 2.7 mW, 310 μW, 17 μW and 130 μW respectively. These values are used to estimate the energy consumption of the different approaches. For each of the three schemes, and at each frequency f_ι, we evaluated: (1) the number of system restores required to complete the computation of the FFT algorithm, (2) the number of times snapshots were stored, or checkpoints were called, (3) the energy overhead, and (4) the processor time overhead. The results were averaged over three complete executions of the test program. The overheads are evaluated with reference to the time and energy for the processor to complete the FFT algorithm with a steady supply: without Mementos or *Hibernus*, it completed in 100 ms.

C. Results

Fig. 6(a) shows how many checkpoints were made by *Hibernus* and Mementos during a single execution of the FFT. As can be seen, *Hibernus* reduces the number of times that checkpoints are taken. This can also be seen from Fig. 7, which shows when *Hibernus* and Mementos checkpoint (for the case when $f_\iota = 6$ Hz), whereas *Hibernus* snapshots (hibernates) only once per interruption (twice in total), Mementos executes a static number of checkpoints (12 and 27 times), although some are repeated when $V_{CC} < V_{min}$ during a snapshot.

Fig. 6(b) shows that, at higher f_ι values, *Hibernus* completes execution of the FFT over fewer power interruptions (3, instead of 5). This is because the mean processor time overheads (Fig. 6(d)) of *Hibernus* are 80-100% shorter than Mementos (function), and 76-100% shorter than Mementos (loop); this leaves more time to execute the application (also shown in Fig. 7, where the arrows denote the total execution time). Furthermore, Fig. 6(c) shows that the energy overheads of running *Hibernus* are 65-79% lower than Mementos (function) and 49-76% lower than Mementos (loop).

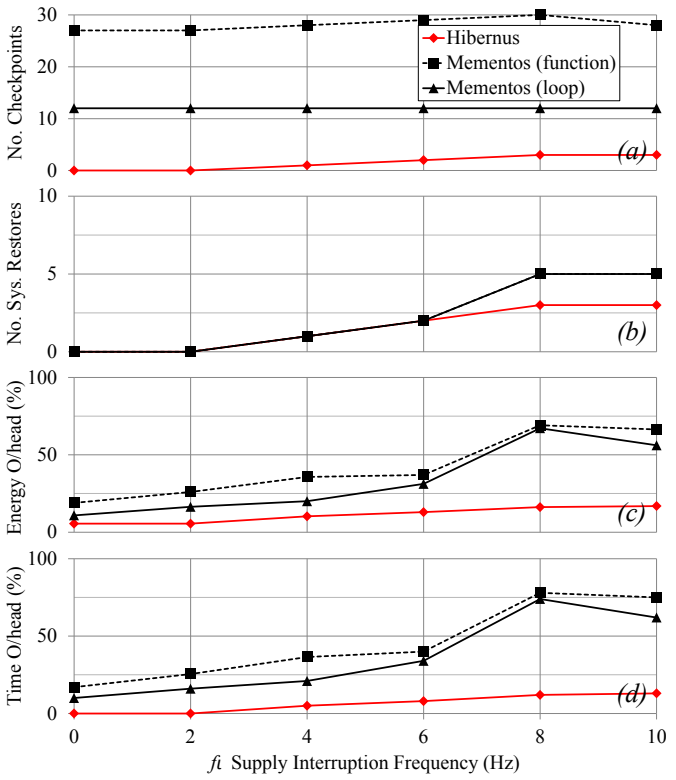

Fig. 6. Comparison of *Hibernus* against Mementos, showing performance when running the FFT text program (averaged over 3 executions): (a) number of checkpoints/snapshot saves, (b) number of times snapshots were restored, (c) energy overhead, (d) time overhead.

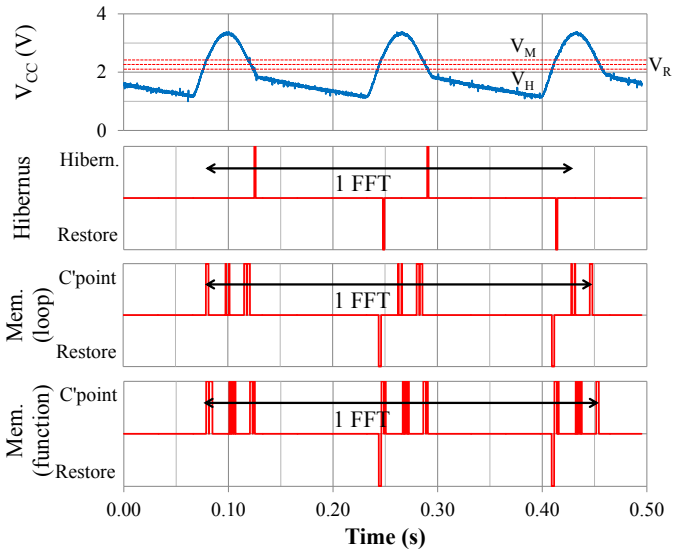

Fig. 7. Results comparing when *Hibernus* and Mementos hibernate, checkpoint, and restore. Results shown were measured over a complete execution of the test FFT algorithm, powered by a sinusoidal supply with f_ι = 6 Hz.

The benefits of *Hibernus* are particularly noticeable at f_ι = 0 Hz (i.e. DC, when V_{cc} is uninterrupted), where negligible time and energy overheads are imposed (see Fig. 6(c) and (d)), while Mementos still requires the same number of checkpoints. This increases the required processor active time and energy by at least 10% and 11% respectively. Table I shows experimentally obtained values for the parameters of (3) and (4). Evaluating these equations support our measured results,

TABLE I
EXPERIMENTALLY MEASURED PARAMETERS (SEE EQUATIONS (3), (4)).

f_ι	T_a	T_s	T_r	T_m	T_λ	*Hib.*	*Loop*			*Function*		
(Hz)	*(ms)*	*(ms)*	*(ms)*	*(ms)*	*(ms)*	n_ι	n_ι	n_m	P_s	n_ι	n_m	P_s
0	100	2.85	2.2	0.65	-	-	-	12	0.00	-	27	0.00
2	100	2.85	2.2	0.65	17	0	0	12	0.08	0	27	0.11
4	100	2.85	2.2	0.65	9.5	1	1	12	0.25	1	27	0.19
6	100	2.85	2.2	0.65	6.5	2	2	12	0.50	2	27	0.33
8	100	2.85	2.2	0.65	3.8	3	5	12	1.00	5	27	0.70
10	100	2.85	2.2	0.65	2.8	3	5	12	0.83	5	27	0.67

and confirm that *Hibernus* spends less time sleeping than Mementos spends on redundant snapshot saves, backtracks, and sampling V_{cc}.

IV. Conclusion

A new approach for sustaining computation during intermittent supply, *Hibernus*, has been proposed. This allows a system to sustain computation through power outages which are common in energy-harvesting systems. It has a lower energy and time overhead than a recently proposed scheme, as demonstrated experimentally. This contributes to the development of future energy harvesting systems.

Acknowledgment

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, and PHIDIAS, an EU 7th Framework Programme project (CA 318013).

References

[1] P. D. Mitcheson *et al.*, "Energy Harvesting From Human and Machine Motion for Wireless Electronic Devices," *Proc. IEEE*, vol. 96, no. 9, pp. 1457-1486, Sept. 2008.

[2] M. R. Mhetre *et al.*, "Micro energy harvesting for biomedical applications: A review," *Proc. ICECT 2011*, vol. 3, pp. 1-5, 8-10 April 2011.

[3] B. A. Warneke and K. S. J. Pister, "An ultra-low energy microcontroller for Smart Dust wireless sensor networks," *Proc. IEEE ISSCC 2004*, pp. 316-317, vol. 1, Feb. 2004.

[4] B. Ransford *et al.*, "Mementos: System Support for Long-Running Computation on RFID-Scale Devices," *ASPLOS11*, Newport Beach, CA, USA, Mar. 5-11, 2011.

[5] P. A. Bernstein *et al.*, "Concurrency Control and Recovery in Database Systems," *Addison-Wesley Longman Publishing*, Boston, USA 1987.

[6] M. Zwerg *et al.*, "An 82μA/MHz microcontroller with embedded FeRAM for energy-harvesting applications," *Proc. IEEE ISSCC 2011*, pp.334-336, 20-24 Feb. 2011.