Using Sleep States to Maximize the Active Time of Transient Computing Systems

Giedrius Lukosevicius, Alberto Rodriguez Arreola, Alex S. Weddell
Department of Electronics and Computer Science
University of Southampton, U.K.
{gl3n15,ara1g13,asw}@ecs.soton.ac.uk

ABSTRACT

Energy harvesters are widely used to power wireless sensor systems, but the produced power is generally low, and can vary abruptly due to changes in the environment or the device's location. Energy buffers (batteries or supercapacitors) are normally incorporated into systems to smooth out these variations. However, they have a limited lifetime and increase system size and cost. Transient computing aims to address these issues by removing the energy buffer, and powering the system directly from the energy harvester. Approaches such as Hibernus++ deal with the resultant power intermittency by 'hibernating', i.e. saving a snapshot of the system state to non-volatile memory before a power failure, and restoring it after the power recovers. The overheads of this can be particularly costly with a low-current harvester, as the system may wake up and hibernate at a high frequency, doing little useful work in each power cycle.

This paper proposes an enhancement to these approaches, providing an efficient method to avoid repeated hibernation. The introduction of a 'sleep' state, which is entered when the power supply is detected to be failing, allows the system's supply voltage to recover without taking a snapshot. Thus, the application can spend more time on useful work rather than checkpointing. If the supply voltage continues to decline, a snapshot will then be taken. The approach has been simulated and experimentally validated, with results demonstrating that the proposed scheme provides up to a 65% improvement in system active run-time with low-current harvesters vs. conventional Hibernus++.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**;

KEYWORDS

transient computing, energy harvesting, checkpointing, Hibernus

ACM Reference Format:
Giedrius Lukosevicius, Alberto Rodriguez Arreola, Alex S. Weddell. 2017. Using Sleep States to Maximize the Active Time of Transient Computing Systems. In *Proceedings of ENSsys'17: The Fifth ACM International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems, Delft, Netherlands, November 5, 2017 (ENSsys'17),* 6 pages.
https://doi.org/10.1145/3142992.3142998

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ENSsys'17, November 5, 2017, Delft, Netherlands
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-5477-6/17/11.
https://doi.org/10.1145/3142992.3142998

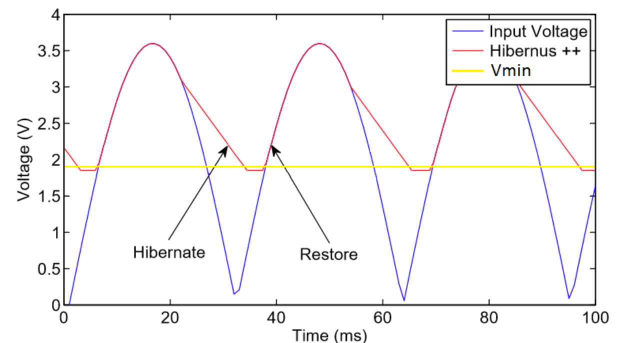

Figure 1: Hibernus++ response to a fully rectified sinusoidal signal (modified from Balsamo, et al. [2]).

1 INTRODUCTION

As the interest in Energy Harvesting (EH) systems integrating with small low-power embedded devices continues to grow, many new technologies which incorporate efficient ways to utilize the harvested energy are being developed. However, batteries and supercapacitors are required to 'buffer' harvested energy to smooth out variations in supply. These energy storage devices increase system size and cost, and have a limited operational lifetime. Removing the energy storage and powering a system directly from an EH source would likely cause the system to become unstable, repeatedly restart, or otherwise malfunction. An emerging concept, termed *transient computing*, enables systems to be directly powered by the EH source without the need for energy storage. Existing approaches including Mementos [1], QuickRecall [3] and Hypnos [4] typically save a snapshot of the system state into a non-volatile memory (NVM), e.g. ferroelectric RAM (FRAM) or flash, before a supply interruption. Thus, when power recovers, the system's state is restored from the saved copy. The general operation of such a scheme is illustrated in Figure 1.

A state-of-the-art approach for transient computing is Hibernus++ [2], which self-calibrates its voltage thresholds to match the platform it is deployed on. However, even with this self-calibration capability, it tends err on the side of caution and hibernate more than necessary; this is a particular problem with low-current power sources. Saving a snapshot of system state each time it hibernates introduces significant time and energy overheads [1].

This paper proposes an algorithm to counter this issue. When the supply voltage is detected to be approaching the hibernate voltage threshold, the system is put into a sleep mode (without taking a snapshot). Provided that the harvested current is higher than the system's sleep current, it will remain in sleep mode until the supply voltage rises to a certain threshold. This avoids a time and energy-intensive snapshot being taken, and maximizes the active time of the system. Should the harvested current be lower than the sleep current, the supply voltage will continue to drop, and prompt a 'fail-safe' hibernation where a snapshot is saved before the power supply is lost.

The novel contributions of this paper are:

- Presentation of an approach to greatly reduce costly system hibernation with low-current EH sources (Section 3).
- A mathematical analysis of the operation of the proposed approach, considering threshold voltages and transitions between states (Section 4).
- Simulation (Section 5) and experimental validation (Section 6) in terms of active run-time of the proposed approach compared to original Hibernus++.

While the paper is pitched as an enhancement to Hibernus++, it is applicable to any transient computing approach that uses threshold voltages to prompt or control checkpointing/hibernation.

2 EXISTING TRANSIENT COMPUTING APPROACHES

This section will explore the various approaches to transient computing. However, it is first useful to define the terminology used:

- **Threshold:** a defined voltage value which triggers a change of state, e.g. to hibernate or restore.
- **Active:** the system being awake and in active mode, carrying out useful work (e.g. processing data).
- **Snapshot:** saving a copy of system state into non-volatile memory (NVM). This is typically the processor registers (e.g. program counter, stack pointer, status register, general purpose registers), along with any volatile memory.
- **Checkpoint:** for some transient computing schemes, the act of checking supply voltage so that a snapshot can be saved if necessary (supply voltage is below a threshold value).
- **Sleep:** entering a low-power mode with the processor inactive and some peripherals/timers/clocks disabled, depending on mode.
- **Hibernate:** to take a snapshot before entering sleep.
- **Restore:** restore state by copying snapshot data from NVM to processor registers and volatile memory, then returning to active mode.

A number of approaches to transient computing have been proposed in the literature:

- **Mementos [5]** places checkpoints at compile-time inside the program, typically before function calls or loops. When the trigger point is reached during program run-time, the execution of it is suspended and a voltage check is performed. If the voltage is found to be below a certain threshold, a snapshot is saved.

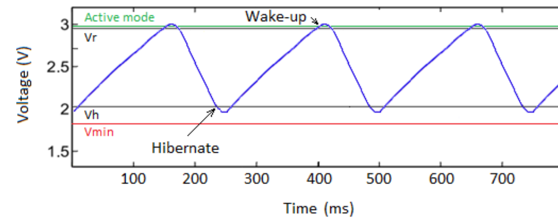

Figure 2: Supply voltage behavior with original Hibernus++, with low current continuous power source.

- **QuickRecall [3]** proposes a 'unified processor' approach, where the system uses a unified memory in FRAM, meaning that when taking a snapshot, only the processor registers need to be saved to NVM. Upon triggering a preset voltage threshold, the system saves the processor registers into FRAM, and the program hibernates.
- **Hypnos [4]** combines on-chip SRAM together with intense voltage scaling techniques during sleep cycle of the microcontroller (MCU) to achieve a reduction in time and energy overheads. However, as it will irreplaceably lose data if the power supply fails, it cannot be considered suitable for transient systems with rapidly-varying supply inputs.
- **Hibernus++ [2]** overcomes the limitations of previous transient computing systems by saving a snapshot only before an expected power failure. Using voltage comparator interrupts, the supply voltage is monitored against hibernate and restore voltage thresholds. When the input voltage drops below the hibernate voltage threshold, the system hibernates by saving a snapshot and entering a deep sleep mode right before a potential power failure. Contrarily, as the power supply recovers from a recent failure, a restore is performed once the voltage rises above the restore threshold.

In addition to these features, Hibernus++ is application and platform agnostic. During the initial self-calibration process, it intelligently sets its restore and hibernate thresholds according to the energy harvesting source's characteristics and system load properties. Every time the system hibernates, it reevaluates its performance and adjusts the thresholds if necessary.

Figure 1 showed an example response of Hibernus++ to a fully rectified sinusoidal input. As can be seen, the system hibernates (taking a snapshot in the process), and when the power supply recovers the system state is restored from the snapshot.

As illustrated in Figure 2, when powered by a low-current source, the system supply voltage can stay above its minimum operating voltage V_{min}. Therefore, when the supply voltage recovers, it is not necessary to restore the state. In effect, the energy expended in taking the snapshot has been wasted. This can cause systems to repeatedly oscillate between hibernate and active modes, with snapshots being taken but not required. This type of problem is a significant issue when the system is powered by low-current energy harvesters, e.g. photovoltaic cells whose output is low-current and relatively slowly varying.

While Hibernus++ identifies whether a source is high-current or low-current, and adjusts thresholds to allow the supply voltage to

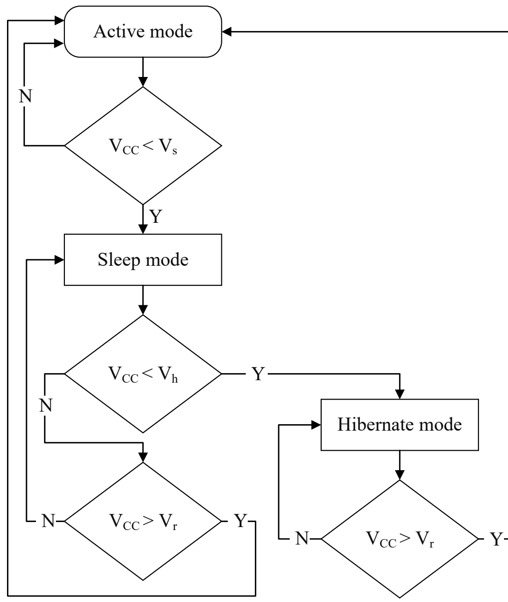

Figure 3: High-level system state diagram for proposed enhancement to Hibernus++ scheme.

build to a higher level for a low-current source (thus increasing the active time of the system), it will still hibernate after the system has been active for a while and the supply voltage has dropped to the hibernate threshold. In most cases, with a low-current harvester, the supply voltage does not drop below V_{min}, therefore the snapshot taken will not be needed. As a result, useful energy is wasted doing very little useful work. Doing this many times without a real power failure incurs comparatively large delays which waste operating time that could be better utilized running the main application.

3 SLEEPING TO AVOID HIBERNATION

As observed in Section 2, a substantial amount of time and energy is wasted by transient computing approaches taking snapshots when they end up not being needed. Figure 3 shows a simplified flow-chart of an approach to address the limitations of Hibernus++ when powered by a low-current energy harvester. In essence, the proposed enhancement enables the system to go to sleep before its supply voltage (V_{cc}) drops below the hibernate threshold (V_h), hoping that it will recover so that this threshold will not be reached.

The system cycle begins in *Active* mode, while the program is executing the application. If V_{cc} is found to be below the sleep threshold (V_s), the program suspends execution and enters *Sleep* mode without saving a snapshot. While in sleep mode, V_{cc} is checked against two thresholds: V_h and V_r (hibernate and restore, respectively). A higher priority is given for V_h, as missing it would delay system hibernation, which could result in a system malfunction. In case the harvested current levels are not sufficient to remain in the sleep state, i.e. if $V_{cc} < V_h$, the system hibernates.

From either *Sleep* or *Hibernation*, if $V_{cc} > V_r$, the system is allowed to go back to *Active* mode. No system restore is required to be performed since the power was not lost at any time. Should V_{cc}

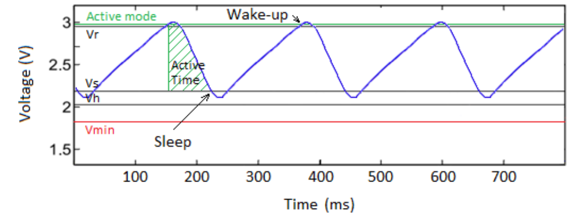

Figure 4: Supply voltage behavior with Enhanced Hibernus++, with low current continuous power source.

have dropped below V_{min}, the system would restore from a valid snapshot if available. Further detail on this behavior is shown later.

Entering *Sleep* avoids the need to hibernate the system, provided that the current generated by EH is less than the sleep-mode current consumption of the system. Voltage thresholds, except for the newly-added V_s, remain the same as in original Hibernus++ [2]. Three states representing different operating modes are shown: *Active*, *Sleep* and *Hibernate*. Throughout different stages of program flow, V_{cc} is being compared against one of the three voltage thresholds: V_s, V_h and V_r.

Figure 4 illustrates the concept of the enhanced approach, with the additional threshold (V_s) using the same low current EH source shown in Figure 2. In contrast to the original Hibernus++, the system is not waiting in *Active* mode until stored voltage levels become critical. Instead, after crossing V_s the system's reduced current consumption in *Sleep* allows V_{cc} to recover. As a result V_{cc} remains above V_h and therefore no snapshot needs to be saved.

In summary, the proposed enhancement allows oscillation between *Sleep* and *Active* modes, rather than *Hibernate* and *Active* modes. Moving from *Sleep* to *Active* is faster and more power-efficient than saving a checkpoint, when entering *Hibernate*, reducing the likelihood of power losses. It should however be noted that the proposed enhancement brings no benefits if the harvesting source is highly variant (e.g. a pulsed source with enough time between each pulse to force hibernation).

4 ALGORITHM DETAILS AND MATHEMATICAL ANALYSIS

Due to the unpredictable behavior of energy harvesters, it is important for the system to switch between different *Sleep* and *Active* modes as quickly as possible. Additionally, to optimize the system's performance, it is desirable for the wake-up time and output current to be as small possible. Therefore a specific low-power mode for the *Sleep* state must be carefully selected.

The MSP430FR5739 development board is selected as the test platform for this work, and is used in the following sections. It was used in the original Hibernus++ paper [2], due to its low power operating modes and built-in FRAM. To implement a *Sleep* state, low-power mode 2 (LPM2) was chosen because it offers a reasonable compromise between current draw and wake-up time.

In this section, the algorithm and design choices are explored using mathematical analysis. The current drawn by the system, I_o, is inversely proportional to the discharge time t_{dis}, which is desired

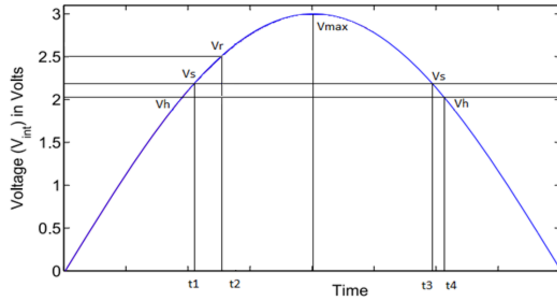

Figure 5: Enhanced Hibernus++ system voltage thresholds, for mathematical analysis.

to be as long as possible. Conversely, the wake-up time $t_{wake-up}$ is directly proportional to the resume time t_{res}, which is required to be as short as possible. By choosing a particular low-power mode, it is impossible to optimise the performance of one parameter without reducing the performance of another.

Figure 5 illustrates an arbitrary voltage input, highlighting V_r, V_s, and V_h thresholds as well as times t_1, t_2, t_3 and t_4. This behavior is analyzed mathematically, based on the equations presented in [2]. Firstly, the time taken to charge the internal capacitance from V_s to V_r is given by:

$$t_{ch} = t_2 - t_1 = \frac{1}{4 f_{source}} - \frac{\sin^{-1} \frac{Vs}{Vr}}{2\pi f_{source}} \tag{1}$$

Here f_{source} is the frequency of the input power source. The time it takes for a microcontroller to go from V_s to V_h, regardless of the power source frequency is given by:

$$t_{dis} = t_4 - t_3 = \frac{C\,(V_s - V_h)}{I_o} \tag{2}$$

The maximum time needed for a microcontroller to fully rsume to *Active* mode from *Sleep* (without hibernating) is given by $t_{res} = t_{ch} + t_{wake-up}$. Every time the voltage threshold is crossed, program appears in the interrupt handler, which temporary disables voltage comparator related interrupts. As a result, additional delays for enabling the comparator and resistor ladder, as well as propagation, need to be taken into account to give the full voltage comparator enable time:

$$t_{en} = \max\{t_{en,comp} + t_{en,rladder}\} + t_{pd} \tag{3}$$

When the system is powered for the first time, a calibration routine is performed to evaluate the power supply characteristics in order to determine the most suitable V_h value [2]. For the system to remain in active mode for as long as possible, $V_s - V_h$ (From equation (1)) is desired to be small. On the other hand, if it is too small, by the time the new threshold parameters are properly configured in hardware, V_{cc} may have already jumped over V_h, in which case system might get stuck in an infinite loop. To avoid this potential problem, combining Equations (2) and (3), the minimum discharge value can be found, which satisfies the following equation:

$$\frac{C\,(Vs - Vh)_{min}}{I_o} < t_{Act \to Sleep} + t_{en} \tag{4}$$

Table 1: Results from 10-minute Simulation

Scheme	Active mode (s)	Sleep mode (s)	Hibernate mode (s)	Active run-time
Original Hibernus++	224.1	-	375.9	37.4%
Enhanced Hibernus++	343.2	275.9	-	57.2%

Here C represents the sum of the MSP430FR5739 board's internal capacitances (characterized by the sum of all system decoupling capacitances), which is found to be 16μF. $(Vs - Vh)_{min}$ is the theoretical minimum difference between the sleep and hibernate voltage thresholds. $t_{Act \to Sleep}$ is the time delay to move from *Active* mode to *Sleep* mode (LPM2).

Assuming that the delay to switch off the processor, system clock generators and oscillator peripherals combined is not greater than two clock cycles (based on the fact that when the clock gating signal is disabled there is no current drawn from the flip-flop), the time it takes to move from *Active* to *Sleep* is less or equal to two high frequency peripheral clock (SMCLK) cycles. This is calculated to be 250 ns.

Expressing the minimum sleep state and hibernate state voltage difference and substituting the values using Equation (4), the minimum difference between voltages is found to be 17.53nV. The MSP430FR5739's internal comparator has a built-in 32-tap resistor ladder which, assuming the absolute maximum voltage threshold for the chosen development board is 3.6V, based on the smallest resolution available $V_s = V_h + 0.11$. As a result, V_s can be safely set to this value, as it is significantly higher than the theoretically calculated minimum value. The system's full state diagram is illustrated in Figure 6.

Voltage comparator interrupts are used to detect when certain voltage thresholds are crossed, depending on the current state of the system. The program starts when power is first applied. After performing self-calibration, power supply and recent failure tests, the program is set to execute the application normally. If the voltage drops below V_s, program execution is suspended and it enters *Sleep*. At this point, if V_{cc} drops below V_h the system will hibernate: i.e. save a snapshot and enter deep sleep mode (LPM4). On the other hand, if V_{cc} rises above V_r, the system will resume to *Active* mode and the whole cycle restarted. If at any time a severe power loss occurs, V_{cc} will drop below V_h, the system will end up in the *Hibernate* state and eventually V_{cc} will drop below V_{min}, meaning that the system state must be restored when the system recovers.

5 SIMULATION

A MATLAB Simulink simulation was used to compare the performance of the proposed Enhanced Hibernus++ system against the original Hibernus++. The simulated system was being powered from a 2 mA Direct Current (DC) current source. The current consumption of the MSP430FR5739 in each power mode was emulated by the circuit in Figure 7; each power mode was assigned a switch

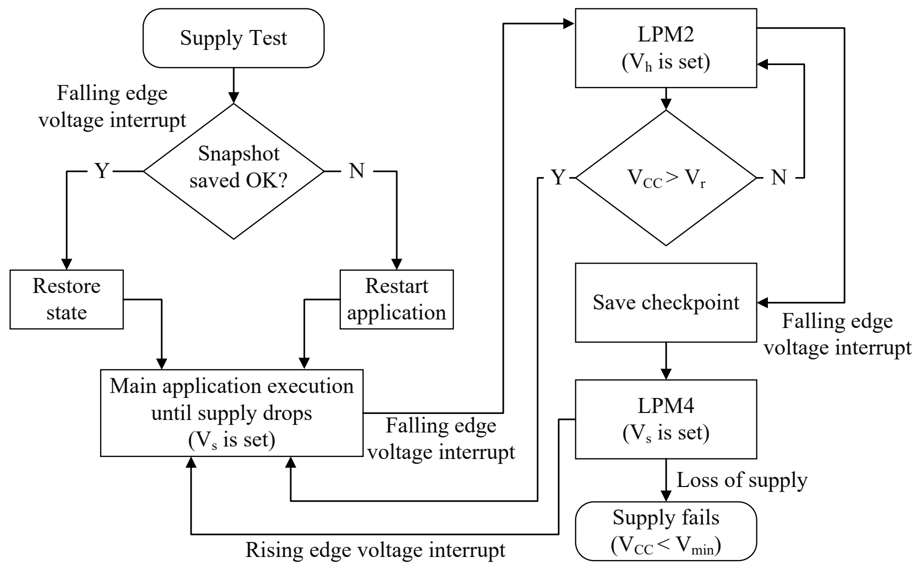

Figure 6: Detailed state diagram for the proposed Enhanced Hibernus++ approach.

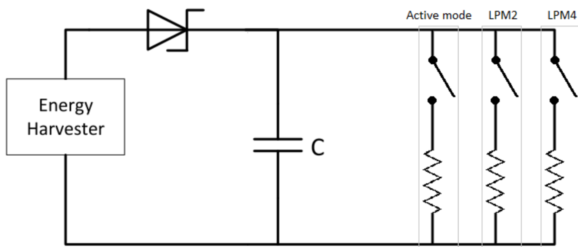

Figure 7: High-level system model schematic, the behavior of the system in each state is represented by a switch and resistor.

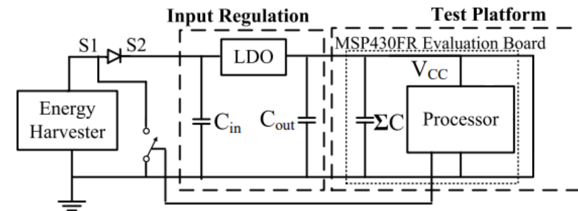

Figure 8: Schematic of test platform circuitry (reproduced from Balsamo, et al. [2]).

and resistor combination. Separate combinational blocks consisting of various control components, such as voltage monitors, edge detectors, and counters were incorporated into the circuit to record the time spent in each mode. Delays (as discussed in Section 4) for switching between power modes, voltage comparator enable times, and system wake-up times were accounted for by MATLAB functions.

The simulation was run for 10 minutes of simulated time, with the results shown in Table 1. Using original Hibernus++ the system spent 37.4% of the time in *Active* mode, while using the Enhanced Hibernus++ approach this increased to 57.2%; this yields an overall 65% improvement in *Active* run-time.

6 EXPERIMENTAL VALIDATION

The program was implemented and evaluated on an MSP430FR5739 development board, connected as shown in Figure 8. The low-dropout (LDO) voltage regulator limits the maximum voltage from the energy harvester, and a Schottky diode prevents the energy harvester from discharging the system capacitance during each negative power cycle. A voltage switch is used to short-circuit the source during the Hibernus++ calibration routine in order to determine V_h and resultantly V_s, as described in Section 4.

To test the system with a representative workload, an 8-bit FFT analysis algorithm was used. The mean *Active* current consumption during algorithm execution was measured as 0.71 mA.

Figure 9 shows the platform's response to a sinusoidal 6 Hz input from a signal generator. Digital output pins were used to indicate the system's operating state (going high when it enters a certain state, and reverting to low after it leaves the state). During the negative power cycle, V_{cc} is discharged to 1.5V. As a result, the system goes through the cycle of *Active* → *Sleep* → *Hibernate*, until it shuts down completely when $V_{cc} < V_{min}$.

Figure 10 shows the performance of the system when powered by a 2 mA DC source (identical to that used in the MATLAB Simulink simulation). Since the supplied current is smaller than the current required to operate the system continuously in *Active* mode, the system oscillates between *Sleep* and *Active* modes. This shows the system operates as predicted by the simulations in Section 5.

Figure 11 shows the performance of the system when powered directly from Schott Solar indoor photovoltaic (PV) module using a

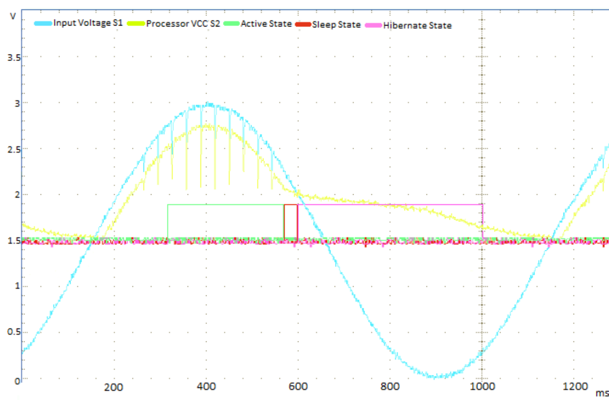

Figure 9: Enhanced Hibernus++ system response to sinusoidal variation in incoming supply voltage.

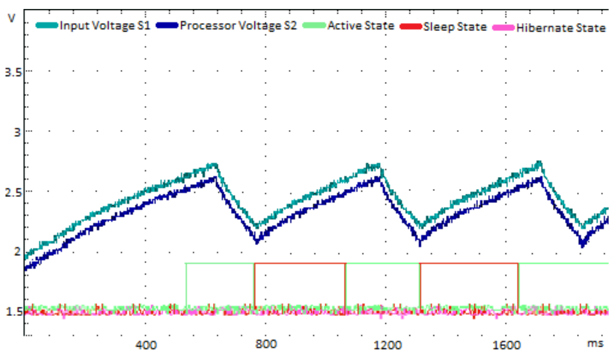

Figure 10: Enhanced Hibernus++ system response to 2 mA constant-current source.

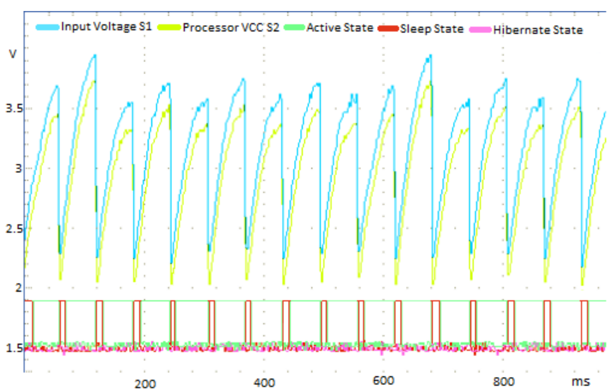

Figure 11: Enhanced Hibernus++ system response when powered from photovoltaic cell.

halogen desk lamp. The maximum voltage was limited to 4 V by the LDO regulator. The light source intensity was set to supply enough power for the system to reach *Active* mode, but not enough to stay in that mode indefinitely. Some variation was added by adjusting the proximity of the lamp to the PV module.

As can be seen from Figure 11, the system oscillates between *Active* and *Sleep* modes, with *Hibernation* completely avoided. As a result, the system was able to spend more time in *Active* mode carrying out useful work, than would otherwise have been the case with the original Hibernus++.

7 CONCLUSIONS

The original Hibernus++ approach is relatively inefficient with low-current harvesting sources, as a substantial proportion of time and energy is wasted in taking snapshots which are not used. The proposed enhancement avoids excessive hibernation cycles by simply going to sleep, reducing the system power consumption and allowing V_{cc} to recover without needing to hibernate.

The enhanced approach adds a sleep mode to Hibernus++, providing a smaller wake-up delay, and avoids saving a redundant snapshot without a power failure. This was found through simulation to be particularly beneficial when the system is powered from low current DC sources, e.g. PV modules. In the simulated scenario, the application run-time was 65% higher than original Hibernus++, indicating that such a system could do substantially more useful work, making more forward progress through an application. This contributes to future improvements of other transient computing systems, e.g. Mementos, where a similar enhancement can be applied.

The proposed method brings benefits where the harvester output is consistently above the system's sleep current draw. In this case, V_s can be set very close to V_h. If the harvesting source output is expected to regularly drop below this value, this could either require V_s to be increased, or result in the system hibernating frequently; either could reduce the active time of the system per cycle. Trade-offs between these have not yet been fully evaluated, and are the subject of future work to find the best way to set V_s for variable harvesting conditions.

REFERENCES

[1] D. Balsamo, A. Das, A. S. Weddell, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini. 2016. Graceful Performance Modulation for Power-Neutral Transient Computing Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 5 (May 2016), 738–749. https://doi.org/10.1109/TCAD.2016.2527713

[2] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini. 2016. Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 12 (2016), 1968–1980. https://doi.org/10.1109/TCAD.2016.2547919

[3] Hrishikesh Jayakumar, Arnab Raha, Woo Suk Lee, and Vijay Raghunathan. 2015. QuickRecall: A HW/SW Approach for Computing Across Power Cycles in Transiently Powered Computers. *J. Emerg. Technol. Comput. Syst.* 12, 1, Article 8 (Aug. 2015), 19 pages. https://doi.org/10.1145/2700249

[4] H. Jayakumar, A. Raha, and V. Raghunathan. 2014. Hypnos: An ultra-low power sleep mode with SRAM data retention for embedded microcontrollers!. In *2014 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 1–10. https://doi.org/10.1145/2656075.2656089

[5] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2011. Mementos: System Support for Long-running Computation on RFID-scale Devices. *SIGARCH Comput. Archit. News* 39, 1 (March 2011), 159–170. https://doi.org/10.1145/1961295.1950386