Improving the Forward Progress of Transient Systems

Tim Daulby, Anand Savanth, *Member, IEEE,* Geoff V. Merrett, *Senior Member, IEEE,* and Alex S. Weddell, *Member, IEEE*

***Abstract*—Emerging applications for Internet of Things devices demand smaller mass, size and cost whilst increasing capability and reliability. Energy harvesting can provide power to these ultra-constrained devices, but introduces unreliability, unpredictability and intermittency. Schemes for wireless sensors without batteries or supercapacitors overcome intermittency through saving system state into non-volatile memory before the supply drops below the minimum operating voltage, termed transient or intermittent computing. However, this introduces significant time and energy overheads. This paper presents two schemes that significantly reduce these overheads: entering a sleep mode to avoid saving state and utilising direct memory access (DMA) when state saves are required. Time and energy previously wasted on state saves can instead be used to perform useful computation, termed "forward progress". We practically validate the proposed approaches across a range of energy sources and IoT benchmarks and demonstrate up to 46.8% and 40.3% increase in forward progress and up to 91.1% and 85.6% reduction in overheads for each scheme respectively.**

***Index Terms*—Embedded systems, energy harvesting, low-power design, transient computing, intermittent computing**

I. Introduction

THE Internet of Things (IoT) is a fast expanding and developing field, ranging from connected homes to concepts of smart cities with a unity of physical world and cloud [1]. A key area of growth is ultra low power sensor devices [2]. To maximise the potential deployment opportunities for these devices, they must be autonomously powered and long-running. Some deployments necessitate a small cost, size and mass; examples include biomedical implants [3], data-rich radio frequency identification and structural monitoring. Powering these types of devices is a significant challenge. Batteries alone incur high maintenance overheads with frequent replacement or charging, or must be much larger, which is impractical. This has prompted developments in energy harvesting (EH) [4].

There are a range of approaches for incorporating EH into IoT devices. Energy neutral computing [5] matches the system demand to the incoming power over a given time period by buffering energy. However, these systems depend on rechargeable batteries that suffer from charge-cycling problems [6] or supercapacitors that increase the size, mass and cost. Transient (or intermittent) computing (Section II) is a new paradigm, powering devices from harvested energy without batteries or supercapacitors. They operate directly from the supply; processing, storing and sending data only when power is available.

A leading transient computing approach is the use of checkpointing and copying a snapshot of the system state to a separate non-volatile memory (NVM), first demonstrated for transient systems by Mementos [7]. At a checkpoint, data is copied from volatile memory, including registers, to a NVM before power failure, incurring an energy (E_{ss}) and time overhead. This data is then copied back when restoring system state when power returns. The smaller these overheads, the more energy and time can be put towards forward progress, that is computation beneficial to the progress of the active applications. To maximise the energy available for forward progress, E_{fp}, all other overheads must be reduced. Equation 1 shows this for a harvested energy budget, E_{harv}.

$$E_{fp} = E_{harv} - (n_{ss} \cdot E_{ss} + E_w + E_R) \qquad (1)$$

Where energy is spent on: forward progress, E_{fp}; a number, n_{ss}, of snapshots (copying data to NVM), E_{ss}; wasted time due to re-execution[1], E_w; and restoring system state, E_R. This establishes a number of trade-offs. Some works, such as Clank [8], have frequent small checkpoints giving a high n_{ss} but low E_{ss}. Other works, such as Hibernus [9], aim for a single state-save per power cycle, greatly reducing n_{ss} and eliminating E_w, but increasing the size of the checkpoint, and therefore the energy to save it, E_{ss}. This single save is achieved by monitoring the supply voltage, checkpointing only when it drops below a set threshold close to the minimum operating point of the processor. This is known as a *reactive checkpointing* approach because it responds to a change in supply voltage, and the single state-save per power-cycle is known as *hibernation*.

EH supplies are typically unreliable, unpredictable and dependent on the environment they are deployed in, and transient systems have been designed to overcome this [10]. However, some EH sources do have predictable traits. Solar and thermoelectric harvesters typically have a slow-varying DC output, particularly outdoor solar. Wind and vibration

Manuscript received September 19, 2019. Accepted May 17 2020. This work was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/P010164/1.

T. Daulby, G. Merrett and A. Weddell are with the School of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, U.K. (e-mail: td2g14@soton.ac.uk)

A. Savanth is with both the University of Southampton and Arm Ltd., Cambridge CB1 9NJ, U.K.

Experimental data used in this article can be found at: https://doi.org/10.5258/SOTON/D1394

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

[1]Re-execution is repeating processing that was lost due to power failure occuring whilst data was held only in volatile memory.

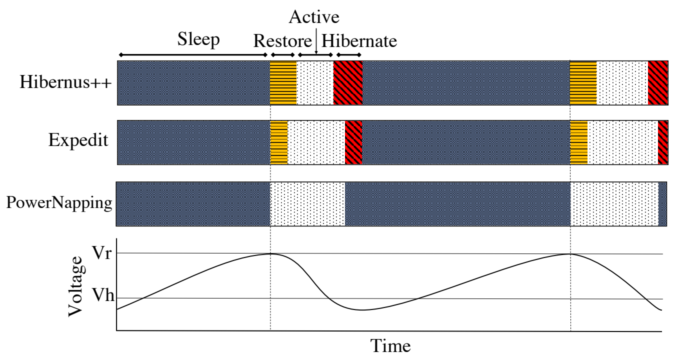


Fig. 1. Illustration of system operation for a given supply with the proposed schemes. Hibernus++ and Expedit restore at a voltage threshold, V_r, and hibernate at V_h. PowerNapping, when enabled, instead of hibernating enters a sleep mode and retains data in volatile memory, simply resuming at V_r.

harvesters may have a more intermittent AC output. Existing reactive transient systems do not account for these traits, and checkpoint when the supply voltage is low, despite the fact that in many cases the voltage would recover naturally, making the checkpoint redundant, if computation was simply suspended in favour of sleep.

This work makes two novel contributions to increase the forward progress of existing transient systems:

1) **PowerNapping.** Works alongside existing reactive checkpointing routines to enter a sleep mode **before** checkpointing. For consistent low-power harvested supplies, e.g. indoor solar, hibernations are avoided, reducing n_{ss} and therefore increasing E_{fp}. Additional safeguards are included to avoid missing a state-save on total power failure (Sec III).
2) **Expedit.** Uses a direct memory access (DMA) controller to handle state-saves and restores, reducing the time and energy overheads of each routine (Sec IV). Decreased E_{ss} and E_R leads to increased E_{fp}.

Figure 1 gives an illustration of the way active time is increased through these two contributions for a voltage trace that decreases when the processor is active, or hibernating/restoring and increases when in sleep. PowerNapping enables supply recovery without resource-intensive state saves, by entering sleep, leading to up to 28.8% improvement in forward progress. Expedit increases the efficiency of hibernation and restore, reducing time and energy overheads, leading to up to 40.3% improvement in forward progress. Both contributions are implemented on an MSP430FR5739 experimenter board with built-in FRAM (ferro-electric RAM) (Sec V)

II. Background and related work

The range of transient computing approaches in the literature can typically be classified as non-volatile processors (NVPs), task-based programming models or checkpointing strategies. The ideal case for these systems is retaining all relevant information in memory on every power loss, therefore being able to resume immediately, with no overheads.

NVPs introduce non-volatility at the processor level by developing state-retentive hardware. This involves flip-flop level NVM and parallel backup strategies [11]. NVPs range from one state-save per power interruption, to back-up every cycle [12]. However, these systems are still in the experimental stage and are not yet commercially available. They also have a larger on chip footprint [13], increasing costs. These systems additionally face idempotency problems and in-rush current peaks due to the large parallel back-up [14].

Task-based programming models such as Mayfly [15], Chain [16], Flexicheck [17] and Alpaca [18] overcome intermittency by ensuring atomic computing tasks are completed each power cycle. There is a large burden on the programmer to break the code into tasks and complex code structures.

In contrast, checkpointing-based systems require little additional complexity at compile time, and can be implemented using commercially available hardware. These systems run intermittently, only when harvested power is available, achieving forward progress by saving the system state to NVM before supply failure. When the supply recovers, the system restores its state from NVM, allowing computation to resume from the point it was interrupted. One of the earliest works, Mementos [7], used static checkpoints, configured at compile time on function returns, every loop, with a timer or manually, to test the supply voltage and save state if below a static threshold voltage. This resulted in a number of checkpoints per power cycle, and data consistency violations[2].

Hibernus [9] instead uses a hardware interrupt to prompt a state save immediately before power failure, giving a single checkpoint per power cycle. A hibernate voltage threshold is set, to ensure sufficient energy in the decoupling capacitance of the system (ΣC) to save state before the supply drops below the minimum operating voltage of the processor (V_{min}). The minimum threshold to save state successfully can be determined by setting the energy required equal to the energy in the capacitance, as given below.

$$n_\alpha E_\alpha + n_\beta E_\beta = \frac{V_h^2 - V_{min}^2}{2} \times \sum C \qquad (2)$$

Where V_h is the threshold voltage, 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 NVM (J/byte).

Energy is not lost on roll-backs, i.e. re-executing code run since the last checkpoint, because the system stops execution, having pre-empted power failure, and waits until the supply voltage exceeds a restore threshold (V_r) or safely dies. This also removes the risk of data consistency violations. Hibernus requires characterisation of the system before compile time. This lack of runtime adjustment could create instability if there is a change to system properties, or a 'safe' hibernate threshold (V_h) must be set to allow changing characteristics at the expense of efficiency. For example changes in the dynamics of the power source, or increased system power consumption due to additional peripherals.

Hibernus++ [19], introduced runtime adjustment of the thresholds (V_r and V_h). When the system is first powered on, it runs a calibration routine that sets V_h, further adjusting if

[2]Data consistency violations can occur when re-initialised volatile values no longer correspond with persistent variables in NVM, and a state is reached that is inconsistent with continuous operation.

a checkpoint ever fails to complete. Hibernus++ anticipates power failure more accurately, but still takes a conservative approach, calibrating and setting voltage thresholds for the worst case power interruption: sudden incoming supply drop to 0 V. This leads to the system hibernating earlier than necessary if supply instead declines slowly. There are further situations where Hibernus++ does not perform well, for example, when the device has a constant low-power source; where the system does not die after checkpointing, but instead begins to recover. If this incoming power is sustained, the data in volatile memory has not been lost, and therefore the effort of taking a checkpoint has been wasted. The MSP430FR5739 used in Hibernus++ and throughout this paper has a number of low-power modes that retain data in registers and RAM with low current consumption ($6\mu A$ in low power mode 4). [20].

It was suggested that this wasteful checkpointing could be solved by introducing a third threshold where the system enters a sleep state without saving state [21]. This could allow the system supply voltage to recover without dropping below V_{min}. This threshold was set statically, as in Hibernus. Additionally, for intermittent high-power sources, where hibernations occur frequently, this technique further reduces system performance. Execution is traded for sleep at a higher voltage, despite hibernation being inevitable, losing forward progress with wasted time and energy.

Hibernus++ remains a key work in this field, with only a few authors re-visiting reactive approaches [22][23]. The key benefit of these reactive systems is that they place little-to-no burden on the programmer, instead allowing standard embedded programs to be compiled and run. They also require little hardware adaptation, unlike NVPs, whilst also avoiding data consistency violations, unlike other checkpointing approaches. The key drawback of reactive systems is their large time and energy overheads compared with other schemes, so this paper demonstrates a clear improvement over the state-of-the-art.

III. PowerNapping

Transient systems are designed to overcome intermittency in the supply, however the nature of this intermittency can vary greatly. We classify three states of harvested power input, P_{Harv}, for the purposes of this paper.

$$P_{Harv} < P_{Slp} : \text{Insufficient power} \tag{3}$$
$$P_{Act} > P_{Harv} > P_{Slp} : \text{Low-power} \tag{4}$$
$$P_{Harv} > P_{Act} : \text{High-power} \tag{5}$$

Where P_{Act} is the active consumption of the device and P_{Slp} is the consumption of the device when in sleep mode. The operating voltage is entirely dependent on the energy stored in the system capacitance. Where the harvested power, P_{Harv}, exceeds the power consumption of the system (P_{Act} or P_{Slp} depending on system state), the energy, and therefore operating voltage, rises; on the other hand, operating voltage will drop, eventually leading to the system responding by sleeping/hibernating. *Low-power* is sufficient to retain state in volatile memory if in sleep mode [20]. If *low-power* is sustained, existing systems still complete resource-intensive hibernations which are unnecessary if the supply can be prevented from dropping below the minimum operating voltage of the processor, V_{min}, with sleep, and therefore retaining volatile data. This energy could otherwise be used to achieve forward progress. Furthermore, for these devices to meet the criteria of small mass, size and cost, the energy harvesters they employ must also be small. These smaller harvesters can be expected to deliver *low-power* more frequently, meaning that an increasing number of checkpoints are taken unnecessarily, since volatile memory contents are not lost. For example a real $1cm^2$ PV cell ranges from $120\mu W$ on a sunny day to $\leq 40\mu W$ when cloudy [24].

We propose pre-emptively triggering sleep instead of hibernation when there is a *low-power* harvested supply. This allows the supply voltage to recover without dropping below the threshold V_h, as defined in Equation 2. The operation of the enhanced PowerNapping approach is detailed in Figure 2. By identifying which power state the system is in (Eq 3-5), and the current supply voltage, this system determines whether it is most effective to sleep, hibernate or restore. High or low power supply is determined at run time by the calibration routine, as explained later, and insufficient power is detected by dropping below V_h during sleep. PowerNapping introduces additional states and decisions as shown within the dotted box, when compared to Hibernus++. This PowerNapping state adaptation could be applied to other schemes such as QuickRecall [25], but Hibernus++ is used throughout this work.

The voltage thresholds V_s, V_h and V_r in Figure 2 are required for reactive checkpointing, and are set at runtime according to the system properties with a calibration routine run when the system is first powered on:

1) An ADC is used to measure voltage (V_1). The supply is then isolated with a diode.
2) The system sleeps and wakes before taking a second reading (V_2).
3) The system hibernates and then takes a third reading (V_3).

The voltage threshold indicating there is sufficient energy for sleep/wake, V_s, is characterised by the voltage drop (V1-V2) and for hibernation, V_h, (V2-V3), giving the following thresholds:

$$V_h = V_{min} + (V_2 - V_3) \tag{6}$$
$$V_s = V_h + (V_1 - V_2) \tag{7}$$

On system start-up, it tests the supply by taking two ADC readings, as shown in Figure 3. If the second is higher than the first, it identifies that there is *a high-power* harvested supply and the system restores state immediately, utilising the ‘abundant’ power to maximise forward progress. There is no downside to this, since, in a transient system, excess energy cannot be stored. If the second reading is lower, there is a *low-power* supply and the system will wait until there is sufficient charge on the capacitor before restoring. Trying to restore immediately would draw power, resulting in a voltage drop below the hibernate threshold. To avoid this, there is an incrementing voltage and timer interrupt. Whilst the voltage

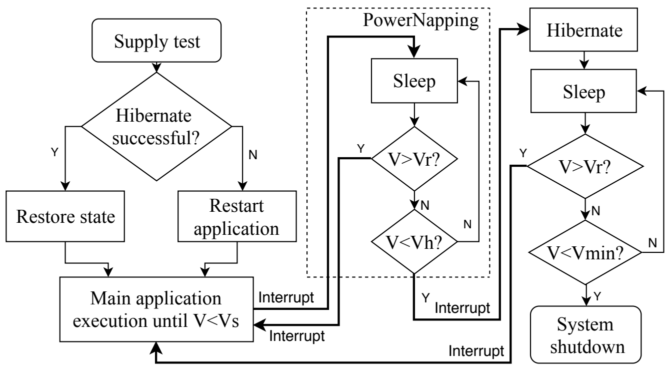


Fig. 2. State diagram of PowerNapping approach

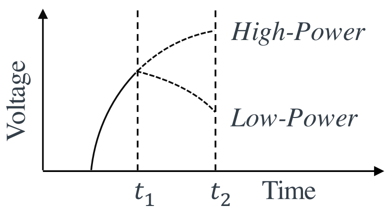


Fig. 3. At t_1 the ADC takes a reading and the CPU begins a test computation. At t_2 another reading is taken, and high/low-power can be established.

interrupts first, the charge is increasing steadily and it is better to continue charging the capacitance. When the timer triggers first, this means that the rate of charge has plateaued and the system restores. The voltage at which this occurs is set as the restore threshold, V_r, used to exit subsequent sleep modes. This will be greater than V_s and V_h.

The system should always save state before power failure, and this is confirmed with a flag triggered at the end of hibernation and cleared on restore. If this flag is not set, the checkpoint did not complete and the application must restart. V_h and V_s are also increased to trigger hibernation earlier on subsequent supply failures, whilst there remains greater energy stored in the capacitance.

When the supply voltage drops below the calibrated threshold, V_s, the system enters sleep, expecting supply recovery. As explained, this only occurs for low-power sources, however P_{Act} is orders of magnitude higher than P_{Slp}, so a *low-power* (Eq 4) supply can occur frequently.

If the supply voltage recovers, the system again waits for sufficient charge on the capacitor. If the supply instead decreases below V_h, i.e. under the *insufficient power* condition, the system wakes, hibernates and returns to sleep. If *insufficient power* continues, the voltage will drop below V_{min} and the system will shut down losing all data in volatile memory.

PowerNapping has a safeguard against missing a state-save. Existing transient systems monitor the restore threshold when sleeping, but with PowerNapping the hibernate threshold is also monitored. This guarantees that when in sleep mode data will not be lost, even on sudden power-failure, whilst also waiting for the optimal restore voltage. The need to monitor a second threshold is the primary overhead of this scheme. After a hibernation, the system can safely lose power without data loss, therefore only the restore threshold is monitored.

A. *Mathematical Analysis*

In Hibernus++, hibernation was completed every time the supply voltage dropped below a certain threshold. For *low-power* sources such as photovoltaic cells, the supply voltage often recovers after this. To maximise active time we want to reduce the time overhead of unnecessary state saves. Equation 8 shows the time available for making forward progress for a given supply profile that leads to a total system on-time, T_{on}. This will be dependent on the number of supply interruptions. We can break this into interruptions that are:

1) avoidable (n_{ia}) - *low-power*, where the supply voltage wouldn't drop below V_{min} if sleep mode was entered.
2) unavoidable (n_{iu}) - *insufficient power*, supply will certainly drop below V_{min}.

$$T_\phi = T_{on} - T_a - n_{ia}(T_h + T_{sw} + T_\lambda) - n_{iu}(T_h + T_r + T_\lambda) \quad (8)$$

Where T_ϕ is the Hibernus++ forward progress, T_a is the time overhead introduced by the Hibernus++ algorithm, T_h is the time taken to hibernate, T_{sw} is the overhead of sleeping and waking, T_r is the time to restore system state and T_λ is the time spent sleeping or shut down before restoring.

The equivalent for PowerNapping is given by:

$$\begin{aligned} T_{\phi pn} = T_{on} - T_{apn} - n_{ia}(T_{sw} + T_\lambda + T_{\lambda pn}) \\ - n_{iu}(T_h + T_r + T_\lambda + T_{sw}) \end{aligned} \quad (9)$$

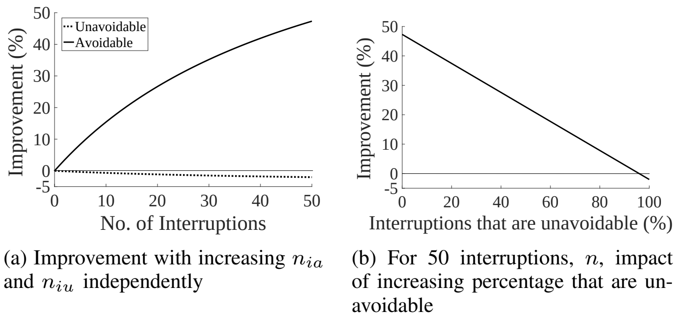
Where $T_{\phi pn}$ is the PowerNapping forward progress, T_{apn} is the time overhead of including the PowerNapping algorithm and $T_{\lambda pn}$ is the additional time in sleep due to PowerNapping. For this system to improve upon Hibernus++, it requires $T_{\phi pn} > T_\phi$. For a given time, T_{on}, by removing equivalent terms we can see the difference in forward progress between PowerNapping and Hibernus++:

$$\begin{aligned} T_{\phi pn} - T_\phi = (T_a + n_{ia}(T_h)) \\ -(T_{apn} + n_{ia}(T_{\lambda pn}) + n_{iu}(T_{sw})) \end{aligned} \quad (10)$$

The algorithm time difference, $T_a - T_{apn}$, is negligible. Using an MSP430FR5739 experimenter board, we found that T_h and $T_r \approx$ 1.87ms and from the data sheet T_{sw} is $78\mu s$ for LPM4. An MSP430FR5739 is chosen for this work due to its built in FRAM memory and prevalence in related transient-compute works. Equation 11 demonstrates how the improvement is dependent on the quantity of n_{iu} and n_{ia}.

$$T_{\phi pn} - T_\phi = n_{ia}(T_h - T_{\lambda pn}) - n_{iu}(T_{sw}) \quad (11)$$

This relationship is visualised in Figure 4, demonstrating that PowerNapping is most beneficial in situations where V_{min} is unlikely to be reached i.e. *low-power* sources, where n_{ia} >> n_{iu}. This benefit increases proportionally with the number of interruptions for a given time, T_{on}. When the majority of interruptions are unavoidable, as for intermittent *high-power* sources, PowerNapping is no longer beneficial and will be disabled. Figure 4b shows that for 50 interruptions PowerNapping is useful until >96% of interruptions are unavoidable.



(a) Improvement with increasing n_{ia} and n_{iu} independently

(b) For 50 interruptions, n, impact of increasing percentage that are unavoidable

Fig. 4. Improvement of PowerNapping with changing n_{ia} and n_{iu} over 50 interruptions

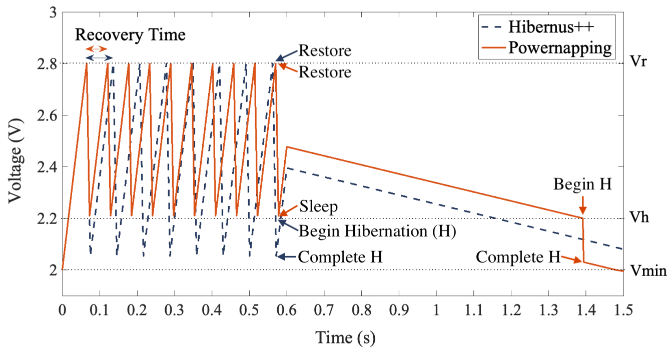


Fig. 5. Comparison of Hibernus++ and PowerNapping for a constant current source interrupted at ~600ms

If the checkpoint is avoidable, then all the energy that would have been wasted on a state save is retained. From the MSP430FR5739 data sheet, E_α = 4.2 nJ/byte and E_β = 2.7 nJ/byte, with a total RAM size of 1024 bytes and register size of 512 bytes. Compared with a complete state save, there should be 5.7μJ more energy available for forward progress. $T_{\lambda pn}$ is minimised by having the PowerNapping threshold (V_s) as close to the hibernate threshold (V_h) as possible, achieved at runtime using the calibration routine described previously.

B. Simulation

The objective of PowerNapping is to increase forward progress for *low-power* supplies. To verify the behaviour, and predict performance, mathematical simulation was used. Energy consumption for active, sleep, hibernation and restore were obtained from both the MSP430FR5739 data sheet [20] and physical characterisation. MATLAB was used to run these values through the mathematical models with various test conditions and input sources as demonstrated below.

Figure 5 shows the results of a simulation of the system response to an ideal 200 μA source that drops to 0 μA at ~600 ms. This is the worst case test, as typically *low-power* EH sources have slower current variation over time. An example case where this could occur is an indoor PV cell, powered with a consistent incoming light, having the light suddenly switched off. Each time the supply drops to the threshold V_h the Hibernus++ algorithm hibernates; moving the contents of RAM, registers and CPU state to FRAM, using energy from the decoupling capacitance of the system. The hibernation routine completes before reaching V_{min} (2 V in this case) at which point the system enters sleep and the supply recovers. PowerNapping instead enters sleep immediately and therefore recovers more quickly. For this supply profile, PowerNapping increases the forward progress by 24.4%, because less energy is wasted on hibernating. When the supply does drop below V_h, due to *insufficient power* during sleep, the system is able to wake and hibernate as seen around 1400 ms.

These two approaches were also compared using a double diode PV cell model [26] as the supply input. PV cells are low current DC sources, but do not supply a constant current. The current varies according to the IV characteristics of the cell. A cell was selected with a maximum power point (MPP) voltage within the operating voltage range of the MSP430. The model uses the open-circuit voltage, short circuit current, MPP voltage and MPP current to generate an IV curve, fed into the MSP430 MATLAB model. With the PV cell, PowerNapping increased forward progress by 18.3% with ~300 μA supply and 27.5% with ~100 μA supply.

The PowerNapping strategy should allow the supply voltage to recover without saving system state, but in the case where the supply does not recover, will hibernate successfully. From simulation, it is expected that the lower the input current, the greater the benefit of PowerNapping. Wasteful hibernations occur more frequently for Hibernus++ in this case, since the lower supply current leads to faster discharging of the capacitance whilst making forward progress.

PowerNapping has little overhead. The system characterises the source on start-up. For bursty *high-power* supplies, where the supply typically dies without recovery, PowerNapping is disabled and there is no additional overhead. When PowerNapping is utilised, the overhead is (1) an increase to the hibernation time due to sleeping first, and (2) the inclusion of a second voltage threshold monitoring circuit using the internal comparator. This overhead is included in simulation and the physical test setup.

It has been shown that decreasing the number of resource intensive hibernations leads to an increase in forward progress. This increase in forward progress leads to faster program execution, and a more efficient system. Improvements would therefore demonstrate an increased active time, and reduced completion time for benchmarks on a physical system, as demonstrated in Section V.

IV. Expedit - Reducing state save overheads

DMA controllers allows data to be accessed and copied without intervention from the CPU. They are useful in embedded systems for controlling memory transfers between peripherals and main memory. Their function is typically for speed-up, multi-tasking or reducing load on the processor, however for large transfers it is possible for the processor to enter sleep, in order to save power. Moving the contents of registers and RAM to NVM via DMA has not been done in any transient computing papers to date. Equation 8 shows that time spent on hibernate and restore has a direct impact on forward progress. By making hibernate and restore routines quicker, there will be more time spent on useful computation, T_ϕ, for a given time, T_{on}.

Reactive transient systems necessitate large memory transfers from main memory into NVM. By transferring the registers and RAM into FRAM with the DMA, time and energy overheads can be reduced. The DMA is used in block-transfer mode. It is initialised with a source address in VM, and a destination address in NVM (or vice versa) and the size of transfer. The RAM can be copied in a single block, however the important registers in the MSP430 are distributed, and are transferred in multiple blocks. For the MSP430, a CPU-controlled data transfer uses 5 cycles, whereas DMA uses 2 [20]. Additionally, DMA is more power efficient, whilst also allowing the CPU to enter sleep. This gives a lower total power consumption when DMA is in use. The on-chip area overhead is also small, DMAs with a gate count of 3-10k are commercially available [27].

The benefits of Expedit are applicable any time data is copied from volatile to non-volatile memory or vice-versa. This applies to the majority of the leading transient papers, across both reactive and task-based approaches. The larger the block of data, the lower the relative impact of initialising the DMA. Reactive transient systems which copy the entire contents of RAM and registers, such as Hibernus++, therefore gain the greatest benefit. Expedit is simulated and practically validated as an extension to this system, and PowerNapping.

The Hibernus++ code uses a number of 'for' loops, arrays and temporary variables to copy the registers and RAM. The code for hibernate and restore with Expedit is simpler and so we can expect the size of the code written to the MSP430 to be significantly reduced. PowerNapping introduces additional interrupts and configuration of the internal comparator, so we can expect an increase to the code size for this scheme.

A. Simulation

In simulation, the overhead of initialising the DMA is considered negligible, because the cycles required to move data are orders of magnitude greater than initialising. Because the DMA uses 2 cycles per word, instead of 5, hibernate and restore times are reduced by 60%. The energy consumption of the MSP430FR5739 was measured to be 10% less with the DMA active and CPU in sleep, so this is also included.

For *low-power* supplies, the system does not drop below V_{min}. Therefore only T_h is relevant, and the improvement is expected to be less than for *high-power* supplies. Figure 8 shows the expected improvement of Expedit in simulation, alongside the practical validation.

Expedit is expected to perform better against Hibernus++ when both T_h and T_r are significant, such as with frequently interrupted *high-power* sources. The simulated improvement is given in Figure 11, alongside the practical validation.

V. Practical Validation

The Hibernus++ code was adapted and programmed onto an MSP430FR5739 experimenter board. This was connected to an external threshold detection circuit, monitoring V_H or V_R respectively. The internal comparator is only used for V_S.

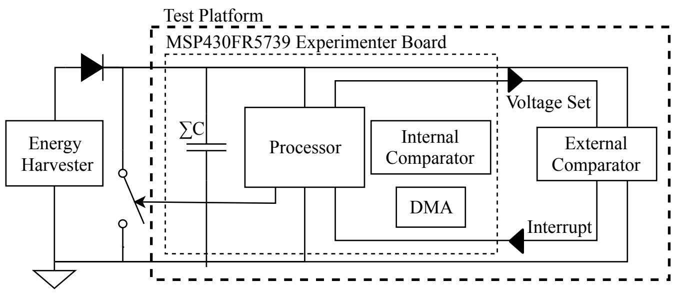


Fig. 6. Schematic of the test platform

TABLE I
Total overhead comparison of PowerNapping (PN) and Expedit (Exp) against Hibernus++ (H++)

	H++ (μs)	Exp (μs)	PN (μs)	Exp+PN (μs)
Restore	1874	306	1876	306
Hibernate	1876	555	1878	555
Sleep	N/A	N/A	166	166

(a) Time to complete listed functions

	H++ (μJ)	Exp (μJ)	PN (μJ)	Exp+PN (μJ)
Restore	7.30	1.05	7.31	1.05
Hibernate	7.31	1.91	7.32	1.91
Sleep	N/A	N/A	0.65	0.65

(b) Energy to complete listed functions

The experimental setup is shown in Figure 6. The test setup for Hibernus++ and the proposed approach is kept the same for all experiments and an FFT analysis is used as a test bench for comparing forward progress with a range of supply conditions. The FFT is completed three times emulating a realistic load: processing data from a 3-axis accelerometer. Additional benchmarks are included to demonstrate the performance of this scheme across different compute loads.

The proposed scheme is a reactive transient system that saves the entire contents of RAM and registers during hibernation. For this reason, time and energy overheads of hibernation and restore are not application dependent. Percentage improvement to completion time seen in these results is also independent of application, since the entire state is saved regardless of what that state is. For this reason other benchmarks are not included for clarity of results.

The energy harvester output is rectified by a Schottky diode. A low-dropout regulator (LDO) could be included to regulate the voltage within safe limits, however none of the sources used in testing exceeded the maximum operating voltage of the MSP430. The internal comparator is only active when in sleep, so there is additional power consumption in this state. For a 2 V supply, the internal comparator consumes 75 μA whereas the external comparator consumes 1 μA. PowerNapping requires both simultaneously. Two external comparators could be used in future implementations. If re-designed for on-chip implementation, more efficient comparators could be used, further reducing the overhead of PowerNapping. Table I compares the restore, hibernate and sleep for each scheme. These remained constant across all benchmarks presented.

The expectation is that Expedit reduces the hibernate and

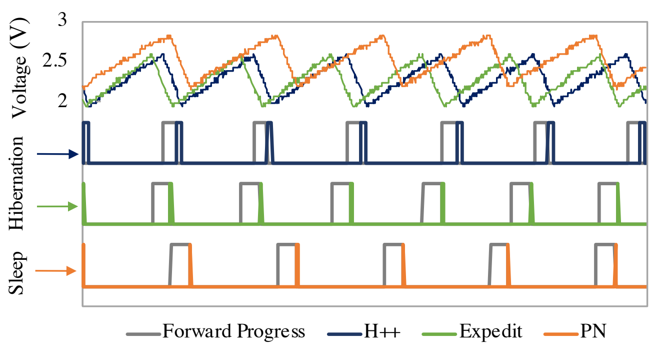


Fig. 7. Comparison of 3 schemes with PV cell input energy source, showing an increase in forward progress for PowerNapping and Expedit.

restore times, while PowerNapping has little overhead, but implements a sleep state that is reached much more quickly than hibernation. This is shown with Expedit improving hibernation time by 70.4% and restore, 83.6%. PowerNapping, which operates only when the harvester is generating a *low-power* supply, is 91.1% faster than hibernating.

A. *Low-power Sources*

Energy harvesters such as photovoltaic (PV) cells and thermo-electric generators (TEGs) typically supply power with lower current, but a more stable supply profile than other harvesters. PV cells also have the highest power density of mainstream energy harvesters [24]. PowerNapping is most effective for these *low-power* sources, but Expedit's reduced hibernate time is also beneficial.

Figure 7 compares Hibernus++ (H++), Expedit and PowerNapping (PN) with a 10x3cm PV cell under 500 lux indoor light, to demonstrate how hibernations impact active time. This delivers ~300 μA of current. Both Expedit and PowerNapping allow the processor to remain active for longer, seen by the increased width of forward progress, because they reduce costly hibernations. The decoupling capacitance charges earlier, leading to more frequent periods of forward progress, because less energy is spent on hibernations. This leads to improvement in the completion time of the FFT.

Figure 8 demonstrates this for a range of source conditions. For *low-power* supplies PowerNapping allows immediate supply recovery, increasing forward progress. The restore routine isn't utilised for *low-power* supplies, since the processor doesn't drop below V_{min}, and volatile data is not lost. PowerNapping removes resource-intensive hibernations by instead entering sleep. This occurs much more quickly as seen in Table I(a), leading to this improvement in the completion time. For lower currents this improvement exceeds our expectation from simulation. In simulation, V_r is approximated as a fixed value, however as explained in Section III, V_r varies according to the source characteristics. At lower currents, the capacitor charges more slowly, leading to a lower V_r and consequently more frequent hibernations. This favours PowerNapping and Expedit with their lower overheads.

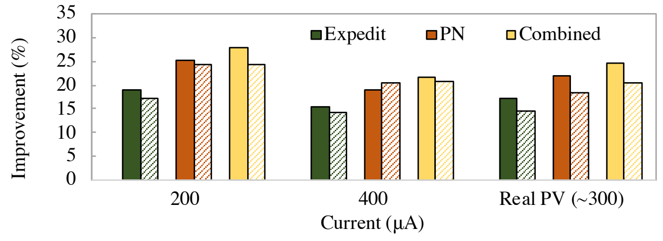


Fig. 8. Improvement in FFT complete time for the two schemes over Hibernus++ for simulation and practical validation with *low-power* sources. Solid colour represents practical validation, and cross-hatch, simulation results.

TABLE II
COMPARISON OF CODE OVERHEAD FOR PROPOSED SCHEMES.

Additional Overhead	Code Bytes	Data Bytes
FFT alone	1852	1248
Hibernus++	+3486	+1508
Expedit	+3348	+586
PowerNapping	+4038	+1584
Combined	+4136	+1124

B. *High-power Sources*

Table I(b) shows the improvement in energy consumption. Expedit not only improves energy efficiency by completing the hibernate/restore more quickly, but the DMA also consumes less power. This leads to a hibernate energy cost reduction of 73.9% and restore, 85.6%. By reducing these overheads, more time and energy goes towards forward progress.

Table I(a) shows that Expedit is faster than expected in simulation. This is thought to be due to the additional algorithm simplifications, such as not reading register locations from a look up table. Table II shows how this also affects the size of code written to the MSP430 when programming the device. FFT is also given as an example base application code size.

As can be seen, Expedit reduces the size of the code overhead by over 1000 bytes (11%). This may enable Hibernus++ to be implemented on more tightly constrained systems. Hibernus++ contained large arrays of address locations (increasing data bytes), and 'for' loops for moving data (increasing code bytes). These elements are removed by using DMA with optimised address location initialisations.

Sources such as wind harvesters and piezo-electric vibration harvesters generate power with high current, but are typically bursty sources, with intermittent power. In a reactive transient system with little energy storage, the processor is likely to hibernate and restore frequently.

In this validation, an ideal DC voltage is interrupted with varying frequency to approximate the system response to these types of harvester. Figure 9 further demonstrates the impact of these reduced times by comparing the percentage overhead for a range of supply interruption frequencies. The higher the supply interruption frequency, the more state saves are required, and the greater the benefit of Expedit. Figure 9a demonstrates how the percentage of the active time spent on hibernate/restore is increasingly less than Hibernus++ and Figure 9b also, with percentage of energy.

The increase in active time can more clearly be seen for one

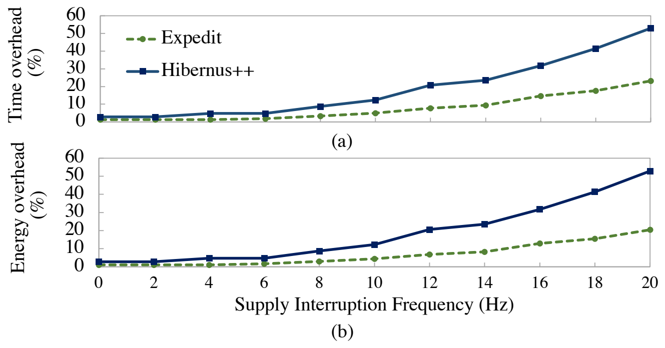


Fig. 9. Overhead Comparisons for Expedit and Hibernus++

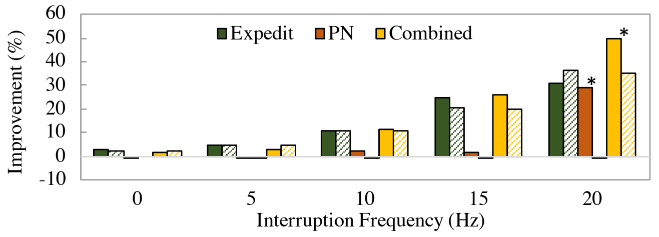


Fig. 11. Improvement in FFT complete time for the two schemes over Hibernus++ for simulation and practical validation with *high-power* sources. The solid colour represents practical validation, and the cross-hatch is simulation results. The * signifies a result where PowerNapping is enabled and by entering sleep the system avoids dropping below V_{min} across some interruptions.

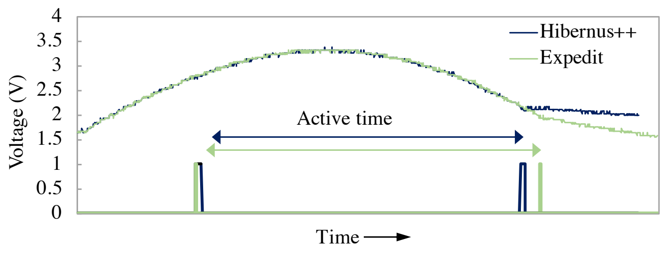


Fig. 10. Expedit Increase in Active Time with sinusoidal voltage input

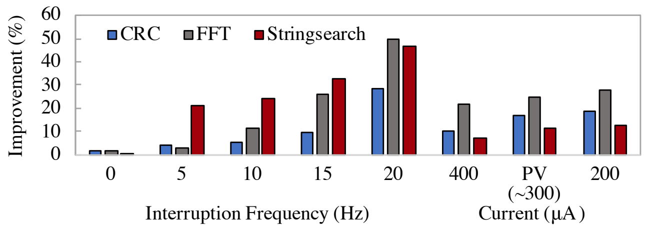


Fig. 12. Improvement of Expedit and PowerNapping combined for FFT, CRC and Stringsearch with high and low-power sources

cycle of a *high-power* AC input, as in Figure 10. By reducing the time and energy required, the hibernation can begin later, at a lower voltage threshold. This is established when first calibrating the system as explained in Section III-A.

Reduced time and energy overheads, and consequently increased active time have both been shown. The total time to complete 3 FFTs was also measured for a range of interruption frequencies. This further highlights the benefits of this scheme since faster task completion means that the processor is able to handle more complex tasks for a given supply profile.

Figure 11 compares the total on-time, T_{on}, of the processor when completing 3 FFTs. Removing time when supply is not available helps clarify the comparison. This figure shows that Expedit is able to complete the FFTs up to 30.8% faster than Hibernus++. As expected, PowerNapping does not give an improvement for high-power supplies. However, the results demonstrate that the associated overheads are minimal, except in the case of high-frequency interruptions which instead show significant improvement: up to 50.8% for PowerNapping alone, 51.7% when combined with the faster restore time of Expedit. At these high frequencies, the supply is interrupted faster than the ADC startup test can occur, defaulting to the system assuming *low-power* and enabling PowerNapping, unlike in simulation (Section III-B).

C. Additional Benchmarks

Figure 12 shows these schemes provide improvement for a range of IoT benchmarks, giving up to 35.2%, 50.0% and 46.8% improvement respectively. With the addition of both Expedit and PowerNapping, Hibernus++ is improved for both high and low-power sources. As shown in Figure 4a, greater benefits are gained when more interuptions occur during a compute cycle, so results vary across benchmarks. Stringsearch takes 1045ms to complete, much longer than CRC at 103.8ms, and therefore more interruptions occur, across all frequencies, hence a greater improvement is observed for *high-power* sources. Conversely for *low-power* sources, longer base completion time means that hibernations make up a smaller percentage of total completion time, leading to lower percentage improvement being observed.

VI. Conclusions

With the two schemes, PowerNapping and Expedit, it has been demonstrated that the forward progress of reactive transient systems can be improved. PowerNapping allows time and energy previously wasted on hibernation to be used to increase the forward progress of the system. Improvements are particularly significant with *low-power* supplies, with up to 28.8% improvement in forward progress over the state-of-the-art. At high interruption frequencies, with *high-power* sources, PowerNapping can achieve up to 46.8% improvement. Expedit reduced the time and energy overheads of saving state by up to 83.6% and 85.6% respectively. This gives up to a 40.3% improvement in completion time over the state-of-the-art. The benefits of these two schemes have been shown experimentally with both ideal and real EH sources across multiple benchmarks. On the whole, results align with those expected from the simulation, with the schemes being more effective at higher interruption frequencies, or with lower current inputs. In combination they bring improvement with a full range of high and low power supplies of up to 50.0%. Tested with Hibernus++ [28] in this work, these schemes demonstrate a reduction in overheads that would be beneficial to other reactive transient works such as QuickRecall [25], or task-based approaches where large checkpoints occur.

References

[1] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Comput. Networks*, vol. 54, pp. 2787–2805, Oct. 2010.

[2] 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, pp. 1645–1660, Sept. 2013.

[3] P. D. Mitcheson, “Energy harvesting for human wearable and implantable bio-sensors,” in *2010 Int. Conf. of the IEEE Eng. in Med. and Biol*, pp. 3432–3436, IEEE, 2010.

[4] H. Jayakumar, K. Lee, W. S. Lee, A. Raha, Y. Kim, and V. Raghunathan, “Powering the internet of things,” in *2014 Int. symp. on Low power electron. and des.*, ISLPED ’14, (New York, NY, USA), pp. 375–380, ACM, June 2014.

[5] M. Rossi, L. Rizzon, M. Fait, R. Passerone, and D. Brunelli, “Energy neutral wireless sensing for server farms monitoring,” *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 4, pp. 324–334, Sept. 2014.

[6] A. Somov and R. Giaffreda, “Powering iot devices: Technologies and opportunities,” *IEEE IoT Newsletter*, Nov. 2015.

[7] B. Ransford, J. Sorber, and K. Fu, “Mementos: System support for long-running computation on RFID-scale devices,” *ACM SIGPLAN Notices*, vol. 46, pp. 159–170, May 2012.

[8] M. Hicks, “Clank: Architectural support for intermittent computation,” in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, pp. 228–240, June 2017.

[9] D. Balsamo, A. S. Weddell, and G. V. Merrett, “Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems,” *IEEE Embedded Syst. Lett.*, vol. 7, no. 1, pp. 15–18, 2015.

[10] A. Rodriguez Arreola, D. Balsamo, and A. K. Das, “Approaches to Transient Computing for Energy Harvesting Systems: A Quantitative Evaluation,” *Proc. Int. Workshop Energy Harvesting Energy Neutral Sens. Syst. (ENSsys)*, pp. 3–8, Nov. 2015. ISBN: 9781450338370.

[11] K. Ma, X. Li, K. Swaminathan, Y. Zheng, S. Li, Y. Liu, and Y. Xie, “Nonvolatile Processor Architectures: Efficient, Reliable Progress with Unstable Power,” *IEEE Micro*, vol. 36, pp. 72–83, May 2016.

[12] K. Ma, X. Li, S. Li, Y. Liu, J. J. Sampson, Y. Xie, and V. Narayanan, “Nonvolatile Processor Architecture Exploration for Energy-Harvesting Applications,” *IEEE Micro*, vol. 35, pp. 32–40, Sept. 2015.

[13] Y. Wang, Y. Liu, S. Li, D. Zhang, and B. Zhao, “A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops,” in *Proc. ESSCIRC*, pp. 149–152, Sept. 2012.

[14] Y. Liu, F. Suy, Z. Wangy, and H. Yang, “Design exploration of inrush current aware controller for nonvolatile processor,” in *Proc. IEEE Non-Volatile Memory Syst. Appl. Symp. (NVMSA)*, (Hong Kong, Hong Kong), pp. 1–6, IEEE, Aug. 2015.

[15] J. Hester, K. Storer, and J. Sorber, “Timely Execution on Intermittently Powered Batteryless Sensors,” in *Proc. ACM Sensys*, (Delft, Netherlands), pp. 1–13, ACM Press, 2017.

[16] A. Colin and B. Lucia, “Chain: tasks and channels for reliable intermittent programs,” in *Proc. OOPSLA*, (Amsterdam, Netherlands), pp. 514–530, ACM Press, Nov. 2016.

[17] P. Singla, S. S. Singh, and S. R. Sarangi, “Flexicheck: An adaptive checkpointing architecture for energy harvesting devices,” in *Proc. Des. Autom. Test Eur. Conf. Exhibit.(DATE)*, pp. 546–551, IEEE, Mar. 2019.

[18] K. Maeng, A. Colin, and B. Lucia, “Alpaca: intermittent execution without checkpoints,” *Proc. ACM Progam. Lang.*, pp. 96:1–96:30, Oct. 2017.

[19] D. Balsamo, A. S. Weddell, and A. Das, “Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, pp. 1968–1980, Mar. 2016.

[20] Texas Instruments, *MSP430FR573xx Mixed-Signal Microcontrollers*. http://www.ti.com/lit/ug/slau272d/slau272d.pdf, Dec. 2017.

[21] G. Lukosevicius, A. R. Arreola, and A. S. Weddell, “Using Sleep States to Maximize the Active Time of Transient Computing Systems,” in *Proc. Int. Workshop Energy Harvesting Energy Neutral Sens. Syst. (ENSsys)*, (Delft, Netherlands), pp. 31–36, ACM Press, Nov. 2017.

[22] S. Ahmed, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola, “Efficient intermittent computing with differential checkpointing,” in *Proc. Int. Conf. Languages, Compilers, and Tools for Embedded Systems*, pp. 70–81, ACM, June 2019.

[23] K. Maeng and B. Lucia, “Supporting peripherals in intermittent systems with just-in-time checkpoints,” in *Proc. 40th ACM SIGPLAN Conf. Program. Lang. Design and Implementation (PLDI)*, (New York, NY, USA), pp. 1101–1116, June 2019.

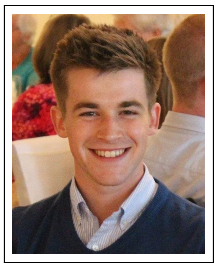
[24] A. Savanth, A. Weddell, J. Myers, D. Flynn, and B. Al-Hashimi, “Photovoltaic cells for micro-scale wireless sensor nodes: Measurement and modeling to assist system design,” in *Proc. Int. Workshop Energy Harvesting Energy Neutral Sens. Syst. (ENSsys)*, (New York, NY, USA), pp. 15–20, Nov. 2015.

[25] H. Jayakumar, A. Raha, and V. Raghunathan, “QUICKRECALL: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers,” *Proc. IEEE Int. Conf. VLSI Design*, pp. 330–335, Jan. 2014.

[26] M. C. Di Piazza and G. Vitale, *Photovoltaic sources: modeling and emulation*. Springer Science & Business Media, 2012.

[27] “PrimeCell uDMA Controller (PL230) Technical Reference Manual,” 2007.

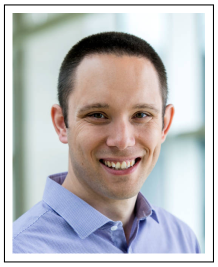
[28] D. Balsamo, A. Das, and A. S. Weddell, “Graceful Performance Modulation for Power-Neutral Transient Computing Systems,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, pp. 738–749, May 2016.



Tim Daulby received the B.Eng. degree (1st class honors) in electronic engineering from the University of Southampton, U.K., in 2017. He is currently pursuing a Ph.D. in electronic engineering with the ECS Group, University of Southampton. His research is focused on transient computing checkpointing strategies and hardware, software co-design for efficient batteryless embedded systems.

Anand Savanth received the master’s degree in microelectronics from the University of Liverpool in 2010 where he received the Sir Robin Saxby Award. He is currently pursuing the Ph.D. degree with the ECS Group, University of Southampton, with a focus on custom and analog-assisted circuits for IoT platforms and energy harvesting applications. He has been with the Devices, Circuits and Systems Group, ARM Research, developing ultra-efficient circuits for Arm embedded processors and holds over 10 granted patents.

Alex S. Weddell (GSM’06-M’10) received the M.Eng. degree (1st class honors) and Ph.D. in electronic engineering from the University of Southampton, U.K., in 2005 and 2010. His main research focus is in the areas of energy harvesting and energy management for future Internet of Things devices. He has over 15 years’ experience in energy harvesting systems, and has published over 65 papers in the area. He is now a Lecturer at the University of Southampton, involved with three projects funded by EPSRC, EU Horizon 2020 and Clean Sky 2.



Geoff V. Merrett (GSM’06-M’09-SM’19) is Professor of Electronic and Software Systems at the University of Southampton, U.K., where he previously received the B.Eng. (Hons) and Ph.D. degrees in 2004 and 2008, respectively. He is co-director of the Arm-ECS Research Centre, an award winning collaboration between the university and Arm Research, Cambridge. His current research interests are in energy management of mobile/embedded systems and self-powered devices, and he has published over 200 journal and conference articles on these topics.