Exploring ARM mbed Support for Transient Computing in Energy Harvesting IoT Systems

Domenico Balsamo, Ali Elboreini, Bashir M. Al-Hashimi, Geoff V. Merrett
Department of Electronics and Computer Science
University of Southampton
Southampton, United Kingdom, SO17 1BJ
Email:{db2a12, ae5g14, bmah, gvm}@ecs.soton.ac.uk

***Abstract*—Energy harvesters offer the possibility for embedded IoT computing systems to operate without batteries. However, their output power is usually unpredictable and highly variable. To mitigate the effect of this variability, systems incorporate large energy buffers, increasing their size, mass and cost. The emerging class of transient computing systems differs from this approach, operating directly from the energy harvesting source and minimizing or removing additional energy storage. Different transient computing approaches have been proposed which enable computation to be sustained despite power outages. However, existing approaches are largely designed for specific applications and architectures, and hence suffer from not being broadly applicable across multiple embedded IoT platforms. To address this challenge, transient approaches need to be integrated within a general IoT programming framework such as ARM's mbed IoT Device Platform. In this paper, we explore how state-of-art transient computing approaches can be integrated into mbed, increasing ease-to-use and deployment across different platforms. This support is offered through libraries and application programming interfaces (APIs) provided by the ARM mbed OS, which enable transient computing to be implemented as a service on top of IoT application protocols. We demonstrate the ability for a transient approach to operate effectively on mbed, by practically implementing it on a low-power NXP microcontroller (MCU) with Flash memory, operating from only 1 mF additional capacitance.**

I. Introduction

Energy harvested from the environment can be used to power autonomous IoT computing systems, alleviating the burden of periodic battery replacement [1]. A primary challenge in developing embedded IoT systems with energy harvesting (EH) is the unpredictable and highly variable nature of the sources [2], [3]. Fig. 1 shows experimentally obtained power output traces from three photovoltaic (PV) cells located outdoor in different places. Here, the harvested outputs vary by many orders of magnitude over the experimental time period, illustrating significant spatial variability.

To accommodate this variability and sustain computation, systems incorporate large energy buffers (such as rechargeable batteries or supercapacitors) [4]. This approach is known as *energy-neutral* operation, where the energy consumed equals the energy harvested over a period of time. Energy neutral systems can overcome the variability in EH supplies, but they also suffer from increased volume, weight and cost due to the requirement for additional energy storage. Moreover, these buffers need time to charge, pose environmental issues, and deteriorate in performance over time [5]. For embedded IoT systems with constrained dimensions, such as implantable biosensors for medical applications [6], smart energy metering [7], personal healthcare [9], building automation [8] and RFID applications [10], it is desirable to reduce the size of the system. However, systems with small energy buffers are subjected to frequent power interruptions due to the highly variable supply.

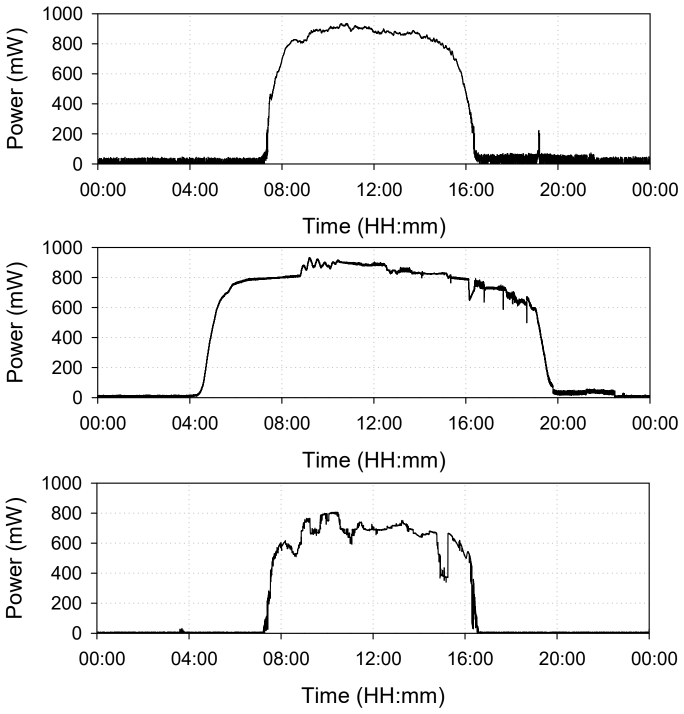

Fig. 1: Experimentally obtained power outputs from three photovoltaic (PV) cells located outdoor in three different sites.

Transient computing approaches enable computation to be sustained despite power outages, allowing systems to achieve forward application execution. This is obtained by retaining the system state before a power failure occurs, and restoring it once the power supply recovers. Existing transient computing approaches are largely designed for specific applications (such as RFID applications) or specific platforms (which use fast low-power non-volatile memories (NVMs) [11]), whilst they

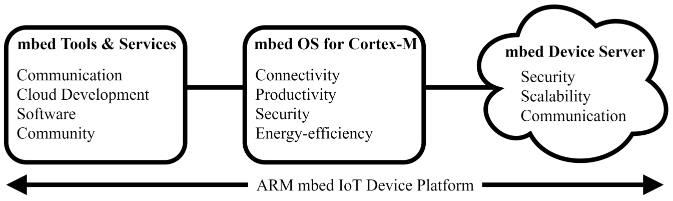

Fig. 2: ARM mbed ecosystem architecture.

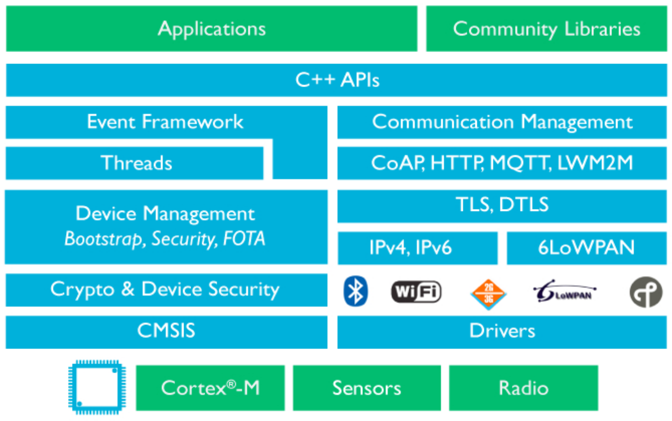

Fig. 3: ARM mbed OS (from ARM mbed website [14]).

suffer from not being broadly applicable across multiple embedded IoT platforms (such as platforms using standard Flash memories). This limits their use as they require considerable time and expertise to implement them on a new platform.

To address this challenge, we propose that it is useful for transient computing to be integrated as part of IoT programming frameworks. In this context, an IoT programming framework is defined as a set of principles, standards and protocols, which facilitate the implementation of IoT applications [12], [13]. A common example of an IoT programming framework is ARM's mbed IoT Device Platform [14], a tool-kit providing an open-source embedded operating system (ARM mbed OS), in addition to a large number of tools and libraries designed for standard-based IoT applications. It also provides drivers for different existing IoT communication protocols such as WiFi, Bluetooth Low Energy, Thread, 6LoWPAN and Ethernet.

In this paper, we explore how existing transient computing strategies can be integrated into mbed, aiming to increase ease-of-use across different IoT platforms. To achieve this, the novel contributions reported in this paper are:

1) An exploration of existing transient computing approaches and their suitability for working within the ARM mbed ecosystem;
2) A validation of this through the implementation and evaluation of the *Hibernus* [20] transient approach on an NXP mbed device, utilizing libraries and APIs function from the mbed OS;
3) Demonstration of the *Hibernus* transient approach working with Flash memory (existing systems have used FRAM memory).

The remainder of the paper is organized as follows. In Section II, ARM mbed properties and design requirements for transient applicability are presented. In Section III, an evaluation of the *Hibernus* transient approach is presented and implemented using the mbed OS, while its operation on a mbed device with Flash memory is presented in Section IV. Finally, Section V concludes the paper.

II. Transient Computing Compatibility with ARM mbed

In this section, the ARM mbed ecosystem is discussed, highlighting the main properties which affect our design decisions for transient computing (section II-A). Following this, an exploration of the existing transient approaches and their suitability for working within mbed is presented (section II-B). Finally, consideration for the necessary requirements for the selected transient approach to be integrated as part of mbed are discussed (section II-C).

A. ARM mbed properties

The mbed IoT Device Platform is an IoT programming framework proposed by ARM, which provides full support for a large variety of embedded devices. As shown in Fig. 2, it provides an operating system (ARM mbed OS), cloud services (server and client-side components to enable communication between mbed devices) and a complete set of libraries, APIs and tools for programming, building and testing standard-based IoT applications for embedded systems.

The mbed OS is the fundamental component as it includes all the features needed for developing IoT applications, comprising security, connectivity and drivers for I/O peripherals and sensors (Fig. 3). The mbed ecosystem provides an online integrated development environment (IDE) which gives access to the mbed OS without requiring specific pre-settings. This allows fast prototyping, without using expensive debug hardware (targeted for low-cost development boards).

In the following, the main properties which affect our design decision are presented:

- The mbed OS has been tested across ARM Compiler 5, GCC and IAR compilers, and mbed projects can be built using these tool-chains. Moreover, projects can also be exported to be used with other IDEs, such as the Keil MDK. However, this may impact the applicability of transient approaches which require modifications at the compiler-level;
- It is targeted for standard ARM Cortex-M MCU platforms with Flash memory. This may impact the performance of transient control approaches which require fast non-volatile memory (NVM);
- It provides a hardware-enforced security level which restricts access to memory and peripherals (uVisor supervisory kernel). This may impact the applicability of

transient approaches which require access to protected memory segments for state backup and restore.

B. Exploring transient approaches and their sustainability

Different transient computing approaches have been proposed to facilitate system state retention and restore [15]. In this section, these approaches are explored, aiming to identify their suitability for operating with mbed.

Broadly, these can be classified into two categories: 1) software-based approaches, where the system state retention is enabled using a software strategy and without requiring dedicated hardware support, and 2) hardware-assisted approaches, where the system is partially or entirely designed to be non-volatile such as Non-Volatile Processors (NVPs) [18], minimizing the overhead needed for state retention.

An early software-based approach was *Mementos* [16], which places static trigger points at compile time, following various strategies (e.g. before a function call or inside each loop). When a trigger point is reached, *Mementos* saves an image of the system state (snapshot) into NVM, if the supply voltage is below a static save threshold. However, *Mementos* requires compiler-level changes (LLVM compiler modification) to enable automatic placements of trigger points, making this approach unsuitable for mbed which supports different compilers that do not allow significant modifications. Moreover, *Mementos* only saves the core registers, part of the main memory, the stack and the global variables (.bss and .data segments), without including the .heap segment which is allocated dynamically. This limits its range of applicability to mbed platforms, as the usage of the .heap segment is suggested for higher programming flexibility with more complex IoT applications (such as machine learning [17]). Finally, *Mementos* does not operate correctly in general cases where the main IoT application can update both the main memory and NVM, as it allows applications to continue forward execution after saving the system state (only the main memory but not the NVM), creating inconsistent states.

QuickRecall [19] is a hardware/software-assisted approach which addresses these issues by using an interrupt-driven strategy for saving the system state, and NVM as a unified memory (using low-power and fast FRAM memory). Therefore, when the supply voltage drops below the minimum operating voltage, it only needs to save the core registers and general-purpose registers (GPRs), as the RAM is already stored in the NVM. However, this approach is applicable only with fast low-power NVMs and customised hardware, making it unsustainable for mbed devices with Flash memories. Moreover, using NVM as unified memory is not efficient due to the higher power consumption and lower access time when compared to SRAM memory, making this approach unsuitable for EH low-power systems [23].

Hibernus [20] is a software-based interrupt-driven approach which addresses *Mementos* and *QuickRecall* shortcomings, by introducing guard bands. A guard band is a voltage threshold (V_H) that relates to the amount of energy required

TABLE I: mbed requirements for transient.

	Transient Approaches			
mbed requirements for transient	Mementos [16]	QuickRecall [19]	Hibernus [20]	Hibernus++ [21]
No compiler-level modification required	🟥	✓	✓	✓
No specific architectures and extra hardware required	✓	🟥	✓	🟥
Applicable to Flash memory	✓	🟥	✓	✓
Compatible with mbed hardware security level	✓	✓	🟥	🟥

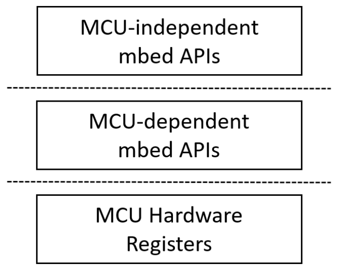

Fig. 4: The mbed APIs divided into three macro-layers for hardware abstraction.

to hibernate the entire system state to NVM (fast and low-power FRAM memory) by using only the on-board decoupling capacitance. This ensures that a snapshot is saved before a power failure occurs. The snapshot is restored when the supply rises above a second guard band, the restore voltage threshold (V_R). *Hibernus* significantly improves *Mementos* in terms of performance, as it employs a comparator (typically available on-chip), allowing a reactive response when the supply voltage drops below V_H or rises above V_R. This removes the software overheard due to the periodic checking of the supply voltage. However, this approach has been only implemented on systems with FRAM memories, and saves the entire main SRAM memory to NVM. This can impact its performance in terms of time and energy overhead due to the state retention process when applied to devices with Flash memory (e.g. mbed-based devices). Moreover, by saving the entire state, this approach requires access to protected memory areas, which violates the hardware-enforced security level provided by the mbed OS.

Hibernus++ [21] is an adaptive version of *Hibernus* which enables greater forward progress in application execution by self-calibrating V_H and V_R, depending on the platform and EH source. However, this approach requires extra-circuitry, which is not available on-chip for typical mbed platforms and undesirable for an easy-to-use approach across different IoT platforms.

Table I summarizes the main requirements of mbed which affect the presented transient approaches. This table is useful to identify the most suitable transient approach for mbed in the next section.

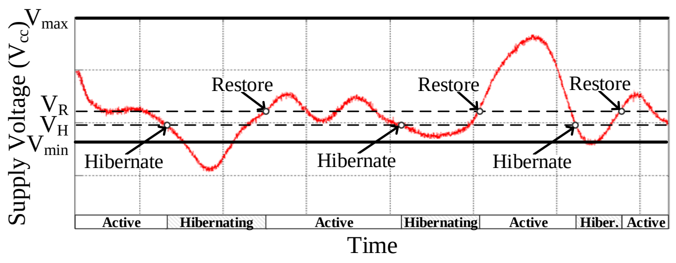

Fig. 5: Operation of *Hibernus* in response to a variable supply voltage [20].

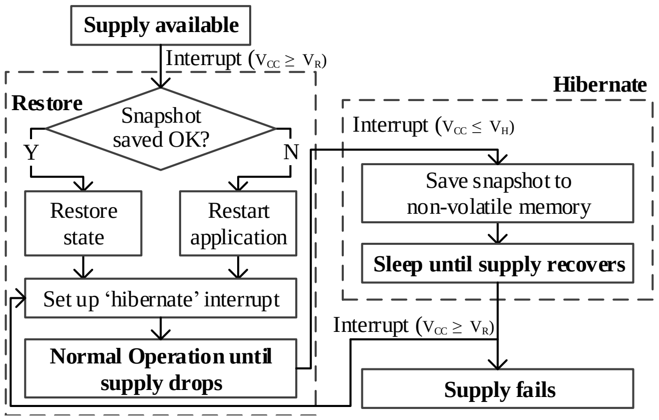

Fig. 6: Flow-chart illustrating the *Hibernus* approach.

C. Requirements for transient approach with mbed

As the selected transient approach aims to be broadly applicable, without dictating specific hardware constrains or extra-hardware (such as *QuickRecall* and *Hibersus++*), and without requiring modifications at compiler-level (such as *Mementos*), the previous exploration indicates that *Hibernus* is the most suitable approach to be incorporated as part of the mbed ecosystem (see Table I). However, *Hibernus* requires some modifications in order to be suitable for mbed devices with Flash memory and restricted access to the the main memory due to the hardware security level.

As already underlined in section I, mbed OS support is offered through APIs, which enables *Hibernus* to be implemented as a service on top of IoT application protocols. As shown in Fig. 4, these APIs are divided into three macro layers: MCU-independent APIs, MCU-dependent APIs and MCU hardware registers. Specifically, *Hibernus* will require MCU-dependent mbed APIs to access to the Flash memory for writing and reading operations, as this operation is platform and memory-dependent, while it will require MCU-independent mbed APIs to trace the allocated main SRAM memory to be saved, without violating the hardware-enforced security level. This is discussed in detail in section III.B, where the *Hibernus* transient approach implementation and evaluation on an mbed NXP device is presented.

III. Hibernus evaluation and implementation on ARM mbed device

Fig. 5 illustrates the desired behaviour of *Hibernus* in response to a variable supply voltage. When the voltage falls below V_H, the system saves a snapshot and hibernates. When the voltage rises above V_R, the system restores a snapshot and continues operation.

Fig. 6 outlines the operation of *Hibernus*. When the power is first applied to the system and the supply voltage rises above the restore threshold V_R, the system checks whether a valid snapshot was previously stored in NVM: if not, the system sets the hibernate interrupt and continues normal operation until the supply voltage drops below V_H, at which point the system hibernates. If the system recovers without dropping below the MCU's minimum operating voltage, the system resumes operation without the need to restore its state. Otherwise, if the supply voltage has dropped below the minimum operation voltage causing the volatile memory contents to be lost, the system restores its state, provided that it was saved successfully.

As already underlined in section II-C, *Hibernus* has been previously implemented on systems with fast and low-power FRAM memory, copying all registers and main memory to NVM using only energy stored in the decoupling capacitance available on these platforms. However, to make it suitable for mbed devices such as NXP MCUs with Flash memory, extra capacitance may be required to guarantee that snapshots can be taken reliably. This aspect is discussed in section III-A. *Hibernus* also needs to address the issue related to the restricted access to the main memory provided by mbed, by dynamically tracing the allocated memory to be saved during hibernation and without violating the mbed hardware-security level. This aspect is discussed in section III-B.

A. Energy Storage

To address the challenge of retaining the system state reliably, the relevant parameters of Flash memories which impact the hibernate and restore processes have to be considered. Specifically, Flash memory operates by erasing a block of memory cells (a page) before writing. The size of this page can range between several bytes to a few kilobytes. A given number of pages form a sector, which is a larger block of memory, and can typically be erased at the same cost of erasing a single page. Flash memory is asymmetrical, meaning that reading is faster and more power efficient than writing.

The energy consumed by the hibernate process with Flash memory, E_σ, depends on the erasing cost and the energy consumption for copying the RAM and registers to Flash

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

Here, n_s is the number of sectors to be erased and n_α is the size of the RAM and registers (in bytes). E_s is the energy required to erase one sector (J) and E_α are the energy required to copy each RAM byte to NVM (J/byte).

Hibernus requires enough energy to be stored in the capacitance between the supply rails to save a snapshot (see test

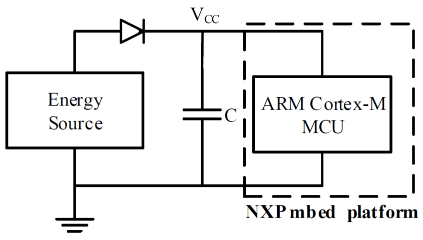

Fig. 7: Test platform used to validate *Hibernus* on mbed.

platform in Fig 7). Given a defined V_H, the energy E_δ stored between V_H and systems's minumum opration voltage V_{min} is:

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

To ensure stability, sufficient C must be present or added so that $E_\sigma > E_\delta$, to enable complete hibernation (even with a sudden loss of supply).

B. Selective Memory Saving

As mentioned in section II-C, *Hibernus* can only access the allocated memory, whilst the unallocated memory can not be accessed without violating mbed hardware level security. To overcome this constraint, a strategy for state retention which dynamically identifies the allocated space and only saves to Flash memory the parts of the main memory being used by the main application has to be considered.

The content of the RAM that must be saved in Flash memory includes the .bss, .data, .heap, and .stack segments, in addition to the GPRs and stack pointer (SP), program counter (PC), and link register (LR) (previously saved in RAM) before hibernating. To identify these segments, the end of the .heap segment and the top of the .stack need to be tracked. This can be done by using a combination of *malloc()* and *free()* functions to locate the end of the .heap segment and by tracking the SP to locate the top of the .stack segment.

The ARM mbed library provides support for memory tracing to extract information about the address and size of dynamic memory allocations. Specifically, the *mbed_mem_trace* library contains a set of callback functions are linked to standard memory allocation functions (e.g. malloc, calloc, free) to return this information. These are MCU-independent mbed API functions which can be activated through the use of the compile-time macro MBED_MEM_TRACING_ENABLED.

Information about the stack size can be accessed through the *mbed_mem_stats* library. In the same way, this can be activated through the use of the compile-time macro MBED_HEAP_STATS_ENABLED.

In Application Programming (IAP) can be used to write to internal flash during application execution. Library functions to perform these operations exist within mbed, however at present they are not fully MCU-independent but target a single group of MCUs. Specifically, IAP for NXP MCUs is supported by mbed within a single library. This process will be used to demonstrate the saving and restoring of data to Flash during hibernation.

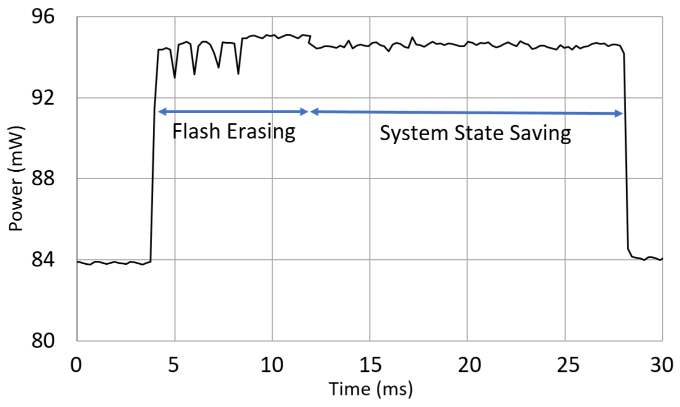

Fig. 8: Time and power consumption for hibernating the system state with Flash memory including erasing.

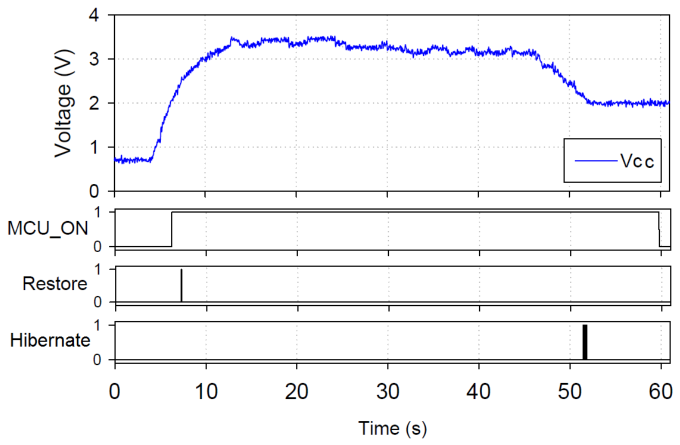

Fig. 9: Demonstration of *Hibernus* on the mbed NXP platform.

IV. Experimental Validation

Hibernus has been validated with an mbed development board, the NXP FRDM-KL05Z MCU built on the ARM Cortex-M0+ processor. Fig. 7 shows the test platform used for this validation, where the MCU input is connected to the output of the energy source through a diode, which prevents back-flow of charge to the harvester. Moreover, an extra capacitance C is added to enable a complete hibernation with Flash memory. The evaluation test case represents a simple binary counter, which uses basic mbed API functions such as *DigitalOut()* and *Wait()* functions. A synthesized trace (shown in Fig. 9) has been used to demonstrate the correctness of *Hibernus* with this mbed platform. This trace was obtained from real EH (PV cell) and replayed via a source-measurement unit.

To calculate the proper value of C which enables complete hibernation, the energy consumed by the hibernate process has to be considered (Eq. (1)). Fig. 8 shows the time and the power needed for hibernating the system state to Flash. As underlined in section III.A, Flash memory requires a significant amount of energy due to the erasing process ($n_s E_s$). In this case, the amount of RAM memory that needs to be saved in Flash, n_α, is smaller than 1 kB (which is the sector size for this Flash memory) due to limited use of the .bss and .data segments by the main application, requiring only one segment to be erased before writing (n_s = 1). Specifically, the time needed to save the system state is equal to 24 ms, consuming an average power of 94 mW corresponding to an energy E_σ equal to 2.2 mJ. Using Eq. 2, the required capacitance C is equal to 1.5 mF, considering a value of V_H equal to 2.4 V and V_{min} equal to 1.7 V for this MCU. However, this value can be smaller with a higher V_H. For example, setting a V_H equals to 2.7 V, the required extra capacitance is lower than 1 mF. It was verified experimentally that V_R = 2.3 V delivers stable operation, even with sudden loss of supply at the beginning of a restore operation.

Fig. 9 demonstrates the correctness of *Hibernus* while operating with this mbed platform. Here, the system remains in sleep mode until the voltage V_{CC} reaches the restore threshold. Once the system state has been restored, the system continues normal operation until the voltage drops below the hibernate threshold. After saving the system state, the MCU is placed in sleep mode before the power outage.

V. Conclusion

In this paper, we explored how state-of-art transient computing approaches can be integrated into the ARM mbed ecosystem, to increase easy-of-use support across different platforms. To achieve this, an exploration of the mbed ecosystem has been presented to highlight the main properties which affect the design decisions for transient computing. Following this, an exploration of existing transient computing approaches has been conducted underlining their suitability for operating with mbed and the necessary requirements for the selected transient approach to be integrated as part of mbed. A validation of the transient approach with mbed was presented through the implementation and evaluation of the *Hibernus* approach on an NXP device utilizing mbed libraries and APIs. Specifically, this has been implemented by using both MCU-dependent mbed APIs to access to the Flash memory, and MCU-independent mbed APIs to trace the allocated main SRAM memory to be saved, demonstrating that *Hibernus* operates correctly on mbed devices with Flash memory (existing systems have used FRAM memory) from only 1 mF of added capacitance. Our continuing work is extending *Hibernus* as a general mbed library, by splitting the MCU-dependent mbed APIs and MCU-independent mbed APIs and validating it on a larger number of mbed platforms.

Acknowledgment

This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) Grants EP/P010164/1 and EP/K034448/1 (the PRiME Programme www.prime-project.org). Experimental data used in this paper can be found at DOI:10.5258/SOTON/D0102 (https://doi.org/10.5258/SOTON/D0102).

References

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

[2] H. Jayakumar *et al.*, "Powering the Internet of Things," *2014 IEEE/ACM Int. Symp. on Low Power Electronics and Design (ISLPED)*, La Jolla, CA, 2014, pp. 375-380.

[3] P. Bogdan *et al.*, "Making the internet-of-things a reality: from smart models, sensing and actuation to energy-efficient architectures," *In Proc. of the Eleventh IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES '16)*, ACM, New York, NY, USA, Article 25, 10 pages.

[4] F. Ongaro *et al.*, "Li-Ion Battery-Supercapacitor Hybrid Storage System for a Long Lifetime, Photovoltaic-Based Wireless Sensor Network," *IEEE Trans. Power Electron.*, 27, 9 (Sept 2012).

[5] V. Raghunathan *et al.*, "Design considerations for solar energy harvesting wireless embedded systems," *IPSN 2005. Fourth Int. Symp. on Inform. Process. in Sensor Networks, 2005*, 2005, pp. 457-462.

[6] P. D. Mitcheson, "Energy harvesting for human wearable and implantable bio-sensors, *2010 Annu. Int. Conf. of the IEEE Eng. in Medicine and Biology*, Buenos Aires, Argentina, 2010, pp. 34323436.

[7] D. Porcarelli *et al.*, "Perpetual and low-cost power meter for monitoring residential and industrial appliances," *2013 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, Grenoble, France, 2013, pp. 1155-1160.

[8] D. Balsamo *et al.*, "Long term, low cost, passive environmental monitoring of heritage buildings for energy efficiency retrofitting," *in Proc. IEEE Workshop Environ. Energy Struct. Monitor. Syst. (EESMS)*, Trento, Italy, 2013, pp. 16.

[9] G. Acampora *et al.*, "A survey on ambient intelligence in healthcare," *Proc. IEEE*, vol. 101, no. 12, pp. 24702494, Dec. 2013.

[10] S. Naderiparizi *et al.*, "WISPCam: A battery-free RFID camera," *2015 IEEE Int. Conf. on RFID (RFID)*, San Diego, CA, 2015, pp. 166-173.

[11] J. S. Meena *et al.*, "Overview of emerging nonvolatile memory technologies, *Nanoscale Research Letters*, 2014, vol. 9.

[12] H. Derhamy *et al.* "A survey of commercial frameworks for the internet of things," *in Emerging Technologies Factory Automation (ETFA)*, 2015 IEEE 20th Conference on, Sept 2015, pp. 18.

[13] L. F. Rahman *et al.*, "Choosing Your IoT Programming Framework: Architectural Aspects," *2016 IEEE 4th Int. Conf. on Future Internet of Things and Cloud (FiCloud)*, Vienna, 2016, pp. 293-300.

[14] "Welcome to ARM mbed" http://www.mbed.com/en.

[15] B. Lucia *et al.*, "Intermittent Computing: Challenges and Opportunities," *In Summit on Advances in Programming Languages (SNAPL)*, 2017.

[16] B. Ransford *et al.*, "Mementos: system support for long-running computation on RFID-scale devices," *SIGARCH Comput. Archit., News 39, 1 (March 2011)*, 159-170.

[17] C. Leech *et al.*, "Real-time room occupancy estimation with Bayesian machine learning using a single PIR sensor and microcontroller," *2017 IEEE Sensors Applications Symposium (SAS)*, Glassboro, NJ, 2017, pp. 1-6.

[18] K. Ma *et al.*, "Nonvolatile Processor Architectures: Efficient, Reliable Progress with Unstable Power," *in IEEE Micro*, vol. 36, no. 3, pp. 72-83, May-June 2016.

[19] H. Jayakumar *et al.*, "QuickRecall: A HW/SW Approach for Computing across Power Cycles in Transiently Powered Computers," *J. Emerg. Technol. Comput. Syst.*, 12, 1, Article 8 (August 2015), 19 pages.

[20] D. Balsamo *et al.*, "Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems," *in IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 15-18, March 2015.

[21] D. Balsamo *et al.*, "Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices," *in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 1968-1980, 2016.

[22] K. Ma *et al.*, "Nonvolatile Processor Architectures: Efficient, Reliable Progress with Unstable Power," *in IEEE Micro*, vol. 36, no. 3, pp. 72-83, May-June 2016.

[23] A. R. Arreola *et al.*, "Approaches to Transient Computing for Energy Harvesting Systems: A Quantitative Evaluation," *In Proceedings of the 3rd International Workshop on Energy Harvesting & Energy Neutral Sensing Systems (ENSsys '15). ACM*, New York, NY, USA, 3-8.