Dynamic Voltage and Frequency Scaling for Intermittent Computing

ANDREA MAIOLI, Politecnico di Milano, Italy
KEVIN A. QUINONES, Politecnico di Milano, Italy
SAAD AHMED, Georgia Institute of Technology, U.S.
MUHAMMAD H. ALIZAI, Lahore University of Management Sciences, Pakistan
LUCA MOTTOLA, Politecnico di Milano, Italy and Uppsala University, Sweden

We present hardware/software techniques to intelligently regulate supply voltage and clock frequency of intermittently-computing devices. These devices rely on ambient energy harvesting to power their operation and small capacitors as energy buffers. Statically setting their clock frequency fails to capture the unique relations these devices expose between capacitor voltage, energy efficiency at a given operating frequency, and the corresponding operating range. Existing dynamic voltage and frequency scaling techniques are also largely inapplicable due to extreme energy scarcity and peculiar hardware features. We introduce two hardware/software co-designs that accommodate the distinct hardware features and function within a constrained energy envelope, offering varied trade-offs and functionalities. Our experimental evaluation combines tests on custom-manufactured hardware and detailed emulation experiments. The data gathered indicate that our approaches result in up to 3.75× reduced energy consumption and 12× swifter execution times compared to the considered baselines, all while utilizing smaller capacitors to accomplish identical workloads.

CCS Concepts: • **Computer systems organization → Embedded hardware**; **Embedded software**.

Additional Key Words and Phrases: Battery-less devices, dynamic voltage and frequency scaling, intermittent computing.

1 INTRODUCTION

Ambient energy harvesting enables battery-less embedded sensing [2, 30, 35, 36, 40, 79, 86]. However, energy from the environment is generally erratic, causing frequent and unanticipated energy failures. Executions thus become *intermittent*, as they consist of intervals of active operation interleaved by periods of recharging energy buffers [16].

Battery-less devices typically employ capacitors as energy buffers. As intuitively shown in Fig. 1, as long as the capacitor voltage is below a predetermined *boot threshold*, the device rests dormant until the buffered energy is sufficient to boot. An *energy cycle* then starts when the device actively operates. The energy consumption during this cycle typically exceeds the ambient energy intake, leading to a net negative energy balance. Consequently, the capacitor voltage drops below the *operating voltage*, causing the device to shut down, at which point a new charging phase begins.

Due to extreme resource constraints of the target platforms, applications run on bare hardware without proper operating system support. Energy failures thus normally cause devices to lose computational and peripheral states. To ensure forward progress across energy failures, techniques [9, 11, 12, 14, 17, 18, 21, 38, 61–63, 65–67, 75, 83] exist that, at the cost of significant overhead, allow the system to save the computational and peripheral state onto non-volatile memory (NVM) locations, which persist across energy failures. Once the boot threshold is attained again, the state is restored from NVM, and execution picks up near the point where the energy failure occurred.

Frequency, voltage, and the rest. With low-power microcontrollers, system efficiency is typically dictated by the rate of energy consumption and execution speed. These parameters are influenced by the running frequency, supply voltage, and operating range [4]. Consider the MSP430-G2553 [43] microcontroller unit (MCU) of the TI MSP430 series,

Authors' addresses: Andrea Maioli, andrea1.maioli@polimi.it, Politecnico di Milano, Italy; Kevin A. Quinones, kevinalessandro.quinones@mail.polimi.it, Politecnico di Milano, Italy; Saad Ahmed, sahmed@gatech.edu, Georgia Institute of Technology, U.S.; Muhammad H. Alizai, hamad.alizai@lums.edu.pk, Lahore University of Management Sciences, Pakistan; Luca Mottola, luca.mottola@polimi.it, Politecnico di Milano, Italy and Uppsala University, Sweden.

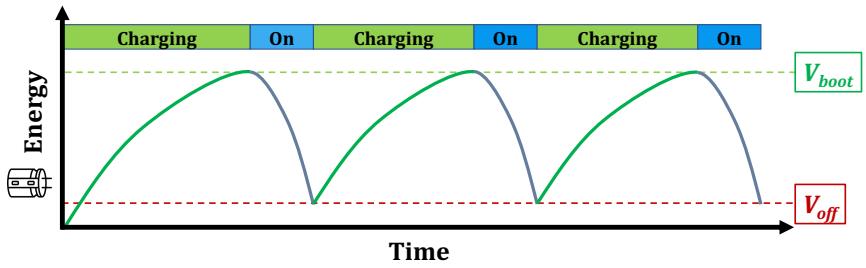


Fig. 1. Example of intermittent execution.

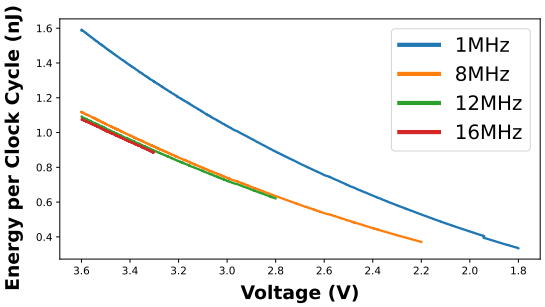


Fig. 2. Energy consumption per clock cycle at various voltage and frequency ranges for the MSP430-G2553 [4, 43].

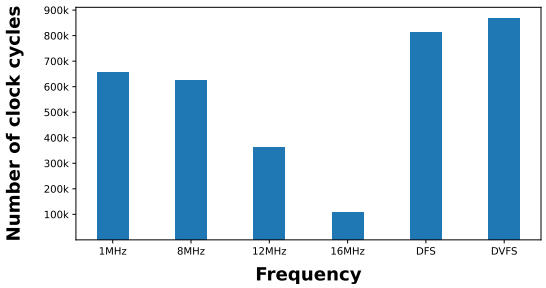


Fig. 3. Clock cycles executed in a single discharge from $3.6V$ of a $100\mu F$ capacitor for various frequency configurations for the MSP430-G2553 [43].

that is, arguably the most used MCU platform in battery-less devices. Fig. 2 shows the energy consumption per clock cycle at the four factory-calibrated operating frequencies. The higher the frequency, the faster the computation and the lower the energy consumption per clock cycle. For example, running the MCU at $16MHz$ is on average 47% more energy efficient per clock cycle and $16x$ faster than the $1MHz$ setting. However, compared to the latter, running the MCU at $16MHz$ limits the operating voltage range: as soon as the supply voltage falls below $3.3V$, the MCU shuts down. Differently, if the MCU is set to run at $1MHz$, it can continue operating until the supply voltage reaches $1.8V$.

Fig. 3 demonstrates the impact of these trade-offs on the number of clock cycles the MCU can execute, given a fixed energy budget. Although the $16MHz$ setting offers faster execution and superior energy efficiency per clock cycle, its narrowed operating voltage range results in $3.75x$ fewer clock cycles compared to the slower, yet less energy-efficient $1MHz$ setting. This latter configuration enables the MCU to compute for an extended duration, specifically as long as the supply voltage remains above $1.8V$. Fundamentally, the $1MHz$ setting allows the system to harness more energy—and consequently more useful work—from an identical initial capacitor charge.

Challenge. Similar trade-offs are seen also in regular processors and routinely exploited to improve execution speed and/or energy consumption [81]. In mobile platforms, for instance, the dynamic adjustment of operating frequencies and supply voltage enables systems to respond to sudden surges in system load, while conserving energy during periods of lighter loads [51]. To achieve this, dedicated hardware and software components are employed, collectively referred to as Dynamic Voltage and Frequency Scaling (DVFS) [25].

DVFS techniques used in mainstream platforms are not applicable to battery-less devices. Resource constraints and different performance metrics demand a different design rationale. As an example, employing hardware support for DVFS from mainstream platforms in battery-less devices would be impractical due to the excessive energy consumption [25]. Conversely, the lack of a proper operating system renders existing software drivers outright unusable.

Crucially, the application and system requirements of battery-less embedded computing diverge significantly from those in mainstream computing. Energy consumption is the primary, and often *only* metric of interest. To conserve energy [13], application developers often prioritize energy savings over other metrics of interest, such as execution speed or data processing accuracy. Conserving energy extends the duration of energy cycles, consequently reducing the overhead associated with NVM operations.

Further, charge-discharge cycles are frequent in battery-less devices, as the push for miniaturization prompts energy storage facilities to be minimized as well. For example, harvesting energy from RF transmissions to compute a simple CRC may lead to 16 energy failures over a 6 seconds period [16]. The improvements in energy consumption, leading to prolonged energy cycles and lower overhead, are going to have a magnifying effect on other metrics of interest, including data throughput.

Contribution and road-map. As we discuss Sec. 2, only a few efforts exist to apply DVFS to battery-less devices [10, 27]. Research most similar to ours primarily targets multi-core processors equipped with DVFS hardware support, which are distinctly different from MSP430-class microcontrollers. While their focus is on achieving power neutrality by adjusting power consumption to match harvested energy, they do not account for the implications of NVM operations.

We demonstrate that it is possible to achieve DVFS functionality in a much more limited energy envelope, throughout intermittent operations, and consequently unlock significant performance gains. Sec. 3 illustrates the design rationale, whereas Sec. 4 provides concrete evidence based on two hardware/software co-designs that expose different trade-offs and functionality. The two distinct implementations, D^{2}VFS and FBTC, were developed to balance simplicity, efficiency, and configurability in achieving DVFS in battery-less embedded devices. D^{2}VFS serves as a reference design, straightforward but occasionally less efficient, emphasizing the gains in performance even with the energy costs of its DVFS circuitry. On the other hand, FBTC improves upon D^{2}VFS by reducing energy overhead and introducing a configurable startup voltage threshold, offering developers a means to tailor energy dynamics to specific deployment scenarios. This design choice underscores a pragmatic approach: providing a baseline system that demonstrates the benefits of DVFS while also offering a more advanced alternative that optimizes for energy efficiency and provides greater flexibility for real-world applications. Both implementations use the same MCU and voltage regulator, but their different architectures highlight the balance between energy efficiency, system responsiveness, and hardware complexity, addressing distinct use cases and optimization priorities in the domain of energy-harvesting systems.

Sec. 5 presents an extensive evaluation of both designs. We compare their performance against a stock MSP430 microcontroller that is statically set to one of the four factory-calibrated frequencies. This configuration fails to capture the trade-offs illustrated in Fig. 3. Our results demonstrate that both D^{2}VFS and FBTC can achieve up to 3.75x lower energy consumption and 12x faster execution time than the considered baselines, while requiring a smaller energy buffer and thus reducing recharging times and mitigated energy waste due to leakage.

2 BACKGROUND AND RELATED WORK

Embedded sensing devices form the backbone of the Internet of Things (IoT) [31]. Most IoT devices are battery-powered. Batteries, even rechargeables, must be periodically recharged, replaced, and eventually disposed, polluting

the environment [19, 87]. The battery-less IoT [7] liberates IoT devices from batteries by enabling them to harvest energy from the environment. This design leads to a broad range of applications, including space applications [24], smart buildings [28, 78], precision agricolture [40], and supervision of archaeological sites [2]. Deployments in these application scenarios can potentially work for years without requiring any maintenance [2].

System support plays a key role in enabling such applications as it helps maintain forward progress despite frequent power failures [5, 6, 22, 63]. We offer next a primer on intermittent computing and delve into the challenges and prevailing solutions for DVFS, in both mainstream computing platforms and battery-less devices.

2.1 Intermittent Computing

The pattern of intermittent computing necessitates specialized system support to bridge periods of energy scarcity. Numerous techniques have been developed to ensure forward progress in battery-less devices despite energy disruptions. Some strategies implement checkpoints at compile-time based on execution patterns [63, 75] or program structures [5, 17, 75], while others utilize supplementary hardware to initiate proactive checkpointing [11, 12, 50]. There are also approaches that offer developers task-based programming abstractions with transactional semantics [21, 62, 68]. Specialized solutions have been designed to preserve peripheral states through energy disruptions [9, 14, 18].

However, the majority of techniques in intermittent computing primarily aim to minimize the energy overhead associated with maintaining application progress. They often overlook the dynamics of supply voltages and MCU frequency adjustments. Thus, the application of DVFS presents a distinct challenge, influencing system performance *within* an energy cycle—by enhancing energy efficiency, for instance—rather than spanning multiple energy cycles.

2.2 DVFS

DVFS includes two key mechanisms: voltage and frequency scaling. Each processor possesses distinct operational ranges, with each range characterized by a frequency and voltage tuple (f, V). Mainstream computing platforms utilize advanced software and hardware mechanisms that allow for precise control over voltage and frequency configurations [23, 33].

In the following, we will focus our discussion on related works pertaining to embedded systems, as they closely align with battery-less devices.

Real-time embedded systems. Salehi et al. [76] present an adaptive voltage and frequency scaling technique that rapidly tracks the workload changes to meet soft real-time deadlines. Their work demonstrates considerable energy savings and fewer frequency updates compared to DVFS systems based on fixed update intervals. HyPowMan [15] considers the problem of minimizing energy consumption for periodic real-time tasks scheduled over multiprocessor platforms. The technique takes a set of well-known existing DVFS policies, each performing well for given conditions, and adapts at runtime to the best-performing policy for a given workload.

Huang et al. [39] apply DVFS to mixed-criticality systems and show that DVFS helps critical tasks meet deadlines by speeding up the processor when it is bound to miss a deadline. Liu et al. [59] employ DVFS to optimize system thermal profiles to prevent run-time thermal emergencies and to minimize cooling costs. RT-DVFS [73] modifies the OS's real-time scheduler and task management service to provide energy savings while maintaining real-time deadline guarantees. Generalized Shared Recovery (GSHR) [88] efficiently uses DVFS techniques to achieve a given reliability goal for real-time embedded applications.

While these works offer essential insights into the application of DVFS in embedded systems, their design objectives diverge significantly, rendering their techniques less suited for direct application to battery-less devices. The latter rarely deal with real-time deadlines, whereas reducing energy consumption for a fixed workload is key.

Wireless sensor networks. Kulau et al. [52–54] analyze the effects of undervolting a wireless sensor node. They show that such a device can still work reliably, even if the voltage recommendations are violated, because a correlation exists between temperature and probability of error at a given voltage level. Powell et al. [74] design DVFS hardware to meet battery life and form factor expectations of body area sensor networks. Similar to these works are also the efforts on developing DVFS techniques in distributed microsensor networks [70] and in sensor networks with deadlines [8].

As most of these works aim to conserve energy, many of them are similar to ours in spirit, yet the authors consider battery-powered devices with *finite* energy supplies and tend to accept performance penalties to increase lifetime. On the contrary, we deal with intermittent but unbounded energy supplies, with the goal of increasing the amount of work achieved in an energy cycle.

Battery-less devices. EA-DVFS [58] presents a high-level simulation study on the advantages of DVFS for real-time operation in battery-less devices. Due to the lack of a corresponding hardware implementation, it does not serve as a suitable baseline for our investigation. Lin et al. [57] model a framework for concurrent task scheduling and dynamic voltage and frequency scaling in real-time embedded systems with energy harvesting. Li et al. [55] also provide early insights into jointly scaling workload, voltage, and frequency in multi-core sensor networks using energy harvesting.

These studies offer valuable preliminary perspectives on the application of DVFS in energy harvesting devices. However, our work is the first concrete implementation of any such technique, complemented by a comprehensive evaluation that distinctly underscores the advantages of applying DVFS in battery-less environments.

Summary. Numerous efforts exists to enhance energy efficiency, particularly in environments with stringent energy constraints. The primary focus of these works is on devices with *finite* energy sources. These works, although foundational, often diverge in design goals and cannot be applied "as-is" to battery-less devices.

Our research pivots from these traditional paradigms. Instead of finite energy reserves, we consider intermittent, yet potentially perpetual energy supplies. Our primary objective is not merely to conserve energy but to maximize the amount of useful work accomplished within each active cycle.

3 DESIGN RATIONALE

The fundamental element enabling DVFS for a target MCU is the identification of the available *performance windows*, which consist in a platform-specific combination of voltage and frequency settings.

Indeed, most low-power MCUs feature dozens of possible frequency settings. We concentrate on a subset of them, usually the factory-calibrated ones, where the datasheet also explicitly reports the corresponding minimum supply voltage. At a given frequency setting, the minimum supply voltage yields the lowest energy consumption [4]. For instance, with the MSP430-G2553 [43] MCU, we examine the four factory-calibrated frequency settings with the corresponding minimum supply voltages, thereby determining *four (ordered) performance windows*: (i) $16MHz$ at $3.3V$, (ii) $12MHz$ at $2.8V$, (iii) $8MHz$ at $2.2V$, and (iv) $1MHz$ at $1.8V$.

Scaling down. The blue and orange curves depicted in Fig. 4 illustrate the expected performance across the four performance windows of the MSP430-G2553 during capacitor discharge.

As long as the capacitor voltage is above the minimum supply voltage of a certain performance window, the supply voltage is regulated to *exactly* this minimum, which provides the best energy efficiency at the corresponding frequency.

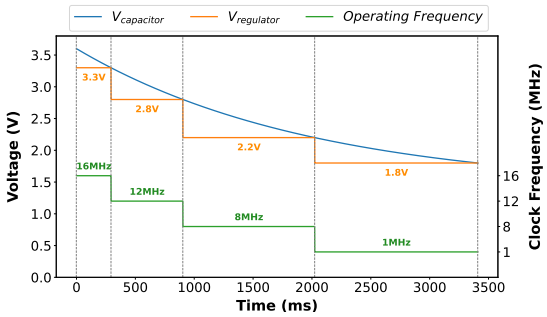


Fig. 4. System behavior when capacitor discharges.

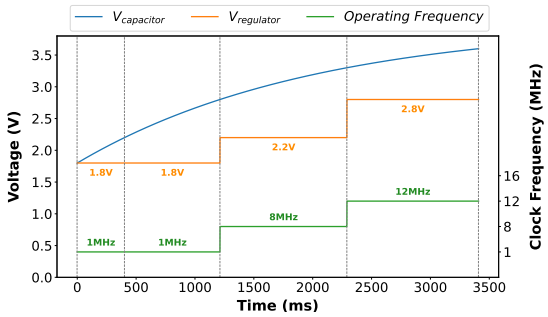


Fig. 5. System behavior when capacitor charges.

As soon as the capacitor voltage crosses the lower bound of the *current* performance window, frequency and voltage settings are scaled to enter the *lower* performance window. For example, when the capacitor discharges from $3.6V$ to $3.3V$, frequency changes to $12MHz$ and supply voltage is scaled to $2.8V$, thus moving from window (i) to window (ii). Transitioning to a lower performance window necessitates altering the frequency settings prior to adjusting the supply voltage; reversing this sequence would result in device shutdown due to the supply voltage dipping below the minimum threshold for the given frequency setting.

Scaling up. This rationale is also applicable when the capacitor voltage rises, albeit with a nuance as depicted in Fig. 5.

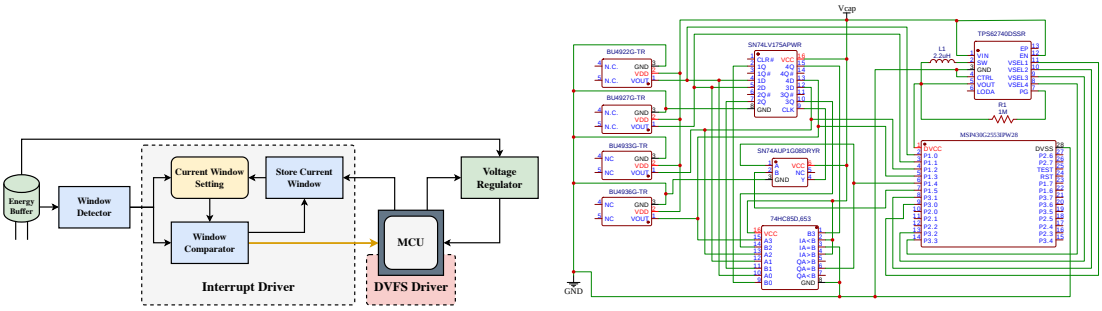
Energy consumption per clock cycle increases when moving from a lower to a higher frequency setting. Should the device fail to harvest sufficient energy, the heightened energy consumption per clock cycle could precipitate an immediate reduction in capacitor voltage, thereby compelling the system to revert promptly to a lower performance window. Following the adjustment, as the energy consumption per clock cycle decreases, the net energy balance may shift to positive, leading to a subsequent rise in capacitor voltage. This increase can trigger a transition back to the higher performance window. This behavior may repeat indefinitely, entering a sort of livelock. To avoid this, we cautiously wait until the capacitor voltage reaches the *upper bound* of the upper performance window before changing frequency and voltage settings accordingly. Symmetrically, to avoid shutting down the system when transitioning to the upper performance window, we change supply voltage first, then frequency.

Towards implementation. Realizing this behavior concretely hinges on a careful consideration of trade-offs between the energy overhead attributed to supplementary hardware components and the resulting gain in flexibility.

For example, to change supply voltage, an external voltage regulator may be required, as regular low-power MCUs are usually not equipped with it. Detection of the capacitor voltage reaching a threshold that necessitates a change in performance window can be accomplished either by periodic polling or by employing specialized circuitry that asynchronously alerts the MCU of particular conditions occurring at the capacitor. Conversely, existing low-power MCUs are capable of altering frequency settings via software: using MSP430-class MCUs [43], frequency settings are programmatically set by changing the values of specific registers.

4 IMPLEMENTATION

The design rationale of Sec. 3 is materialized in two distinct implementations, each elucidating different trade-offs and functionalities. The first implementation we present is called D^2VFS (Discrete Dynamic Voltage and Frequency Scaling) and may be regarded as a reference implementation of sorts. It achieves DVFS functionality in the simplest, but not



(a) D^{2}VFS logic. (b) D^{2}VFS schematics.

Fig. 6. D^{2}VFS design.

necessarily the most efficient or flexible way. As illustrated in Sec. 5, despite the energy overhead due to the circuitry realizing DVFS functionality, D^{2}VFS already provides great performance advantages compared to a static setting.

The second implementation is called FBTC (Fixed Boot Threshold Controller) and improves over D^{2}VFS *in three ways*. The circuitry realizing DVFS functionality imposes a much lower energy overhead compared to D^{2}VFS. Further, FBTC avoids the fluctuation problem mentioned in Sec. 4 by design, without requiring a delay in the changes to upper performance windows during the capacitor charge. This results in a faster and more efficient change of operating setting compared to D^{2}VFS. The corresponding energy savings are spent in useful application processing, boosting the overall energy efficiency. Finally, FBTC allows developers to configure the voltage threshold to boot the system, providing a knob that may be useful to capture deployment-specific energy dynamics [2].

Both implementations are centered around the MSP430-G2553 [43] MCU and use the TPS62740 [45] voltage regulator. The performance windows are those in Sec. 3.

4.1 D^{2}VFS

Fig. 6 illustrates the design of D^{2}VFS; Fig. 6(a) describes the logical components and Fig. 6(b) shows the schematics.

Logical components. The *Window Detector* in Fig. 6(a) determines the valid performance window based on capacitor voltage. To circumvent the energy-intensive process of periodic polling by the MCU's ADC, we employ four TI BU49XXG [80] voltage detectors, as illustrated in Fig. 6(b); one for each performance window. Each detector takes as input the capacitor voltage V_{cap} and outputs a signal that indicates if V_{cap} is higher than the threshold.

The MCU is required to ascertain shifts in the current performance window to adjust its operating frequency and supply voltage appropriately. One approach could involve periodic software polling of the *Window Detector*'s output. However, this method is fraught with several drawbacks: it imposes extra latency dependent on the polling interval, risks interrupting the flow of application processing, and leads to superfluous energy expenditure, as each non-revealing check essentially constitutes wasted effort. We anticipate that such unproductive checks would predominate.

We opt for a design that employs a hardware interrupt mechanism to notify the MCU of a change in the performance window. This functionality is shown as *Interrupt Driver* in Fig. 6(a). The key is to maintain a small dedicated memory that reflects the active performance window—specifically, the current configuration of the MCU's frequency and supply voltage—as depicted in *Current Window Setting* in Fig. 6(a). A dedicated *Window Comparator* monitors both the output of the *Window Detector* and the *Current Window Setting*; whenever the two differ, it signals an interrupt to the MCU.

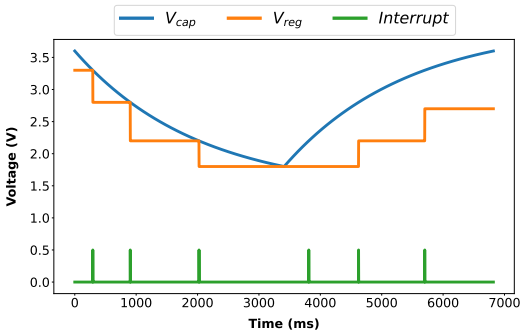


Fig. 7. Example of D^{2}VFS behavior.

This informs the MCU that the capacitor voltage entered a new performance window. As a result, the *Store Current Window* function updates the *Current Window Setting* to reflect the new information accurately.

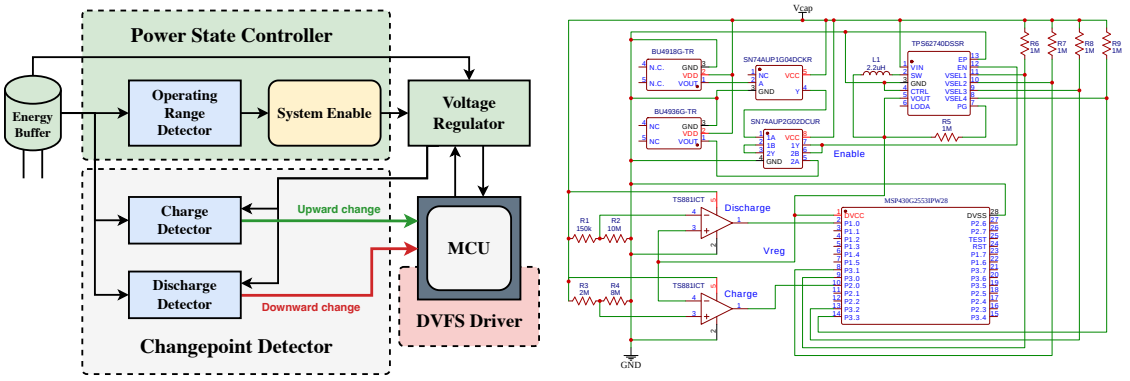
The *Interrupt Driver* is implemented using three hardware components, each chosen mainly because of energy efficiency, as depicted in Fig. 6(b). The SN74LV175A [41] D-type flip-flop stores the *Current Window Setting* as a sequence of bits, where the $i-th$ bit represents the output of the $i-th$ voltage detector. The 74HC85 [71] 4-bit comparator works as the *Window Comparator*, which compares the output of voltage detectors against the state saved in the D-type flip-flop and outputs a changed signal when they differ. The SN74AUP1G08 [46] `AND` gate operates as the *Store Current Window* block, which allows the MCU to set the new state of the D-type flip flop after the performance window changes.

Run-time behavior. Fig. 7 shows an example execution. The capacitor voltage V_{cap} starts at $3.6V$ and the DVFS driver sets the voltage regulator to $3.3V$ with the MCU operating at $16MHz$. As soon as V_{cap} reaches $3.3V$, the *Interrupt Driver* fires an interrupt, shown in green in Fig. 7. The D^{2}VFS driver identifies the new performance window by checking the outputs of the voltage detectors and regulates supply voltage to $2.8V$ first, then sets the operating frequency to $12MHz$. The same behavior repeats when V_{cap} reaches $2.8V$ and $2.2V$, corresponding to two more interrupts.

To avoid the fluctuations mentioned in Sec. 3, the D^{2}VFS driver delays the change to the upper performance window when V_{cap} increases. Let us focus on Fig. 7 when V_{cap} is at $1.8V$ and rising. The MCU is running at $1MHz$ and supply voltage is regulated at $1.8V$. Whenever V_{cap} reaches $2.2V$, the *Interrupt Driver* fires an interrupt. The D^{2}VFS driver discerns the appropriate new performance window by monitoring the outputs from the voltage detectors. To avoid the risk of fluctuations, an immediate transition to a higher performance window is deferred. The driver awaits a subsequent interrupt to initiate this change. Thus, when V_{cap} rises to $2.8V$, the *Interrupt Driver* issues a new interrupt, prompting the D^{2}VFS driver to adjust the supply voltage to $2.2V$ and the MCU frequency to $8MHz$.

4.2 FBTC

Fig. 8 shows the design of FBTC. Fig. 8(a) illustrates the logic and Fig. 8(b) shows the corresponding schematics. Two macro components drive the functioning of FBTC. The *Power State Controller* of Fig. 8(a) turns the system on whenever the capacitor voltage rises above a given boot threshold. Unlike D^{2}VFS, this threshold is hardware-configurable in FBTC. The *Changepoint Detector*, instead, manages the detection of changes in the performance window. We consider the same performance windows of D^{2}VFS.



(a) FBTC logic. (b) FBTC schematics.

Fig. 8. FBTC design.

Power state controller. The *Operating Range Detector* in Fig. 8(a) identifies if V_{cap} is within the considered operating range. It does so by relying on two BU49XXG [80] voltage detectors, as shown in Fig. 8(b). The first detector triggers when V_{cap} reaches the MCU minimum operating voltage $V_{min} = 1.8V$, whereas the second detector triggers when V_{cap} reaches the hardware-configurable boot threshold V_{on}. Although Fig. 8(b) shows a $3.6V$ setting for the second voltage detector, when fabricated, FBTC allows users to select among four different voltage detectors to configure V_{on}, as indicated by the *PVComp* and *PVT* ports of Fig. 9.

The *System Enable* function, as illustrated in Fig. 8(a), determines the conditions to activate the system. This operation utilizes a SN74AUP1G04 [44] NOT gate in conjunction with a SN74AUP2G02 [42] 2-input NOR gate, configured as a set-reset flip-flop, which is detailed in Fig. 8(b). The NOT gate takes as input the signal of the first voltage detector, that is, the one identifying if V_{cap} exceeds V_{min}. The NOT gate thus verifies if V_{cap} falls below V_{min}, resetting the flip-flop output. Instead, the signal of the second voltage detector sets the flip-flop output. When V_{cap} exceeds the configured V_{on}, the flip-flop output is set to a logical high and the voltage regulator is powered on. When V_{cap} goes below V_{min}, the flip-flop output is reset to a logical low and the voltage regulator is powered off.

To initialize the output voltage of the voltage regulator at startup, we employ four pull-up resistors, designated as $R6 - R9$ in the schematic depicted in Fig. 8(b) and as $R1 - R4$ in the actual prototype shown in Fig. 9. This step is necessary because the voltage regulator's output is governed by the MCU, which is incapable of setting the output voltage until it has completed its startup sequence.

Changepoint detector. Unlike D^{2}VFS, FBTC does not keep track of the current performance window in hardware; instead, it merely detects the conditions that trigger any change in the current performance window and whether this change is towards an upper or lower window. This indication reaches the MCU through a hardware interrupt: by keeping track of the current performance window and by learning whether the change being detected is upwards or downwards, the MCU changes voltage and frequency settings.

The *Interrupt Driver* of Fig. 8(a) provides this functionality through a *Charge (Discharge) Detector* detecting upward (downward) changes in the performance window. The two detectors are based on the same logic, which we accomplish with two components: (i) the $R3 - R4$ ($R1 - R2$) resistors of Fig. 8(b), which act as a voltage divider that reduces the

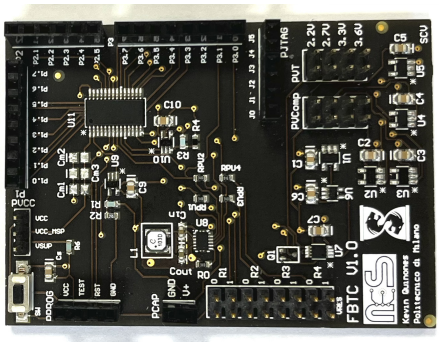


Fig. 9. FBTC prototype.

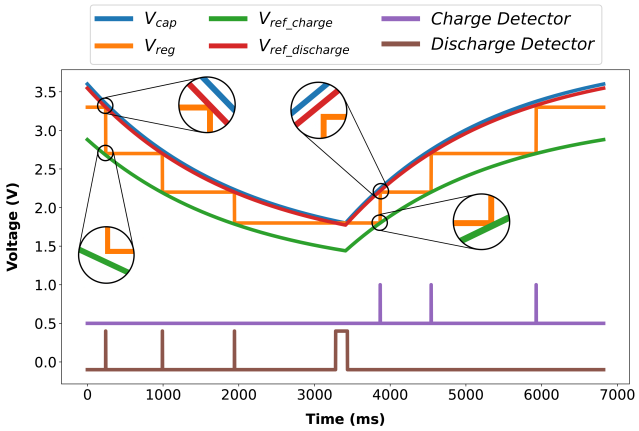


Fig. 10. Example of FBTC behavior.

V_{cap} signal to V_{ref_charge} ($V_{ref\_discharge}$), that is, the maximum (minimum) voltage level that triggers a change in the performance window, and (ii) a TS881 [82] operational amplifier that compares V_{cap} against V_{ref_charge} ($V_{ref\_discharge}$) and outputs the signal indicating to step up (down) the performance window.

To detect a discharge, the output of the voltage regulator V_{reg} is connected to the non-inverting input of the operational amplifier and the reduced V_{cap} signal is connected to the inverting input, as shown near the *discharge* label of Fig. 8(b). To detect the energy buffer charge, the connections to the operational amplifier are inverted. We discuss later how to dimension $R1 - R2$ and $R3 - R4$, as well as the need for *both* reference signals for discharging and charging.

Fig. 10 shows an example execution. The blue curve represents the original V_{cap}, whereas the orange one represents V_{reg}. The signals representing the reference voltage for charging or discharging are V_{ref_charge} and $V_{ref\_discharge}$, shown in green and red, respectively. Initially, the frequency is set to $16MHz$, V_{reg} is set to $3.3V$, V_{cap} is $3.6V$, and the capacitor is discharging. When V_{cap} reaches $3.3V$, the V_{reg} signal, corresponding to the orange curve, exceeds the $V_{ref\_discharge}$ signal, corresponding to the red curve, as shown in Fig. 10. The *Discharge Detector* outputs a logical high, indicated with the brown line in Fig. 10, triggering an interrupt. Knowing the current performance window and learning that a downward change is detected, the MCU switches to a configuration running at $12MHz$ with V_{reg} set to $2.8V$.

The same operations repeat throughout the discharge phase until the MCU switches to a configuration running at $1MHz$ with V_{reg} set to $1.8V$. When V_{cap} approaches $1.8V$, V_{reg} constantly exceeds $V_{ref\_discharge}$. This time there is no lower performance window to change to, as the MCU is already at $1MHz$ and V_{reg} at $1.8V$. To avoid unexpected behaviors, the software driver disables the interrupts from the *Discharge Detector* when it sets the lowest possible performance window and enables them back whenever scaling upwards again.

The behavior when charging is dual: the *Charge Detector* triggers an interrupt when V_{reg} intersects V_{ref_charge}. Different than D^{2}VFS, FBTC need not to delay changes to the upper performance window when V_{cap} increases, as the charge detector avoids bouncing between two adjacent performance windows by design, as detailed next.

Voltage divider configuration. The efficient operation of FBTC rests on one key aspect: the dimensioning of $R1 - R2$ and $R3 - R4$. Multiple reasons concur to this:

(1) Properly setting the values of $R1 - R2$ ensures that V_{cap} never comes too close to V_{reg}, giving the MCU enough margin to trigger a switch to a lower performance window before $V_{cap} < V_{ref}$ for the current performance

window. The $V_{ref\_discharge}$ signal exists precisely for this: if we were to compare directly V_{cap} with V_{reg}, the time taken by the MCU to switch towards a lower performance widow would become (too) critical.

(2) The reciprocal setting of $R1-R2$ and $R3-R4$ allows the system to avoid fluctuations between adjacent performance windows. For example, when switching to a lower performance window, V_{reg} must not intersect V_{ref_charge}, or the MCU would trigger an immediate switch back to the upper performance window. Otherwise, FBTC may end up in a sort of livelock bouncing back and forth between adjacent performance windows.

(3) By accurately tuning the V_{ref_charge} signal, that is, the values of $R3-R4$, we may ensure sufficient energy margin in the upper performance window to prevent an immediate downward transition. This addresses the problem we discuss previously with D^{2}VFS possibly bouncing between two adjacent performance windows when switching from a lower to an upper window.

For a clearer illustration, we now describe the method for quantitatively determining the values for $R1-R2$, taking into account the considerations mentioned above. The reasoning to ascertain the values for $R3-R4$ is entirely dual. Based on the schematics of Fig. 8(b), the operational amplifiers inputs are:

$$V_{ref\_discharge} = \frac{R2}{R1+R2} \cdot V_{cap} = \delta_d \cdot V_{cap}, \tag{1}$$

$$V_{ref\_charge} = \frac{R4}{R3+R4} \cdot V_{cap} = \delta_c \cdot V_{cap}, \tag{2}$$

where δ_c (δ_d) indicates the charge (discharge) voltage divider ratio.

Let the performance windows be ordered by ascending operating voltage and let $V_{reg}[i]$ be the voltage regulator output of the i-th performance window. The interrupt signaling a change from the i-th performance window to the $i+1$-th performance window is triggered whenever $V_{ref\_charge} > V_{reg}[i]$. FBTC may, however, immediately bounce back to the i-th performance window if $V_{ref\_discharge} < V_{reg}[i+1]$. In summary, we must avoid

$$\text{when } \; V_{ref\_charge} > V_{reg}[i] \rightarrow V_{ref\_discharge} < V_{reg}[i+1] \tag{3}$$

that we can rewrite, based on Eq. (1) and Eq. (2), as

$$\text{when } \; \delta_c \cdot V_{cap} > V_{reg}[i] \rightarrow \delta_d \cdot V_{cap} < V_{reg}[i+1] \tag{4}$$

To avoid undesired bouncing behaviors, for any performance window i, Eq. (3) must never hold. This means

$$\text{when } \; \delta_c \cdot V_{cap} > V_{reg}[i] \rightarrow \delta_d \cdot V_{cap} \geq V_{reg}[i+1] \tag{5}$$

Say the operating range of the i-th performance window is $(V_{max}[i], V_{min}[i])$. To satisfy Eq. (5) for any performance window i, we introduce a margin ϵ_c that represents the minimum voltage sensitivity we wish to obtain for the charge detector. This means that, for a given performance window i, we substitute $V_{cap} = V_{cap} = V_{min}[i] + \epsilon_c$ as long as there exists a performance window $i-1$.

To reason quantitatively, consider the four performance windows of the MSP430-G2553 [43] introduced earlier:

1) $1MHz$ with $V_{reg} = 1.8V$ and V_{cap} in $(2.2V, 1.8V)$
2) $8MHz$ with $V_{reg} = 2.2V$ and V_{cap} in $(2.8V, 2.2V)$
3) $12MHz$ with $V_{reg} = 2.8V$ and V_{cap} in $(3.3V, 2.8V)$
4) $16MHz$ with $V_{reg} = 3.3V$ and V_{cap} in $(3.6V, 3.3V)$

and assume $\epsilon_c = 50mV$. We return soon to how to determine ϵ_c.

Consider now performance windows with $i = 1, 2, 3$ and Eq. (5), obtaining the following constraints on δ_d:

- $V_{cap} = 2.20V + 50mV = 2.25V$, $V_{reg}[1] = 1.8V$, $V_{reg}[2] = 2.2V \rightarrow \delta_d \geq \frac{2.2V}{2.25V}$
- $V_{cap} = 2.80V + 50mV = 2.85V$, $V_{reg}[2] = 2.2V$, $V_{reg}[3] = 2.8V \rightarrow \delta_d \geq \frac{2.8V}{2.85V}$
- $V_{cap} = 3.30V + 50mV = 3.35V$, $V_{reg}[3] = 2.8V$, $V_{reg}[4] = 3.3V \rightarrow \delta_d \geq \frac{3.3V}{3.35V}$

These constraints collectively determine a lower bound for δ_d. To ensure all constraints are satisfied, we pick the highest value for δ_d, that is, $\delta_d \geq \frac{3.3V}{3.35V} = 0.9851$. because $\delta_d = \frac{r2}{r1+r2}$, a possible selection is $R1 = 150k\omega$ and $R2 = 10M\omega$.

Determining the values for $R3 - R4$ requires dual reasoning, where the resulting constraints identify an upper bound for δ_c. Therefore, we pick the lowest value for δ_c, that is, $\delta_c \geq \frac{1.8V}{2.25V} = 0.8$. Similarly to the previous case, we consider a margin $\epsilon_d = 50mV$ that represents the minimum voltage sensitivity we wish to obtain for the discharge detector. Because $\delta_c = \frac{R4}{R3+R4}$, a possible selection is $R3 = 2M\Omega$ and $R4 = 8M\Omega$.

Selecting ϵ_c. To prevent an immediate transition back to a lower performance window, we must ensure that the capacitor stores sufficient energy to sustain the computation in the upper performance window for a reasonable amount of time. An extra voltage of ϵ_c in a capacitor corresponds to $\frac{1}{2}C{\epsilon_c}^2$ energy. Say the maximum energy consumption per clock cycle is e_{cc}, the number of extra clock cycles n_{clock_cycles} that an extra voltage ϵ_c allows the MCU to execute is

$$n_{clock\_cycles} = \frac{\frac{1}{2}C{\epsilon_c}^2}{e_{cc}} \tag{6}$$

The software driver of FBTC requires 18 machine-code instructions to change the performance window, that is, 18 clock cycles. To justify switching to an upper performance window, we must satisfy

$$n_{clock\_cycles} * p_{lower} \geq 18 + n_{clock\_cycles} \tag{7}$$

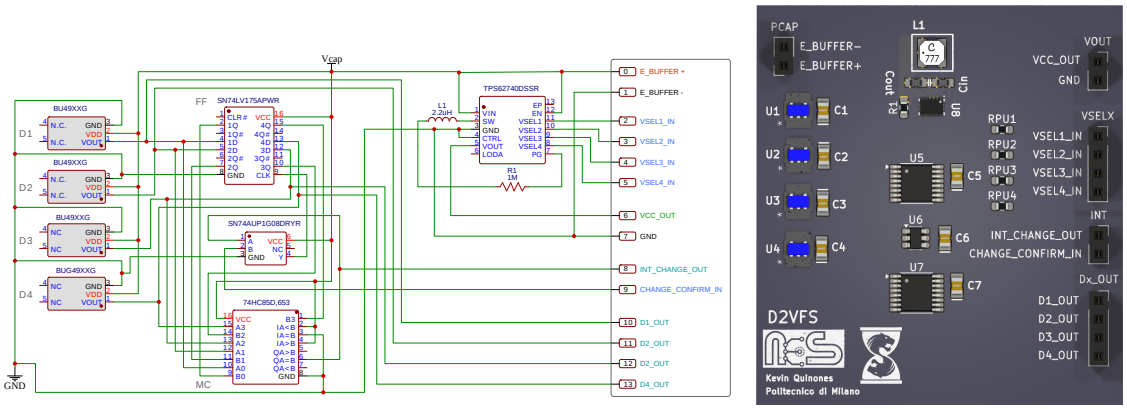
where p_{lower} represents the energy consumption increase at a lower operating frequency compared to the higher one, sustained at the same voltage level. For the MSP430-G2553 [43], the average p_{lower} between the three switching points, that is, $1MHz - 8MHz$, $8MHz - 12MHz$, and $12MHz - 16MHz$ is 1.17. This means that switching to a higher frequency provides, on average, a 17% better energy efficiency; hence $n_{instr} \geq 106$ clock cycles.

FBTC sets the MCU to operate at the minimum possible voltage for each performance window. To identify the highest energy consumption per clock cycle of the MCU, we consider the operating frequency with the highest energy consumption at the corresponding minimum operating voltage, that is, $16MHz$ with a $3.3V$ voltage supply, corresponding to $0.85nJ$ energy consumption per clock cycle, as shown in Fig. 2. By substituting these values in Eq. (6) and by considering a target capacitor of $100\mu F$, ϵ_c must be at least $0.042V$.

4.3 Base Design as Expansion Board

The definition of performance windows is platform-dependent, as they consist of pairs of operating frequencies and minimum operating voltages specific to the hardware features. Although the specific designs of D^{2}VFS and FBTC we present are specific to the MSP430-G2553 [43], their underlying logic is platform-independent. Only two elements of D^{2}VFS and FBTC are specific to a platform: the voltage and frequency pairs in the DVFS driver and the hardware components that identify the voltage range associated with performance windows. Developers can set the former in software. The latter requires circuit designers to carefully dimension a subset of D^{2}VFS and FBTC hardware components.

To facilitate this process and allow developers and circuit designers to use D^{2}VFS and FBTC with their platform of choice, we devise and implement a base design of two expansion boards that capture the core logic of D^{2}VFS and FBTC, depicted in Fig. 11 and Fig. 12, respectively. These expansion boards can be attached to a arbitrary evaluation board as peripheral devices using dedicated PIN headers, and their circuit schematics isolate platform-specific components.



(a) D^{2}VFS expansion board schematics.

(b) D^{2}VFS expansion board implementation for the MSP430-G2553.

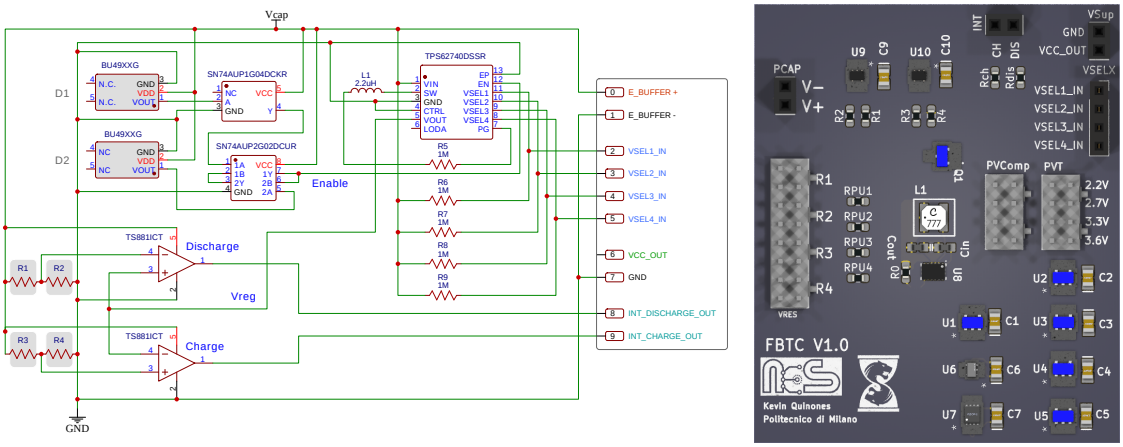
Fig. 11. D^{2}VFS expansion board design.

D^{2}VFS expansion board. Fig. 11 shows an overview of the D^{2}VFS expansion board, where Fig. 11(a) provides the corresponding circuit schematic and Fig. 11(b) depicts its implementation for the MSP430-G2553 [43]. The circuit schematic of Fig. 11(a) captures the core design of D^{2}VFS, where the grey elements represent platform-specific components, consisting of the four BU49XXG [80] voltage detectors ($D1 - D4$) that dictate performance window changes.

Circuit designers must dimension the voltage detector as follows. $D4$ must detect the power-on voltage, whereas the voltage detectors $D1 - D3$ must detect the operating voltages of the first three most-efficient performance windows, $D1$ targets the operating voltage associated with the performance window with the second lowest operating frequency, whereas $D3$ targets the operating voltage associated with the performance window having the highest operating frequency. For the MSP430-G2553 [43], we consider $3.6V$ as power-on voltage and the performance windows set to $16MHz$ at $3.3V$, $12MHz$ at $2.7V$, $8MHz$ at $2.2V$, and $1MHz$ at $1.8V$. Therefore, $D1$ detects $2.2V$, $D2$ detects $2.7V$, $D3$ detects $3.3V$, and $D4$ detects $3.6V$.

The D^{2}VFS expansion board may be connected to the target device using the dedicated pins: VCC_OUT and GRN provide regulated voltage and must be used for supplying power to the MCU, whereas $E_{BUFFER}+$ and $E_{BUFFER}-$ must be connected to the corresponding ends of the energy buffer. The other pins allow the D^{2}VFS driver to interact with the D^{2}VFS expansion board: the input pins $VSEL1\_IN - VSEL4\_IN$ control the voltage regulator output, the output pin INT_CHANGE_OUT fires an interrupt that signals the D^{2}VFS driver to change the performance window, the input pin $CHANGE\_CONFIRM\_IN$ signals to the expansion board the change to the performance window, and the output pins $D1\_OUT - D4\_OUT$ signal the current performance window information. Fig. 11(b) shows the D^{2}VFS expansion board implemented for the MSP430-G2553 [43].

The expansion board of Fig. 11 demonstrates a design for platforms with *four* performance windows. This design can be adapted to support a different number of performance windows by changing the number of voltage detectors and their corresponding output pins. To support more than *four* performance windows, circuit designers must swap the FF flip-flop holding the current performance window and the MC magnitude comparator detecting performance window changes with corresponding components that support the increased number of information bits. For example, with *six* performance windows, FF and MC must support 6 bits.



(a) FBTC expansion board schematics.

(b) FBTC expansion board implementation for the MSP430-G2553.

Fig. 12. FBTC expansion board design.

FBTC expansion board. Fig. 12 shows the design of the FBTC expansion board, with Fig. 12(a) illustrating the corresponding circuit schematic and Fig. 12(b) depicting its implementation for the MSP430-G2553 [43] including the configurable power-on voltage. The circuit schematic of Fig. 12(a) captures the core design of FBTC, where the grey elements represent platform-specific components, consisting of the two BU49XXG [80] voltage detectors ($D1 - D2$) defining the operating voltage range and the four resistors ($R1 - R4$) dimensioning the reference voltages for the discharge ($R1 - R2$) and charge ($R3 - R4$) detectors.

Circuit designers must dimension the voltage detectors as follows: $D1$ must detect the power-off voltage, whereas $D2$ must detect the power-on voltage. We recall that for the MSP430-G2553 [43] we consider an operating range of $3.6V - 1.8V$. Therefore, $D1$ detects $1.8V$ and $D2$ detects $3.6V$. Instead, the four resistors $R1 - R4$ must comply with the constraints derived from Eq. (5), using the process described in Sec. 4.2. For the MSP430-G2553 [43] we set $R1 = 150k\Omega$, $R2 = 10M\Omega$, $R3 = 2M\Omega$, and $R4 = 8M\Omega$.

The FBTC expansion board is connected to the target device using the corresponding pins. The pins VCC_OUT, GRN, $E_{BUFFER}+$, $E_{BUFFER}-$, $VSEL1\_IN - VSEL4\_IN$ are set with the same logic as the D^{2}VFS expansion board. Instead, the output pin INT_CHARGE_OUT ($INT\_DISCHARGE\_OUT$) fires an interrupt that signals the FBTC driver to step up (down) the performance window.

5 EVALUATION

We evaluate the performance of D^{2}VFS and FBTC under different system settings and energy harvesting scenarios. We describe next the experiments and system setup, the considered energy scenarios, and the results of the experiments.

Our setup is designed to investigate a broad spectrum of energy conditions, ranging from energy-rich sources that prevent energy failures to energy-poor sources that result in frequent energy failures, with various intermediate scenarios in between. Benchmarks comprise a diverse array of embedded programs, each exposing a variety of programming

structures and workloads. Our evaluation includes more than $500k$ data points. Despite the extreme diversity of the setup and the quantity of experimental data at hand, the results allow us to conclude that:

(1) FBTC and D^{2}VFS significantly surpass all static configurations *at both extremes*—with energy-rich or energy-poor sources—as their capacity to maximize the number of instructions executed per active cycle results in substantially reduced energy consumption and completion times;
(2) with setups lying between the two extremes, the performance of FBTC and D^{2}VFS is on par with the best-performing static configuration;
(3) The best performing static configuration *differs* across setups; for instance, the static 16 MHz configuration excels with an energy-rich source but turns into the least effective baseline with an energy-poor one;
(4) FBTC outperforms D^{2}VFS in diverse contexts with its energy-efficient design that diminishes external circuitry overhead, reducing energy use and quiescent current.

Our primary conclusion from the above is that given the variable nature of ambient energy, FBTC either significantly outperforms or matches static configurations in most scenarios. Real-world deployments often show drastic changes in energy supply [2, 30, 35, 36, 40, 79, 86], and may even be approximated to either of the two extremes we consider *at different times* of the system lifetime. Deploying FBTC enables the system to adapt to prevailing energy conditions, maximizing the amount of useful work derived from a given energy budget.

5.1 Setting

Accurately measuring the performance of D^{2}VFS and FBTC is a challenge per se. Collecting metrics and state from a system powered with harvested energy is a non-trivial process that may interfere with the intermittent execution of the system and generally requires significant hardware-software modifications [20]. Further, reproducing ambient energy sources is indeed extremely difficult, as their behavior is generally erratic [29, 34]. We thus opt for software-based system emulation, as this not only enables fine-grained control of experiments but most importantly ensures reproducibility *by us and others*. The code, documentation, and datasets we use are publicly available [64].

We describe next the experimental setting, the benchmarks we run, the baselines we compare with, and the energy environment that systems are exposed to.

Platform and emulation. We employ ScEpTIC [67], an extendable emulator for intermittent programs previously utilized in various studies [65–67] We extend ScEpTIC to emulate the functioning and energy consumption of the circuitry enabling D^{2}VFS or FBTC functionality. We emulate ambient energy sources by replaying voltage traces [3, 34, 75] that are either synthetic or gathered from a real harvester. Throughout program execution, ScEpTIC monitors the capacitor voltage, taking into account the total device energy consumption and harvested energy. Whenever the capacitor voltage falls below a threshold, ScEpTIC emulates an energy failure.

We emulate the MSP430-G2553 [43] MCU from the MSP430 family [48], attached to a $8Kbyte$ MB85RC64V [56] non-volatile FRAM chip through I^2C operating at $1MHz$. We incorporate an energy model of the MCU into ScEpTIC, which considers the various operating modes, and leverages established experimental data [4] to simulate active mode behavior. Evidence exists that during active mode this MCU experiences fluctuations in power consumption that are not represented in its datasheet [4]. We instead rely on the latter [43] to model its energy consumption in low-power mode as well as the energy consumption and latency of peripheral accesses.

We model the latency and energy consumption of the FRAM chip and of the additional components in D^{2}VFS and FBTC using a combination of datasheet information and real measures taken from the fabricated board for FBTC.

Despite ScEpTIC enables the simulation of energy sources, device energy consumption, and circuitry functionality, it does not account for real-world phenomena, including capacitor leakage and circuitry non-linearities, which are inherently difficult to model. To validate the results, we experimentally verify for FBTC that the discharge patterns observed by relying on datasheet information mirror those of the fabricated board. Further details about these aspects are available in Sec. 5.2. We also note that the ADC minimum operating voltage is $2.2V$ on the MSP430-G2553 we consider. Should V_{cap} be lower than $2.2V$, the ADC may return unreliable values, causing unexpected system behaviors, including unnecessary state-save operations. To account for this, we consider three possible settings for Mementos: (i) `Default`, where every function call performs a state-save operation as soon as V_{cap} is lower than $2.2V$, yet the execution continues until $V_{cap} < 1.8V$, (ii) `NOADCOFF`, where we pretend the ADC can operate in the same voltage range of the MCU, and (iii) `ADCMINV`, where we set the MCU to power off at $2.2V$.

We consider two well-established techniques to ensure forward progress: Hibernus [12] and Mementos [75]. Both save the program state on the FRAM chip, including the register file, special registers, and main memory, whenever V_{cap} falls below a specified threshold V_{save}. Hibernus relies on system interrupts that fire whenever the V_{save} is reached; Mementos relies on special function calls, statically placed at specific program locations, that probe V_{cap} through the ADC and accordingly determine whether to save the state. We use ScEpTIC itself to determine an efficient setting for V_{save}, empirically exploring different possible values and eventually settling on the one providing the best energy efficiency to complete a given workload.

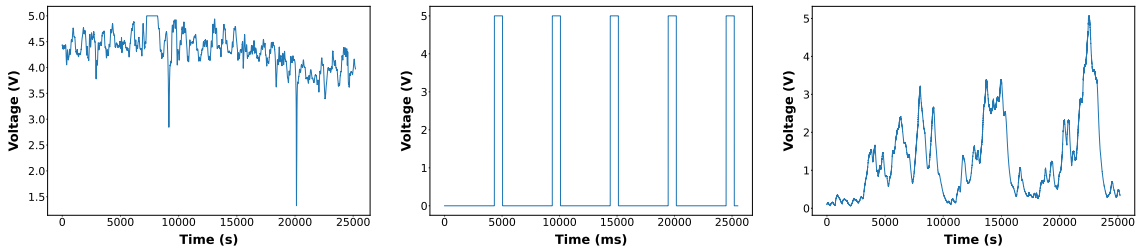
For Hibernus, we consider an external voltage divider of $200K\Omega$ as in the original setup [12] and we use ScEpTIC to model the execution of state-save operations whenever V_{cap} falls below V_{save}. For Mementos, we use the *loop-latch* placement strategy [75] to insert function calls in the source code that probe the value of V_{cap} and compare it with V_{save}. In line with the behavior of a real deployment, we also assume that Hibernus operations to save the system state only cover the *used* portion of main memory, that is, the one delimited by the stack pointer, instead of the whole memory content [12] including unused segments.

Benchmarks, metrics, and baselines. Battery-less devices usually run a periodic sense-process-transmit loop to gather data from the environment and relay that to a collection point [2]. Sensing and data transmission employ external peripherals, such as sensors and radio transceivers; their performance is thus not a function of MCU behavior. Therefore, we focus on benchmarks that represent processing, which execute entirely on the MCU.

We have chosen a suite of benchmarks that exemplify the diverse processing tasks typical in intermittent computing environments [11, 12, 21, 38, 50, 62, 65, 75, 85]: (i) the Dijkstra algorithm for computing the shortest path between two nodes of a graph, (ii) a Fast Fourier Transform (FFT) for signal analysis, and (iii) the RSA for data encryption. We consider the open-source implementation of each benchmark available in the MiBench2 [32, 37] benchmark suite and we compile them using Clang [60] version 8.0.1 with default compiler settings.

We prioritize the metrics of *completion time*—the duration to finish a workload—and *energy consumption*, which are directly influenced by the voltage and frequency adjustments in D^{2}VFS and FBTC. We compare them against a baseline that uses static frequency configurations for the MSP430-G2553 [43], including $1MHz$, $8MHz$, $12MHz$, and $16MHz$.

When quantifying the duration to complete a workload, we distinguish between *execution time* for active periods and *recharge time* for inactive periods. This allows us to identify (i) whether performance is lost or gained in either or both of the phases, (ii) how different configurations of voltage and frequency affect the execution time, and (iii) how the external circuitry of D^{2}VFS and FBTC affect the recharge time. This separation also allows us to identify how different



(a) Solar (walking outdoor) (b) Synthetic (c) RF

Fig. 13. Voltage traces of the considered energy sources.

voltage operational ranges affect performance, as different frequencies have different voltage ranges that affect both the execution and recharge time.

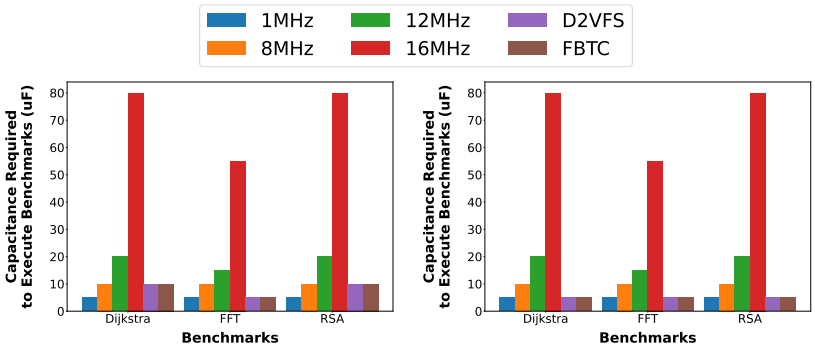
We also track the *number of energy failures* occurring while completing a workload. We consider this metric as an indicator showing how energy consumption affects performance. Given the same initial energy budget, a higher energy consumption leads to shorter energy cycles and thus the system experiences more energy failures. This increases both the execution and recharge time due to additional restore operations and capacitor recharges.

Energy sources and system settings. The characteristics of the energy source largely determine the system's performance. We investigate the system performance with three diverse energy sources.

(1) An *energy-rich* source, whose trace is shown in Fig. 13(a), which enables long energy cycles and yields a low energy failure rate. We reproduce this scenario with the voltage trace of a solar energy source, measured from a solar panel outside our lab while walking [3].
(2) At the opposite extreme, we consider an *energy-poor* source, whose trace is shown in Fig. 13(b), which only produces short energy cycles and yields a high energy failure rate. Similar to previous works [49], we reproduce this scenario with a synthetic $5V$ energy source that supplies energy only when the device is powered off.
(3) The *energy-moderate* source, whose trace is found in Fig. 13(c), represents a middle point between the two extremes. We reproduce this scenario by considering the voltage trace of an RF energy source, taken from Mementos [1, 75].

Capacitor size C and boot threshold V_{boot} determine the length of energy cycles and the time required to recharge after an energy failure. Large capacitors increase the duration of an energy cycle, as they store more energy, yet they also increase the time to reach V_{boot}. Similarly, a high V_{boot} extends the duration of an energy cycle by providing a larger initial energy budget, but it also increases the recharge time. There also exist lower bounds for C and V_{boot}, depending on frequency setting and workload. Their setting determines the energy available in an energy cycle, which we call e_{active}, which must be strictly larger than the sum of the energy consumed by state-save and state-restore operations. Otherwise, a device would not achieve forward progress across energy failures.

To evaluate the performance of D^{2}VFS and FBTC under different conditions, we consider multiple combinations of lower bounds for C and V_{boot}. We use ScEpTIC to determine these settings, running repeated experiments to measure the performance of the various possible configurations. Fig. 14 shows the lower bound for C for the systems we consider and across all benchmarks and system support configurations. The execution of benchmarks at a static frequency of $16MHz$ or $12MHz$ requires at least a $80\mu F$ or $20\mu F$ capacitor, respectively. Instead, the static setting at $1MHz$ or



(a) Hibernus (b) Mementos

Fig. 14. Minimum capacitance required to execute benchmarks at a given frequency.

$8MHz$, D^{2}VFS, and FBTC require no more than a $10\mu F$ capacitor, that is, the minimum decoupling capacitance of the MSP430-G2553 suggested by TI [47].

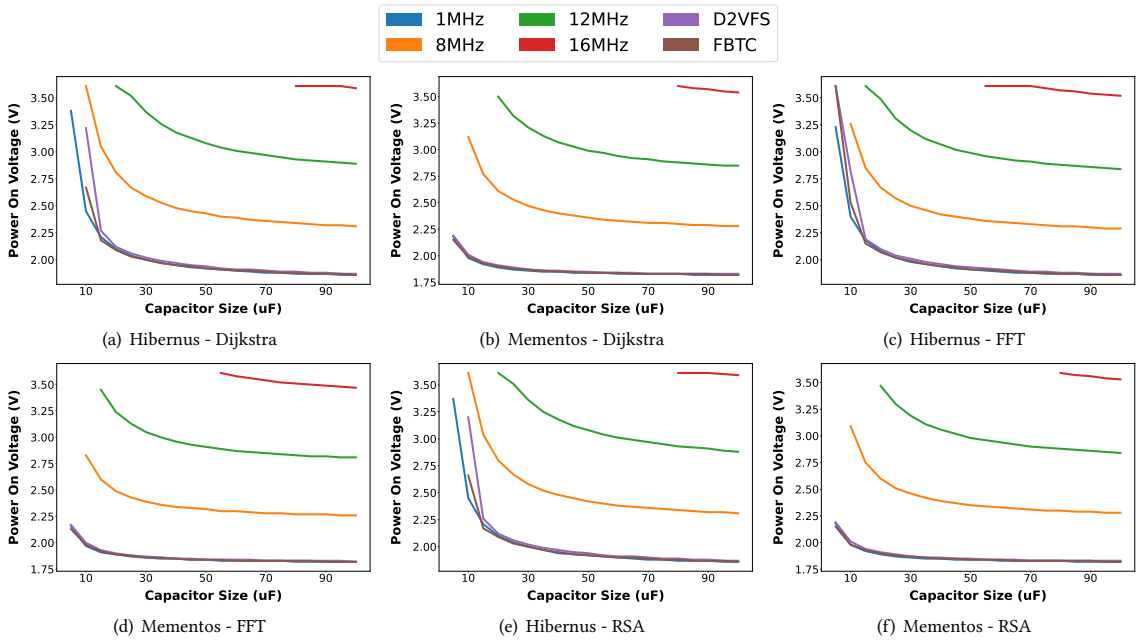
Based on these results, we use two capacitor sizes: (i) $80\mu F$ to run experiments for all baselines and settings, and (ii) $20\mu F$ to run experiments using all baselines except $16MHz$. Then, we identify the minimum V_{boot} for each possible capacitor size. Fig. 15 shows the V_{boot} setting across benchmarks and capacitor sizes. In general, the trend is consistent with the voltage operating range at a given frequency: the $16MHz$ configuration has the highest V_{boot}, whereas the $1MHz$ configuration has the lowest. Note that the curves for D^{2}VFS and FBTC closely align with that of the $1MHz$ configuration, due to their similar voltage operating ranges..

Quiescent current. Our models in ScEpTIC account for the quiescent current $I_{quiescent}$ due to external circuitry, which causes the capacitor to discharge even when the MCU is off. This applies to Hibernus [12], D^{2}VFS, and FBTC. Note that we ignore the capacitor leakage current, as it is negligible compared to $I_{quiescent}$. Due to $I_{quiescent}$ and depending on the other system parameters, the energy source may be unable to make the system eventually reach V_{boot}, potentially leading to a scenario where the device never powers on. This is the case of the *energy-poor* source with $C = 100\mu F$ and $V_{boot} = 3.6V$. The short energy bursts rarely exceed the capacitor voltage V_{cap}, and contribute no additional charge.

To address this issue, we integrate into ScEpTIC a model of a voltage doubler between the energy harvester and the capacitor, as used in the WISP platform [69, 77]. Using the voltage doubler, energy bursts exceeding V_{cap} are both more frequent and longer, allowing the capacitor to eventually reach V_{boot} despite the influence of $I_{quiescent}$. This addition is unnecessary for the *energy-rich* and *energy-moderate* sources, but mandatory for the *energy-poor* one when using $20\mu F$ capacitors. Using a voltage doubler may not always be an option, because (i) voltage doublers usually require AC input currents [26], whereas an energy harvester may output DC current [16], and (ii) similarly to voltage regulators, voltage doublers never have a 100% efficiency [26] and thus waste energy.

5.2 Energy Model Validation

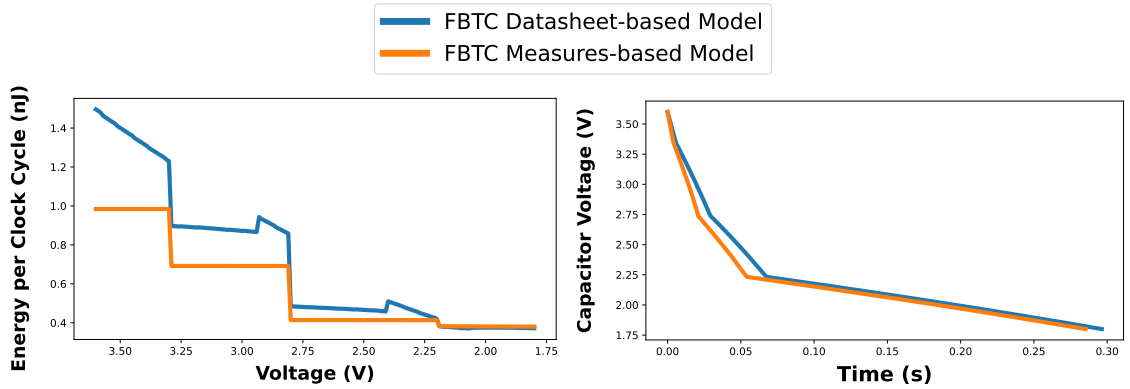
We model D^{2}VFS and FBTC energy consumption using real measures of the MSP430-G2553 [43] MCU and the datasheet information for the various circuitry components of D^{2}VFS and FBTC. To validate the model, we measure the energy consumption of the FBTC board we fabricated. We use a PeakTech 6225A [72] variable power supply to vary the voltage of the FBTC board between $3.6V$ and the minimum operating voltage for the considered clock frequency, using steps of



(a) Hibernus - Dijkstra (b) Mementos - Dijkstra (c) Hibernus - FFT

(d) Mementos - FFT (e) Hibernus - RSA (f) Mementos - RSA

Fig. 15. Minimum V_{boot} required for benchmark execution.

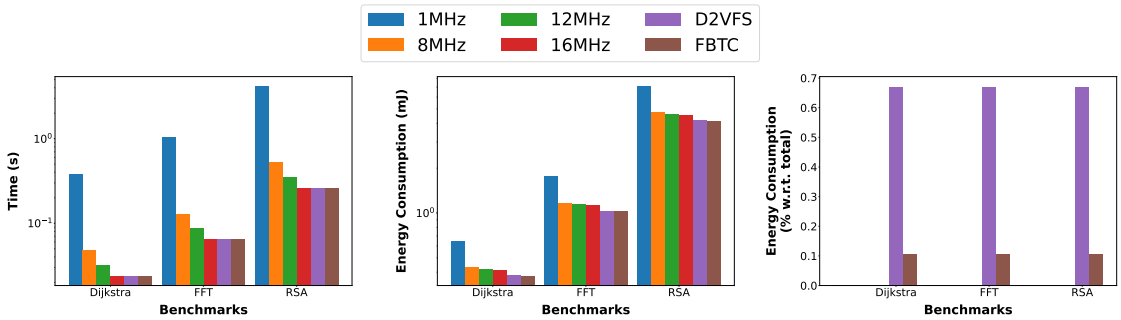


(a) Energy per cycle (b) Capacitor discharge

Fig. 16. Comparison of FBTC datasheet-based model against FBTC measures-based model.

$0.01V$. We measure the FBTC board current draw using a UNI-T UT61E multimeter [84]. We repeat the measures for each operating frequency we consider, namely, $16MHz$, $12MHz$, $8MHz$, and $1MHz$.

Fig. 16 compares FBTC datasheet-based model against the fabricated FBTC board. Fig. 16(a) compares the energy consumption per clock cycle of the datasheet-based FBTC model against our measures. Our model considers an average efficiency of 90% for the TPS62740 [45] voltage regulator [45]. However, this does not represent the actual behavior of the voltage regulator. The measures of Fig. 16(a) show that the voltage regulator has a non-linear behavior and its



(a) Completion time (b) Energy consumption (c) Impact of external circuitry

Fig. 17. Results with the energy-rich source and Hibernus, $\mathbf{C = 80\mu F}$, and $\mathbf{V_{boot} = 3.6V}$.

efficiency depends on the input/output voltages. In particular, between $3.6V$ and $3.3V$, that is, the operating voltage range of the $16MHz$ configuration, our model underestimates the energy consumption by up to 50% and, on average, by 38%. This discrepancy decreases down to 34% (23%) in the voltage range associated to $12MHz$ ($8MHz$), that is, between $3.3V$ ($2.8V$) and $2.8V$ ($2.2V$), with an average underestimation of 28% (13%). Conversely, between $2.2V$ and $1.8V$, that is, the voltage range associated to the $1MHz$ configuration, our model overestimates the energy consumption by up to 2%.

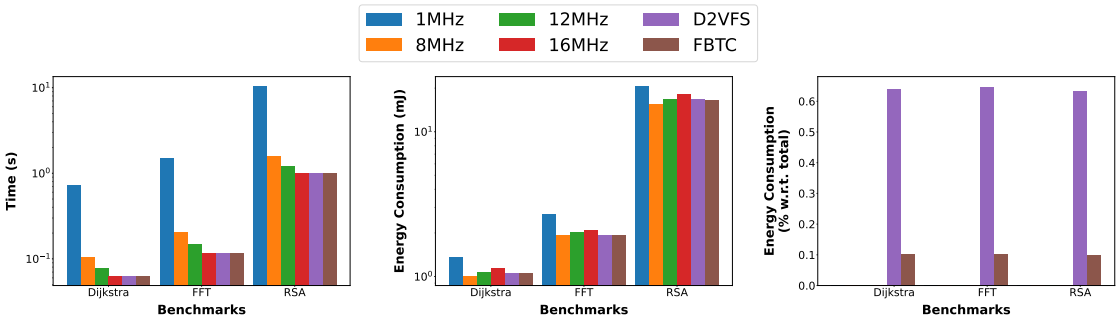
To evaluate the impact of these inaccuracies, we compare the workload achieved in a single discharge of a $100\mu F$ capacitor between the fabricated board the FBTC model. The lower energy consumption of the model results in the execution of 16% more clock cycles. Interestingly, the capacitor discharge time depicted in Fig. 16(b) shows an interesting behavior. The significant difference in the energy estimation between $3.6V$ and $3.3V$ barely affects the discharge time. The overall difference between the discharge times is only 4%, which is mainly caused by the differences in the energy estimation between $3.3V$ and $2.2V$. This is due to the non-linear relation between the capacitor voltage and the capacitor energy, which makes the MCU sustain lower frequencies for longer periods. Consequently, the discrepancy in the energy estimation of higher frequencies bears a very limited impact.

For these reasons, despite the energy estimation difference, there is essentially no difference in the performance trend of the FBTC models against static frequencies and D^{2}VFS across our experiments. Therefore, the results we report next are obtained using the datasheet-based FBTC model, making the results also comparable with those of D^{2}VFS and enabling a per-component analysis of the FBTC energy consumption, which would be unfeasible otherwise.

5.3 Results → Energy-rich Source

Experiments with the energy-rich source experience no energy failures, as sufficient energy is available to complete the workload in a single energy cycle in any configuration. Thus, we do not report on the number of energy failures and the recharge times. Similarly, we do not report on the execution time, as it corresponds to the completion time. In these experiments, the energy source always keeps the capacitor at its maximum voltage, independently of size. We discuss only the experiments with a $80\mu F$ capacitor, as the $20\mu F$ capacitor produces the same results.

Hibernus. Fig. 17 shows the results with Hibernus. Fig. 17(a) depicts the completion time of each benchmark. D^{2}VFS and FBTC require the same time of the static $16MHz$ configuration and are up to $16x$ faster than the other baselines. Under conditions where the harvested energy maintains the capacitor fully charged throughout the experiment, both



(a) Completion time (b) Energy consumption (c) Impact of external circuitry

Fig. 18. Results with the energy-rich source and Mementos, $\mathbf{C = 80\mu F}$, and $\mathbf{V_{boot} = 3.6V}$.

D^{2}VFS and FBTC consistently select the $16MHz$ frequency for its optimal speed and energy efficiency. This results in up to $1.7x$ lower energy consumption, as Fig. 17(b) shows.

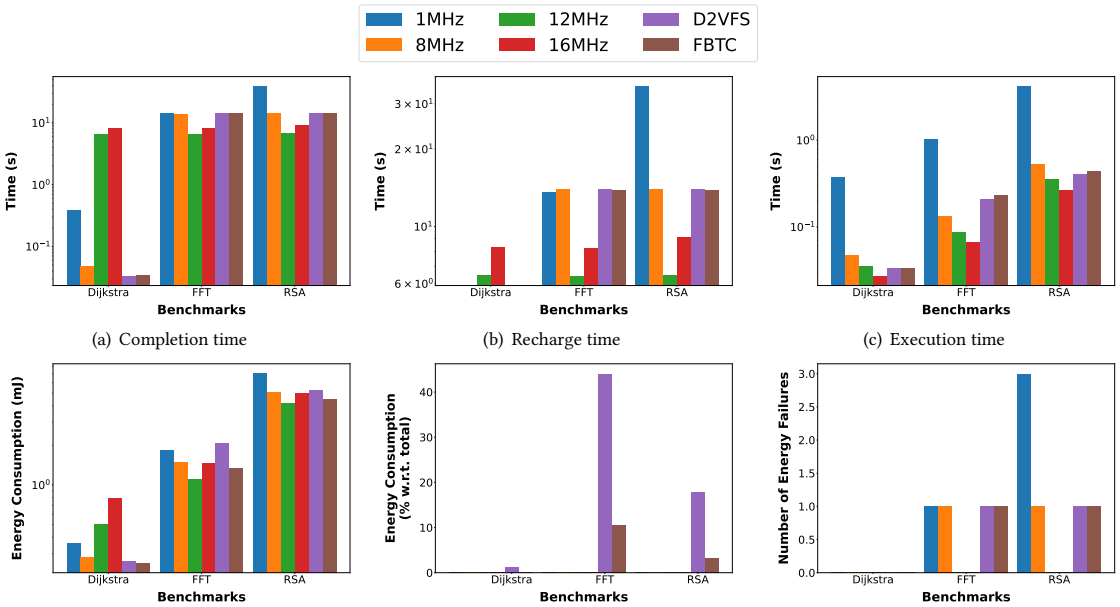
Despite constantly executing at $16MHz$, we note that D^{2}VFS and FBTC show a 9% lower energy consumption than the static $16MHz$ configuration. Both D^{2}VFS and FBTC regulate the supply voltage to the lower bound of the current performance window, that is, $3.3V$. The static configuration running at $16MHz$, instead, does not regulate the supply voltage and provides energy in the range $3.6V - 3.3V$ as the capacitor discharges, ultimately consuming more energy despite the energy overhead of (i) the voltage regulator and (ii) the circuitry of D^{2}VFS and FBTC.

D^{2}VFS and FBTC custom circuitry also bears a negligible impact. Across all benchmarks, Fig. 17(c) shows that it is responsible for just 0.67% and 0.1% of the overall energy consumption, respectively. FBTC has a 0.57% lower energy impact than D^{2}VFS while achieving the same completion time.

Mementos. As the energy-rich source never yields energy failures, the three ADC configurations for Mementos produce the same results, as the voltage is always in the correct ADC operating voltage range. We report only the results of the `Default` configuration, shown in Fig. 18 with the $80\mu F$ capacitor.

Fig. 18(a) shows the same patterns of the experiments with Hibernus: D^{2}VFS and FBTC require the same time of the static $16MHz$ configuration to complete the benchmarks and they are up to $12x$ faster than the other baselines. However, as Fig. 18(b) shows, D^{2}VFS and FBTC no longer show the same marked improvement in energy consumption as with Hibernus. This is due to Mementos' probe function, which turns the ADC on, waits for a sample of capacitor voltage, and turns the ADC back off. These operations introduce an overhead consisting of mandatory wait states that the MCU fills up by executing null operations (NOPs). The number of NOPs is proportional to the MCU operating frequency, thus higher frequencies are subject to a higher penalty. The cost for these NOPs partially outweighs the gains due to regulating the input voltage at the lower bound of the performance window.

Despite the penalty of ADC accesses, D^{2}VFS and FBTC still consume less energy than the static $1MHz$, $12MHz$, and $16MHz$ configurations across all benchmarks, as Fig. 18(b) shows. This is again mainly due to the voltage regulation. Instead, FBTC (D^{2}VFS) consumes, on average, 3.7% (4.29%) more energy than the static $8MHz$ configuration, with a maximum of 7.6% (8.22%) more in RSA. Here again, the cost of ADC accesses at higher frequencies, that is, $16MHz$ compared to $8MHz$, represents a cost that makes the static $8MHz$ configuration more efficient. However, FBTC and D^{2}VFS are, on average, 67% faster than the static $8MHz$ configuration. The decrease in completion time may compensate



(a) Completion time (b) Recharge time (c) Execution time

(d) Energy consumption (e) Impact of external circuitry (f) Number of energy failures

Fig. 19. Results with the energy-moderate source and Hibernus, $\mathbf{C = 80\mu F}$, and $\mathbf{V_{boot} = 3.6V}$.

for the small increase in energy consumption, especially considering that the energy source supplies more energy than the device can buffer anyways. Therefore, an increase in energy consumption does not cause any energy failure.

D^{2}VFS and FBTC custom circuitry bear negligible impact as in the case of Hibernus, that is, 0.64% and 0.1% of the total energy consumption, respectively. FBTC again has a 0.55% lower energy consumption than D^{2}VFS, with the same completion time.

5.4 Results → Energy-moderate Source

We discuss next the results for the experiments with the energy-moderate source, obtained using the voltage traces of an RF energy harvesting system [1, 75]. We set $V_{boot} = 3.6V$ in these experiments.

Hibernus with C = 80μF. Fig. 19 shows the results. The completion times shown in Fig. 19(a) indicate two different trends. With the implementation of the Dijkstra algorithm, D^{2}VFS and FBTC outperform all baselines, whereas with the implementation of FFT and RSA they are on par with the baselines. The two trends deserve separate discussions.

When executing the Dijkstra algorithm, both D^{2}VFS and FBTC surpass the highest-performing static benchmark—the $8MHz$ configuration. They offer a 42% and 41% improvement in completion time, as shown in Fig. 19(a), and consume 8% and 11% less energy, as Fig. 19(d) demonstrates, respectively. Moreover, D^{2}VFS and FBTC are up to two orders of magnitude faster than the baselines and consume up to $3x$ less energy than the static frequency configurations.

The enhanced performance can be attributed to the voltage and frequency scaling capabilities of D^{2}VFS and FBTC. Fig. 19(c) shows that scaling the frequency grants D^{2}VFS and FBTC a shorter execution time than the static $8MHz$ configuration, as they can execute a portion of the code faster. Additionally, by transitioning to the most efficient performance window based on the current capacitor voltage, they maximize the number of instructions executed in each

energy cycle. With this, D^{2}VFS and FBTC show lower energy consumption than the baselines, as shown in Fig. 19(d), allowing both to complete the execution in a single energy cycle, as Fig. 19(f) shows. Note that the static $1MHz$ and $8Mhz$ configurations show a similar behavior. However, due to frequency scaling, D^{2}VFS and FBTC execute faster.

Unlike the Dijkstra algorithm, the FFT and RSA implementations encompass a significantly larger number of machine instructions. This prevents D^{2}VFS and FBTC from completing their execution in a single energy cycle, despite frequency and voltage scaling. As a result, they no longer perform better than all static configurations. Compared to the best-performing baseline, that is, $12MHz$, D^{2}VFS and FBTC are $2.1x$ slower, as shown in Fig. 19(a), and consume, on average, 56% and 15% more energy, as shown in Fig. 19(d), respectively.

The efficacy of D^{2}VFS and FBTC arises from the nature of the energy source coupled with their limited voltage span when activating *hibernation mode*. This mode, unique to Hibernus, transitions the system to a low-power state without full shutdown, allowing for energy accumulation before a checkpoint is imperative. The initiation of hibernation mode is contingent on the minimum voltage required for MCU operation, which is in turn determined by the operating frequency of the MCU.

The higher static frequency configurations, such as $12MHz$ and $16MHz$, enter hibernation mode at a higher voltage level than D^{2}VFS and FBTC. In contrast, D^{2}VFS and FBTC enter hibernation mode with a lower energy reserve. The energy source supplies short energy bursts that are $5s$ apart from each other, as shown in Fig. 13(c), which is insufficient to let D^{2}VFS and FBTC wait in hibernation mode, as the bursts are too far from each other, eventually causing an energy failure. Instead, the $12MHz$ and $16MHz$ static configurations have sufficient energy to wait for the next energy burst and therefore experience no energy failures. Fig. 19(f) provides evidence for this analysis.

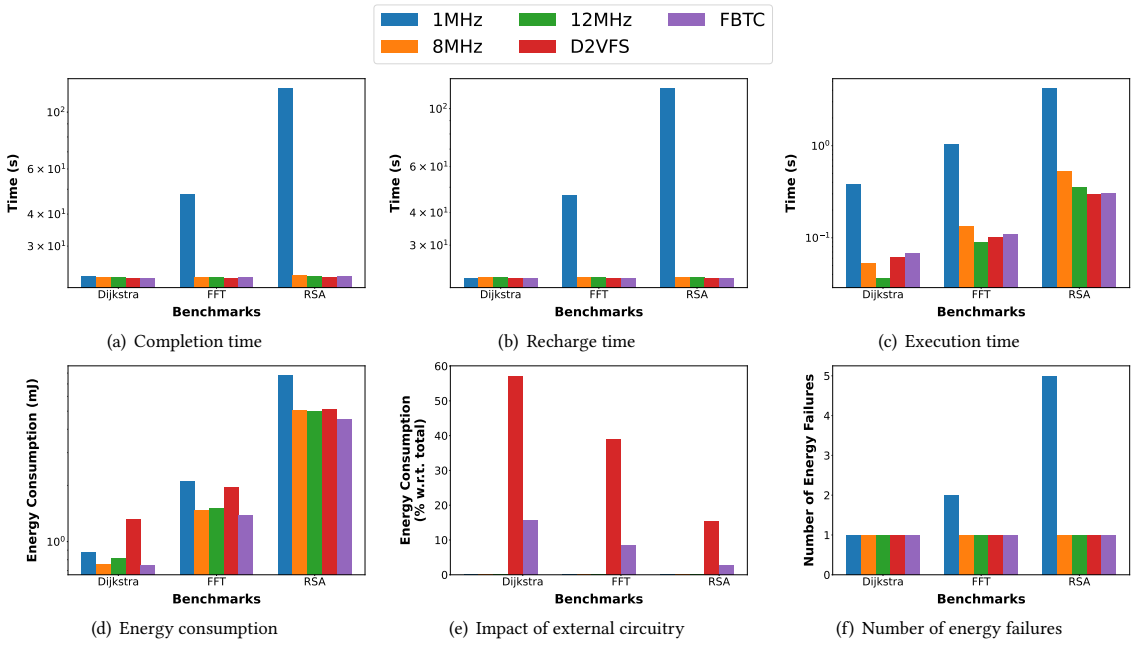
Fig. 19(a), Fig. 19(c), and Fig. 19(b) also indicate that the recharge times represent most of the completion time, whereas the execution times contribute in a limited way. In RSA, the recharge times of the best static frequency configuration, that is, $12MHz$, are 95% of the total completion time, whereas in D^{2}VFS and FBTC the recharge times are 97% of the completion time. The increase in recharge times is another consequence of D^{2}VFS and FBTC entering hibernation mode with lower energy compared to the $12MHz$ static configuration. D^{2}VFS and FBTC show $2.1x$ higher recharge times than the latter configuration, as both must recharge the capacitor to V_{boot} starting from a lower voltage.

On average, FBTC achieves a 0.01% faster completion time and exhibits a 24% reduction in energy usage compared to D^{2}VFS across all evaluated benchmarks. Fig. 19(e) shows that D^{2}VFS external circuitry bears a higher impact on overall energy consumption than in the case of FBTC. D^{2}VFS external circuitry is indeed responsible for up to 44% of the total energy consumption, whereas this figure is limited to 11% for FBTC.

Hibernus with C = 20μF. The smaller $20\mu F$ capacitor setting allows us to run tests with RF energy harvesting without using a voltage doubler, as discussed in Sec. 5.1. We do not consider the static $16MHz$ configuration here, as it cannot complete the workload with such a small capacitor size.

Fig. 20 shows the results. Unlike the case with $C = 80\mu F$, D^{2}VFS and FBTC outperform all baselines. The capacitor size determines this performance, as it causes all systems to enter hibernation mode with little energy. In fact, as Fig. 20(b) shows, the recharge times of D^{2}VFS and FBTC are close to the best-performing baseline and overall account for up to 99% of the total completion time.

Fig. 20(a) shows that D^{2}VFS and FBTC complete the benchmarks $5.4x$ times faster than the static $1MHz$ configuration, with a performance similar to the two static $8MHz$ and $12MHz$ configurations. FBTC also shows the lowest energy consumption across the board, as shown in Fig. 20(d): it consumes *at least* 22% less energy than the baselines. Instead, D^{2}VFS higher quiescent current results, on average, in a 22% higher energy consumption than the baselines.



(a) Completion time (b) Recharge time (c) Execution time

(d) Energy consumption (e) Impact of external circuitry (f) Number of energy failures

Fig. 20. Results with the energy-moderate source and Hibernus, $\mathbf{C = 20\mu F}$, and $\mathbf{V_{boot} = 3.6V}$.

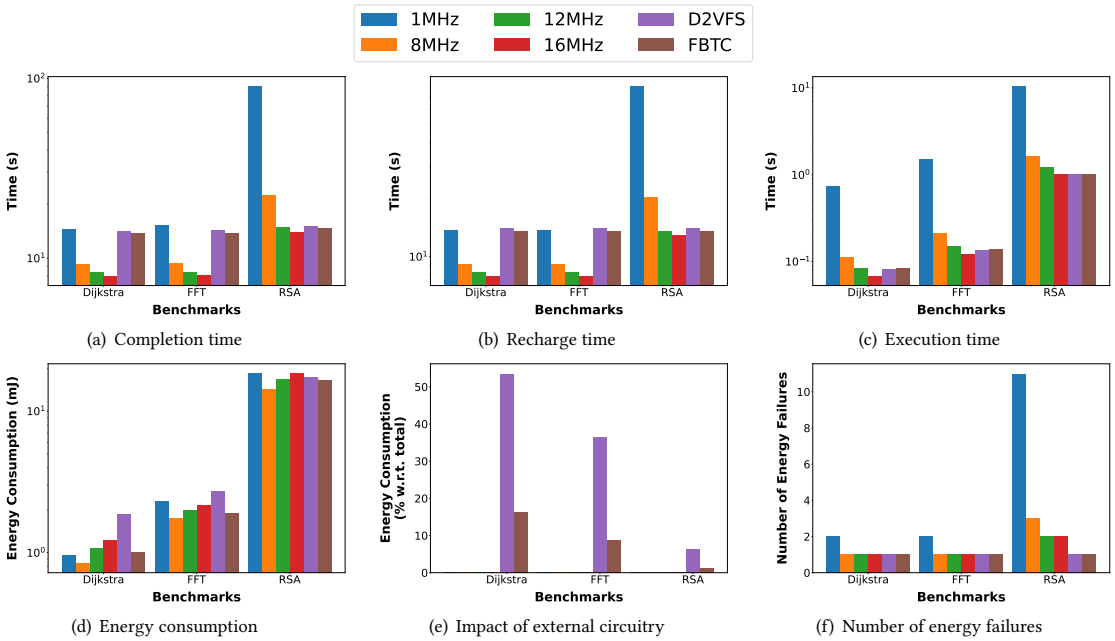
These results are due to voltage and frequency scaling, as D^{2}VFS and FBTC can temporarily set the MCU to run at $16MHz$, operating in a more efficient condition than the baselines. Compared to the $80\mu F$ case, this produces a shorter execution time, as Fig. 20(c) shows. Further, the higher the number of clock cycles in the workload, the faster D^{2}VFS and FBTC complete the benchmarks compared to static configurations. This is the case in the RSA implementation, as opposed to Dijkstra and FFT implementations.

The smaller capacitor impacts the number of energy failures the system is subject to, shown in Fig. 20(f): all the baselines now experience an energy failure, whereas with a $80\mu F$ capacitor the static $12MHz$ configuration did not. No system can now complete the Dijkstra implementation in one energy cycle.

The same performance difference of the $80\mu F$ capacitor case remains here between D^{2}VFS and FBTC. On average, FBTC is 0.27% slower than D^{2}VFS, while consuming 44% less energy. However, there is now an increase in the overall energy consumption of D^{2}VFS and FBTC components due to higher recharge times. Fig. 20(e) shows that D^{2}VFS circuitry is now responsible for up to 57% of the total energy consumption, whereas FBTC circuitry is responsible only for up to 15% of it.

Mementos with C = 80μF. We run the experiments considering the three Mementos configurations, namely `Default`, `NOADCOFF`, and `ADCMINV`, described in Sec. 5.1. The results show no significant change in performance between these configurations. For these reasons, we report here only the results for `ADCMINV`, as it represents the most reasonable choice for a real-world deployment.

Fig. 21 summarizes the results. Fig. 21(a) indicates that the static $16Mhz$ configuration has the shortest completion time. Analogous to the Hibernus experiments, the reduced operating range of this configuration enables the MCU to resume computation sooner after an energy failure, as the capacitor needs less energy to attain the boot voltage



(a) Completion time (b) Recharge time (c) Execution time

(d) Energy consumption (e) Impact of external circuitry (f) Number of energy failures

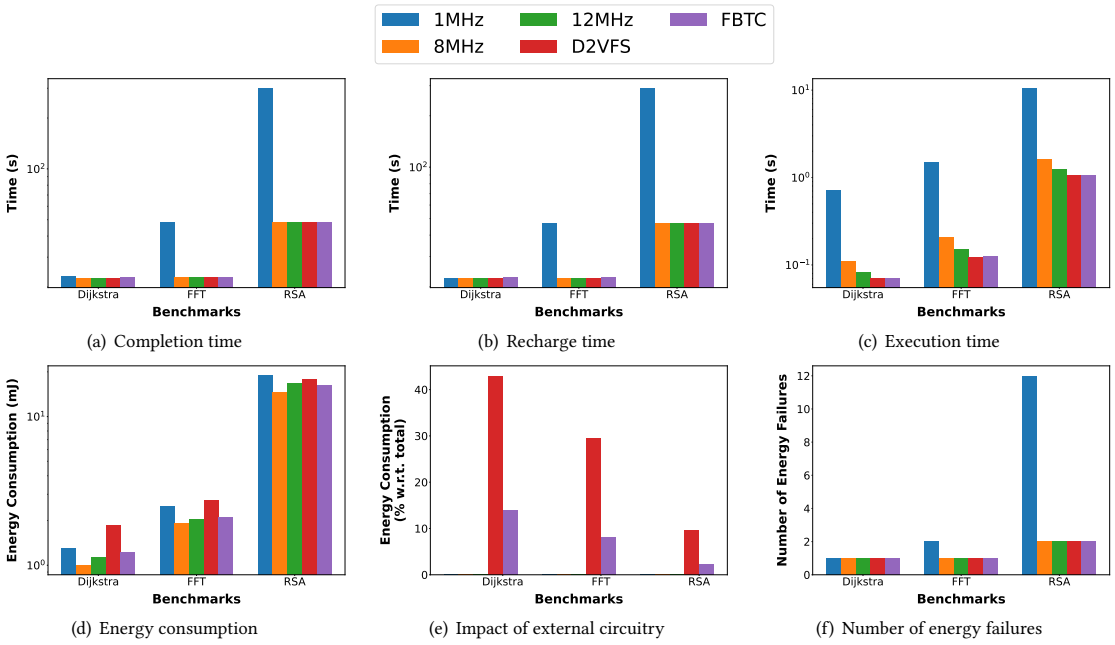
Fig. 21. Results with the energy-moderate source and Mementos with ADCMINV, $\mathbf{C = 80\mu F}$, and $\mathbf{V_{boot} = 3.6V}$.

V_{boot} again. With Mementos, the MCU shuts down without entering a hibernation mode, thus avoiding the capacitor discharge caused by the quiescent current consumption of Hibernus's external comparators, since Mementos does not rely on any such components. The baselines thus recharge back to V_{boot} faster than D^{2}VFS and FBTC, which may be functioning at a slower, less efficient frequency or are turned off while awaiting the energy buffer to refill to V_{boot}.

Similarly to the Hibernus experiments, the completion time is mainly affected by the recharge time, as Fig. 21(b) and Fig. 21(c) jointly demonstrate. When running the the Dijkstra and FFT implementations, D^{2}VFS and FBTC execution times are within the execution time of the $12MHz$ static configuration, whereas in the RSA implementation they match the one of the $16MHz$ static configuration. Considering that a deployed system runs the same workload indefinitely, in the long run D^{2}VFS and FBTC may show significantly shorter overall completion times compared to the baselines.

Fig. 21(d) shows that the static $8MHz$ configuration results in the most efficient energy performance, which however does not translate into the shortest completion time, as seen in Fig. 21(a). Among the three benchmarks, D^{2}VFS always shows one of the highest energy consumption, consuming on average 66% more energy than the static $8MHz$ configuration. Instead, on average, FBTC consumes 45% less energy than D^{2}VFS and 12% more energy than the static $8MHz$ configuration, always resulting among the most efficient configurations.

Two factors influence D^{2}VFS and FBTC energy performance in this setting. As we point out in the experiments with the energy-rich source of Sec. 5.3, ADC probing introduces a clock cycle penalty that increases with the MCU operating frequency. Hence, ADC probing makes D^{2}VFS and FBTC pay a higher penalty than the static $8MHz$ configuration, as the former execute a portion of the program at $16MHz$ and $12MHz$, which incur in a higher penalty than the static $8MHz$ configuration. Second, D^{2}VFS and FBTC have a quiescent current draw that does not impact the baselines.



(a) Completion time (b) Recharge time (c) Execution time

(d) Energy consumption (e) Impact of external circuitry (f) Number of energy failures

Fig. 22. Results with the energy-moderate source and Mementos with ADCMINV, C = **20μF**, and $\mathbf{V_{boot}}$ = **3.6V**.

Despite the higher energy consumption, when running the Dijkstra and FFT implementations, D^{2}VFS and FBTC experience the same number of energy failures of the baselines, as Fig. 21(f) shows. Instead, with the RSA implementation, D^{2}VFS and FBTC experience only one energy failure, whereas the baselines experience at least twice that. This demonstrates that, despite the higher energy consumption, D^{2}VFS and FBTC can manage energy more efficiently, as they experience fewer energy failures.

The more efficient voltage and frequency scaling circuitry of FBTC demonstrates, on average, a 45% lower energy consumption and a 3.6% faster completion time than D^{2}VFS. The higher quiescent current draw of D^{2}VFS components is responsible, on average, for the 33% of the overall energy consumption, whereas FBTC components bear only a 9% impact, as shown in Fig. 21(e). This causes D^{2}VFS to consume 4.5x more energy than FBTC when the MCU is powered off and recharges its energy buffer, causing the recharge time of D^{2}VFS to be 4% higher than FBTC, as seen in Fig. 21(b).

Mementos with C = 20μF. As before with Hiubernus, we run experiments with a 20μF capacitor, which does not require a voltage doubler with RF energy harvesting. For the reasons outlined earlier, we discuss only the results for the ADCMINV configuration.

Fig. 22 summarizes the results with this configuration. We note again a different performance compared to the experiments with a 80μF capacitor. Fig. 22(a) shows that there is negligible difference in the completion time between all configurations but the static 1MHz one. With the RSA implementation, that is, the benchmark with the highest number of required clock cycles, FBTC and D^{2}VFS are 0.11% and 0.37% faster than the static 12MHz configuration, respectively, which is the fastest baseline.

The reason for the different performance is the same as with the Hibernus experiments: the capacitor size no longer represents a disadvantage for D^{2}VFS and FBTC extended voltage range. D^{2}VFS and FBTC recharge times are now

on par with the baselines, as Fig. 22(b) shows. This also demonstrates that the quiescent current of D^{2}VFS and FBTC external circuitry bears a limited impact on the performance while the MCU is off. The recharge times of D^{2}VFS and FBTC are similar to the baselines, which have no additional hardware and hence no quiescent current draw.

Fig. 22(c) indicates that the execution time of D^{2}VFS and FBTC is, on average, 3.5x shorter than the baselines and at least 16% faster than the best-performing baseline, which is the static 12MHz configuration in this case. The key behind this performance is D^{2}VFS and FBTC voltage and frequency scaling technique. Despite the inability of the static 16MHz configuration to complete the workload with the 20μF capacitor, D^{2}VFS and FBTC can set the MCU to operate at 16MHz *for a portion* of each energy cycle, which is the fastest and most efficient operating frequency. This makes D^{2}VFS and FBTC able to extract the most possible performance out of available energy.

Compared to the experiments with a 80μF capacitor, there is limited difference among the different system configurations in other performance metrics. The same performance difference between D^{2}VFS and FBTC with the 80μF capacitor is visible here too, as FBTC is only 0.28% slower than D^{2}VFS, while demonstrating a 30% lower energy consumption, as shown in Fig. 22(a) and Fig. 22(d). The higher quiescent current draw of D^{2}VFS components is responsible, on average, for 27% of the overall energy consumption, whereas FBTC components bear only a 8% impact, as shown in Fig. 22(e).

5.5 Results → Energy-poor Source

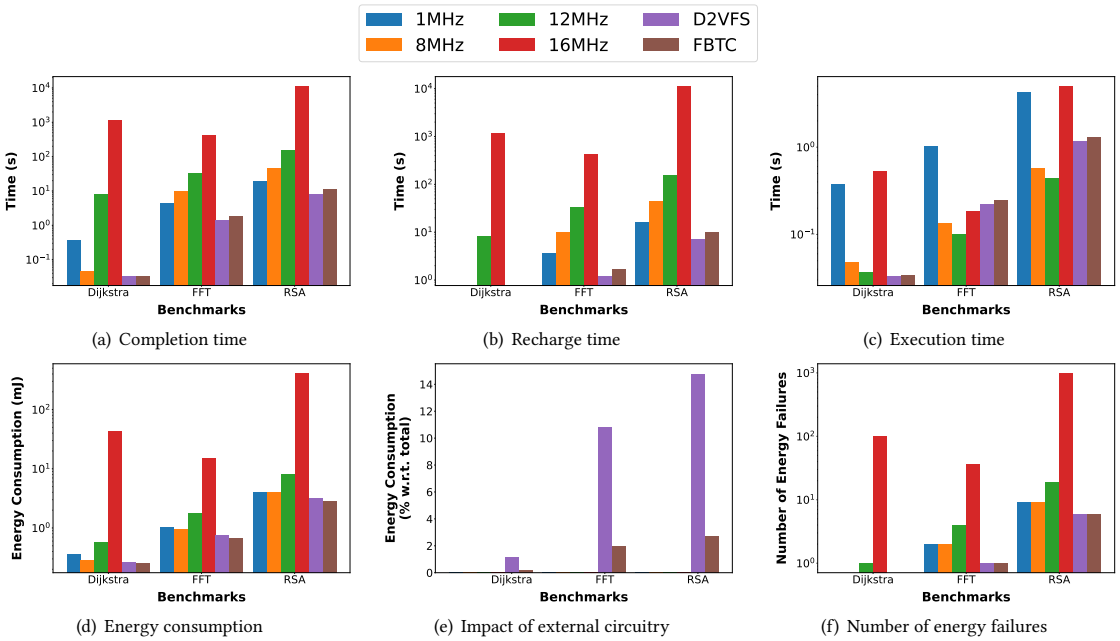
We discuss here the results of the experiments with the energy-poor source, which only produces short energy cycles and yields a high energy failure rate. We reproduce this scenario with a synthetic 5V energy source that supplies energy only when the device is powered off. We set V_{boot} to 3.6V.

In the following and for both Hibernus and Mementos, we only discuss results with a 80μF capacitor. The results with the 80μF show almost identical trends, leading to the same conclusions.

Hibernus. Fig. 23 depicts the results. D^{2}VFS and FBTC demonstrate the best overall performance against all baselines. As the energy-poor source does not supply energy unless the device is off, the duration of an energy cycle only depends on the minimum operating voltage of the selected MCU frequency. D^{2}VFS and FBTC ensure that the MCU consistently operates at the maximum possible frequency and minimum possible voltage. This extends the number of clock cycles executed within a single energy cycle.

Fig. 23(a) depicts the completion time of each benchmark. D^{2}VFS and FBTC are, on average, three orders of magnitude faster than the baselines. Extending the energy cycle by lowering the clock frequency also increases, however, the time required to execute, as Fig. 23(c) depicts. D^{2}VFS and FBTC indeed often show longer execution times than some of the baselines. For example, when running the FFT and RSA implementations, D^{2}VFS and FBTC are respectively 91% and 111% slower than the static 12MHz configuration, that is, the best-performing baseline with this metric. The increase in execution time comes in exchange for a higher number of instructions executed within an energy cycle, which significantly lowers the number of energy cycles required to complete the workload. D^{2}VFS and FBTC also take less time than the baselines in waiting for new incoming energy, abating recharge times up to two orders of magnitude, as shown in Fig. 23(b).

Most importantly, D^{2}VFS and FBTC show a significantly lower energy consumption than all the baselines. Fig. 23(d) shows that D^{2}VFS and FBTC consume, on average, 27x and 29x less energy than the static frequency configurations, respectively. The voltage and frequency scaling techniques allow them to operate in the most efficient conditions. Further, Fig. 23(f) shows that D^{2}VFS and FBTC can complete the Dijkstra algorithm implementation within a single energy cycle, whereas with the FFT and RSA implementations, D^{2}VFS and FBTC experience, on average, 26x fewer



(a) Completion time (b) Recharge time (c) Execution time

(d) Energy consumption (e) Impact of external circuitry (f) Number of energy failures

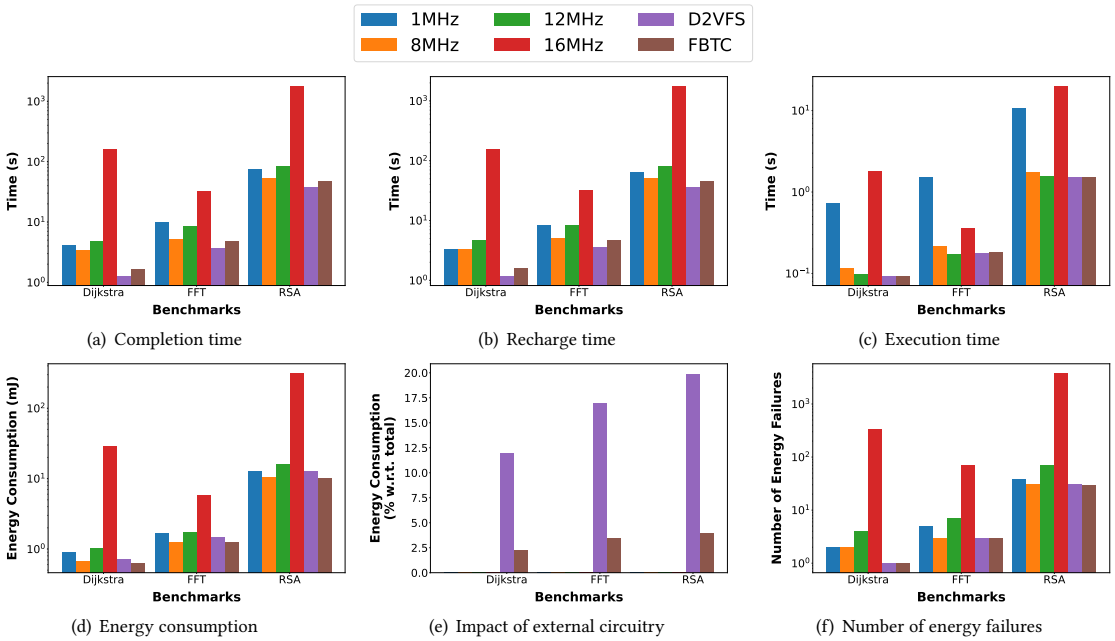
Fig. 23. Results with the energy-poor source and Hibernus, $\mathbf{C = 80\mu F}$, and $\mathbf{V_{boot} = 3.6V}$.

energy failures than the baselines. This behavior is a consequence of D^{2}VFS and FBTC ability to extend the number of instructions executed within an energy cycle, which also results in a reduction of the number of energy cycles required to complete a workload.

The lower quiescent current of FBTC results, on average, in a 9% lower energy consumption than D^{2}VFS, as shown in Fig. 23(d). D^{2}VFS components are responsible for up to 16% of the total energy consumption, wheras FBTC components do not exceed 3% of it, as shown in Fig. 23(e). A higher energy consumption also means a lower equivalent resistance that enables a faster capacitor recharge. Fig. 23(b) shows that the lower resistance of D^{2}VFS results, on average, in a 37% faster recharge time than FBTC. This affects the completion time, as D^{2}VFS shows, on average, a 33% shorter completion time than FBTC, as Fig. 23(a) shows.

Mementos. For reasons similar to Sec. 5.4, Fig. 24 only reports the results with the `ADCMINV` ADC configuration. The performance difference between D^{2}VFS, FBTC, and the baselines generally shows a trend similar to the Hibernus experiments. The execution of Mementos' probe function, however, introduces an additional overhead, because each ADC access introduces a latency that increases with the MCU frequency. Compared with Fig. 23, here the $1MHz$ static configuration pays the highest penalty due to ADC accesses, as the completion times in Fig. 24(d) demonstrate.

Both D^{2}VFS and FBTC outperform the best-performing baselines depending on the metric at hand. They show, respectively, a 42% and 84% shorter completion times than the static $8MHz$ configuration, as Fig. 24(a) demonstrates. Fig. 24(d) also indicates that, on average, FBTC (D^{2}VFS) has a 3.5% (0.81%) lower (higher) energy consumption than the same baseline. Collectively, the metrics of completion times and energy consumption suggest that FBTC and D^{2}VFS outperform the static $8MHz$ configuration by finishing tasks more rapidly while consuming comparable amounts



(a) Completion time (b) Recharge time (c) Execution time

(d) Energy consumption (e) Impact of external circuitry (f) Number of energy failures

Fig. 24. Results with the energy-poor source and Mementos with ADCMINV, $\mathbf{C = 80\mu F}$, and $\mathbf{V_{boot} = 3.6V}$.

of energy. Their ability to dynamically scale voltage and frequency enables them to sustain longer energy cycles by operating in the most efficient settings.

Similarly to the Hibernus experiments with the energy-moderate source, FBTC demonstrates, on average, a 19% lower energy consumption but 29% longer completion times than D^{2}VFS. The lower quiescent current of FBTC, as evidenced in Fig. 24(e), accounts for no more than 4% of the overall energy consumption. In contrast, the components of D^{2}VFS contribute up to 20% of the energy use.

5.6 Summary and Performance Trends

Our experimental results demonstrate the consistent advantages of D^{2}VFS and FBTC over static configurations across various energy conditions. Table 1 provides a concise overview of the key metrics—completion time and energy consumption—summarized across different benchmarks and energy scenarios.

A few notable performance trends emerge. Static configurations, particularly those operating at higher frequencies, excel in energy-rich environments where capacitor charging is frequent, leading to reduced execution times. However, they fall behind in energy-poor scenarios due to increased energy consumption and frequent state-save operations. On the other hand, D^{2}VFS and FBTC consistently show superior performance across all scenarios. These systems dynamically adjust voltage and frequency, maximizing the number of instructions executed per energy cycle. In energy-poor environments, FBTC stands out, offering the most significant energy savings by reducing the overhead of DVFS circuitry and effectively managing performance windows.

Overall, both techniques demonstrate a clear advantage in balancing energy efficiency and execution speed, especially in fluctuating or constrained energy environments. FBTC achieves up to 45% lower energy consumption than D^{2}VFS in

Figure ID	Experiment	Completion Time	Energy Consumption	Key Insights
Fig. 17	Hibernus with energy-rich source	D^2VFS & FBTC up to 16x faster than baselines	Up to 1.7x lower for D^2VFS & FBTC	D^2VFS and FBTC operate at optimal frequency (16MHz) with 9% lower energy than the static 16MHz configuration
Fig. 18	Mementos with energy-rich source	D^2VFS & FBTC up to 12x faster than baselines	Slightly higher than 8MHz static configuration	D^2VFS & FBTC are penalized by ADC accesses in energy but still outperform lower static frequencies in speed
Fig. 19	Hibernus with energy-moderate source (80μF)	D^2VFS & FBTC outperform static baselines for Dijkstra	About 8%-11% lower energy consumption for D^2VFS & FBTC in Dijkstra	D^2VFS and FBTC excel for smaller workloads like Dijkstra but are slower for larger workloads like RSA
Fig. 20	Hibernus with energy-poor source (80μF)	About 5.4x faster than 1MHz static configuration	FBTC shows 22% lower energy consumption than baselines	Frequency scaling gives D^2VFS and FBTC an advantage; capacitor size impacts failures
Fig. 21	Mementos with energy-moderate source (20μF)	Static 16MHz configuration is fastest	FBTC has 45% lower energy consumption than D^2VFS	FBTC benefits from lower quiescent current; D^2VFS incurs higher energy consumption due to scaling
Fig. 22	Mementos with energy-moderate source (80μF)	FBTC is 0.28% slower than D^2VFS	FBTC is 30% more energy efficient than D^2VFS	Quiescent current difference leads to significantly different energy profiles between D^2VFS and FBTC
Fig. 23	Hibernus with energy-poor source	D^2VFS & FBTC complete Dijkstra by 3 orders of magnitude faster	D^2VFS and FBTC consume 27x and 29x less energy than baselines	Frequency scaling allows for maximizing instructions per energy cycle in energy-constrained environments
Fig. 24	Mementos with energy-poor source	D^2VFS and FBTC outperform static configurations for FFT/RSA	Slightly higher energy consumption than the best-performing baseline	D^2VFS and FBTC dynamically scale voltage and frequency to handle intermittent energy supplies

Table 1. Summary of experimental results.

some cases while maintaining comparable completion times, making these approaches, particularly FBTC, effective in scenarios where energy efficiency is crucial.

6 CONCLUSION

In this paper, we delved into the unique challenges faced by intermittently computing devices that harness ambient energy and utilize small capacitors as energy buffers. Traditional methods of setting clock frequency fall short in addressing the intricate relationship between capacitor voltage, operational frequency's energy efficiency, and the associated operational range. Existing techniques, designed for conventional devices, prove to be ill-suited due to the extreme energy limitations and distinct hardware attributes of energy-harvesting devices.

Through our exploration, we introduced two innovative hardware/software co-designs that recognize these distinct hardware characteristics. These designs operate effectively within a constrained energy envelope, each offering its own set of trade-offs and functionalities. Our experimental assessments, grounded in a mix of real-world and synthetic benchmarks, underscore the potential of these techniques to reshape the landscape of intermittent computing. As ambient energy-harvesting devices continue to gain traction, the strategies presented in this paper lay a foundation for their efficient and sustainable operation.

REFERENCES

[1] 2011 (last access: Jul 1st, 2022). RF energy traces used in Mementos. https://github.com/ransford/mspsim/tree/mementos/traces.

[2] M. Afanasov, N. A. Bhatti, D. Campagna, G. Caslini, F. M. Centonze, K. Dolui, A. Maioli, E. Barone, M. H. Alizai, J. H. Siddiqui, and L. Mottola. 2020. Battery-Less Zero-Maintenance Embedded Sensing at the Mithræum of Circus Maximus. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys '20).*

[3] S. Ahmed, Q. Ain, J. H. Siddiqui, L. Mottola, and M. H. Alizai. 2020. Intermittent Computing with Dynamic Voltage and Frequency Scaling. In *Proceedings of the 2020 International Conference on Embedded Wireless Systems and Networks (EWSN '20).*

[4] S. Ahmed, A. Bakar, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola. 2019. The Betrayal of Constant Power × Time: Finding the Missing Joules of Transiently-powered Computers. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and*

Tools for Embedded Systems (LCTES).

[5] S. Ahmed, M. H. Bhatti, N. A. Alizai, J. H. Siddiqui, and L. Mottola. 2019. Efficient Intermittent Computing with Differential Checkpointing. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2019).*

[6] S. Ahmed, N. A. Bhatti, M. Brachmann, and M. H. Alizai. 2021. A survey on program-state retention for transiently-powered systems. *Journal of Systems Architecture* (2021).

[7] S. Ahmed, B. Islam, K. S. Yildirim, M. Zimmerling, P. Pawełczak, M. H. Alizai, B. Lucia, L. Mottola, J. Sorber, and J. Hester. 2024. The Internet of Batteryless Things. *Commun. ACM* 67, 3 (2024), 64–73.

[8] R. Antonio, R. Costa, A. Ison, W. Lim, R. Pajado, D. Roque, R. Yutuc, C. Densing, M. T. de Leon, M. Rosales, and L. Alarcon. 2017. Implementation of dynamic voltage frequency scaling on a processor for wireless sensing applications. In *TENCON.*

[9] A. R. Arreola, D. Balsamo, G. V. Merrett, and A. S. Weddell. 2018. RESTOP: Retaining External Peripheral State in Intermittently-Powered Sensor Systems. *Sensors* (2018).

[10] 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* (2016).

[11] 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* (2016).

[12] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini. 2015. Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems. *IEEE Embedded Systems Letters* (2015).

[13] F. Bambusi, F. Cerizzi, Y. Lee, and L. Mottola. 2022. The Case for Approximate Intermittent Computing. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN).*

[14] G. Berthou, T. Delizy, K. Marquet, T. Risset, and G. Salagnac. 2018. Sytare: a Lightweight Kernel for NVRAM-Based Transiently-Powered Systems. *IEEE Trans. Comput.* (2018).

[15] M. K. Bhatti, C. Belleudy, and M. Auguin. 2010. Power Management in Real Time Embedded Systems through Online and Adaptive Interplay of DPM and DVFS Policies. *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing* (2010).

[16] N. A. Bhatti, M. H. Alizai, A. A. Syed, and L. Mottola. 2016. Energy Harvesting and Wireless Transfer in Sensor Network Applications: Concepts and Experiences. *ACM Transactions on Sensor Networks* (2016).

[17] N. A. Bhatti and L. Mottola. 2017. HarvOS: Efficient Code Instrumentation for Transiently-powered Embedded Sensing. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN).*

[18] A. Branco, L. Mottola, M. H. Alizai, and J. H. Siddiqui. 2019. Intermittent Asynchronous Peripheral Operations. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems (SENSYS).*

[19] C. Church and L. Wuennenberg. 2019. Sustainability and Second Life: The Case for Cobalt and Lithium Recycling. https://www.iisd.org/publications/sustainability-and-second-life-case-cobalt-and-lithium-recycling

[20] A. Colin, G. Harvey, B. Lucia, and A. P. Sample. 2016. An Energy-interference-free Hardware-Software Debugger for Intermittent Energy-harvesting Systems. *SIGOPS Operating Systems Review* (2016).

[21] A. Colin and B. Lucia. 2016. Chain: Tasks and Channels for Reliable Intermittent Programs. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA).*

[22] A. Colin and B. Lucia. 2018. Termination Checking and Task Decomposition for Task-based Intermittent Programs. In *Proceedings of the 27th International Conference on Compiler Construction (CC 2018).*

[23] H. David, C. Fallin, E. Gorbatov, Ulf R. Hanebutte, and O. Mutlu. 2011. Memory Power Management via Dynamic Voltage/Frequency Scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC '11).*

[24] B. Denby and B. Lucia. 2020. Orbital edge computing: Nanosatellite constellations as a new class of computer system. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems.*

[25] S. Eyerman and L. Eeckhout. 2011. Fine-grained DVFS using on-chip regulators. *ACM Transactions on Architecture and Code Optimization (TACO)* (2011).

[26] P. Favrat, P. Deval, and M.J. Declercq. 1998. A high-efficiency CMOS voltage doubler. *IEEE Journal of Solid-State Circuits* (1998).

[27] B. J. Fletcher, D. Balsamo, and G. V. Merrett. 2017. Power Neutral Performance Scaling for Energy Harvesting MP-SoCs. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE).*

[28] F. Fraternali, B. Balaji, Y. Agarwal, L. Benini, and R. Gupta. 2018. Pible: Battery-Free Mote for Perpetual Indoor BLE Applications. In *Proceedings of the 5th Conference on Systems for Built Environments (BUILDSYS).*

[29] M. Furlong, J. Hester, K. Storer, and J. Sorber. 2016. Realistic Simulation for Tiny Batteryless Sensors. In *Proceedings of the 4th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems (ENSsys'16).*

[30] A. Gomez, L. Sigrist, T. Schalch, L. Benini, and L. Thiele. 2017. Efficient, Long-Term Logging of Rich Data Sensors Using Transient Sensor Nodes. *ACM Transactions on Embeddded Computing Systems* (2017).

[31] B. Guo, D. Zhang, Z. Yu, Y. Liang, Z. Wang, and X. Zhou. [n. d.]. From the internet of things to embedded intelligence. *World Wide Web* ([n. d.]).

[32] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. 2001. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop.*

[33] S. Herbert and D. Marculescu. 2007. Analysis of Dynamic Voltage/Frequency Scaling in Chip-Multiprocessors. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design (ISLPED '07)*.

[34] J. Hester, T. Scott, and J. Sorber. 2014. Ekho: Realistic and Repeatable Experimentation for Tiny Energy-harvesting Sensors. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems (SenSys '14)*.

[35] J. Hester and J. Sorber. 2017. Flicker: Rapid Prototyping for the Batteryless Internet-of-Things. In *Proceedings of the 15th ACM Conference on Embedded Networked Sensor Systems (SenSys '17)*.

[36] J. Hester and J. Sorber. 2017. The Future of Sensing is Batteryless, Intermittent, and Awesome. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems (SENSYS)*.

[37] M. Hicks. 2016 (last access: Oct 15th, 2021). MiBench2 - MiBench porting to IoT devices. https://github.com/impedimentToProgress/MiBench2.

[38] M. Hicks. 2017. Clank: Architectural Support for Intermittent Computation. In *Proceedings of the 44th annual International Symposium on Computer Architecture (ISCA)*.

[39] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele. 2014. Energy Efficient DVFS Scheduling for Mixed-criticality Systems. In *Proceedings of the 14th International Conference on Embedded Software (EMSOFT '14)*.

[40] N. Ikeda, R. Shigeta, J. Shiomi, and Y. Kawahara. 2020. Soil-Monitoring Sensor Powered by Temperature Difference between Air and Shallow Underground Soil. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)* (2020).

[41] Texas Instruments. 1998 (last access: Dec 22nd, 2022). SN74LV175A Quadruple D-Type Flip-Flops. https://www.ti.com/lit/ds/symlink/sn74lv175a.pdf.

[42] Texas Instruments. 2009 (last access: Dec 22nd, 2022). Low-Power Dual 2-input Positive-NOR Gate. https://www.ti.com/lit/ds/symlink/sn74aup2g02.pdf.

[43] Texas Instruments. 2013 (last access: Jul 1st, 2022). MSP430-G2553 datasheet. https://www.ti.com/lit/ds/symlink/msp430g2553.pdf.

[44] Texas Instruments. 2014 (last access: Dec 22nd, 2022). SN74AUP1G04 Low-Power Single Inverter Gate. https://www.ti.com/lit/ds/symlink/sn74aup1g04.pdf.

[45] Texas Instruments. 2014 (last access: Dec 22nd, 2022). TPS6274x Step Down Converter Datasheet. https://www.ti.com/lit/ds/symlink/tps62740.pdf.

[46] Texas Instruments. 2016 (last access: Dec 22nd, 2022). SN74AUP1G08 Low-Power Single 2-Input Positive-AND Gate. https://www.ti.com/lit/ds/symlink/sn74aup1g08.pdf.

[47] Texas Instruments. 2020 (last access: Jul 1st, 2022). MSP430 Hardware Design Tips. https://www.ti.com/seclit/ml/sprpe57/sprpe57.pdf.

[48] Texas Instruments. (last access: Jul 1st, 2022). MSP430 family of MCUs. https://www.ti.com/msp430.

[49] B. Islam and S. Nirjon. 2020. Scheduling Computational and Energy Harvesting Tasks in Deadline-Aware Intermittent Systems. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*.

[50] H. Jayakumar, A. Raha, W. S. Lee, and V. Raghunathan. 2015. QuickRecall: A HW/SW Approach for Computing Across Power Cycles in Transiently Powered Computers. *ACM Journal on Emerging Technologies in Computing Systems* (2015).

[51] J. M. Kim, Y. G. Kim, and S. W. Chung. 2013. Stabilizing CPU frequency and voltage for temperature-aware DVFS in mobile devices. *IEEE Trans. Comput.* (2013).

[52] U. Kulau, F. Büsching, and L. Wolf. 2015. Undervolting in WSNs: Theory and Practice. *Internet of Things Journal, IEEE* (2015).

[53] U. Kulau, F. Büsching, and L. Wolf. 2016. IdealVolting: Reliable Undervolting on Wireless Sensor Nodes. *ACM Trans. Sen. Netw.* (2016).

[54] U. Kulau, S. Rottmann, S. Schildt, J. Balen, and L. Wolf. 2016. Undervolting in Real World WSN Applications: A Long-Term Study. In *DCOSS*.

[55] X. Li. 2017. Dynamic Voltage-Frequency and Workload Joint Scaling Power Management for Energy Harvesting Multi-Core WSN Node SoC. *Sensors* (2017).

[56] Fujitsu Semiconductor Limited. 2015 (last access: Jul 1st, 2022). MB85RC64V 8Kb I^2C FeRAM datasheet. https://www.fujitsu.com/jp/group/fsm/en/documents/products/fram/lineup/MB85RC64V-DS501-00013-7v0-E.pdf.

[57] X. Lin, Y. Wang, S. Yue, N. Chang, and M. Pedram. 2013. A framework of concurrent task scheduling and dynamic voltage and frequency scaling in real-time embedded systems with energy harvesting. *Proceedings of the International Symposium on Low Power Electronics and Design*.

[58] S. Liu, Q. Qiu, and Q. Wu. 2008. Energy Aware Dynamic Voltage and Frequency Selection for Real-Time Systems with Energy Harvesting. In *2008 Design, Automation and Test in Europe*.

[59] Y. Liu, H. Yang, R. P. Dick, H. Wang, and L. Shang. 2007. Thermal vs Energy Optimization for DVFS-Enabled Processors in Embedded Systems. In *Proceedings of the 8th International Symposium on Quality Electronic Design (ISQED '07)*.

[60] llvm 2003 (last access: Oct 15th, 2021). The LLVM Compiler Infrastructure. https://llvm.org/.

[61] B. Lucia and B. Ransford. 2015. A Simpler, Safer Programming and Execution Model for Intermittent Systems. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*.

[62] K. Maeng, A. Colin, and B. Lucia. 2017. Alpaca: Intermittent Execution Without Checkpoints. *Proceedings of the ACM Programming Languages* (2017).

[63] K. Maeng and B. Lucia. 2018. Adaptive dynamic checkpointing for safe efficient intermittent computing. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[64] A. Maioli. 2022 (last access: Jul 1st, 2022). ScEpTIC extension implementing system energy emulation. http://sceptic.neslab.it/.

[65] A. Maioli and L. Mottola. 2021. ALFRED: Virtual Memory for Intermittent Computing. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems (SenSys '21)*.

[66] A. Maioli, L. Mottola, M. H. Alizai, and J. H. Siddiqui. 2019. On Intermittence Bugs in the Battery-Less Internet of Things (WIP Paper). In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*.

[67] A. Maioli, L. Mottola, M. H. Alizai, and J. H. Siddiqui. 2021. Discovering the Hidden Anomalies of Intermittent Computing. In *Proceedings of the 2021 International Conference on Embedded Wireless Systems and Networks (EWSN 2021)*.

[68] A. Y. Majid, C. Delle Donne, K. Maeng, A. Colin, K. S. Yildirim, B. Lucia, and P. Pawelczak. 2020. Dynamic Task-Based Intermittent Execution for Energy-Harvesting Devices. *ACM Transactions on Sensor Networks* (2020).

[69] R.han Menon, R. Gujarathi, A. Saffari, and J. R. Smith. 2023. Wireless Identification and Sensing Platform Version 6.0. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems (SenSys '22)*.

[70] R. Min, T. Furrer, and A. Chandrakasan. 2000. Dynamic Voltage Scaling Techniques for Distributed Microsensor Networks. In *Proceedings of the IEEE Computer Society Annual Workshop on VLSI (WVLSI'00) (WVLSI '00)*.

[71] Nexperia. 2021 (last access: Dec 22nd, 2022). 74HC85 4-bit Magnitude Comparator. https://www.mouser.it/datasheet/2/916/74HC_HCT85-1541793.pdf.

[72] PeakTech. 2022 (last access: Dec 22nd, 2022). PeakTech 6225A Variable Power Supply. https://peaktech-rce.com/en/laboratory-power-supplies/441-peaktech-6225a-laboratory-switching-power-supply-dc-0-30v-0-5a-digital-meters.html.

[73] P. Pillai and K. G. Shin. 2001. Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP '01)*.

[74] H. C. Powell, A. T. Barth, and J. Lach. 2009. Dynamic Voltage-frequency Scaling in Body Area Sensor Networks Using COTS Components. In *Proceedings of the Fourth International Conference on Body Area Networks (BodyNets '09)*.

[75] B. Ransford, J. Sorber, and K. Fu. 2011. Mementos: System Support for Long-running Computation on RFID-scale Devices. *ACM SIGARCH Computer Architecture News* (2011).

[76] M. E. Salehi, M. Samadi, M. Najibi, A. Afzali-Kusha, M. Pedram, and S. M. Fakhraie. 2011. Dynamic Voltage and Frequency Scheduling for Embedded Processors Considering Power/Performance Tradeoffs. *IEEE Trans. Very Large Scale Integr. Syst.* (2011).

[77] A. P. Sample, D. J. Yeager, P. S. Powledge, A. V. Mamishev, and J. R. Smith. 2008. Design of an RFID-Based Battery-Free Programmable Sensing Platform. *IEEE Transactions on Instrumentation and Measurement* (2008).

[78] N. Saoda and B. Campbell. 2019. No Batteries Needed: Providing Physical Context with Energy-Harvesting Beacons. In *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems (ENSSYS)*.

[79] E. Sazonov, H. Li, D. Curry, and P. Pillay. 2009. Self-Powered Sensors for Monitoring of Highway Bridges. *IEEE Sensors Journal* (2009).

[80] ROHM Semiconductor. 2015 (last access: Dec 22nd, 2022). BU49XXG CMOS Voltage Detector. https://www.mouser.it/datasheet/2/348/bu48xxg_e-1874410.pdf.

[81] M. H. Shirvani, A. M. Rahmani, and A. Sahafi. 2020. A survey study on virtual machine migration and server consolidation techniques in DVFS-enabled cloud datacenter: taxonomy and challenges. *Journal of King Saud University-Computer and Information Sciences* (2020).

[82] STMicroelectronics. 2013 (last access: Dec 22nd, 2022). TS881 Comparator. https://www.st.com/resource/en/datasheet/ts881.pdf.

[83] M. Surbatovich, L. Jia, and B. Lucia. 2021. Automatically Enforcing Fresh and Consistent Inputs in Intermittent Systems. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2021)*.

[84] UNI-TREND Technology. 2022 (last access: Dec 22nd, 2022). UNI-T UT61E Digital Multimeter. https://meters.uni-trend.com/product/ut61plus-series/.

[85] J. Van Der Woude and M. Hicks. 2016. Intermittent Computation Without Hardware Support or Programmer Intervention. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI)*.

[86] K. Vijayaraghavan and R. Rajamani. 2010. Novel Batteryless Wireless Sensor for Traffic-Flow Measurement. *IEEE Transactions on Vehicular Technology* (2010).

[87] Y. Yang, E. G. Okonkwo, G. Huang, S. Xu, W. Sun, and Y. He. 2021. On the Sustainability of Lithium Ion Battery Industry – A Review and Perspective. *Energy Storage Mater.* (2021).

[88] B. Zhao, H. Aydin, and D. Zhu. 2011. Generalized Reliability-oriented Energy Management for Real-time Embedded Applications. In *Proceedings of the 48th Design Automation Conference (DAC '11)*.