Energy-Driven Computing: Rethinking the Design of Energy Harvesting Systems

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

***Abstract*—Energy harvesting computing has been gaining increasing traction over the past decade, fueled by technological developments and rising demand for autonomous and battery-free systems. Energy harvesting introduces numerous challenges to embedded systems but, arguably the greatest, is the required transition from an energy source that typically provides virtually unlimited power for a reasonable period of time until it becomes exhausted, to a power source that is highly unpredictable and dynamic (both spatially and temporally, and with a range spanning many orders of magnitude). The typical approach to overcome this is the addition of intermediate energy storage/buffering to smooth out the temporal dynamics of both power supply and consumption. This has the advantage that, if correctly sized, the system 'looks like' a battery-powered system; however, it also adds volume, mass, cost and complexity and, if not sized correctly, unreliability. In this paper, we consider energy-driven computing, where systems are designed from the outset to operate from an energy harvesting source. Such systems typically contain little or no additional energy storage (instead relying on tiny parasitic and decoupling capacitance), alleviating the aforementioned issues. Examples of energy-driven computing include transient systems (which power down when the supply disappears and efficiently continue execution when it returns) and power-neutral systems (which operate directly from the instantaneous power harvested, gracefully modulating their consumption and performance to match the supply). In this paper, we introduce a taxonomy of energy-driven computing, articulating how power-neutral, transient, and energy-driven systems present a different class of computing to conventional approaches.**

Keywords*—power neutral; transient computing; energy harvesting; battery-free computing; energy-driven computing.

I. Introduction

The proliferation of mobile, autonomous and wearable devices is creating a dramatic increase in the number of battery-powered computing systems. While battery-powered devices can offer untethered operation, they bring with them issues including battery replacement (and the personnel costs associated with this), frequent recharging, and environmental factors. One potential solution to these is energy harvesting, where electrical energy is scavenged from a device's surrounding environment in order to provide it with a power supply [1]. At first glance, energy harvesting appears to be a very attractive offering, and has gained increasing traction over the past decade; however it is not without its own challenges.

This work was supported in part by the Engineering and Physical Research Council (EPSRC) for their financial support under grant numbers EP/K034448/1 "PRiME: Power-efficient, Reliable, Many-core Embedded systems" (www.prime-project.org) and EP/L000563/1 "Continuous on-line adaptation in many-core systems: From graceful degradation to graceful amelioration". Experimental data used in this paper can be found at DOI: 10.5258/SOTON/404058.

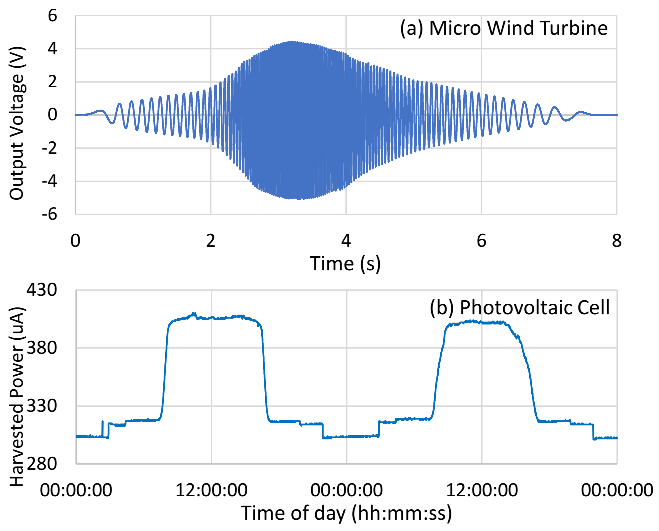

Fig. 1. Example energy harvesting source outputs, showing: (top) the voltage output of a micro wind turbine during a single 'gust', and (bottom) the available power from an indoor photovoltaic cell over a period of two days [2].

Arguably the greatest challenge is the change in the dynamics of the power supply that has to be handled by the system designers when moving from a battery-powered system to an energy-harvesting system. A battery is an energy source that provides virtually unlimited power for a reasonable amount of time, until it becomes exhausted. An energy harvester, on the other hand, is typically a power source that is highly unpredictable, and varies by many orders of magnitude both temporally and spatially. Two examples of this are shown in Fig. 1, illustrating the temporal characteristics of a micro wind turbine (providing an AC voltage with a frequency of many Hz) and an indoor photovoltaic cell (providing an output power that changes throughout the period of a day. Furthermore, both of these examples would likely display significant spatial variation. Compounding the problem further, the computational loads that operate on these low-power mobile, autonomous and wearable devices typically have highly variable power consumptions that are, themselves, unpredictable or event-driven. A key question to solve thus becomes how to enable computation in such systems, despite this highly variable and intermittent supply.

The most commonly adopted solution to this conundrum is offered by energy-neutrality [3], effectively trying to make the energy harvester 'appear' like a battery to the computational

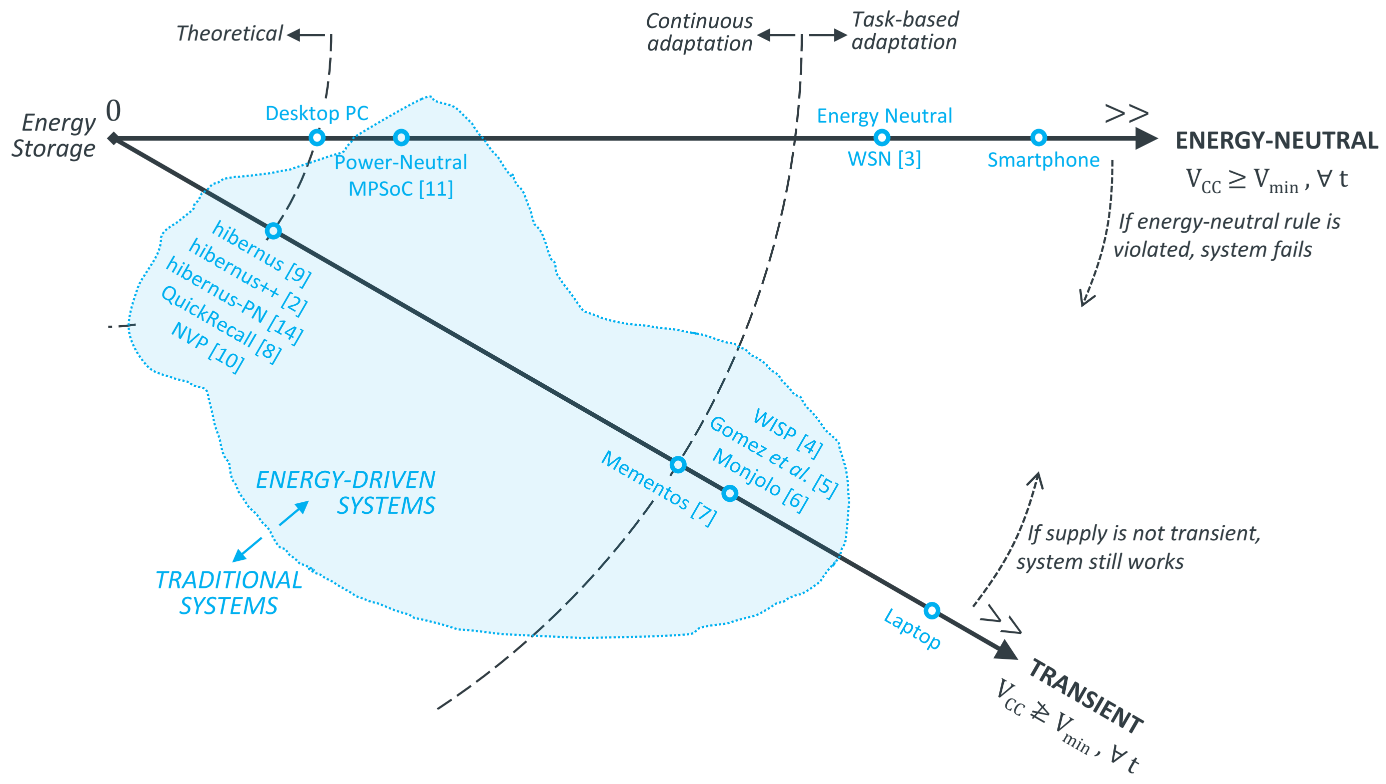

Fig. 2. Taxonomy of energy-neutral, transient, energy-driven and power-neutral computing systems

load. This is undertaken by smoothing out the temporal dynamics of both power supply and consumption through the addition of intermediate energy storage or buffering (e.g. by adding a supercapacitor or rechargeable battery) and appropriate power conversion and conditioning circuitry. These additional components typically increase the cost, volume, weight and complexity of the systems. Increases in these aspects of a system are commonly contradictory to the requirements of wearable, implantable and pervasive systems.

As an alternative approach, we believe that the design of energy-harvesting systems needs to be rethought where, instead of making the energy harvester look like a battery so that it can be connected to a traditional battery-powered load, we need to redesign our complete system with energy-harvesting in mind.

In this paper, we present a taxonomy of computing systems from the perspective of contained energy storage and their ability to operate despite an intermittent supply (Section II). This taxonomy considers different overlapping classes of computing, including those which are energy-neutral, transient, energy-driven, and power-neutral. We describe and illustrate this taxonomy with examples from recent literature, and then present a system (Section III) which incorporates power-neutral, transient, and energy-driven behavior.

II. An Energy-Based Taxonomy of Computing Systems

In this section, we present a taxonomy of computing systems, based on two aspects:

1. The amount of energy storage that they contain;
2. Whether or not operation can be sustained despite an intermittent supply to the computational load (i.e. once any energy storage available is depleted).

Fig. 2. illustrates our taxonomy, where the two axes (*energy-neutral* and *transient*) classify the system's ability to operate correctly despite an intermittent supply to the load, and the distance from the origin (left) depicts the amount of energy storage present in the system. To explain the taxonomy, different classes of system will be considered in turn in the following subsections.

A. *Energy-Neutral Computing*

In an energy-neutral system, the following expression is met:

$$\int_{(n-1)\cdot T}^{n\cdot T} P_h(t)dt = \int_{(n-1)\cdot T}^{n\cdot T} P_c(t)dt \qquad (1)$$

where $P_h(t)$ is the instantaneous harvested power at time t, $P_c(t)$ is the instantaneous power consumption at time t, and T is an appropriate time period over which energy-neutrality is achieved (typically this is related to the periodicity of the energy environment, for example 24 hours in the case of outdoor solar harvesting). Sufficient energy storage (e.g. a battery or supercapacitor) is added to the system to buffer or 'smooth out' the temporal differences in supply and consumption during the period T, such that expression (1) is met. This is illustrated by

Fig. 3, whereby the power supply (e.g. an energy harvester) is separated from the computational load by energy storage and power conversion circuitry, to efficiently condition the output from the power supply for the energy storage element, and then subsequently condition it for efficient use by the computational load.

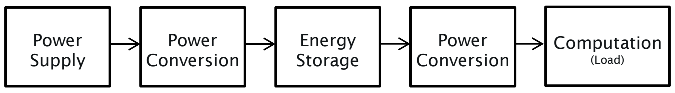

Fig. 3. Block diagram of an energy-neutral system's energy subsystem.

If sufficient energy storage is added to the system, the following expression is met:

$$V_{CC}(t) \geq V_{min} \ , \ \forall t \qquad (2)$$

where V_{min} is the voltage below which the system stops operating. If expression (2) is ever violated, the system can no longer be considered energy-neutral and hence the system fails.

Referring back to Fig. 2, consider only the horizontal line at the top of the figure (the *Energy-Neutral* axis). The right-hand-side of this axis represents a system with a large amount of energy storage, while the left-hand-side represents a system containing zero energy storage/buffering. It should be noted that there is a practical minimum to this, whereby energy storage elements (capacitance) is there for other purposes, e.g. power supply decoupling, parasitic capacitance in electronic components, etc. This is marked on the diagram by the '*Theoretical*' arc, beyond which implementations are unlikely in practice.

The concept of energy-neutrality has been around for many years, and has been widely used in fields such as energy harvesting Wireless Sensor Networks (*WSN*) [3]. In these approaches, added energy storage allows the device to buffer differences in supply and consumption such that expression (2) is ensured. Furthermore, expression (1) is met by adaptively adjusting the consumption of the load such that, over a reasonable period of time T, the energy consumed approximately equals that harvested. Methods for achieving this include adjusting device activity (e.g. changing sample/transmit duty cycles) or participation in network activity (e.g. packet routing).

If we consider a typical *Smartphone* (a mobile computing device), it can also appear to function using energy-neutral operation. The smartphone contains added energy storage (the battery) such that the differences between its variable and intermittent supply (times when it is plugged into the mains) and its variable consumption (phone calls, web browsing, games, standby, etc) are buffered over the period of a day; hence meeting expression (1). If the difference between these becomes too great and the battery is depleted, expression (2) is violated and the system fails – it no longer operates as it was designed to.

Furthermore, considering a *Desktop PC*, it operates at the theoretical minimum of energy storage, primarily consisting of the large decoupling capacitors in the power supply. This is shown on the *Energy-Neutral* axis of our taxonomy (Fig. 2) as expressions (1) and (2) are both met. Over any time T, the energy consumed by the Desktop PC equals that 'harvested' (drawn from the mains grid) – meeting (1). If supply to the load is ever lost, i.e. expression (2) is violated, the system no longer operates correctly (for example, in the case of a power-outage).

The final point on the *Energy-Neutral* axis, *Power-Neutral MPSoC*, will be discussed later in Section II.C.

B. Transient Computing

The fundamental difference between a transient system and the energy-neutral systems that we have considered so far is that, in a transient system, expression (2) can be violated and yet the system can still operate correctly and within the requirements of the application. Consider the other (angled) axis in Fig. 2; the *Transient* axis. Here, the right-hand-side of this axis continues to represent a system with a large amount of energy storage, while the left-hand-side represents a system containing zero energy storage/buffering.

At the rightmost point on this axis is a *Laptop Computer*. It could be argued that a desktop PC should also appear here (or vica-versa), but for the purposes of explanation we consider that the laptop supports the 'hibernation' feature (where all memory and other system state is transferred to the HDD so that the power can be removed) whereas the desktop PC did not. Hence our laptop, much like the smartphone considered earlier, has a large battery to attempt to buffer differences between supply (when it is connected to the mains) and consumption. However, if the battery is not sufficient and is nearly depleted, the operating system automatically instigates a hibernation operation such that its operational state remains intact despite expression (2) being violated.

More commonly however, transient computing is used to overcome some of the problems of adding additional energy storage to a system: i.e. increased complexity, cost, mass and volume. By being able to operate despite expression (2) being violated, transient systems can minimize the energy storage present, reducing it by many orders of magnitude.

The *WISPCam* device [4] is a wireless camera that is powered from harvested RF energy, and supports data-transfer from Non-Volatile Memory (NVM) using RFID. The system contains enough energy storage (a 6mF supercapacitor) to enable a single photo to be taken and stored in NVM. If expression (2) is violated, the system does not fail as the photo is stored in NVM and, when the supercapacitor is later charged again, another photo will be captured. *Gomez et al.* [5] propose a system based on a similar concept, whereby tasks in a wireless sensing system (e.g. sampling data, transmitting data, etc) are not performed until there is enough energy stored in a small (80μF) capacitor. *Monjolo* [6] follows a similar approach for a home energy monitor system, whereby a current clamp around a mains cable harvests energy through induction and charges a 500μF capacitor. When the energy stored reaches a preset value, the system transmits a wireless packet, thus discharging the stored energy in the process. This process repeats as the capacitor is repeatedly charged from the harvested power. The wireless receiver can then use the frequency of received 'pings' to estimate the power being harvested, and hence the power passing through the mains cable.

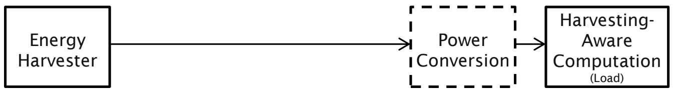

Fig. 4. Block diagram of a power-neutral system's energy subsystem.

These transient approaches are all located on the right hand side of the *Continuous/Task-Based Adaptation* arc on Fig. 2. This is because they all accommodate the intermittent supply by buffering enough energy in storage to complete a single task, be it taking a photo, sampling a sensor, or transmitting data. While these approaches have significantly reduced the amount of added energy storage, some works (including our own [12]) seek to remove energy storage altogether and, where possible, remove external power conversion circuitry; see Fig. 4. This significantly reduces the complexity of the system, and its cost, mass and volume. It also means that there is no longer enough energy stored to execute a complete 'task', and hence they operate on a basis of *Continuous Adaptation* (left side of the arc in Fig. 2). Such systems operate by performing computation when enough power is harvested, while also 'checkpointing' the volatile state of the system into NVM. When the supply is interrupted, the state can later be restored and computation continued from where it left off – similar to hibernation on a laptop (discussed above). However, a question remains of when and how to take a checkpoint to maximize system efficiency.

Mementos [7] places these checkpoints at design/compile time using various heuristics, e.g. at the start of a loop or function. When a checkpoint is reached, Mementos saves ('snapshots') the volatile system state into NVM if V_{cc} is less than a predefined threshold. After a supply interruption, Mementos restores the most recent snapshot from NVM. This approach has three downsides: 1) redundant snapshots add both time and energy overhead; 2) a snapshot might be started but not completed before the supply is interrupted; 3) after a snapshot is restored, any code that was executed since the last snapshot has to be repeated. Mementos appears in Fig. 2 at the boundary between continuous and task-based adaptation. This is because of the way in which checkpoints are placed, and the segments of code in-between checkpoints could be considered mini 'tasks'.

QuickRecall [8] overcomes many of Mementos' shortcomings by using FRAM NVM for both data and program memory. As a result, the only volatile state left is in the registers, which are copied to NVM when an imminent power outage is detected using a voltage interrupt monitoring V_{cc} (the system's decoupling capacitance allows this decay to be detected and the volatile state saved before the decreasing V_{cc} causes the system to lose power). This is similar to that of our own work [2][9], discussed in Section III. One disadvantage of this approach is that NVM typically consumes greater power than SRAM, hence a quiescent overhead is always incurred in terms of energy.

Architectural approaches (sometimes referred to as Non-Volatile Processors, or NVPs) have also been proposed for transient computing, which provide hardware support for maintaining and saving state using NVM elements [10].

C. Power-Neutral Computing

The Energy-Neutral systems considered in Section II.A were defined by expression (1) whereby, over a reasonable period of time T, the energy harvested equaled the energy consumed with any short-term differences smoothed by energy storage. Power-neutral systems, however, can use the zero-storage approach illustrated in Fig. 4. However, because they have no storage, T has to be infinitesimally small; hence expression (1) becomes:

$$P_h(t) = P_c(t) \quad (3)$$

In practice, systems cannot react this quickly, and T is a sufficiently small period of time such that variations in $P_h(t)$ and $P_c(t)$ can be accommodated using minimal amounts of energy storage, ideally only that already in the system as parasitic or decoupling capacitance (i.e. the practical limit).

In order to meet expression (3), the system has to modulate its own power consumption and performance, using various controls available on the hardware, for example Dynamic Frequency Scaling (DFS) or Dynamic Voltage and Frequency Scaling (DVFS), and enabling/disabling different processing elements and peripherals (sometimes referred to as 'hot-plugging'). An example of this can be seen in Fig. 6, which plots the power consumption of an ODROID XU-4 platform (containing an ARM eight-core big.LITTLE processor) against performance when executing a raytracing application [11]. The points plotted are different operating points obtained through combinations of core frequency (DVFS) and enabled CPU cores; the power consumption can be modulated by an order of magnitude through this. By modulating this performance at runtime to keep V_{cc} constant (i.e. not charging or discharging the decoupling capacitance), we achieve power-neutral behaviour. This is the approach taken by the *Power-Neutral MPSoC* point on Fig. 2 (note, this particular point is on the Energy-Neutral axis as it is not equipped with transient functionality, and not quite at the practical minimum of energy storage is the decoupling capacitance alone is not sufficient to smooth dynamics). Another example of power-neutral operation (the *hibernus-PN* point on Fig. 2) is given in Section III.

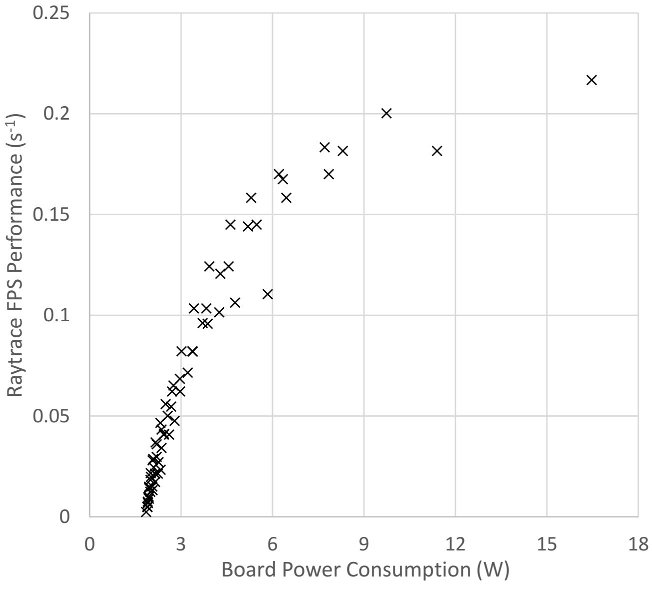

Fig. 5. Control over 'hooks' such as DVFS and disabling combinations of processing elements allows control over the power-consumption of the device (at the expense of performance) [11].

D. *Energy-Driven Computing*

Energy-driven systems are those in which both the application requirements and energy sub-system are driving factors in the way in which the system is designed. It cannot be designed solely from the consideration of the application's requirements, without giving consideration to the energy environment. Many of the examples that have been discussed above are energy-driven systems, as highlighted by the shaded area in Fig. 2.

The following section illustrates an approach for energy-driven systems which is both transient (can operate correctly despite power outages), and power-neutral (adapts its power consumption to match the power being harvested).

III. Achieving Transient and Power Neutral Operation

Hibernus [9] is an approach for transient computing systems which uses *continuous adaptation* and hence can operate with no added energy storage. Unlike Mementos [7] (but similar to QuickRecall [8]), it snapshots volatile state to NVM every time V_{cc} drops below a threshold (i.e. the supply is likely to be imminently interrupted). This means that it usually only makes a single snapshot per supply failure, which removes wasted snapshots (increasing efficiency) and ensures a valid snapshot is always made (improving reliability). To detect the drop in V_{cc}, a voltage interrupt is used where the hibernation threshold, V_H, is chosen such that [9]:

$$E_\sigma \leq \frac{V_H^2 - V_{min}^2}{2} \sum C \tag{4}$$

where E_σ is the energy required to save the system state, and $\sum C$ is the total capacitance of the system. By using an interrupt-driven approach, the snapshot is made as late as possible which avoids re-executing code (increasing efficiency) and maximizes execution time (increasing efficiency). On making a snapshot, it saves all RAM (including the data memory and stack), and CPU registers (core registers and peripherals/special function registers) to NVM. A waveform illustrating a system using *hibernus* to execute an FFT across an intermittent supply is shown in Fig. 7.

Little modification needs to be made to the application code to support *hibernus*; the library must be included and the initialization/restore routine included as the first thing in the `main` function, as shown in Fig. 6. The *hibernus* library, and other resources, can be downloaded from [12]. Of course, the application itself must be able to accommodate transient operation, and this may require substantial modification.

```
#include "hibernation.h" // hibernus library

int main(void)
{
        Hibernus();

        // main code goes here
}
```

Fig. 6. Required additions to the application code in order to support *hibernus*.

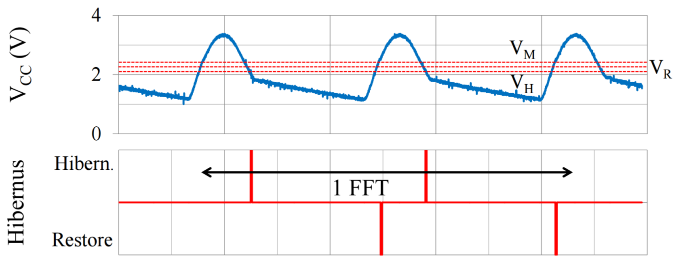

Fig. 7. Example waveform showing a *hibernus* system operating directly from a half-wave rectified sine-wave voltage. When V_H is crossed, the system takes a snapshot (hibernates), and when it recovers (here, crossing V_R) the snapshot is restored. During the third cycle, an FFT that began at the beginning of execution is completed [9].

Hibernus has been validated with multiple applications, and powered from multiple sources including controlled sources (signal generator at DC-20Hz), synthesized energy harvesters, and real energy harvesters (wind, kinetic, RFID, and photovoltaic).

QuickRecall primarily differs from *hibernus* in its use of unified NVM. As previously mentioned, while this reduces the time needed to snapshot (as there is no need to copy the RAM to NVM), NVM typically has a higher power consumption than SRAM. For scenarios where the frequency of power interruption is low, it could be expected that *hibernus* will perform better, and vice versa. This is summarized by [13]:

$$f_{crossover} = \frac{P_{FRAM} - P_{SRAM}}{E_{hibernus} - E_{QuickRecall}} \tag{5}$$

where P_{FRAM} and P_{SRAM} are the power consumptions of FRAM and SRAM respectively and $E_{hibernus}$ and $E_{QuickRecall}$ are the energy consumed per snapshot for *hibernus* and QuickRecall respectively. *hibernus* requires design-time calibration to:

1. Select the hibernate threshold V_H based on $\sum C$ (i.e. characterizing the properties of the platform it is to be executed on);
2. Select the restore threshold, where a previously made snapshot is copied back to volatile memory following a supply interruption, based on the expected dynamics of the energy harvesting source (i.e. characterizing the energy harvesting source).

As an extension to *hibernus*, *hibernus*++ [2] performs adaptive, run-time calibration and management of the platform and energy harvesting source, to avoid the need to provide the design-time calibration mentioned above. Through this approach, *hibernus*++ allows the system to operate effectively with an amount of energy storage that was unknown at design-time. Compared to a *hibernus* system where the storage has been manually characterized and its operation optimized for it, *hibernus*++ will operate slightly less efficiently due to the overheads of the online characterization process. However, if there is greater storage than it was pre-characterized for, *hibernus*++ will operate more efficiently (as it will increase the active time). If there is less storage than it was pre-characterized for, *hibernus*++ will still operate (whereas *hibernus* will not have enough time to save state, and hence will no longer be able to operate correctly).

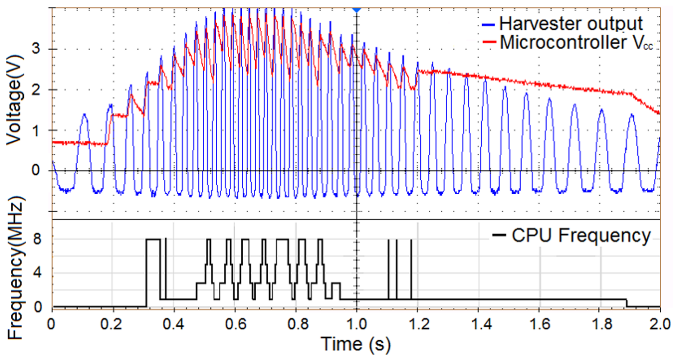

Fig. 8. Example of power-neutral operation [14], where a microcontroller dynamically adapts its core frequency to modulate its power consumption in response to the half-wave rectified signal from a micro wind turbine.

hibernus and *hibernus*++ enable transient computing using continuous adaptation, and hence can operate on systems with no added energy storage. However, when the system is operating, its performance (hence power consumption) is static; this is likely to either waste power or draw too much (causing the supply to drop and operation be interrupted). To alleviate this and improve performance, the principles of power-neutral operation can be used to adjust the clock frequency (DFS) to modulate power consumption to match harvested power [14]. This approach is the *hibernus-PN* point on Fig. 2.

Figure 8 shows the operation of a system using *hibernus-PN* to operate directly from the half-wave rectified output of a micro wind turbine (blue trace). By adapting its frequency through DFS (bottom) in response to the power being harvested, it is able to gracefully increase and degrade its performance to ensure power-neutral operation. Between 0.4 and 1.1 seconds, power-neutral operation allows it to modulate its performance and power consumption such that V_{CC} is not interrupted and hence does not incur the overheads of saving and restoring state.

IV. Discussion

This paper has presented a taxonomy of computing systems from the perspective of contained energy storage and their ability to operate despite an intermittent supply. This taxonomy considers different overlapping classes of computing system, including those which are energy-neutral, transient, energy-driven, and power-neutral. Energy-harvesting systems have have considerable promise, particularly in certain application domains, if associated challenges can be overcome. These are not insignificant, and there are key issues surrounding the design of energy-harvesting computing systems, as they are inherently different to the battery- or mains-powered counterparts.

Energy-neutral computing is excellent for many scenarios, but causes problems for some application domains (e.g. implantable, wearable, and pervasive) as it typically increases the complexity, volume, mass and cost. In many cases, energy-neutral systems appear to be over-engineered for the task at hand, where such effort has gone into making the energy subsystem appear like a battery, that it may have been easier to design it as an energy-driven system from the outset. Transient approaches are such a way of re-thinking how we design energy-harvesting computing systems. However, work to date has primarily focused on computation, and not the plethora of peripherals that are typically present in embedded systems. Power-neutral operation is a complementary approach to transient computing, where the power consumption of the device is continually modulated at run-time (usually sacrificing performance) to match it to the instantaneously harvested power. While promising, better power proportionality (i.e. the range over which the power can be controlled) is needed.

Finally, energy-driven systems are those where the energy environment and subsystem has been an integral part of the design process, right from the stage of application requirements. These systems usually overlap with transient and power-neutral approaches, as the application and its performance has to consider the dynamics of the energy-environment; this often requires a dramatic rethink of what the application requirements truly are. Clearly there are many applications where energy-driven systems are simply not suitable; however, we believe that there are a subset that could realize significant benefit.

References

[1] P. D. Mitcheson, E. M. Yeatman, G. K. Rao, A. S. Holmes, T. C. Green, "Energy Harvesting From Human and Machine Motion for Wireless Electronic Devices," in Proc IEEE, vol. 96, no. 9, pp. 1457-86, Sept. 2008.

[2] D. Balsamo et al., "Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices," in IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 35, no. 12, pp. 1968-1980, 2016.

[3] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava. 2007. Power management in energy harvesting sensor networks. ACM Trans. Embed. Comput. Syst. 6, 4, Article 32 (Sept 2007).

[4] S. Naderiparizi, A. N. Parks, Z. Kapetanovic, B. Ransford and J. R. Smith, "WISPCam: A battery-free RFID camera," 2015 IEEE Int'l Conf. RFID, San Diego, CA, 2015, pp. 166-173.

[5] A. Gomez, L. Sigrist, M. Magno, L. Benini, L. Thiele, "Dynamic energy burst scaling for transiently powered systems," in Proc. Conf. Design, Automation & Test in Europe (DATE'16), Dresden, Germany, Mar 2016.

[6] S. DeBruin, B. Campbell, and P. Dutta, "Monjolo: an energy-harvesting energy meter architecture," in Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys '13), Rome, Italy, Nov 2013.

[7] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on rfid-scale devices," ACM SIGPLAN Notices, vol. 47, no. 4, pp. 159–170, 2012.

[8] 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. J. Emerg. Technol. Comput. Syst. 12, 1, Article 8 (Aug 2015), 19 pages.

[9] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli and L. Benini, "Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems," in IEEE Embedded Systems Lett., vol. 7, no. 1, pp. 15-18, March 2015.

[10] K. Ma, X. Li, K. Swaminathan, Y. Zheng, S. Li, Y. Liu, Y. Xie, J Sampson, and V. Narayanan. Nonvolatile processor architectures: Efficient, reliable progress with unstable power. IEEE Micro, 2016.

[11] B. Fletcher, D. Balsamo, and G. V. Merrett, Power neutral performance scaling for energy harvesting MP-SoCs. Proc. Conf. Design, Automation & Test in Europe (DATE'17), Lausanne, Swizterland, Mar 2017.

[12] University of Southampton. *Energy-Driven Computing* [Online]. Available: www.transient.ecs.soton.ac.uk (accessed 01 December 2016).

[13] A. A. Rodriguez, D. Balsamo, A. Das, A. S. Weddell, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, Approaches to transient computing for energy harvesting systems - a quantitative evaluation. In Proc. Int'l Workshop Energy Harvesting & Energy Neutral Sensing Systems (ENSsys'15), Seoul, Korea, 01-04 Nov 2015.

[14] D. Balsamo et al., "Graceful Performance Modulation for Power-Neutral Transient Computing Systems," in IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 35, no. 5, pp. 738-749, May 2016.