Flicker: Rapid Prototyping for the Batteryless Internet-of-Things

Josiah Hester
Northwestern University
josiah@northwestern.edu

Jacob Sorber
Clemson University
jsorber@clemson.edu

ABSTRACT

Batteryless, energy-harvesting sensing systems are critical to the Internet-of-Things (IoT) vision and sustainable, long-lived, untethered systems. Unfortunately, developing new batteryless applications is challenging. Energy resources are scarce and highly variable, power failures are frequent, and successful applications typically require custom hardware and special expertise. In this paper, we present Flicker, a platform for quickly prototyping batteryless embedded sensors. Flicker is an extensible, modular, "plug and play" architecture that supports RFID, solar, and kinetic energy harvesting; passive and active wireless communication; and a wide range of sensors through common peripheral and harvester interconnects. Flicker supports recent advances in failure-tolerant timekeeping, testing, and debugging, while providing dynamic federated energy storage where peripheral priorities and user tasks can be adjusted without hardware changes. Flicker's software tools automatically detect new hardware configurations, and simplify software changes. We have evaluated the overhead and performance of our Flicker prototype and conducted a case study. We also evaluated the usability of Flicker in a user study with 19 participants, and found it had above average or excellent usability according to the well known System Usability Survey.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; *Architectures*; • **Human-centered computing** → *Ubiquitous and mobile computing systems and tools*;

KEYWORDS

Flicker, Batteryless, Intermittent, Energy Harvesting, Wearable

ACM Reference format:
Josiah Hester and Jacob Sorber. 2017. Flicker: Rapid Prototyping for the Batteryless Internet-of-Things. In *Proceedings of SenSys '17, Delft, Netherlands, November 6–8, 2017,* 13 pages.
DOI: 10.1145/3131672.3131674

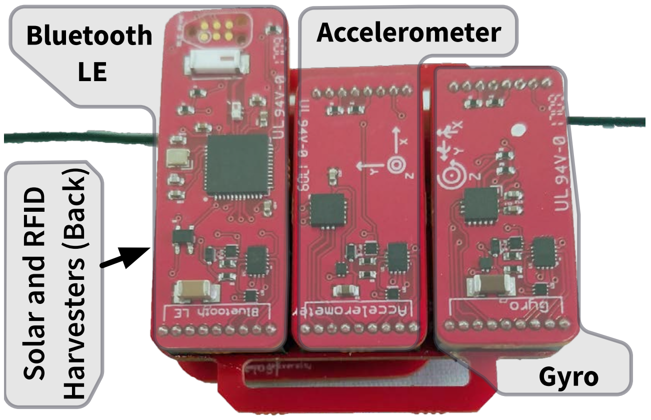

Figure 1: Flicker enables rapid prototyping of batteryless, intermittently powered sensing systems. This figure shows a Flicker device equipped to harvest RFID and solar energy (harvesters are placed on the back side), sense acceleration and rotational velocity, and communicate via Bluetooth LE.

1 INTRODUCTION

The Internet-of-Things (IoT) vision is both exciting and elusive. Billions or trillions of sensors collecting, processing, and communicating data throughout the world will fundamentally change how we manage our health [10], our water [33], our buildings [8, 13], and our interactions with the natural world [27]. But, the IoT has an energy problem. Batteries are large, heavy, expensive and short lived—even rechargeables wear out after a few years—and the maintenance and environmental costs of replacing trillions of batteries every few years are prohibitive.

Batteryless computing devices—devices that gather energy from the environment and execute opportunistically—promise a more sustainable, maintanence-free, and environmentally-friendly alternative, but batteryless computing systems are challenging for developers. Power failures can be frequent and difficult to predict. Data processing, sensing, and communication are often disrupted, clocks reset, and volatile memory lost. Recent advances in checkpointing [2, 29], consistent execution [11, 24], timekeeping [21], energy management [20], testing [19], and debugging [12] address key challenges; however, developing a new batteryless application still often requires specialized expertise, custom hardware development, and considerable hardware tuning. This paper specifically addresses the following key challenges:

Limited Hardware Options: The Intel WISP [31] and its descendants [35], have long been the platform of choice for computational RFID (CRFID) research, but the wide range of energy harvesters, sensors and other peripherals, and exciting new applications that are available to today's batteryless system designers is not well-supported by the WISP's RF-centric design. Application designers that need solar, kinetic, or thermal energy or can't depend on a close-range RFID reader are left to either hack the WISP [18] to suit their needs or create new hardware from scratch.

Limited Flexibility: The success of a batteryless sensing application often hinges on a variety of hardware adjustments that affect how energy is harvested and managed, how tasks are configured, and how data is gathered. Unfortunately, existing batteryless platforms are monolithic, tightly integrating energy harvesting, energy management, sensing, data processing and communication onto a single circuit board that is difficult to modify. Consequently, development is typically slow and developers may not consider promising design alternatives in fear of long delays.

Lacking Modern Amenities: Current platforms also lack recently-developed features, like hardware-assisted zero-power timekeeping [21] that allows devices to measure time across power failures. Timekeeping is incredibly important for sensing, security, data provenance, and data utility. Without a sense of time, batteryless active RFID cards could be brute force attacked for passwords, or endlessly tasked with authentication requests in a DoS attack. Data provenance and utility is usually tied to the time a data point was gathered. Federating energy storage for individual peripherals [20] not only simplifies software development and improves system availability, but improves energy harvesting efficiency. Timekeeping and federated energy storage are particularly important for batteryless applications that rely on ambient energy sources and can't rely on an RF reader to provide time.

Poor Usability or Community: Each of the previous factors contribute to the lack of usability in current hardware platforms, from the programming tools to the hardware inflexibility. Novice developers find it difficult to construct useful devices or verify that they work. The emergence of the maker movement has shown that hobbyists, as well as researchers and industry, are interested in building for the Internet-of-Things. Without a usable platform, no community will emerge.

We have an answer to these deficiencies: Flicker, a flexible, modular hardware platform for energy harvesting, intermittently powered, batteryless sensing devices. With the help of common interconnects, Flicker developers assemble batteryless sensors from a set of interchangable modules—computational cores, energy harvesters, sensors, communication peripherals. Flicker is the first general platform for batteryless, energy harvesting sensing. In this paper, we present twelve different modules, and describe how others can extend this set by developing their own compatible peripherals and harvesters. Flicker supports federated energy storage [20] and features a novel extension of the state of the art, allowing developers to programmatically set wakeup and task trigger points and change the relative priorities of individual sensors and other peripherals. Flicker also supports failure-resistant timing with an onboard capacitive timekeeper[21].

Contributions: At the time of publication, we will release the Flicker platform as an open source, open hardware resource for the research community. We envision Flicker as a catalyst for batteryless sensing research that enables a larger community of developers to easily build and test new batteryless applications. We make the following contributions in this paper:

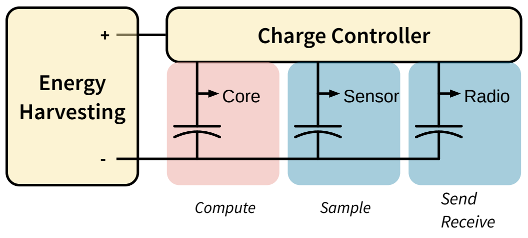

Figure 2: A conceptual view of federated energy storage for capacitor based, energy harvesting sensors. Energy storage is separated per peripheral. Each peripheral maps to a set of of sensing tasks, making energy management more straightforward.

(1) A modular, extensible, hardware platform design and implementation for batteryless, energy harvesting sensors: the first general platform for these devices. This platform will be open source and open hardware[1].
(2) A novel extension of federated energy storage [20] that automatically manages peripheral charging and allows for dynamic retasking and reprioritization of energy resources at runtime.

2 BUILDING FOR BATTERYLESS

The future of sensing likely depends on tiny, batteryless, energy harvesting devices because of the poor economics, sustainability, and scaling of batteries. Today, building and deploying untethered, batteryless sensors is challenging because 1) design-time decisions make prototyping slow and expensive, 2) flexibility of tasks and hardware is lacking, 3) few people have the expertise to build and deploy these sensors, and 4) few readily available hardware options exist, short of building custom hardware.

Modules from Sparkfun[2], or Arduino[3] shields can't just be wired to a batteryless sensor, deployed, and then expected to work. Arduino developers cannot replace the battery with a postage stamp sized solar panel and expect it to function just the same. The architectural, operating systems, and language support for intermittent sensors is not widely available or explored, except in custom configurations of hardware and software. The ultra low operating power requirements, variable energy environments, and difficult to use tools, discourage most developers from ever trying to work with batteryless sensors unless they can afford to design and assemble their own hardware from scratch.

Using recent energy management techniques, capable designers can reduce power failures and make better use of sensing components and other peripherals. For example, federated energy storage [20] (concept shown in Figure 2) assigns a dedicated capacitor for every hardware peripheral, such as the radio, microcontroller, or accelerometer. These capacitors are sized to hold enough energy for a single task using the hardware peripheral it powers. With federated energy, sensors avoid the tragedy of the commons (or coulombs), where a *single* faulty or mismanaged component can

[1]Find the Flicker platform release at https://github.com/PERSISTLab/FlickerPlatform
[2]Sparkfun is a popular company for DIY and maker electronics.
[3]Arduino is the definitive microcontroller ecosystem for embedded electronics makers.

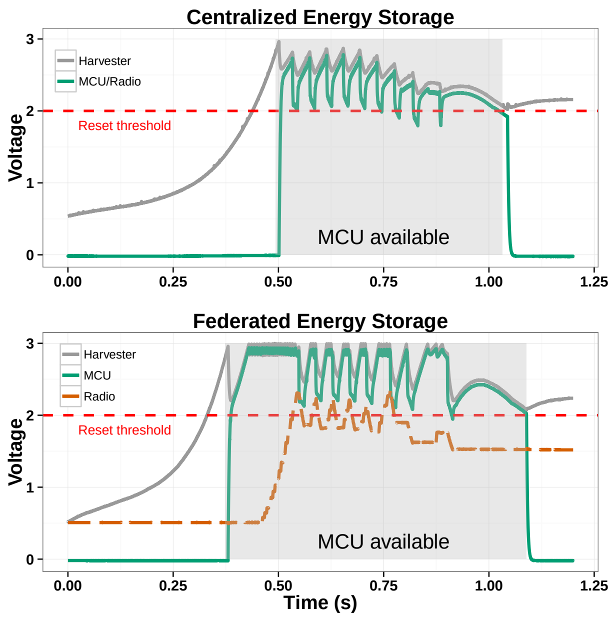

Figure 3: This figure shows a "sense-and-send" program executing with federated energy vs centralized energy storage. Federated energy storage (bottom) allows useful work to start sooner, as smaller capacitors sized to a particular peripheral charge faster than a single shared reservoir. FedEnergy also reduces power failures from coupling of hardware peripherals, and harvests more energy.

deplete the shared energy supply—rendering the *entire* sensor inoperable. With federated energy, devices charge more quickly (small capacitors charge faster than large capacitors), as each component only needs to harvest the energy it needs, not the energy required to support *all* components. Federated energy allows peripherals with different operating voltages to function on the same device: capacitors can be tuned exactly to the peripheral specifications. Federated energy functions by setting voltage thresholds for each capacitor at design time, these thresholds are a direct measure of the amount of energy in a capacitor. The microcontroller monitors the charge state of each capacitor and peripheral-dependent tasks are executed when the needed peripheral is sufficiently charged. An execution of a federated energy and centralized energy (traditional) sensor is shown in Figure 3, the traditional variant is more susceptible to failures, and takes longer to harvest and then store enough energy to start useful computation and sensing.

2.1 Limitations of Static Federated Energy

Federating a batteryless sensor's energy storage improves the reliability, efficiency, and energy use of the whole sensor; however, the federated approach as implemented in UFoP [20] is difficult to use, especially by non-experts, for three reasons described below.

Program-specific hardware designs are brittle. In order to use federated energy effectively, a developer needs to determine the right size for peripheral capacitors, the voltage at which each peripheral should start charging, and the voltage at which a peripheral should be deemed charged and ready to use. In UFoP, capacitor sizes and charging thresholds are static and set at design time, and software changes often require hardware changes—soldering or even circuit board revisions—to ensure good performance. Task priorities cannot be changed once deployed. The result is brittle systems with tight hardware and software dependencies and long, expensive development and debugging cycles. Even for those with hardware expertise, these systems are difficult to maintain and modify.

Static UFoP is inflexible at runtime. When using static UFoP, programmers cannot turn off peripherals when they are no longer in use or assigned to a task. These peripherals continue to charge, storing energy that may never be used and delaying more important tasks. Programmers cannot change the relative priorities of different peripherals adapting tasks at runtime—limitations that fundamentally bound application complexity and the ability to retask deployed sensors in the field.

Significant hardware complexity remains. Hardware complexities and unintended interactions can interfere with the operation of the sensor; in static UFoP this is seen in the approach to voltage regulation, logic levels, and the energy management interface. In order to maximize efficiency and availability, most batteryless sensors do not regulate supply voltages. For some peripherals (especially with RF components), voltage fluctuations will affect accuracy. Other components (like an MSP430 MCU), draw more power at higher voltages. Static UFoP does not propose a standard way to deal with these conflicting peripheral requirements. In its current form, UFoP does not manage logic levels for communication between the MCU and peripherals, and designers must carefully tune capacitors and harvesters to keep voltage level within logic bounds, concurrently. Additionally, UFoP's energy management interface uses a polling-based approach to manage energy levels, which wastes energy in frequent threshold checking.

Today, prototyping batteryless sensing devices is challenging enough to discourage all but the most determined developers. This paper addresses this problem by 1) improving the flexibility and efficiency of federated energy storage and 2) integrating these improvements into a novel and general platform, called Flicker, for flexible and rapid prototyping of the batteryless Internet of Things.

3 FLICKER

We have developed Flicker for IoT application designers who want to develop batteryless energy harvesting devices. Flicker's goals are (1) to provide multiple hardware options in terms of peripherals and harvesting technologies, (2) realize runtime and design time flexibility, (3) enable recent advances in timekeeping and energy management on a general purpose platform, and (4) provide a platform focused on entire stack (software and hardware) usability. Instead of a single, monolithic hardware platform, we present a system of hardware modules that can be used interchangeably and software tools that can automatically analyze hardware configurations, detect incompatibilities, and help developers more easily create new applications. Flicker rethinks Federated Energy to meet the requirements of a reconfigurable platform, adds failure tolerant

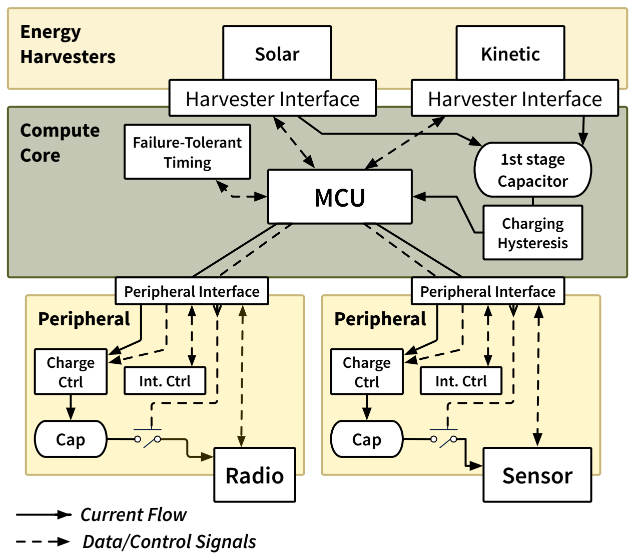

Figure 4: Flicker harware architecture. A multi source energy harvesting interface feeds into the first stage capacitor that powers the Compute Board. Peripherals are connected through a standard interface that maps power, and control signals.

timekeeping, and exposes multiple standard interfaces to peripherals. An overview of Flicker's design is shown in Figure 4.

3.1 Flicker Modules

Flicker modules come in three distinct varieties — compute cores, peripherals, and harvesters. A viable device configuration consists of a single compute core, one or more harvesters, and one or more peripherals.

Compute core modules include a microcontroller (MCU) (the device's main controller, programmed by the application developer and responsible for peripheral control and application logic), timekeeping functionality, ports for attaching harvesters and peripherals, and hardware support for managing federated energy stores. Any MCU can be used to create a core module, but we recommend low-power processors with on-chip FRAM, like the MSP430 FRAM series [22], that work well with a wide range of small harvesters and support efficient checkpointing. Core modules also control how peripherals are used and charged, and serve as the central hub, around which harvesters and peripherals are connected.

Peripherals connect to the core modules' peripheral interface ports, which provide power, control, and signaling for peripheral charging, in addition to analog signal lines, digital signal lines, and digital bus lines (SPI, I2C, and UART) for peripherals that communicate digitally. Peripherals can be radios, sensors, and actuators — in this paper, we focus primarily on radios and sensors, since most actuators are too power hungry for batteryless operation, but this is not a fundamental limitation. While peripheral behaviors and needs will vary significantly, each peripheral stores its own energy and contains circuitry that controls how that energy is stored and used.

Harvester modules harvest energy from a variety of environmental sources. A variety of energy sources — solar, kinetic, vibration, radio frequency (RF), and thermal — are available to application designers, but available harvesters provide the energy they harvest differently, as direct current (DC) or alternating current (AC) and at a variety of different voltages. In Flicker, harvesters are designed to provide energy in a form that can be used directly by the system. Specifically, harvester modules provide DC electricity at voltages that are high enough to support common modules and protect against reverse current flow, using a blocking diode or other similar mechanism. Reverse current protection is common in energy harvesting devices, and critical when multiple harvesters are used simultaneously (preventing one harvester from draining the energy harvested by another).

Some harvesters (namely RF energy harvested from a reader, and NFC) also combine data and energy. Flicker's harvester interface includes optional data lines, specifically to support these harvesters.

3.2 Reconfigurable Federated Energy

Reconfigurable Federated Energy is the crucial innovation allowing quick prototyping with a modularized platform. Converting what is usually a rigid hardware platform to one able to support a multitude of peripherals and energy harvesters. "Reconfigurable" means that programmers (or compilers) can assign (at runtime or compile time) the amount of energy to be harvested for each peripherals capacitor, the priority of charging of each peripheral, and the trip point where enough energy is stored to execute a task. This allows applications to tailor charging behaviors for different configurations of peripherals and harvesters at the prototyping stage, at compile time, or even at runtime. This makes it easy to efficiently support longer tasks that require more energy, without incurring long charging delays for shorter tasks that need less energy. This makes it easy to mix and match different tasks, and peripherals, and removes the hard coupling between peripherals and their code. Reconfiguring previously was a tedious, time-consuming chore that, using static Federated Energy [20], required hardware modifications. Many of these adjustments can now be performed easily in software.

The key challenge is in implementation — developing a Reconfigurable Federated Energy mechanism while keeping overhead (in terms of energy, processing requirements, and cost) low. We discussed the flaws in implementation in Section 2. To implement Reconfigurable Federated Energy we depart philosophically from the original in three ways: (1) we move from a *polling* to *interrupt* based "peripheral ready" signal generated by custom hardware, (2) we enable changing voltage thresholds (a direct proxy for energy stored in a capacitor) using digitally programmable resistor dividers, (3) we use dedicated voltage regulation depending on the peripheral. Flicker's support for reconfigurable federated energy is split between core and peripheral modules. Core modules include the first stage storage capacitor and the hysteresis control that support the microcontroller, as well as power and control lines for the Flicker peripheral interface.

Peripherals each have an individual storage capacitor, which stores the harvested energy that will be used for that peripheral and a programmable **charge controller**, which charges the capacitor only when its input voltage reaches a particular threshold set by the MCU (allowing assignment of priority, peripherals with higher priority start charging at lower voltage thresholds). In contrast to earlier federated energy systems that use static thresholds set in hardware [20], Flicker thresholds can be changed by an application over time as priorities and energy availability change, or baked in at compile time depending on the hardware modules used. Flicker also uses a programmable **interrupt controller**, that signals the MCU with an interrupt whenever the charge level exceeds a set threshold, which is also set in software at runtime. This approach replaces the more energy expensive technique used in [20] that polled the charge level on the ADC continuously.

3.3 Peripheral Ports

Flicker peripherals are designed to be flexible with little energy and computational overhead. As such, the peripheral interface requires common functionality to support energy harvesting, but does not constrain how peripheral-specific functions interact with the core. Instead, each peripheral port provides a wide range of connectivity options to each peripheral (such as GPIO, SPI, I2C, and reference voltages), with most peripherals only using a subset of the available pins. Ideally, enough pins would be provided to support any peripheral on any port. In reality, pins and other hardware resources are often limited. Some Flicker implementations may provide some limited hardware resources on a subset of its peripheral ports, and may not support some complicated peripherals that require an excessive number of control signals. Section 4 describes how we addressed this challenge in our implementation and how, in practice, we are able to support many common radios and sensors.

3.4 Failure-Tolerant Timing

Timekeeping is incredibly important for sensing, security, data provenance, and data utility, especially in the face of intermittent power with undefined (to the runtime) lengths of time between power failures. Failure-tolerant timing also allows for continuous user interface and response, and is a functional enabler for strengthening user privacy. Each core module (the compute core with the MCU) is responsible for providing a timekeeping mechanism that is robust *even in the face of power failures.* A variety of timing techniques can be used with Flicker, including remanence-based SRAM timing [28] which uses the decay characteristics of SRAM memory cells to determine the duration of power loss events. Timing failures with custom circuitry using capacitors with stable thermal properties [21] is more reliable and allows timing longer outages, but with precision reduced the longer the outage.. Powering an ultra-low power real-time clock (RTC) off a small independent capacitor provides even more fine grained timing information, but at increased cost and space. The Flicker framework is compatible with any of these approaches. Our current implementation provides hardware support for all three to give broad application.

3.5 Auto-Detecting Configurations

Hardware changes often require software changes. In order to make batteryless prototyping fast and easy, Flicker harvesters and peripherals contain circuitry that allows the Flicker toolchain to

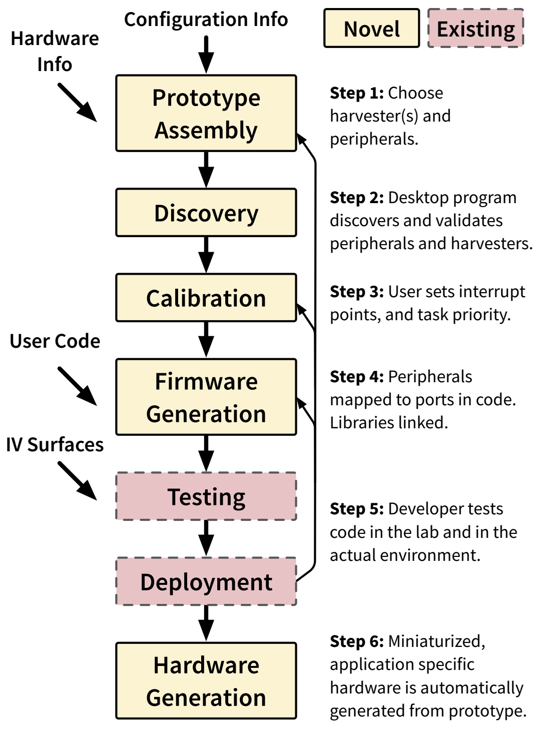
Configuration Info
Novel
Existing
Hardware Info
Prototype Assembly
Step 1: Choose harvester(s) and peripherals.
Discovery
Step 2: Desktop program discovers and validates peripherals and harvesters.
Calibration
Step 3: User sets interrupt points, and task priority.
User Code
Firmware Generation
Step 4: Peripherals mapped to ports in code. Libraries linked.
IV Surfaces
Testing
Step 5: Developer tests code in the lab and in the actual environment.
Deployment
Hardware Generation
Step 6: Miniaturized, application specific hardware is automatically generated from prototype.

Figure 5: Flicker Workflow.

automatically detect which modules are attached to which ports. For simplicity, Flicker uses resistor dividers to identify modules. Each module is identified by a single resistor value, which is measured by a special calibration firmware that also tests a variety of hardware functions. The autodetection process is not particularly energy efficient, and is designed to be done during calibration and testing, and not at runtime. We favor this approach over more sophisticated techniques (like using serial ID chips), to minimize cost and board size.

By autodetecting hardware configurations, many software updates can be made automatically, or by updating a simple port mapping.

3.6 Flicker Workflow

Flicker's modular design lends itself to the workflow shown in Figure 5. A developer initially selects a particular configuration she wants to try out from available peripherals and harvesters, based on her application goals and intuition. She assembles her Flicker sensor by attaching peripherals and energy harvesters. With the core module attached to a programmer, the discovery process detects which peripherals and harvesters are attached, produces a file that describes the configuration, and detects any incompatible connections (for example, a peripheral requiring I2C attached to a port that doesn't support I2C).

The configuration file contains a mapping between peripherals and ports, as well as the default threshold voltages for each peripheral's charge controller and interrupt controller. The developer can update or change this configuration file to suit her needs and her domain knowledge of the deployment environment and tasks that will be executed. During calibration, thresholds are adjusted based on application priorities, developer intuition, and prior testing. The developer's code is then combined with library code and threshold initialization code, compiled, linked, and installed on the device for testing and deployment.

Flicker's harvester interface is compatible with existing debugging tools like the Ekho [19] energy harvesting emulator and the EDB [12] debugger, for in-lab testing with I–V surfaces that are appropriate for the attached harvesters.

This process is meant to be iterative. Testing and deployment often indicate needed changes in the voltage thresholds or even the modules that are used. The developer makes adjustments to their configuration and application code as many times as it takes to produce a configuration that works well. With Flicker, these design iterations, which have traditionally taken days or weeks per iteration, can often be done in minutes. Consider that manufacturing custom printed circuit boards with the same capability as Flicker can take two weeks (standard lead time for batch PCB suppliers like OSH Park) or hundreds of dollars, and then many hours more to assemble sensors by hand, debug hardware and test solder connections, then design custom firmware to manage energy and interface with peripherals. With Flicker, this process is sped up because developers can reuse existing code, can attach the peripherals they need, and will never have to solder.

When testing is complete, the developer may want to take the final step and generate her own custom hardware version of her configuration. Flicker's strength is flexibility and rapid prototyping, but many of its signal traces, connectors, and discovery hardware components increase device size and cost and are often not needed in a finalized device. In order to allow developers to further miniaturize their designs and adapt them to other form factors, the Flicker toolchain also generates a schematic and board layout for a configuration (without unneeded components) that the developer can modify as needed to fit her form-factor of choice.

4 IMPLEMENTATION

We implemented Flicker hardware (shown in Figure 6) and software, in order to 1) evaluate the efficacy of the Flicker approach and 2) provide a set of reference designs to the research community. We developed one core module board with connectors for three peripherals modules and two harvester modules. We also developed tools to ease the prototyping and design process, as well as created runtime libraries allowing for developer controlled adaptation for different energy harvesting scenarios in deployment. In this section, we describe the specifics of our Flicker hardware and software implementation. All hardware, software, and tools, as well as documentation and tutorials on using and extending Flicker will be released at publication time.[4] We plan to make fully assembled boards available for purchase (at cost) at publication time using an online, self-service manufacturer.

In the following sections, we detail implementation decisions balancing the architecture design, usability, and generality requirements — we note specific tradeoffs in translating Flicker to a practical implementation.

4.1 Hardware

The hardware modules, shown in Figure 6 enable a wide range of energy harvesting options, sensing activities and communication channels in order to support a broad range of applications, enabling flexibility, a key goal of Flicker. We plan to expand on this initial set of modules, and anticipate additional hardware contributions from the research community.

Compute Core: The compute core module is centered on a Texas Instruments ultra low power FRAM-enabled MSP430FR5989 microcontroller with 128K of FRAM, 2K of SRAM, and multiple communication and analog ports. This iteration of the core has three peripheral slots, each peripheral slot has an SPI and analog connection, while two have UART connections, and one has an I2C connection. Due to limited MCU resources, not every peripheral slot has I2C or UART. Any peripheral can be attached to any port, but some peripherals will not function correctly on all ports. Our prototype supports only a single I2C peripheral at a time — an artifact that is handled by the calibration stage; if a peripheral is attached to an incompatible port, the compilation process is stopped and the developer informed. The compute core also has a voltage reference, an Abracon AB0805 RTC supplied by a small 10 µF capacitor, and a remanence timekeeper. The latter able to time power outages up to 19 minutes, providing the developer a sense of time. A low profile Tag-Connect programming interface on the PCB allows firmware upload by the developer and calibration routines to be executed.

Universal Peripheral Interface: Each peripheral has a charge and priority controller, an interrupt controller, a storage capacitor, identification circuitry, and a power gating switch. The controllers are implemented with a dual digital potentiometer and network of comparators which gate harvested energy, based against a stable reference voltage supplied by the Compute Board. These potentiometers set the voltage on the capacitor that triggers an *interrupt* to the MCU, and the voltage on the first stage capacitor that triggers *charging* of the peripheral capacitor. Additional circuitry gates the power to the actual peripheral (radio or sensor), controlled by a pin from the MCU, and handles the identification voltage divider.

Each peripheral and harvester has an identification voltage divider (and therefore a set voltage out of the voltage divider) that can be read in discovery mode. The mapping of voltages to individual modules is given as input to the discover stage from a static file. Because of ADC voltage constraints, reference accuracy, and voltage divider noise, the maximum number of peripherals possible (without risk of mis-identification) in the Flicker ecosystem is 128.

Finally, the peripheral interface breaks out control pins, analog pins for direct sensing, and digital pins for communication with peripheral components. This common interface enables a broad array of sensing and communication peripherals as detailed below.

Environmental Sensing: Gathering information about a sensors outdoor environment is a common need for many sensing deployments, including greenhouse sensing, geo-spatial deployments, and

[4]All hardware, software, and tools, as well as documentation and tutorials on using and extending Flicker are open-source and on our website.
https://github.com/PERSISTLab/FlickerPlatform

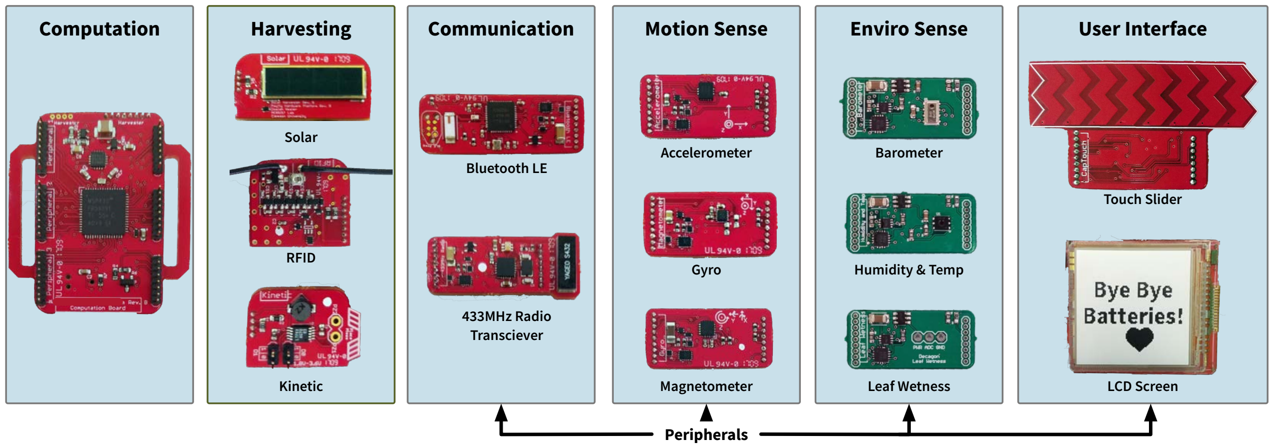

Figure 6: Flicker hardware modules, including a single compute core board, three harvesters, and ten peripherals for sensing, communication, and interacting with users. Peripherals and harvesters attach to the core module, and use federated energy storage with charge and interrupt thresholds to support a wide variety of user tasks.

others. Our current Flicker prototype has three peripherals for sensing information about the environment. A low power NXP MPL115A digital Barometer is used for pressure sensing, it communicates over SPI to the MCU. The barometer operates up to 5.5 V. The dynamic federated energy circuitry can take advantage of this voltage range, independent of the MCU. A Silicon Labs Si7021 Humidity and Temperature sensor provides relative humidity readings over an I2C interface. Finally an analog peripheral is connected to a Decagon Leaf Wetness sensor for gathering coarse information about the moisture and water needs of plants in a greenhouse. This peripheral can be used with any analog sensor by soldering power and signal lines to the headers.

Motion Sensing: Motion sensing has broad application for mobile sensor networks, wearables, infrastructure monitoring, and even manufacturing. We developed three peripherals for motion sensing using components commonly seen in the sensor networks and batteryless sensing communities. The Analog Devices ADXL362 is used for acceleration measurement, connected via the SPI bus. This IC only draws a few nano-amps when sensing at low speeds, meaning that the size of the Accelerometers peripheral capacitor can be set to less than 1 µF for quick charging and increased availability. The Gyro peripheral is equipped with an STElectronics L3GD20H which enables gathering of angular velocity, which can be used in multiple applications in wearables that monitor the motion of the wrist (example: bite counting). The magnetometer peripheral uses the Honeywell HMC5883L, enabling a 3-axis compass functionality over a I2C interface.

Communication: Communication is a necessary part of nearly every sensing device, therefore it is essential for Flicker to support a broad range of communication modalities. We built a low frequency (433MHz) radio transceiver peripheral using the CC1101, a common radio in the WSN community. The Texas Instruments CC1101 uses a small chip antenna, and communicates via SPI to the microcontroller. Bluetooth Low Energy is a popular communication interface between phones, wearables, and sensors. We built a BLE peripheral that supports peripheral and central modes, using the the Nordic nRF51822 System-on-Chip and a small chip antenna. The nRF51822 SoC is programmable, and the BLE peripheral includes a low profile programming port for changing BLE behaviors. Both active radios voltage is regulated to minimize odd RF behaviors from an unstable supply. In addition to the active communication peripherals described, we have also implemented a passive communication scheme, using RFID backscatter based on the UMich Moo[35], allowing ultra low power two way communication with an RFID reader.

User Interface: We implemented user facing peripherals to allow testing and experimentation in the design fiction of intermittent computing. These peripherals make it simpler to implement wearable or embedded devices that interact with a user. We developed two peripherals, a 1.28 x 1.28 inch low power SHARP display, which draws 5 µW to hold a static image, and a capacitive touch sensor with eight buttons in a slider configuration. These two peripherals will enable a rich set of interactions and allow experimentation inside the new design space of intermittent displays, interaction, and computing.

Energy Harvesting: Our current Flicker Compute Core provides two slots for energy harvesting modules for multi source harvesting. Only one of the slot allows for energy sources that also are used as a communication medium, like RFID backscatter described in the previous section. Currently Flicker supports **three** energy harvesting modalities.

The **solar** harvester module is equipped with a 22 mm by 7 mm Ixys solar cell that supplies up to 4.5 V. Charging is accomplished through a Schottky diode on the positive solar output to prevent reverse leakage from the first stage capacitor. The **kinetic** harvester uses a Linear Technology LTC3588 to harvest energy from piezoelectric materials. The IC is used as a low quiescent current rectifier and settable buck boost regulator. The regulated voltage output of the LTC3588 can be set by the developer before deployment using pin headers, as this voltage will depend on the piezoelectric used.

The **RFID** harvester is based on the UMich Moo[35]. The module harvests energy from an ultra high frequency RFID reader such as the Impinj Speedway. It uses a charge pump built with off the shelf components, and allows for tuning pre-deployment using a variable capacitor accessible to the developer. Additionally, the harvester is equipped with circuitry that enables backscatter communication with the reader, which can be initiated by the MCU.

Customization: Flicker uses a standardized hardware template and universal peripheral interface that allows developers to add or adapt their own hardware components to Flicker easily, using CAD software. With the availability of incredibly cheap batch PCB services, designing and building these custom peripherals is accessible to students, hobbyists, and professionals. In addition, users can make use of a Flicker peripheral breakout board (not shown) which allows a developer to breadboard a new peripheral before committing to a PCB design.

Mechanical Design: The mechanical design of the hardware factors into both the *flexibility* (in terms of deployment ability) and *usability* (in terms of ruggedness and comfort) of a platform. We chose to tradeoff size of the platform for ruggedness, by using larger pin connectors for modules instead of smaller but much more frail, board-to-board connectors. We also put cutouts on the compute board (increasing the size) that allow a watchband to be connected to the platform, enabling a quick wearable.

Cost and Size: The final cost and size of a fully assembled prototype varies with the peripherals chosen. An assembled prototype will have a maximum size of 61 mm by 36 mm if equipped to harvest RFID and solar energy, sense acceleration and pitch, and send sensor data with the CC1101 as shown in Figure 1. For the described prototype, we estimate the cost of components and printed circuit boards to be near $200 a piece in a small batch of ten. At scale, we anticipate the cost of prototyping with Flicker to be significantly reduced. This cost in term of time and finance even at small batches is **an order of magnitude lower** than designing and assembling custom hardware when an application changes, or components are found to be defective.

4.2 Software

Flicker includes supporting software and firmware for managing each stage of the prototyping pipeline. These tools are meant to streamline developer effort and save time from design, to runtime, to deployment. These tools support the Flicker workflow described in Figure 5. We describe the implementation details of each piece below.

Discovery and Calibration: A combination of python scripts on the desktop and custom firmware on the Compute Core MSP430 handle the discovery and calibration phases (Step 2 and 3 in Figure 5) of the firmware upload process. When the developer creates their Flicker hardware and initiates the firmware creation process, a special `discovery firmware` is uploaded to the MSP430 on the Compute Core. The `discovery firmware` uses the ADC to read the voltage of a resistive divider on each peripheral and harvester slot to identify the modules mounted on the Computer Core. The voltage of each connected module (harvester and peripheral) is then stored in a predetermined memory location on the MSP430. The MSP430 then goes into a wait mode. On the desktop side, once the wait mode starts, the python script interfaces with the MSP Debug Stack and programmer to download the stored peripheral voltages read by the `discovery firmware`. These voltages are looked up in a preset table, which maps the voltage to the name of the peripheral. The python program outputs configuration information to the developer, and alerts on any incompatibility of peripherals.

After error checking by the toolchain, the configuration file and further prompts from the developer is used to set the voltage thresholds for the interrupts and charging. In this phase `discovery firmware` on the MSP430 writes the non-volatile registers on the digital potentiometers with the values defined in the configuration file (converted from voltage to digital). At this point, the Flicker hardware is configured for the attached peripherals.

Runtime Libraries: We developed runtime libraries for use with peripheral modules, many were adapted from open source code libraries (such as the Accelerometer). These libraries encompass minimum functionality of the components, allowing basic sensing and communication tasks. For the BLE peripheral we have implemented a simple forwarding mechanism over the SPI bus, allowing the MCU to treat the BLE as a radio modem. Currently backscatter is supported in hardware, however runtime libraries have not been ported from the UMich Moo MSP430 code base to the new FRAM MSP430 processor used by Flicker.

In addition to peripheral and harvester runtime libraries, we also have developed control and adaptation libraries for the timekeeper, peripheral energy and priority management function. Programmers can set voltage thresholds for both the interrupt level and the charging threshold using a simple API. They can also write these to non-volatile memory for long term application changes. We envision further runtime library development by ourselves and the community as more applications, peripherals, and harvesting modalities are created for the Flicker ecosystem.

Design Automation: Developers who have prototyped, tested, and even deployed their batteryless sensors using Flicker and need a more permanent, smaller, or easily scalable deployable solution, use the design automation tool to combine peripherals, harvesters, and the MCU to generate the final device. This is implemented with a combination of python and EAGLE CAD scripts that put all peripherals together in a single schematic at the peripheral interface points. We developed a python script that interfaces with EAGLE CAD (a very popular and free PCB design tool), takes the peripherals and harvesters listed by the developer, and the configuration file the developer generated for the calibration phase that defines the voltage thresholds, and creates a single schematic, with proper resistance settings baked in.

5 EVALUATION

In this section we evaluate the overhead and performance of Flicker, and qualitatively evaluate how Flicker simplifies the process of prototyping batteryless, intermittent, energy harvesting sensing devices. Specifically, we quantify the usefulness of dynamically adjusted federated energy, evaluate the overhead of Flicker in terms of energy and user time, illustrate how Flicker simplifies prototyping with a real world use case, and finally discuss the usability

Table 1: Flicker overhead

Parameter	Value
Timekeeper Charge Energy	39 µJ
Timekeeper Startup Time	1.1 s
Volatile Threshold Write Time	197.7 µs
Volatile Threshold Write Energy	69.3 nJ
Peripheral Voltage Range	1.7 to 5.5 V
Peripheral Current Range	0.0 to 40 mA
Compute Board Quiescent Current	5.77 µA
Peripheral Quiescent Current	4.47 µA

perspectives of prototyping with Flicker. In our experiments and experience, we have found that using Flicker dramatically shortens the time to deploying a usable prototype, and enables use cases not possible with current hardware. All with low overhead.

5.1 Overhead

Because of Flicker's reliance on harvesting energy to power all operations, energy efficiency must be high. Table 1 shows the overhead of specific parts of Flicker. This table shows that Flicker trades off some energy-efficiency for flexibility. For example, the steady state quiescent current costs of the peripheral stems from the charge management and interrupt circuitry of dynamic UFoP. In a static UFoP implementation most of this cost would dissapear. However, the overhead is manageable with the current set of energy harvesters. One source of overhead comes from managing the timekeeper, specifically the first time charging the small reservoir capacitor that maintains timekeeper state when the Compute Board is off. Longer off-timekeeping requires larger capacitors, and therefore more charge time and energy. We chose a 10 µF ceramic as a reasonable tradeoff between charge time, and energy cost. Another source of overhead comes from setting the voltage thresholds for charging, interrupts of peripherals. Writing the volatile thresholds must be done every time the MCU returns from a power failure, or if runtime adaptation happens in deployment. The quiescent current in steady-state of the processor, and control components like the timekeeper, and peripheral controllers, is also quite low as shown in the table.

5.2 Performance

In this section we evaluate and discuss different performance metrics pertaining to Flicker.

RF Harvester: The performance of the RF Harvester is comparable to the UMich Moo harvesting performance that it is based on. We connected the Flicker RF harvester to an Ekho device, and recorded the energy harvesting conditions for twenty seconds when placed within one centimeter of a small antenna, connected to an Impinj Speedway Revolution RFID reader outputting at high transmit power. We then captured the maximum power point (MPP), the maximum voltage, and the maximum current, of the energy harvesting environment. The Flicker RF Harvester's MPP was recorded at 1.4 mW, with a maximum voltage of 6.3 V, and maximum current of 1.0 mA

Table 2: System Usability Survey (SUS) scores vs. Flicker

Interface Type	SUS Score
Cell Phone	66.55
Customer Premise Equipment	71.60
Graphical User Interface	75.24
Interactive Voice Response Systems	73.84
Web Pages and Applications	68.05
Flicker Platform	**84.9**

This power level is comparable to the Moo and WISP platforms, enabling a broad range of RFID powered applications. One major difference between the current harvester and the Moo, is that our harvester only has a two layer PCB instead of four, hurting overall RF performance. In additions, antenna tuning and careful design could increase harvesting ability. We expect future revisions to continue to improve performance, however, as is, the RF harvester is comparable to the Moo and useful for RFID powered applications.

Timekeeping: Timekeeping using the ultra low power RTC is one of the critical parts of Flicker. Without an accurate clock, sensor data could be forwarded that is not relevant, and tasks may be executed that are superfluous. We chose a 10 µF ceramic capacitor as the reservoir capacitor, as this size takes 19 minutes and 40 seconds to discharge enough that the RTC fails, it's memory resets, and time is lost. If the capacitor discharges and the RTC resets, on the next power up, the RTC takes 1.1 seconds to wakeup and recharge. If the energy harvesting environment has very little available energy and cannot support the energy requirements of the clock, then Flicker could potentially not get past the initialization stage. However, this is easily overcome in software; once the Compute Board turns on, programs can sleep until the RTC sends the ready signal on it's I/O line, at which point the program can resume operation.

5.3 User Study

We evaluated the usability of Flicker on 19 participant drawn from a junior-level, university Computer Operating Systems course[5]. **We had the 19 students each participate in a half hour session building multiple devices with Flicker. In all, students built 76 devices and spent 9.5 hours working with the platform.** The participants rated the platform as having **excellent usability** according to results from the industry standard System Usability Survey [6] participants completed. The results of this survey are shown in Table 2.

5.3.1 *Methodology:*

Participants were recruited from an undergraduate, junior-level university Computer Operating Systems course consisting of Computer Science and Computer Engineering students. Each participant was asked to fill out an entry survey where students self rated their competency in computer engineering, computer science, and embedded systems, then described their previous experience with platforms like Arduino[1].

They were then given two pieces of documentation 1) a four page instructions handout on the Internet-of-Things and the promise of batteryless operation, including a brief overview of the motivation, applications, and difficulties in deploying and prototyping

[5]This study was approved by our Institutional Review Board.

these devices, 2) a three page platform handout describing and displaying the Flicker platform, including the individual modules and their usage, as well as basic instructions on how to construct a device. Participants were then given Flicker hardware including the computation board, the Kinetic, RFID, and Solar harvesters, BLE and CC1101 radio, the motion sensors, and the leaf wetness sensor. Participants then built each of four devices (two greenhouse monitoring devices, a fitness wearable, and a earth science enabling micro satellite) described in the instruction handout by assembling devices using the Flicker modules and Flicker main compute board, described their justified their selection decisions to the study supervisor.

Finally the participants took an exit survey capturing their experience. The exit survey contained the System Usability Survey (SUS)[6], a Likert scale ten question survey administered to users for measuring the perceived ease of use (usability and learnability) of software, hardware, phones, wearables, and websites SUS is a well tested, standard method in industry and academia for evaluating systems, which provides a quantitative way to demonstrate the usability of Flicker. The exit survey also asked questions about prior knowledge, future interest with the platform, and enjoyment or distaste of the experience.

5.3.2 Sample: Our 19 participants were either juniors (60%) or seniors, with 2 to 5 years of formal computing education, and 2 to 10 years of total programming experience. Participants self-rated programming abilities, and knowledge of systems and computer hardware as average or above average when compared to other students at their university. Participants nearly uniformly rated their knowledge of embedded platforms as slightly below average compared to other students and developers in industry. We note that our sample size of 19 was well above the stable size of five participants, and represents a core community (mid level computer engineering and computer science students) we would like to engage with the Flicker platform.

5.3.3 Results: Tabulating and scoring the Flicker prototyping SUS surveys for each participant gave the mean SUS score of 84.9, with the median score at 87.5. SUS literature [4] states that a score of 70 is considered average, with higher scores meaning higher usability. Each participant scored Flicker above average, with most in the "excellent" usability category. Flicker SUS scores are shown in relation to other interface types in Table 2. This table shows the interface types along with their average SUS scores, the average is derived from years of surveys, and a category had to have at least fifty surveys to be put in the table [4]. The SUS results show that the Flicker hardware platform has usability demonstrably well above average for prototyping batteryless Internet-of-Things devices.

In addition to the the SUS, participants were asked in the entry survey to discuss if they would use the Flicker platform in the future and in what contexts. Nearly 95% of participants agreed or strongly agreed that Flicker could be used to to create devices for many different applications. Nearly 90% or participants agreed or strongly agreed they would use Flicker on a new IoT project if it was available. The same percentage was interested in learning more about the platform and the context in the future. Every participant agreed or strongly agreed that Flicker devices could be deployed in a real environment for *short-term* use. A majority (58%) said the same about *long-term* use in real environments. A plurality (32%) agreed that Flicker devices could be deployed in a safety-critical application.

5.3.4 Caveats and Discussion: Our user study investigated the **usability** and **acceptability** of the core part of the Flicker platform—rapidly prototyping batteryless IoT devices by hand—with one of the major expected user groups (students) of these devices. However, the study did not look at the usability of different parts of the Flicker workflow and toolchain, including configuring the hardware peripherals, and writing software that runs on the device, nor at quantitative measures such as application development time and coding overhead, nor did this study compare to other prototyping platforms. We anticipate future work will attempt to fill this gap, we note that as this platform is the first of its kind, it is difficult to fairly compare to systems like Arduino or mBed. While this study is by no means comprehensive, it demonstrates that the hardware devices themselves are usable, and enable a broad range of applications.

5.4 Case Study

In this section we illustrate the new use cases and applications that Flicker allows for batteryless intermittently powered devices. Often, a single source of energy is not enough to power the tasks that a developer wants to run for a given application. No current platform supports multiple harvesters, so developers must make invasive hardware changes to the platform to integrate a new harvesting modality[18]. Flicker supports up to two harvesters at once, allowing developers to iterate their tasks and programs through different application scenarios.

Application: We used Flicker to prototype an application for greenhouse monitoring at the plant bed level, for the purpose of water conservation. The dual harvesting sensor reads the leaf wetness and barometric pressure opportunistically. The sensor harvests solar, and RFID energy. The small solar panel does not generate enough energy for expensive leaf wetness readings, but does generate enough energy for intermittent barometeric pressure readings, if thresholds are set correctly. Because of the low energy requirement, the barometric peripheral charge threshold is set to the lowest setting, and the interrupt threshold set just above it. The only time enough energy is available to gather wetness readings is when the RFID reader is used to power the sensor, therefore, the leaf wetness interrupt threshold is set to the highest setting so that enough energy is available to complete the task when the Compute Board is interrupted. In this greenhouse monitoring application, the RFID reader sits on the overhead mechanical watering arm that moves across the plant beds a few times a day. This means the only time leaf wetness readings are needed, is when an RFID reader is present to gather those readings. For our application, we simulated the overhead mechanical watering arm movement in the lab using an RFID Reader. The voltage trace of the application running is shown in Figure 7.

This type of application would be very difficult to design, build, test, and deploy with conventional sensing platforms. Setting different trip points pre-deployment for the two peripherals is impossible with the state of the art, as is using multiple harvesters. By allowing

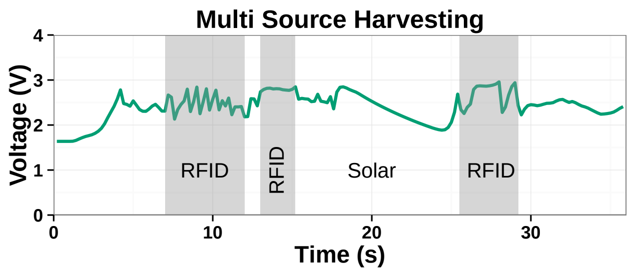

Figure 7: Greenhouse monitoring prototyping case study, capacitor voltage trace shown.

quick prototyping with common peripherals, and adaptation to multiple energy environments, a usable sensor can be deployed.

6 RELATED WORK

Flicker is the first reconfigurable hardware platform designed from the ground up for tiny, batteryless, energy harvesting, intermittently powered sensors. While many platforms have allowed some measure of reconfigurability, none support the constraints imposed by an intermittent power supply requiring careful and direct management of multiple small energy stores and timekeeping. In the sensing community there have been a wide range of platforms, reconfigurable or not, that Flicker builds on. We describe platforms, energy management techniques, and related work from the intermittent computing field and place Flicker in the literature.

Reconfigurable Platforms: In both the commercial and research communities, there are many reconfigurable platforms for sensing, and support of the Internet-of-Things. Epic[15] was one of the first sensing platforms that enabled a degree of freedom in assembling application specific sensor node. Epic required hardware expertise, and CAD tools to generate the final node, however, the core computation, wireless, storage, and programming interface were all standard. TandemStack[32] and other platforms[23] provide either a common interconnect, modular plug and play hardware, or FPGA based modules for faster prototyping and development of application specific sensing devices Recent platform advances have focused on reconfiguring part of the analog sensing component[30] instead of the full device, or focus on ultra low power interconnects for millimeter scale sensing[26]. Commercial platforms like Arduino support the concept of daughter boards, allowing for easy prototyping for the purpose of learning. Other platforms such as the Bosch IoT XDK[5], and EnOcean[16] claim to support Internet-of-Things applications, sometimes without batteries.

Each of these platforms falls short of providing a comprehensive, intentionally designed platform for batteryless, intermittently powered sensor prototyping in key ways. First, none of the platforms allow using different energy harvester, or multiple energy harvesters at the same time. This is critical because of the broad range of applications. Secondly, no platform enables reconfiguration of energy management, charging, and interrupt priority per peripheral. Without this, platform performance suffers from lower energy harvesting efficiency, decreased availability, higher failure rates[20]. Finally, each of these platforms ignores the critical need for timekeeping through power failures. Each of these shortcomings is addressed by Flicker.

Batteryless Platforms: Flicker is inspired by the United Federation of Peripherals (UFoP)[20]. Flicker generalizes the approach in UFoP by allowing changes to charge points and interrupt points, enabling retasking, and dynamic priority, as well as reconfigurability of the hardware platform.

Batteryless platforms have been built that harvest energy from an RFID reader[31, 35], indoor solar[25], thermoelectric energy[9], airflow[34], and power outlets[14]. These single application, single harvester platforms are not extensible or reconfigurable, do not consider timekeeping as a first class priority, and do not separate energy concerns to reduce failures. Energy-Harvesting Active Networked Tags (EnHANTs)[25] can be attached to objects that are traditionally not networked, such as books. The prototypes harvest indoor light energy using custom organic solar cells. Campbell et al.[8] proposed an architecture for energy harvesting in buildings based on event detection, but only allowed for a single energy harvester (indoor solar) and did not develop a prototyping platform.

Unlike previous approaches and platforms mentioned, Flicker provides an all in one plug and play solution to batteryless prototyping that includes energy management, timekeeping, and hardware design, in a modular, quickly reconfigurable, extensible platform. Flicker can be applied to a huge range of future and present applications, energy harvesting modalities, and sensing tasks.

Energy Harvesting: Previous work has characterized kinetic energy harvesting[17], developed techniques for harvesting RFID and Solar energy[18], and even harvesting solar, thermal, and vibration energy[3]. Flicker can harvest from multiple sources, but does so in a naive manner, giving priority to the harvester with most voltage potential. We view these related works as complementary to Flicker, and hope to integrate these novel multi source energy harvesting approaches into future versions of the platform.

Intermittent Computing: Other related work comes from efforts to simplify the programming, testing, and evaluation of batteryless, intermittently powered devices . Recently, tools for debugging (EDB)[12] batteryless programs, and emulating energy harvesting environments have been created (Ekho)[19]. These simplify development and allow rigorous in-lab testing. Other work has tried to simplify the programming and task management for intermittent computing[7, 24, 29]. We view this work as complementary to our own, in fact, each of these techniques could be immediately used with Flicker.

7 DISCUSSION AND FUTURE WORK

Flicker enables many use cases and new applications that were previously impossible without deep hardware knowledge, and large investments of time and money. Flicker While powerful and complete, there are many areas we envision where Flicker could be improved and augmented. In this section we discuss current limitations and tradeoffs of Flicker, and describe some potential avenues for future research.

Balancing Modularity: Flicker currently offers a core of twelve modules that can serve many applications. Additionally, Flicker makes it easy to add new peripherals, or modify existing peripherals. This modularity can be detrimental. Flicker is a step towards building a community around batteryless sensors; part of a community is being able to easily share, reuse, and re adapt code and hardware. If many different generations or versions of hardware are allowed to propagate, or closed source hardware is mixed in, this will only hurt the community. We hope to address this by introducing standard APIs, and solidifying around a few standard modules (like the twelve currently in existence), as well as providing an on-line resource for code examples, best practices, and ordering hardware.

Tuning Thresholds: Just as in previous work on Federating Energy storage for tiny batteryless sensors (FedEnergy)[20], figuring out the best voltage thresholds is difficult and imprecise. Energy environments change, and application requirements vary; for Flicker this can become more complex, as designers now have a choice of harvesters and peripherals. This ability to choose and set voltage thresholds for charging and peripheral wake up interrupts without a hardware revision gives maximum flexibility to the user, but choosing the best set of thresholds still requires careful consideration.

The added flexibility comes at a energy and space cost. Peripherals on Flicker are larger in size than peripherals using FedEnergy, because of the lower component costs of using static, hardware defined thresholds. This also means that static, FedEnergy will always be lower energy overhead than Flicker. However, the flexibility offered by Flicker is worth it, especially for developers not capable of redesigning hardware. Additionally, Flicker software tools can generate a static FedEnergy version from the Flicker version, for long term deployments and sensor building at scale.

Runtime Adaptation: Dynamic Federated Energy allows changing task charging and interrupt thresholds at runtime, enabling adaptation of tasks to the energy environment in-situ. However, as discussed, tuning is hard when not much is known about future energy harvesting conditions or even task schedules. If a simple metric could be developed for deciding when to adapt, and adaptation cost could be lowered by advances in low power memory, runtime adaptation could become incredibly useful.

Toolchain Integration: Flicker can be used with the two main tools for working with intermittently powered devices, Ekho, and the Energy Interference Free Debugger (EDB), without any hardware changes. Ekho emulates energy harvesting conditions, so an Ekho device can be plugged directly into one of the harvesting ports of the Compute Board, just as with any other device. Two Ekho devices are required to emulate multi source harvesting. With EDB, there are some constraints. Since EDB needs access to a few I/O pins, as well as the capacitor voltage for full functionality (breakpoints, watchpoints, and energy guards), an entire peripheral slot must be dedicated to using EDB. In the future, we hope to create a dedicated port for using EDB so that all three peripheral ports can be used when debugging.

Hardware improvements: Now that the baseline hardware, software, and firmware are implemented in the Flicker ecosystem, other harvesting modalities, sensors, and application can be imagined. In the short-term, we plan to support eInk displays that can hold their image even when the device has lost power. We are currently implementing NFC and thermal energy harvesters to support more environments and applications. In order to support long-term deployments with Flicker hardware, we plan to support mass storage peripherals using microSD cards or large FRAM memory ICs.

Community Building: Flicker is an open source, open hardware initiative that seeks to empower sensing experts and non experts alike to build comprehensive batteryless sensing applications for the vision of the Internet-of-Things. Important to our effort is developing materials that help our community, and generate We anticipate immediate future work centered around designing documentation, writing tutorials and new libraries, formalizing the batteryless sensing toolchain, and engaging in community building and outreach.

8 CONCLUSIONS

Batteryless, energy harvesting, intermittently powered sensors are an emerging class of device that defines and enables the vision of the Internet-of-Things. Despite the importance of these devices, current sensing platforms are application specific, lack recent advances in energy management and timekeeping, and are limited in flexibility and usability.

In this paper we have presented Flicker[6], an open-source, open-hardware prototyping platform intentionally built for batteryless, energy harvesting, intermittently powered sensing. With Flicker, developers and system designers can quickly prototype devices for new applications in many fields, with many energy harvesting modalities. Flicker is comprised of a reconfigurable hardware platform that lets designers replace sensor, harvester, and communication peripherals at will, without hardware experience or design abilities. Flicker hardware manages harvested energy in a novel, and dynamic way, allowing for easy adjustment of charging thresholds and interrupt routines depending on application, energy harvester, peripheral, or any other developer constraint. Flicker also includes software tools to streamline the prototyping process, all the way through to deployment.

Flicker is extensible by platform and software developers who want to add new sensors, new runtime techniques, or even new operating systems. We implemented Flicker in a small form factor for multi source harvesting from RFID, Solar, and Kinetic energy sources. Our Flicker implementation supports a broad range of environmental and motion sensors, and communicates through Bluetooth LE, low frequency radios, or RFID backscatter. We evaluated the usability of Flicker in a user study with 19 participants, and found it had above average or excellent usability according to the well known System Usability Survey (SUS). We believe Flicker will support the emerging batteryless sensing community and bring about exciting new applications, and research directions.

ACKNOWLEDGMENTS

The authors would like to thank: Diana Zhang and Tamara Ortega for initial work on Flicker; Kevin Storer for helpful input on designing the user study; our shepherd, Kay Römer and the anonymous reviewers, for helpful comments. This research is based upon work supported by the National Science Foundation under grant

[6]Find the Flicker platform release at https://github.com/PERSISTLab/FlickerPlatform

CNS-1453607. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation

REFERENCES

[1] Arduino. 2016. Arduino: Open-source electronic prototyping platform. https://www.arduino.cc/. (October 2016).

[2] Domenico Balsamo, Alex S Weddell, Geoff V Merrett, Bashir M Al-Hashimi, Davide Brunelli, and Luca Benini. 2015. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters* 7, 1 (2015), 15–18.

[3] Saurav Bandyopadhyay and Anantha P Chandrakasan. 2012. Platform architecture for solar, thermal, and vibration energy combining with MPPT and single inductor. *IEEE Journal of Solid-State Circuits* 47, 9 (2012), 2199–2215.

[4] Aaron Bangor, Philip T. Kortum, and James T. Miller. 2008. An Empirical Evaluation of the System Usability Scale. *International Journal of HumanÂĂŞComputer Interaction* 24, 6 (2008), 574–594. DOI: https://doi.org/10.1080/10447310802205776 arXiv:http://dx.doi.org/10.1080/10447310802205776

[5] Bosch. 2016. XDK Cross Domain Development Kit. http://xdk.bosch-connectivity.com/. (October 2016).

[6] John Brooke and others. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 4–7.

[7] M. Buettner, B. Greenstein, and D. Wetherall. 2011. Dewdrop: An Energy-Aware Runtime for Computational RFID. In *Proc. 8th USENIX Conf. Networked Systems Design and Implementation (NSDI'11)*. ACM, Boston, MA, USA, 197–210.

[8] Bradford Campbell and Prabal Dutta. 2014. An energy-harvesting sensor architecture and toolkit for building monitoring and event detection. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. ACM, 100–109.

[9] Bradford Campbell, Branden Ghena, and Prabal Dutta. 2014. Energy-harvesting Thermoelectric Sensing for Unobtrusive Water and Appliance Metering. In *Proceedings of the 2Nd International Workshop on Energy Neutral Sensing Systems (ENSsys '14)*. ACM, New York, NY, USA, 7–12. DOI: https://doi.org/10.1145/2675683.2675692

[10] Gregory Chen, Hassan Ghaed, Razi M. Haque, Michael Wieckowski, Yejoong Kim, Gyouho Kim, David Fick, Daeyeon Kim, Mingoo Seok, Kensall Wise, David Blaauw, and Dennis Sylvester. 2011. A Cubic-Millimeter Energy-Autonomous Wireless Intraocular Pressure Monitor. *IEEE International Solid-State Circuits Conference* (2011).

[11] Alexei Colin and Brandon 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 2016)*. ACM, New York, NY, USA, 514–530. DOI: https://doi.org/10.1145/2983990.2983995

[12] Alexei Colin, Alanson P. Sample, and Brandon Lucia. 2015. Energy-interference-free System and Toolchain Support for Energy-harvesting Devices. In *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '15)*. IEEE Press, Piscataway, NJ, USA, 35–36. http://dl.acm.org/citation.cfm?id=2830689.2830695

[13] Samuel DeBruin, Bradford Campbell, and Prabal Dutta. 2013. Monjolo: An Energy-harvesting Energy Meter Architecture. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys '13)*. ACM, New York, NY, USA, Article 18, 14 pages. DOI: https://doi.org/10.1145/2517351.2517363

[14] Samuel DeBruin, Branden Ghena, Ye-Sheng Kuo, and Prabal Dutta. 2015. Powerblade: A low-profile, true-power, plug-through energy meter. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 17–29.

[15] Prabal Dutta, Jay Taneja, Jaein Jeong, Xiaofan Jiang, and David Culler. 2008. A building block approach to sensornet systems. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 267–280.

[16] EnOcean. 2016. EnOcean: Self Powered IoT. https://www.enocean.com/en/. (October 2016).

[17] Maria Gorlatova, John Sarik, Guy Grebla, Mina Cong, Ioannis Kymissis, and Gil Zussman. 2014. Movers and shakers: Kinetic energy harvesting for the internet of things. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 42. ACM, 407–419.

[18] Jeremy Gummeson, Shane S Clark, Kevin Fu, and Deepak Ganesan. 2010. On the limits of effective hybrid micro-energy harvesting on mobile CRFID sensors. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 195–208.

[19] Josiah Hester, Timothy Scott, and Jacob Sorber. 2014. Ekho: Realistic and Repeatable Experimentation for Tiny Energy-Harvesting Sensors. In *Proc. 12th ACM Conf. Embedded Network Sensor Systems (SenSys'14)*. ACM, Memphis, TN, USA, 1–15.

[20] Josiah Hester, Lanny Sitanayah, and Jacob Sorber. 2015. Tragedy of the Coulombs: Federating Energy Storage for Tiny, Intermittently-Powered Sensors. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys '15)*. ACM, New York, NY, USA, 5–16. DOI: https://doi.org/10.1145/2809695.2809707

[21] Josiah Hester, Nicole Tobias, Amir Rahmati, Lanny Sitanayah, Daniel Holcomb, Kevin Fu, Wayne P Burleson, and Jacob Sorber. 2016. Persistent Clocks for Batteryless Sensing Devices. *ACM Transactions on Embedded Computing Systems (TECS)* 15, 4 (2016).

[22] Texas Instruments. MSP430FRxx FRAM Microcontrollers. http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/ultra-low_power/msp430frxx_fram/overview.page. (????). Accessed: 2015-10-13.

[23] A. E. Kouche, H. S. Hassanein, and K. Obaia. 2014. WSN platform Plug-and-Play (PnP) customization. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*. 1–6. DOI: https://doi.org/10.1109/ISSNIP.2014.6827642

[24] Brandon Lucia and Benjamin 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 '15)*. ACM, New York, NY, USA, 575–585. DOI: https://doi.org/10.1145/2737924.2737978

[25] Robert Margolies, Maria Gorlatova, John Sarik, Gerald Stanje, Jianxun Zhu, Paul Miller, Marcin Szczodrak, Baradwaj Vigraham, Luca Carloni, Peter Kinget, and others. 2015. Energy-Harvesting Active Networked Tags (EnHANTs): Prototyping and Experimentation. *ACM Transactions on Sensor Networks (TOSN)* 11, 4 (2015), 62.

[26] Pat Pannuto, Yoonmyung Lee, Ye-Sheng Kuo, ZhiYoong Foo, Benjamin Kempke, Gyouho Kim, Ronald G. Dreslinski, David Blaauw, and Prabal Dutta. 2015. MBus: An Ultra-low Power Interconnect Bus for Next Generation Nanopower Systems. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture (ISCA '15)*. ACM, New York, NY, USA, 629–641. DOI: https://doi.org/10.1145/2749469.2750376

[27] Joseph Polastre, Robert Szewczyk, Alan Mainwaring, David Culler, and John Anderson. 2004. Analysis of wireless sensor networks for habitat monitoring. In *Wireless sensor networks*. Springer, 399–423.

[28] Amir Rahmati, Mastooreh Salajegheh, Dan Holcomb, Jacob Sorber, Wayne P Burleson, and Kevin Fu. 2012. TARDIS: Time and remanence decay in SRAM to implement secure protocols on embedded devices without clocks. In *Proceedings of the 21st USENIX conference on Security symposium*. USENIX Association, 36–36.

[29] Ben Ransford, Jacob Sorber, and Kevin Fu. 2011. Mementos: System Support for Long-Running Computation on RFID-Scale Devices. In *Proc. 16th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS'11)*. ACM, Newport Beach, CA, USA, 159–170.

[30] Brandon Rumberg, David W Graham, Spencer Clites, Brandon M Kelly, Mir Mohammad Navidi, Alex Dilello, and Vinod Kulathumani. 2015. RAMP: accelerating wireless sensor hardware design with a reconfigurable analog/mixed-signal platform. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*. ACM, 47–58.

[31] Alanson P Sample, Daniel J Yeager, Pauline S Powledge, Alexander V Mamishev, and Joshua R Smith. 2008. Design of an RFID-based battery-free programmable sensing platform. *IEEE Transactions on Instrumentation and Measurement* 57, 11 (2008), 2608–2615.

[32] Oliver Stecklina, Dieter Genschow, and Christian Goltz. 2012. TandemStack-A Flexible and Customizable Sensor Node Platform for Low Power Applications.. In *SENSORNETS*. 65–72.

[33] Ivan Stoianov, Lama Nachman, Sam Madden, and Timur Tokmouline. 2007. PIPENET: A wireless sensor network for pipeline monitoring. In *2007 6th International Symposium on Information Processing in Sensor Networks*. IEEE, 264–273.

[34] Tianyu Xiang, Zicheng Chi, Feng Li, Jun Luo, Lihua Tang, Liya Zhao, and Yaowen Yang. 2013. Powering indoor sensing with airflows: a trinity of energy harvesting, synchronous duty-cycling, and sensing. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM, 16.

[35] Hong Zhang, Jeremy Gummeson, Benjamin Ransford, and Kevin Fu. 2011. Moo: A batteryless computational RFID and sensing platform. *Department of Computer Science, University of Massachusetts Amherst., Tech. Rep* (2011).