Animal Biotelemetry

METHODOLOGY

Open Access

# An open-source platform for sub-g, sub-μA data loggers

Geoffrey M. Brown[1*], Jiawei Chen[2], Adam Fudickar[3] and Alex E. Jahn[4]

**Abstract**

**Background** Rapid improvements in inexpensive, low-power, movement and environmental sensors have sparked a revolution in animal behavior research by enabling the creation of data loggers (henceforth, tags) that can capture fine-grained behavioral data over many months. Nevertheless, development of tags that are suitable for use with small species, for example, birds under 25 g, remains challenging because of the extreme mass (under 1g) and power (average current under 1μA) constraints. These constraints dictate that a tag should carry exactly the sensors required for a given experiment and the data collection protocol should be specialized to the experiment. Furthermore, it can be extremely challenging to design hardware and software to achieve the energy efficiency required for long tag life.

**Results** We present an activity monitor, BitTag, that can continuously collect activity data for 4–12 months at 0.5–0.8g, depending upon battery choice, and which has been used to collect more than 500,000 h of data in a variety of experiments. The BitTag architecture provides a general platform to support the development and deployment of custom sub-g tags. This platform consists of a flexible tag architecture, software for both tags and host computers, and hardware to provide the host/tag interface necessary for preparing tags for "flight" and for accessing tag data "post-flight". We demonstrate how the BitTag platform can be extended to quickly develop novel tags with other sensors while satisfying the 1g/1μA mass and power requirements through the design of a novel barometric pressure sensing tag that can collect pressure and temperature data every 60s for a year with mass under 0.6g.

**Keywords** Biologging, Dataloggers, Accelerometer, Pressure sensor, Songbird activity

## Background

A wide variety of sensors have been used in studying animal behavior including accelerometers, barometers, temperature sensors [1–5], and various devices used for geolocation [6–8]. [9–11] provide surveys of the available technologies and their use in animal behavior research. While the tag designs we present in this paper utilize accelerometers to measure motion and barometers to measure changes in altitude, the system we present and its corresponding design processes are easily extended to support both other types of sensors and other data collection protocols.

The key attribute of our tag designs is the combination of longevity with low mass. It is difficult to find published specifications for sub-gram archival tags—Table 1 provides details of several archival tags with mass under 2 g including the two tag designs discussed in this paper—BitTag and PresTag. The closest comparable design in terms of mass is the pressure sensor tag developed at Cornell University [5] which, as we will discuss, requires significantly more power than either of the tags we

*Correspondence:
Geoffrey M. Brown
geobrown@indiana.edu
[1] Department of Computer Science, Indiana University, 700 N Woodlawn Avenue, Bloomington, IN 47408, United States
[2] Robotics Department, University of Michigan, 2505 Hayward Street, Ann Arbor, MI 48109-2106, United States
[3] Department of Biology, Indiana University, 1001 East Third Street, Bloomington, IN 47405-7005, USA
[4] Environmental Resilience Institute and Department of Biology, Indiana University, 1001 East Third Street, Bloomington, IN 47405-7005, USA

**Table 1** Example archival tags under 2 g

| Manufacturer | Mass (g) | Memory | Sensors | Reference |
|---|---|---|---|---|
| Cornell University | 0.33-0.67[a] | 40KB | P | [5] |
| Swiss Ornithological Institute | 1.3 | Unknown | ALMPT | [12] |
| UC Berkeley | 1.5-2.5 | 512KB | A | [13] |
| Lund University | 1.2 | Unknown | AL | [14, 15] |
| Wild Byte Technologies | 1.7[b] | MicroSD | AMPT | [16] |
| BitTag | 0.5-0.8 | 240KB | A | This article |
| PresTag | 0.5-0.8[c] | 4MB | P | This article |

*A* accelerometer, *L* light, *M* magnetometer, *P* pressure, *T* temperature

[a] Estimated using same batteries as BitTag

[b] Not including battery

[c] Estimated

present. We excluded tags with only a light sensor from our discussion.

Accelerometers have been especially useful in studies relating to migration, its timing, and flight behavior of small birds [4, 17–21]. Accelerometers can be used to determine when an animal is active and, with additional signal processing, the nature of its activity (e.g., resting, flapping, or gliding). Pressure sensors (altimeters) have been used for studying the flight behavior of animals and for discerning group behavior [22–25]. Remarkably, pressure sensor data can be combined with global weather data to provide relatively accurate global positioning. [26]

It seems natural that the additional burden of carrying a tag could be deleterious to animals in general and birds especially. Although 3–5% has been used as a “rule-of-thumb” for maximum tag mass, it is far from clear what limit is necessary [27]. There have been a number of attempts to quantify the effect of additional mass on the survival of the experimental subjects [28–32].

One might reasonably ask how many species can be studied with a tag of a given mass. There have been several studies that analyzed the available data on the weight of various species [33–35]. The essential takeaway is that the vast majority of bird species (more than 75% of 6209 studied) are in excess of 16 g with a median mass of 37.6 g. Thus, at 3% body weight, a 0.48 g tag can be used with 75% of all bird species.

Because of the engineering effort required to create a novel tag [36], it is tempting to combine a variety of sensors in a single design and indeed such designs have found great utility with larger bird species [37]. However, ultimately biologists are engaged in a battle of weight—improvements in miniaturization have led to smaller tags, but that in turn has led to their use on smaller animals [38]. Rather than attempt to create a multi-purpose tag with a variety of sensors, we address the engineering difficulties inherent in creating ultralight, ultra-low energy tags that can be specialized to their application.

## Methods

The physical design of tags is just a small part of the effort required to create a tag system. To be useful, the tags require a support “system” that enables tag configuration, battery charging, and data recovery. This system includes both software (tag firmware, configuration software, data processing), and support hardware. The BitTag system architecture is designed to be extensible in order to enable the rapid development of tags with other sensors or data collection protocols. The keys to this extensibility are a flexible software architecture that enables adding new tags to the system with minimal software development and a hardware architecture that supports rapid prototyping with accurate power measurement.

In order to demonstrate this extensibility, we present the design of a prototype barometric pressure sensor tag— PresTag—which has 100 times more memory and uses less than 1/10 the power of a recently published design. [5] PresTag can collect altitude (pressure/temperature) data every 60 s for over a year with a 0.23 g battery and an expected mass of under 0.6 g.[1] The large storage and low power requirements of PresTag make it ideal for use with a recently developed geolocation algorithm that utilized global weather data to estimation location using only pressure measurements [26].

In this article, we describe the BitTag system including key architectural decisions that make it easy to create novel tags with different sensors and data collection protocols. We also discuss the tools and techniques we use to optimize their energy usage.

Achieving sub-μA currents requires careful hardware and software design, and the measurement of the dynamic power consumption of actual tags. Very small errors in either software or hardware can completely destroy the energy budget; for example, with a misconfigured pin we have observed excess current consumption of up to 100 μA (our design budget is under 1 μA).[2] In the tags we design, all of the required energy is carried in the form of batteries—for sub-g tags, energy harvesting through solar or other means would add significant additional complexity and mass.

As an example tag, consider BitTag, illustrated in Fig. 1. BitTag utilizes a low-power accelerometer to detect when an animal is active (e.g., when the acceleration due to

[1] While we have prototyped PresTag and designed the tag PCB, the current semiconductor shortage prevents building the final design. The mass estimate is based upon a design with exactly the same outline as BitTag.

[2] As an example, the STM32 pins include configurable (nominally) 40 kΩ pull-up resistors which, at 2.5 V, draw 62 μA.

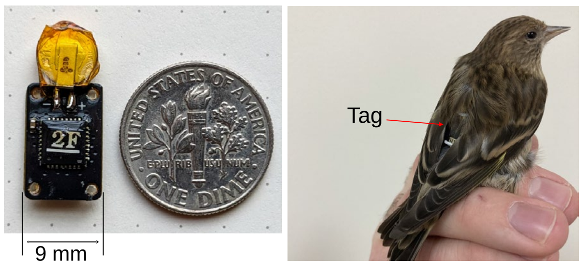


**Fig. 1** BitTag. Left image shows the top side of a BitTag with a 0.23 g, 5.5 mA h battery. Right image shows a Pine Siskin (12–18 g) with tag. Tags are attached with a loop harness [39] constructed from elastic thread. Pine Siskin photo courtesy of Ben Vernasco

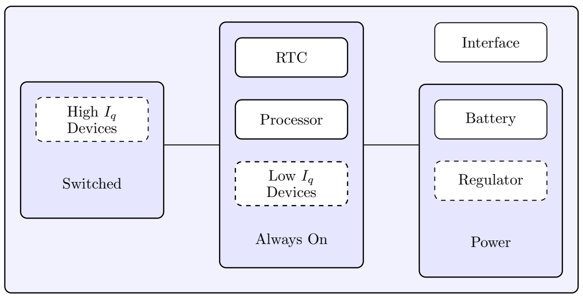


**Fig. 2** Generic tag architecture. Every tag has a processor, real-time clock (RTC), battery, and interface. In addition, tags have devices (e.g., sensors) with low quiescent current $I_q$ which are continuously powered and may have high $I_q$ devices which are powered only when in use. Depending upon battery chemistry, a tag may require voltage regulation

movement crosses a threshold measured in g). Activity is detected on a per-second basis as bits that are aggregated and stored in flash memory. BitTags with 0.5–0.8 g mass (depending upon battery) have been used in experiments with a variety of species (Dark-eyed Junco, Pine Siskin, Great Tit, White-crested Elaenia, and American Robin) with both captive and free animals to collect more than 500,000 h of activity data.

BitTag is part of a larger system architecture that includes supporting software and hardware. The hardware for a tag system consists of a tag, a base board, and a host computer. The base board includes battery charging circuitry, a processor to mediate communication between the tag and a host computer, and hardware to support high-resolution power measurement. In addition to these hardware components, the system includes both host software, for accessing tags, and the software (firmware) executing on the tag. In addition to presenting the components of our BitTag system, we explore our technique for measuring tag power requirements and estimating long-term energy consumption. By design, our power measurement solution utilizes the same hardware that we provide to researchers for configuring tags and that hardware can also be used to program new firmware onto tags.

We begin with a *generic* tag architecture and then discuss the BitTag system hardware and software design. In order to make the BitTag system extensible, the design of the host and tag software builds upon a well-defined software architecture that dictates how the host and tag exchange information. Our software architecture is easily extended to support the configuration of new types of tags and downloading of new types of logging data.

A high-level view of the major components of a generic tag is illustrated in Fig. 2 with optional sub-components indicated by dashed boxes. Every tag has an interface, a battery, a real-time clock (RTC), and a processor. We distinguish between components that are always powered (the "always on" domain), and components that consume too much energy to remain powered when not in use (the "switched" domain). The always on domain includes the core processor, the RTC, and any devices (sensors or flash memory) with sufficiently low quiescent current ($I_q$) to remain powered throughout the tag life. The accelerometer used in BitTags (ADXL362) has a low 10 nA $I_q$. Some tags will have external devices with high $I_q$ requiring them to be electrically disconnected when not in use. For example, we have built prototype tags with pressure and light sensors that have quiescent currents of greater than 0.5 µA which mandates the use of power switching.

Clearly, an important step in tag design is the selection of components that are *capable* of meeting the energy constraints necessary to collect data over long periods. In our tags we use the STM32L43x family of processors, which have extremely low-power "standby" states (under 100 nA ignoring the cost of the RTC reference signal), and, most recently, the RV-3028 RTC, which has 1 ppm accuracy and consumes 45 nA. With these components, our tags require approximately 210 nA (measured) when hibernating. The difference between this number and one reached simply by calculating from a datasheet is an important reminder of the need to measure actual circuits. For example, driving the clock signal from the RTC to the processor consumes energy, real capacitors suffer from leakage, and the cost of driving the processor RTC from an external clock signal cannot be determined from the datasheet.

Furthermore, datasheet estimates frequently assume the specific operating conditions and actual consumption can vary significantly—especially in the extreme low-power domain where our devices operate and hence depend crucially upon the tag software. Examples of software impact include the frequency with which the processor wakes, how often the processor communicates with peripheral devices, and how long the processor is

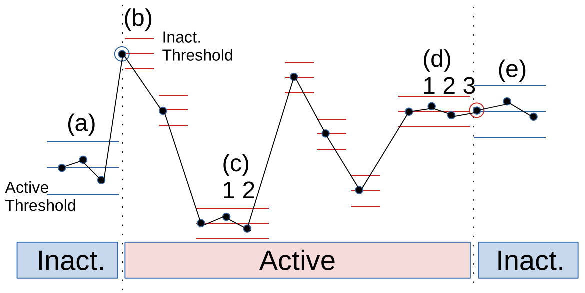


**Fig. 3** ADXL362 activity detector. Consider a tag with two states—inactive and active. An inactive tag (**a**) remains inactive until it experiences acceleration greater than the "active threshold" (**b**). An active tag remains active until it remains within a configured acceleration range for a configured time (d). Notice that whenever an active tag exceeds the configured threshold, the center of the range moves (**c**, **d**)

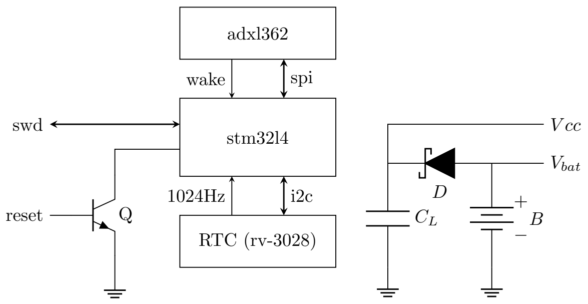


**Fig. 4** BitTag circuit. Circuit illustrating the major components of a BitTag including the processor (stm32l4), real-time clock (rv-3028) and accelerometer (adxl392). The processor reset pin is isolated from the interface with an NPN transistor (*Q*). Power is supplied by battery (*B*). The internal voltage ($V_{CC}$) is separated from the battery voltage ($V_{bat}$) by a Schottky diode (*D*). Only a single load/decoupling capacitor ($C_L$) is illustrated. Various passive devices and supply/ground signals to the block components omitted for clarity

active when it wakes. Measurement of actual energy consumption over a variety of operating modes is a crucial step in tag development and a key topic discussed in this section.

## BitTag

The operational concept of BitTag is simple; the tag contains an accelerometer that detects movement and, based upon the movement dynamics, determines if the subject animal is active. Each second BitTag generates a single bit of information—1 if the animal is active, 0 if it is not. These bits are aggregated (counted) over a measurement period ranging from 1 s to 5 min; at the end of each aggregation period the counts are stored in non-volatile memory. The choice of aggregation period depends upon the length of the experiment and is constrained by the amount of available storage—248 h for raw data (1 s "aggregation") to 8660 h for 5 min aggregation.

The key component in the BitTag is an extremely low energy accelerometer—the ADXL362 [40]. This accelerometer has special activity detection hardware that samples acceleration along 3-axes at 6 Hz to determine transitions between active and inactive states. Briefly, an animal is active if it displays acceleration (changes) above a configured threshold, and it is inactive if its acceleration (changes) remains below a configured threshold. The detection algorithm is illustrated in Fig. 3. This sensor is remarkably efficient (300 nA), yet very effective at tracking bird activity which can result in large changes in acceleration. In a previous article, we discussed our method for selecting the various sensor thresholds [41].

The circuit (block diagram) for BitTag is illustrated in Fig. 4. The various components are connected by standard hardware interfaces—I2C for the RTC, SPI for the accelerometer, and SWD (serial wire debug) [42] for the external processor interface. In our architecture, the external RTC is used only as a reference signal (1 kHz) that drives the processor's clock circuitry. This results in considerable energy savings over using the RTC's clock circuitry due to the inherent inefficiency of using the I2C open-drain architecture to access the current time from the RTC. The battery circuit is designed to allow powering the tag (from the base) while charging. The small batteries used in our tags have strict charging current limitations; hence our design includes a Schottky diode (*D*) to isolate battery voltage ($V_{bat}$) from internal power ($V_{CC}$). Note that the processor reset circuit is isolated from external electrical noise via an NPN transistor (*Q*). Finally, the circuit includes various load and decoupling capacitors (illustrated as a single load capacitor—$C_L$).

The interface to our tags, consisting of two SWD signals, reset, ground, $V_{cc}$ and $V_{bat}$ is accessed through an array of six test points. This is illustrated in Fig. 5 showing both sides of a v6 BitTag. During use in an experiment, these test points are insulated with Kapton® tape. The tags are protected from moisture and dust by a urethane coating (MG Chemicals 4223) developed specifically for electronics The batteries are wrapped with Kapton® tape and, after masking the testpoints, the entire assembly is submerged in the polyurethane. Finally the tags are allowed to air dry.

### *Base board*

The base board for the tags, illustrated in Fig. 6, provides a USB bridge to the tag SWD interface, a configurable battery charger, and an interface to an optional external power source which can be used to accurately measure the dynamic current required by an operating tag. We

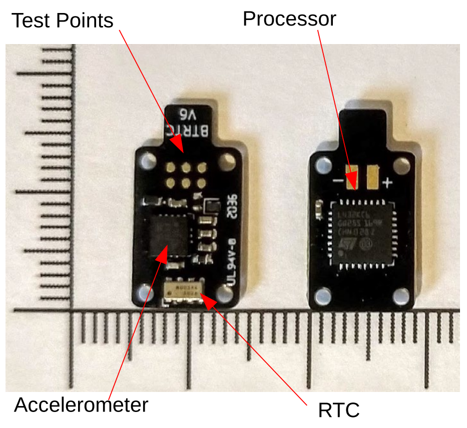


**Fig. 5** BitTags. Two sides of a BitTag illustrating the major components and the array of test points that are used to communicate with the tag. Scale is mm

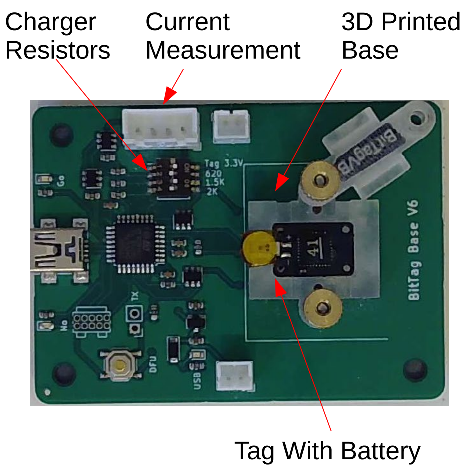


**Fig. 6** Base for tags. The tag base provides a USB interface between a host computer and tag. The tag is supported physically by a 3D printed base that can be modified for new tag designs. The base provides a battery charger that is configured with load resistors and an interface to an external power supply/current monitor. The base is isolated from the tag with low-leakage analog switches

program and communicate with our tags through the base USB connector using existing open-source tools and libraries such as openocd and st-util [43, 44]. To accommodate these tools, our base firmware emulates the st-link protocol used by ST Microelectronics in their proprietary programmers [44].

The tags are connected to the base (illustrated in Fig. 6) by an array of spring loaded "pogo-pins" (these contact the six test points illustrated in Fig. 5) and are supported by tag-specific 3D printed holders. The use of 3D printing for these adapters makes it possible to prototype new tag designs at low cost. The base processor provides

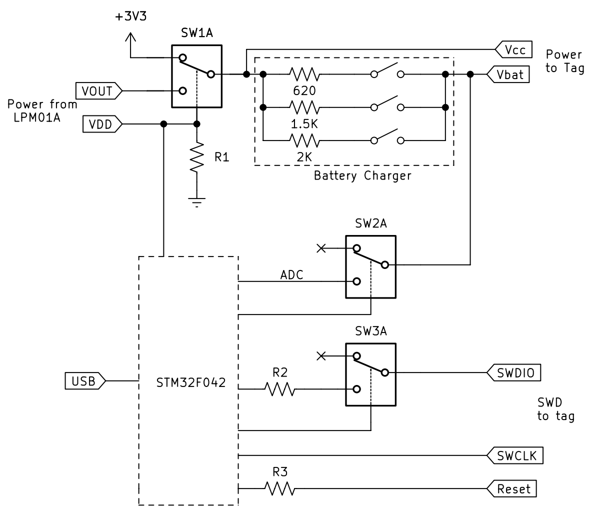


**Fig. 7** Tagbase circuit (simplified). The tag base consists of a processor (STM32F042), circuitry for a battery charger and SWD interface to the tag, and (optional) input from an external power source (LPM01A). Selection of the power source for the tag is controlled through an analog switch (SW1A). The processor can be isolated from the tag through a pair of analog switches (SW2A,SW3A). The external power source provides two power lines with programmable, but equal voltages—VDD and VOUT; the current through VOUT is measured. Voltage regulation and circuit protection components are elided

a "bridge" between the tag SWD interface and the host USB interface.

The tag base circuit, illustrated in Fig. 7, consists of a processor (STM32F042), a configurable battery charger, and signals to control the SWD programmer interface. The standard charging circuit for the LiMnSi batteries used in our tags consists of a voltage source and battery specific series resistor; our circuit provides manually configured switches to select the appropriate resistor. As mentioned, an important design consideration is the ability to connect an external power supply (the LPM01A [45]) providing wide-range dynamic current (power) monitoring. Our base automatically selects between the base-generated 3.3 V source and the external source (VDD) to provide the tag power (Vcc). This is accomplished through a low-leakage analog switch (SW1A). During power measurement, the base processor is disconnected from the tag interface with analog switches SW2A and SW3A. The processor detects connection to the external power source by monitoring VDD. When not connected externally, the processor monitors the tag battery voltage (Vbat) through an internal analog to digital converter (ADC).

Charging the LiMnSi batteries we use is a slow process requiring several days to reach full charge. Consequently, we have also developed a low-cost charger base (under

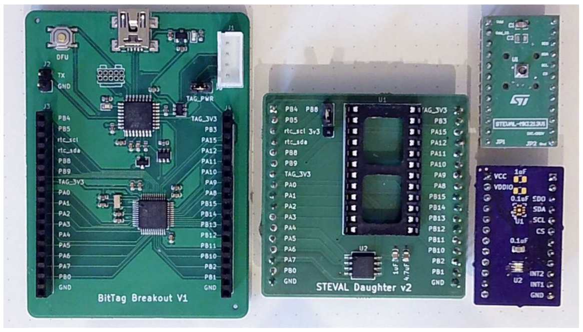


**Fig. 8** Tag breakout board (left), daughter card (center), and sensors (right). The tag breakout board (left) provides all of the functionality of a tag baseboard along with the processor and RTC used in a tag. All of the processor's free pins are brought to connectors enabling the creation of accurate tag prototypes with suitable daughter cards. In this figure, a daughter card (middle) with an external flash and connector is illustrated along with both a commercial sensor board (upper right) and a custom sensor board (lower right)

$30) that can be connected together in groups of 4–5 to be powered from a single USB cable.

### *Custom tags*

In order to facilitate the rapid prototyping of new tag designs and the evaluation of candidate sensors, we created a development platform that combines the interfaces of our base board with the processor and RTC required for our tags. In Fig. 8, this platform is illustrated along with a basic "daughter" card that includes an AT25XE321 flash memory and an adapter to support the large variety of sensor evaluation boards from ST Microelectronics. In addition, one commercial (top right) and one custom (bottom right) sensor evaluation board are illustrated supporting LPS27 pressure sensor (top right) that we use discuss in this article, and an OPT3002 light sensor (bottom right) that we have used to prototype other tag designs. Most commercial sensors are supported by manufacturer or third party evaluation boards (so-called breakout boards). While the platform base required commercial fabrication, the daughter card and OPT3002 breakout boards were fabricated in house with basic soldering equipment.

## Software architecture

The majority of the engineering effort in developing BitTag was expended on software; however, by careful design of the tag firmware and communication architecture that enables host software to control a tag through the tag base, we have greatly reduced the programming effort for creating new tags. Here, we briefly discuss the features of our software architecture and point out which components must change for custom tags. There are three major components to consider—the communication architecture, the tag firmware, and the host software (considered later in this article).

The system software for BitTags consists of firmware (the software running on the tags), a host library supporting communication with tags that are connected through a base, and various host applications utilizing this library including a configuration GUI, command-line tools for testing and commissioning tags, and a data visualization tool (not discussed in this paper).

The key software architecture decision in our system was to implement host/tag communication with a protocol defined using Google Protocol Buffers [46]. Protocol Buffers provides a well-defined interface description language for "Messages" and a "wire" format for serializing those messages. We use Protocol Buffers messages both to control our tags and to transfer configuration parameters and data between the host and tags.

Our use of protocol buffers greatly simplified the problem of providing a coherent and extensible interface between tags and hosts. By compiling from common source—the message definitions—all aspects of shared data are kept coordinated. Further, Protocol Buffers provide a standard way to extend the message definitions without breaking backwards compatibility. Thus, we can add support for new tags by extending the message definitions (for example to support new sensor types), and extend the host tools to utilize these new definitions without "breaking" support for existing tags. This is particularly important where tags "in flight" may be in use for many months during which new tags or iterations of tag firmware may be introduced. For example, our message definitions allow future tags to add configuration and logging messages for new types of sensors and data without breaking the ability of our host tools to configure and access earlier generations of tags. Furthermore, the host software utilizes configuration messages received from a connected tag to determine its type and capabilities.

Host software is built using Google provided C++ libraries while protocol buffer support on the tags is realized with Nanopb—a lightweight C language protocol buffer library [47]. The Protocol Buffer libraries make it easy to build software applications that can communicate with tags through our base hardware.

The set of messages in our communication protocol were designed to match the tag "life cycle" illustrated in Fig. 9 which corresponds to a simplified version of the state machine implementing the tag runtime firmware—several states have been omitted in the interest of brevity. Notice that there are three *categories* of states—*Erased*, *Active*, and *Completed*. In the *Erased* states, no data or configuration is stored in non-volatile memory. The

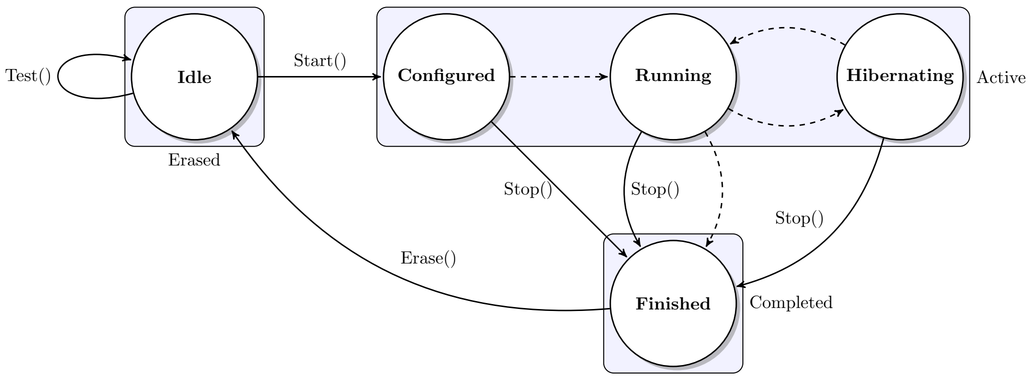


**Fig. 9** Tag life cycle (simplified). The tag firmware is organized around this (simplified) state machine. The key life phases are *Erased* (no data or configuration are stored), *Active* (tag is configured for data collection), and *Completed* (data collection has ceased). State transitions initiated by the host are illustrated with solid arrows. Tag-initiated state transitions are illustrated with dashed arrows

*Active* states correspond to a tag that is configured or is actively collecting data. In these states, configuration, and event and data logs are stored in non-volatile memory. Finally, the *Completed* states are post-experiment in the sense that data collection has ceased by design, by external command, or through an unrecoverable error (e.g., power brown-out). In the *Completed* states, the event and data logs from the last *Active* period are preserved in non-volatile memory.

In addition to organization by groups of states, our figure distinguishes between two types of state transitions. Internal transitions are shown as *dashed* arrows and transitions in response to an external request (message) are shown as *solid* arrows. Every communication between the host and tag is initiated by the host sending a *request* message to the tag and completed by the tag returning a *acknowledgement* message to the host. In the following, when a command is mentioned, it can be assumed to be realized by exchanging a pair of messages between the host and the tag.

The initial state after commissioning and erasure is **Idle**. An experiment (biological or during tag development) is initiated by the host from **Idle**, by configuring the tag with a **Start()** command. From the **Configured** state the tag enters the **Running** state once the configured start time has been reached. In **Running** the tag actively collects data; a tag may *hibernate* for a configured hibernation period during which all data collection ceases. The fundamental difference between **Hibernation** and **Running** is that in the former, all sensors are powered down and the tag enters the lowest possible power (dormant) state while in the latter the tag is either quiescent (waiting for an event) or the processor is actively running. In the case of BitTag, hibernation utilizes approximately 1/4 the power of active data collection and hence may enable lighter tags to operate over significantly longer periods at the expense of continuous data collection.

The **Start()** command takes a configuration object (providing scheduling, sensor configuration, and other inputs) as a parameter. In execution, this configuration object (a Protocol Buffer message) is transferred to the tag; when the tag evaluates the **Start()** command, it records the configuration and enters the Configured state. It is possible to abort execution of a tag by issuing a **Stop()** command.

Under ordinary conditions, a tag remains active until an end condition is met. This can be a configured end time, exhaustion of available data storage, or exhaustion of available energy. In each of these cases, the tag will enter **Finish**. The tag is returned from a completed state to **Idle** via an external **Erase()** command which erases any stored data.

The **Test()** command initiates internal self-test routines that check the basic functionality of sensors and other components. Pre-flight testing is especially important for tags because of the high opportunity cost of a multi-month experiment. As an example, the ADXL362 accelerometer has a known issue that can occur if its power is not properly cycled (for example when attaching batteries to tags). This issue is detected with the test routines in BitTag.

Several additional commands are available for accessing saved data and status logs, for determining the tag state and configuration, and for synchronizing the RTC.

### *Communication*

Communication with the tags is realized through a simple request/acknowledge protocol that is implemented on top of the ARM debug interface using protocol buffer messages. Briefly, each of the procedures described previously (e.g., **Start()**), is implemented by sending a request

```
message Req {
  oneof payload {

    // Information

    Empty get_info = 1;   // static tag information
    Empty get_status = 2; // current tag status
    Empty get_config = 3; // current tag configuration

    // control

    Empty erase = 4;      // erase tag (return to Idle)
    Config start = 5;     // configure & start tag
    Empty stop = 6;       // stop tag
    TestReq test = 7;     // start test
    int64 set_rtc = 8;    // set clock (milliseconds)

    // logs -- see log message for types

    LogReq log = 9;
  }
}
```

**Fig. 10** Request message definition

message to the tag and waiting for an acknowledgment message. The Req(uest) message is defined in Fig. 10. A request message contains one of the various choices. For example, the current tag state is requested with a request containing an (empty) **get_status** message and the current configuration is requested with an (empty) **get_config** message. Some request messages are non-empty. For example, the start request carries a **Config** message, which includes all of the information necessary to configure a tag such as start/stop times, hibernation periods, sensor and data logging configuration. The current system and data logs can be requested with the **log** message.

In the protocol buffer language message fields have unique numerical identifiers (here, the integers 1–9). These numbers play an important role in encoding and decoding messages. Furthermore, (sub)-messages can be defined as new types and their contents embedded in other messages. An important characteristic of the encoding/decoding process for protocol messages is that unknown fields are ignored; the consequence of this is that messages in our application can be extended with new fields to support new types of tags without breaking backwards compatibility with existing tags.

The dual of the request message is the acknowledgment which may return a variety of information from tag status to data logs. While the details have been omitted in this paper, there are a few key ideas that enable support of new tag types. Because sensor configuration is unique to each tag type, the **Config** message must be extended with new fields or sub-messages to support new sensors. To support new log data types, the acknowledgment messages are extended with new log data fields. These changes can be accomplished while maintaining backwards compatibility—new host software can support earlier tag types—and without requiring tag firmware to support message fields that are irrelevant to the tag. This extensibility is *built in* to the protocol buffers implementation. Furthermore, the host software can determine, by reading the tag information and configuration, with what type of tag it is communicating and hence can dynamically customize the host interface to support that tag type.

### *Tag software*

The tag software is organized around two threads—a main thread that implements the state machine illustrated in Fig. 9, which wakes periodically and performs data collection/recording tasks, and a communication thread that is active only when the tag is connected to a host though a tag base. As discussed previously, host software communicates with the tag using protocol buffer messages. These messages are sent to and received from the tag by writing to and reading from a reserved memory area utilizing the ARM SWD hardware interface. The communication thread is awoken by an interrupt routine triggered through the SWD interface.

The main thread spends most of its life sleeping, with the processor in a extremely low-power standby state. The processor wakes from standby whenever an RTC alarm occurs or an external sensor (e.g., the accelerometer in a BitTag) triggers a wakeup event through one of the dedicated wakeup pins. Recovering from standby in a STM32L4xx processor is somewhat complex because the wakeup event appears, to the software, as a reset event and because, by default, the only preserved state is a set of backup registers associated with the processor's RTC. Whenever the processor wakes, the main thread performs the following tasks:

1. Initialize the real-time operating system.
2. Determine the wakeup event cause.
3. Determine the current time.
4. Execute the state machine.
5. Return to standby mode.

The core logic associated with data collection and logging is associated with the "Run" state, which is necessarily different for each tag architecture. In the BitTag, the actions taken in this state include:

1. Determine if accelerometer is active.
2. Record activity in backup register holding current aggregation data.
3. At aggregation boundaries write to persistent storage.

More generally, the run state can be defined by the following steps:

1. Read sensors.
2. Log data.

When we create a new tag, the firmware that implements the tag lifecycle is retained. Only the code that corresponds to configuration, data collection and logging, and log retrieval must be specialized to support new sensors or new data collection strategies.

Similarly, the communication thread code is common across tag types with extensions as needed to support retrieving new data log messages and reading/writing new configuration information.

### *Host software*

The host software consists of a code library that provides a procedural interface to a tag and applications that use this library. These applications include a graphical user interface intended to be used by researchers who need to configure tags and download collected data, and command-line applications that can be used to test tags and configure them from stored configuration files. The host library is written in C++ and provides a tag object model that allows software to use a procedural interface for tag access.

Consider again the state diagram describing the tag "lifecycle" (Fig. 9) and notice the state transitions labeled Test(), Start(config), Stop(), and Erase(). These correspond directly to methods of the tag C++ class which is used to build host software.

```
bool Start(Config &config);
bool Stop();
bool Erase();
bool GetStatus(Status &status);
bool GetConfig(Config &cfg);
bool SetRtc();
```

Additional methods support attaching/detaching from tags (through the tag base) and accessing data logs. We leverage this interface both for the graphical interface and for command-line tools to test and commission tags. The example in Fig. 11 illustrates the simplicity of this interface for creating custom command-line tools—in this case to set and check the RTC for short-term drift. The first step is to Attach to the tag (through a base connected by USB). Once the tag is attached, its real-time clock can be set (based upon the host clock). In this example, the host sleeps for 2 s and then reads the tag status which includes its current RTC value in milliseconds. Finally, the test program computes the clock drift.

```
int main(int argc, char **argv)
{
  using MS = chrono::milliseconds;
  Tag tag;
  Status status;
  int64_t millis;

  if (tag.Attach())
  {

    tag.SetRtc();                       // set the RTC
    this_thread::sleep_for(MS(2000)); // sleep 2 seconds
    tag.GetStatus(status);              // Read the RTC
               // Compute drift
    auto now =  chrono::system_clock::now();
    auto ts =  chrono::time_point_cast<MS>(now);
    float error = (status.millis()
          - ts.time_since_epoch().count())/1000.0;
    cout << "Clock Error: " << error << endl;
  }
}
```

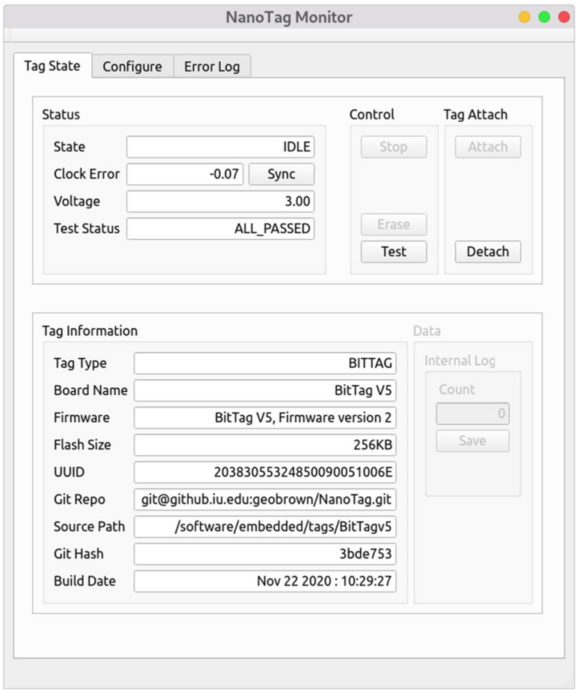
**Fig. 11** Example host application

While this may appear a useless exercise, it can detect whether the tag RTC is misconfigured or not running.

The primary tool for configuring tags and accessing their data is a graphical interface, qtmonitor. All of the functions of qtmonitor can also be accessed through command-line tools which simplifies the tasks for bulk configuration and bulk testing. The qtmonitor application, provides the ability to gather metadata about a tag (for example, its hardware and firmware revisions), to read and write configurations used in experiments, to synchronize the on-board clock, and to access tag data post-experiment. The monitor screen, illustrated in Fig. 12, includes a number of tabs—"Tag State", "Configure", and "Error Log"—to access various features. The Tag State tab provides status information about the tag, control functions, and information about the tag. The status information includes the current execution state (in this case Idle), battery voltage, and current RTC error. Tag information includes details about both the tag hardware and firmware including the processor unique identifier (UUID) and the location and version of the software build. This tab is also used to initiate internal tests and access data from completed experiments. The "Error Log" provides basic debugging information in the event of application errors.

The configuration tab (Fig. 13) includes three sub-tabs for defining the data collection schedule, data format (for BitTag, the aggregation period), and sensor (ADXL362) configuration. The configuration tab for BitTag supports a single sensor—the ADXL362. This configuration tab defines a number of parameters including the sample rate, dynamic range, and activity detection parameters. This configuration tab is only displayed for tags with the ADXL362 sensor.

Our system software architecture has made it possible for us to extend qtmonitor to support other tag designs with different sensors without requiring significant



Status

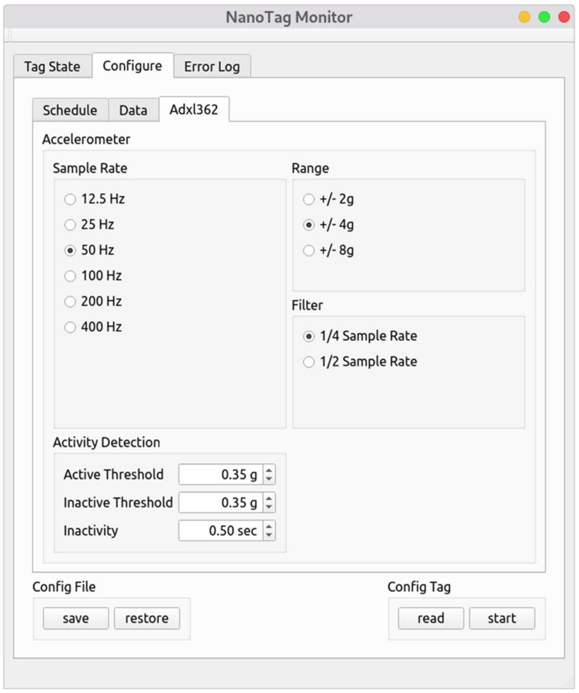
State: IDLE
Clock Error: -0.07
Voltage: 3.00
Test Status: ALL_PASSED

Tag Information

Tag Type: BITTAG
Board Name: BitTag V5
Firmware: BitTag V5, Firmware version 2
Flash Size: 256KB
UUID: 20383055324850090051006E
Git Repo: git@github.iu.edu:geobrown/NanoTag.git
Source Path: /software/embedded/tags/BitTagv5
Git Hash: 3bde753
Build Date: Nov 22 2020 : 10:29:27

Count: 0

**Fig. 12** qtmonitor main tab. Main tab for monitor program. The tag information section is metadata provided from the tag including information about the hardware and the firmware including the specific software revision from the Git repository. The status section provides current status information including tag state, test results, voltage, and real-time clock error. The control section allows execution of self-tests (for an idle tag), halting a running tag, and erasing a completed tag. The Tag Attach section enables attaching/detaching the base from a tag. Finally, for a completed tag, the data section enables data download



Accelerometer

Sample Rate: 12.5 Hz 25 Hz [✓ 50 Hz] 100 Hz 200 Hz 400 Hz

Range: +/- 2g [✓ +/- 4g] +/- 8g

Filter: [✓ 1/4 Sample Rate] 1/2 Sample Rate

Activity Detection

Active Threshold: 0.35 g
Inactive Threshold: 0.35 g
Inactivity: 0.50 sec

**Fig. 13** qtmonitor adxl362 configuration tab. The BitTag configuration provides a number of options for the ADXL362 accelerometer. The most important configurations are the activity detector thresholds and inactivity delay. In addition, the accelerometer sample rate (when active) can be configured along with the range. These configurations are provided to enable support for custom firmware—in the case of BitTag the defaults are generally accepted

redesign. The qtmonitor application configures itself, based upon information obtained by querying a connected tag, to display the appropriate configuration tabs.

## Energy and power

Accurate energy and power estimation for our tags requires active measurement of the various operating modes. The datasheets for the various devices provide architectural guidance, but are not sufficiently accurate for predicting tag lifetime or for determining peak power requirements. Furthermore, achieving the lowest energy utilization requires significant firmware tuning. For example, we minimized the cost of waking up from processor standby by optimizing the firmware initialization code. Power measurement is an ongoing part of our software development process—we iterate the firmware design based upon power measurements. In addition, relatively small configuration errors (for example in processor pin initialization or sensor initialization) can lead to significant excess power usage which would be difficult to diagnose without accurate power measurement.

**Table 2** Power requirements of key components at 2.5V

| Component | Power | Mode |
|---|---|---|
| STM32L432 | 725 μW | Run at 2 MHz |
| | 622 nW | Standby with external 32 kHz signal |
| | 805 nW | Standby with external 32 kHz crystal |
| ADXL362 | 25 nW | Standby |
| | 675 nW | Activity detection |
| | 4.5 μW | Sampling at 50 Hz |
| RV-3028 | 112 nW | Normal operation |

Consider the power requirements for the key components of BitTag illustrated in Table 2. The BitTag firmware uses the processor in two operating modes—Standby when the tag (or bird) is idle and Run when changes in activity occur. Similarly, the ADXL362 accelerometer has several relevant operating modes—Standby when the tag is idle, Activity Detection when waiting for a significant change in acceleration, and Sampling when acceleration exceeds programmed

**Table 3** Seiko small Li–Mn batteries [48]

| Type | Size (mm) | Weight (g) | Capacity (mA h) | Capacity (J) | Ω |
|---|---|---|---|---|---|
| MS414GE | 4.8 × 1.4 | 0.08 | 2.0 | 18 | 100 |
| MS518SE | 5.8 × 1.8 | 0.13 | 3.4 | 30.6 | 90 |
| MS621FE | 6.8 × 2.1 | 0.23 | 5.5 | 49.5 | 80 |
| MS920SE | 9.5 × 2.1 | 0.47 | 11 | 99 | 35 |

thresholds. Notice that the specification for processor Standby power depends upon the source (and frequency) of the reference signal. Furthermore, the cost of *driving* an external signal is not included. Surprisingly, the designs presented in this paper require only 525 nW with the external RTC because we drive the processor with a 1-kHz signal rather than the 32-kHz signal described in the datasheet. Finally, understanding the power required for a tag requires knowing the relative time spent in the various operating modes.

Our tag designs and base support a range of small batteries with capacities of 18–99 J (Table 3). Although BitTag is designed to operate with any of these batteries, we have only used the three larger devices in animal experiments. BitTag has operated in experiments lasting 10–11 months with 0.47 g batteries and in experiments lasting 3 months with 0.13 g batteries.

There are two key energy-related measurements necessary for tag development—average current and peak current. The former is necessary to estimate tag lifetime with a given battery, while the latter is necessary to design reliable circuits. The small batteries that are suitable for our tags can deliver limited peak current which necessitates additional load capacitors to deliver high peak current for sensing and writing flash.

Peak current capacity of a battery is limited by its internal impedance (Ω). For example, the external flash we use has peak currents of 3.4 mA; with the MS518, which has internal impedance of 90 Ω, this would result in a 0.3 V drop without capacitors sized to deliver the necessary peak currents. Accurately sizing these capacitors is crucial to minimizing leakage currents while bounding voltage 'drop" during peak power events. Excessive voltage drop can result in processor shutdown.

The dynamic range of power requirements in our tags is roughly 5 orders of magnitude—200 nA-10 mA—and involves peak power events as brief as 50 µs. In order to accurately measure power requirements, we need a current monitor with both a high sample rate and a large dynamic range. We designed our tag bases to accept connection to the X-Nucleo-LPM01A—an inexpensive (under $100) programmable voltage source that can make dynamic power measurements with currents ranging from 100 nA to 50 mA at 100 kHz sample rates with a claimed 2% accuracy [45].

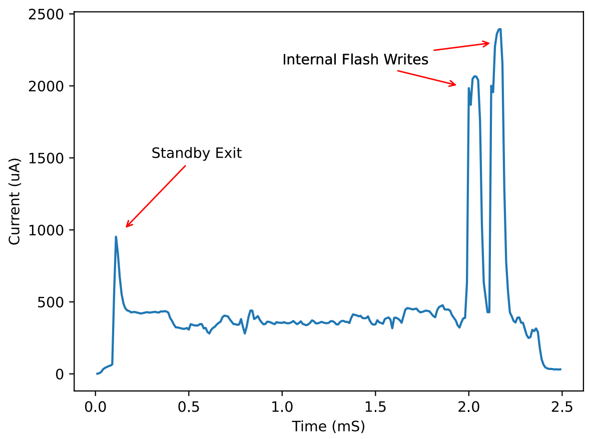


**Fig. 14** BitTag log write power. Power measurement for a single "wake" cycle with three current peaks. This cycle includes a peak due to processor exit from standby, and two peaks due to successive 64-bit flash writes. While awake, the processor averages approximately 400 µA. Flash writes, while brief (80–90 µs), have high peak currents

For example, the STM32L432 datasheet predicts that a 64-bit flash write takes 80–90 microseconds and requires 3.4 mA (average current). Figure 14 illustrates a complete log-write sequence (two 64-bit writes) from processor wake up to return to standby. Although the current during the write pulses (to the extreme right) appears somewhat lower than predicted, the overall behavior is consistent with the datasheet estimate. Notice also the small current spike that occurs at the transition from Standby to running. The datasheet predicts a 1.23 mA spike for 20 µs during this transition—this is at the limit for the 100 kHz sample rate of our measurement board.

In estimating peak power requirements, we utilize a feature of the LPM01A software that integrates current (and energy) over time; for example, the average current in the illustrated example was 424 µA and the total energy was 2.74 µJ at 2.5 V and the pair of flash writes required 1 µJ energy or 0.4 µC charge (1µJ/2.5 V). We can compute the load capacitance required to support these writes for a given voltage drop limit using the equation:

$$Capacitance = \frac{q}{V},$$

(where charge is $q$ and voltage is $V$). For a maximum 0.1V drop, the required load capacitor is then

$$4\,\mu\mathrm{F} = \frac{0.4\,\mu\mathrm{C}}{0.1\mathrm{V}}\,.$$

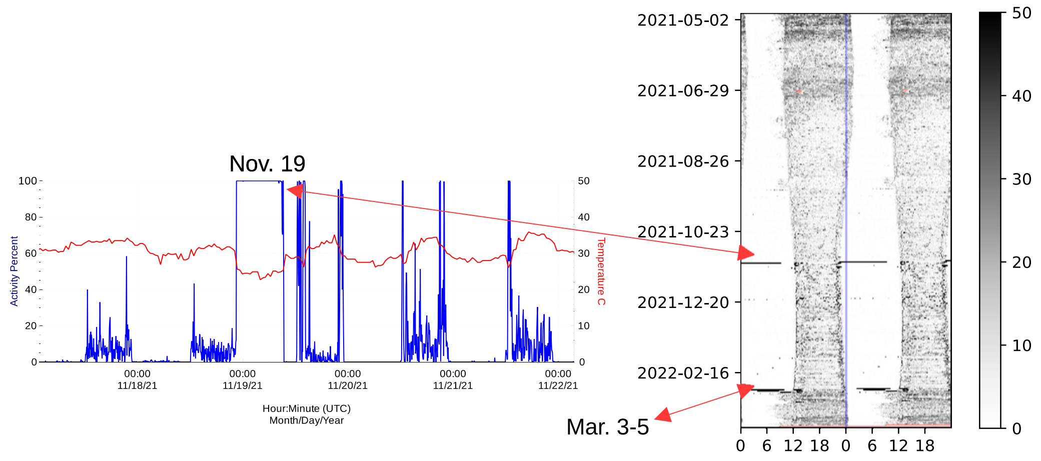


**Fig. 15** American Robin migration. This figure contains two components—an inset actogram displaying the activity of an American Robin over 10 months and detailed activity and temperature data for 5 days centered on November 20, 2021. Rows in the actogram contain 48 h of activity data (the percentage of a sample bucket that the animal was active) and rows are stepped by 24 h. Thus each day appears twice—first on the right of the actogram and then on the left of the next row. The primarily light regions of the actogram correspond to night and the primarily dark regions correspond to day. The activity detail includes both the activity level (the fraction of the integration period in which the animal was active), and the processor temperature. Note that during the night of November 19 (UTC) the animal made an extended flight. During this flight, the processor temperature declined. Also shown on the actogram is nocturnal activity during March 3–5, 2022

In practice, the capacitor capacity must be increased to compensate for physical effects such degradation with temperature and DC load; all of the batteries in Table 3 are capable of delivering peak currents of 1 mA with 0.1 V drop which reduces the required capacitance. BitTag has 5 μF total load capacitance.

## Results

### BitTag

More than 400 BitTags have been used in a variety of experiments with both captive and free animals over a 3-year period. During that time there have been two major hardware releases—V4/V5 and V6/V7; the former used the RV-8803 RTC and the latter the RV-3028 RTC. BitTags are remarkably inexpensive to build—in batches of 100–160 this has averaged $30/tag including battery. Similarly, the support hardware—bases for programming and charging average less than $50/base.

An example of the data that BitTag collects is displayed as an actogram [15] in Fig 15 tracking the activity of an American Robin over 10 months and clearly illustrate a long flight on November 19 (UTC), 2021 with several return flights in March 2022. The figure includes both the long-term data and detailed data from the period surrounding November 19.

We have provided BitTags to several researchers studying a variety of species with both captive and free-range animals. At Indiana University, Devraj Singh has deployed nearly 200 tags on Dark-eyed juncos in aviaries for several extended experiments relating to circadian rhythm. These experiments used two distinct populations of juncos kept in indoor aviaries with varying light conditions monitored over a period of 18 months (with tags replaced at 6-month intervals.) Alex Jahn has deployed more than 40 tags on free American robins to study their activity patterns over the year. Robins in Indiana are partially migratory (some migrate and some do not), and Jahn was able to determine their activity budgets throughout the year and found that robins will, at times, migrate at night (Fig. 15).

At Washington State, Heather Watts has deployed BitTags on two species of small songbirds—Pine Siskins and Red Crossbills—to collect behavioral data on wild-caught birds housed in large outdoor aviaries under semi-naturalistic conditions.

Tim Greives and Emily Elderbrock at North Dakota State have deployed tags on captive juncos in Fargo North Dakota to quantify daily activity and its relationship with physiological measures. They have deployed tags on free-living Great Tits in Bavaria, Germany over multiple years to track behavior over their breeding season.

Victor Cueto has deployed 20 tags on free-living White-crested Elanea in Argentina collecting data over a full migration cycle with 0.5 g tags programmed to hibernate between 2-month windows centered on the expected migration dates.

*Energy and power measurements*

The energy requirements for a BitTag depend both upon the hardware and firmware designs, and upon dynamic characteristics of animal behavior because the

**Table 4** BitTag energy/power usage

| State | Event | Power | Energy |
|---|---|---|---|
| Idle | | 525 $nW$ | |
| ADXL sleep | | 1.1 $\mu W$ | |
| ADXL awake (50 hz) | | 4.5 $\mu W$ | |
| | Wake event | | 1.5 $\mu J$ |
| | Log event | | 2.8 $\mu J$ |

tags require more energy when activity is detected than when idle. Thus, predicting the energy requirements of a biological experiment depends both upon measurable quantities—the power requirements of a hardware/firmware combination in various operating modes—and estimates, of how long we expect a tag to remain in each operating state based upon past experimental data. In the following, we show how we estimate tag energy requirements, and discuss results from biological experiments that may be useful in predicting tag lifetime in a future experiment.

In our firmware, there is normally a single active thread whose lifecycle has three purposes. First, it determines the cause of a wakeup event and then it performs any required actions such as reading the sensor state and writing data logs. Finally, it returns to the Standby state. The possible wakeup events are either an RTC alarm or an external input (e.g., the accelerometer activity state).

BitTags with the RV-3028 RTC have power consumption in various operating modes ranging from 525 nW to 4.5 μW. For example, the ADXL362 has a low-energy sleep state that requires about 260 nA while sampling at 6 Hz. When activity is detected, the accelerometer wakes and samples at a higher data rate requiring 1.8-2.4 μA depending upon sample rate. In addition, wake/sleep transitions trigger the processor to wake. The processor wake cost is 330 μA for 1.8 ms (the energy for this event is 1.5 μJ at 2.5 V).

In computing energy requirements we consider two effects—the steady-state cost of operating in various low-energy states, and the cost of high-energy events (e.g., waking the processor to handle an event). The energy/power for a BitTag V6 are shown in Table 4. The Idle current corresponds to the cost of hibernation, while ADXL Sleep and Awake distinguish the two operating modes of the accelerometer when the animal is inactive or active. There are two types of events of interest—those that wake the processor due to a sensor or RTC event, and those that involve logging to internal flash.

Activity data from over 500,000 h averaged 8.2% activity with a peak of roughly 15%. To understand how these data can be used to predict the energy requirements of an experiment, consider the following. BitTags wake once/minute to summarize data and, in long experiments, twice/hour to log data. Suppose the animal is active 15% (our experimental data suggest this is conservative for juncos) and every active second causes two wakeup events (again, conservative). The average current, neglecting the wake events for the moment, is:

$$0.85 * 1.1\ \mu W\ + 0.15 * 4.5\ \mu W\ =\ 1.61\ \mu W.$$

Every second, we expect 0.3 events of 1.5 μJ:

$$0.3 * 1.5\ \mu J * S\ =\ 0.45\ \mu W.$$

The log events are so infrequent that they can be ignored in this estimate. In this scenario, the tag uses on average 2.06 μW; with a 5.5 mA h battery, we expect a lifetime of 6675 h (278 days). In experiments with American robins, Alex Jahn has achieved over 8000 h (limited by data capacity) using an 11 mA h battery.

In order to enable long experiments with small batteries, our tags have a *hibernation* state during which no data are collected. In this state, requiring 525 nW, the sensors are shut down and the tag wakes once per hour to determine if data collection should resume. Hibernation requires 1/4 the power of active data collection and can be used, for example, to target data collection to the few months surrounding migration. Victor Cuerto recently retrieved two 0.48 g BitTags with 3.2 mA h batteries that collected activity data on White Crested Eleania and which ran for 9 months with a 5-month hibernation period separating two 2-month migration windows.

*Reliability*

Over time we have made minor revisions to improve reliability. Overall reliability has steadily improved with both changes to the coatings and software modifications to enable graceful recovery from more types of unexpected events. An early 60-day experiment with captive animals had a tag failure rate of 23% while a recent experiment with free animals had a 15% failure rate over 90 days. The majority of failures are partial—the tag aborted data collection due to an unexpected event, but the data up to failure remain readable.

The reliability of BitTag has improved significantly since the first biological experiments performed in 2019. Improvements can be attributed to three major factors—small hardware design changes, evolution of the coatings, and, significant changes to the tag firmware. For example, the first 20 tags, used with captive animals, suffered 8 failures in which the tags stopped collecting data prior to the end of the experiment. In contrast, a recent experiment with free Great Tits lasting 1200 h had a failure rate of 3/20 and an experiment with captive Pine Siskins lasting 2200 h had a failure rate of 3/20.

The most significant reliability improvement has been due to changes in passivation (coatings). We initially used a silicone coating that proved to be too soft—the coating contained a fluorescent dye that allowed us to determine that the coating had failed due to abrasion. We subsequently switched to a more durable polyurethane coating. In addition, more recent tags have been deployed with polyimide tape wrapping the batteries to provide additional electrical insulation.

The firmware for first generation BitTags deployed in experiments did not utilize the protocol buffer-based architecture described in this paper. Along with the new communication model, significant improvements have been made to enable recovery from transient errors and to provide more comprehensive self-tests in order to ensure that tags are functioning properly before deployment.

## Custom pressure tag

In order to demonstrate the flexibility of our system architecture, we designed a novel tag with a pressure sensor (PresTag) and a large capacity (4 MB) external memory. The addition of this memory enables a new family of tags capable of storing finer grained data than BitTag. Although the pandemic-related semiconductor shortage has precluded building prototypes, we used the prototyping platform illustrated in Fig. 8 to fully develop the host and tag software and to measure power requirements.

Pressure sensors have been shown to have great utility in understanding the behavior of birds during migration. For example, [24] demonstrated that by comparing pressure measurements over time it is feasible to determine which animals from a given site migrate together, [37] demonstrated that one can reliably use pressure measurements to determine when animals are migrating, and [49] determined that small animals may fly above 5000 ms during migration. Recently, [26] developed a method to combine pressure sensor data with global weather data to accurately predict location. A sub-gram pressure tag has previously been developed, but with significantly less storage (1/100) and greater power requirements (10x) than PresTag [5].

The PresTag circuit, illustrated in Fig. 16 replaces the accelerometer of BitTag with a low-energy waterproof pressure sensor (LPS27) and adds a unique wide-voltage, low-power 4MB flash memory (AT25XE321). In contrast with the ADXL362 accelerometer, the LPS27 has a relatively high (0.9 μA) $I_q$ and hence power to the sensor is controlled through a processor I/O pin. The LPS27 is extremely accurate (0.5 hPa absolute accuracy, 0.025 hPa relative accuracy) and sensitive—as configured in the PresTag, the data collected have 1/32 hPa resolution. This allows tracking *changes* in altitude of approximately 0.25 m—in order to compute absolute altitude knowledge of the local barometric pressure is required.

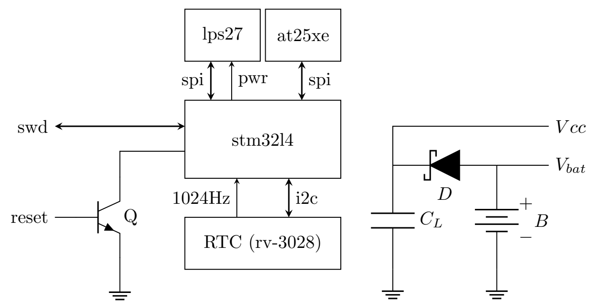


**Fig. 16** Pressure tag circuit. The major architectural differences between the pressure tag and BitTag are the different sensors and the addition of a large (4 MB) NOR flash memory (at25xe). Due to additional energy requirements of the external flash relative to internal flash the net load capacitance ($C_L$) required is significantly higher 23 μF vs. 5 μF. Diode *D* enables charging battery *B* while *Vcc* is connected. Bipolar transistor *Q* provides isolation for the processor reset line. Note that power to the pressure sensor (lps27) is provided by a processor I/O pin. Supply and ground signals to major blocks and various passive components are elided for clarity

**Table 5** PresTag lifetime in days with various batteries (* indicates memory limited)

| Period | Power (μW) | MS414GE | MS518SE | MS621FE | MS920SE |
|---|---|---|---|---|---|
| 10 s | 4.38 | 47 | 81 | 113* | 113* |
| 30 s | 1.18 | 115 | 195 | 316 | 341* |
| 60 s | 1.15 | 181 | 308 | 498 | 682* |
| 120 s | 0.81 | 256 | 435 | 705 | 1364* |

PresTag was designed to record pressure and temperature (16 bits each) at a configurable period. In addition, a block header consisting of timestamp, processor voltage, and processor temperature is stored every 30 periods. The external flash memory can store 32768 blocks (273 h at 1 sample/s; 683 days as 1 sample/min). We measured the energy consumption of our design using our development platform with an STEVAL-MKI213 pressure sensor board from ST Microelectronics (we modified the board by removing a 10 μF capacitor). We used the STM-32CubeMonPwr [50] tool to measure energy consumption over extended periods at various sample rates. The average currents and predicted tag life times are illustrated in Table 5. With a 120 s sample period, PresTag uses less than 1/10 the power of the Shipley design sampling once per day (0.81 μW vs. 8.5 μW) with the ability to store 100 times as much data [5].

We completed the PresTag board design—a 3D model is illustrated in Fig. 17. We chose to design this board with exactly the same outline as BitTag so that we can accurately estimate its mass. The BitTag board is a 4-layer, 0.6 mm substrate; for PresTag we switched to a 2-layer board which makes it feasible to build on 0.4 mm

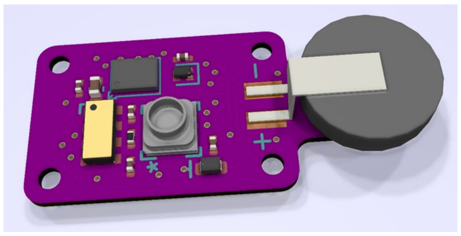


**Fig. 17** PresTag 3D model

substrate using the same low-cost fabricators we have used for BitTag. The expected mass with coatings for the four batteries in Table 3 ranges from 0.4 g (MS414GE) to 0.83 g (MS920SE).

As discussed previously, peak power consumption is also a significant design consideration because of the relatively high impedance of the small batteries we use. In the case of PresTag, peak currents occur during writes to flash. The firmware for PresTag was designed to write to flash in 2-byte blocks and to sleep between writes to allow load capacitors to recharge. Using our development platform, we determined that the energy required for a single 2-byte write is 4 $\mu$J. At 2.5 V, the charge required for this write is 1.6$\mu$C. Delivering this charge with a 0.1 V drop requires a 16 $\mu$F capacitor. To meet this requirement, we included a 22 $\mu$F load capacitor in the PresTag design. The capacitor chosen has a minimum specified insulation resistance of 22 M$\Omega$ and hence a maximum leakage current of 114 nA. At a 60-s sample period, this results in a 20% reduction in the expected tag lifetimes illustrated in Table 5.

As mentioned previously, hardware design represents a small part of the engineering effort required to develop a new tag. In developing PresTag, we extended the host software libraries and leveraged the tag firmware design. Protocol changes consisted of adding a single configuration parameter—the sample period—and adding a new data log message type. Modifications to the host library and qtmonitor required fewer than 100 lines of code. Modifications to the protocol required fewer than 20 lines of code. The new tag firmware required fewer than 1200 lines of new code including drivers for flash and the pressure sensor. Thus, the majority of software engineering effort was dedicated to the tag firmware with relatively little effort necessary to support the new tag on the host software.

## Discussion

We have presented the architecture of a tag system to enable the design and deployment of sub-g, sub-$\mu$A data loggers and discussed the detailed design of two tags—BitTag, an activity logger that has been used in several extended experiments; and PresTag, a prototype pressure (altitude) measurement tag capable of storing pressure data 1/min for a year at less than 0.6 g. Throughout this paper we have focused on three related design constraints—mass, energy, and power.

The designs we presented are the result of considerable optimization of both hardware and software. Hardware optimizations include our use of an external RTC in a hybrid configuration, capacity sizing of load capacitors to satisfy peak power requirements, limited use of power-hungry communication (I2C), and the use of I/O pins to power sensors. Software optimizations include tuning the use of initialized memory, the embedded operating system, and substituting low-power wait states for active polling of sensors. In addition, we utilized our power monitoring tools to find and eliminate unexpected power usage.

Our initial tag designs utilized an external crystal to provide a reference signal to the processor RTC. We found this to be extremely unreliable—low-power crystal circuits are very sensitive to stray capacitance (e.g., contact with a living organism)—sensitive to temperature changes, and difficult to calibrate. By switching to a hermetically sealed external RTC we resolved all these issues with significantly reduced power. While the RTC components we use are extremely efficient, the hardware protocol used to communicate with them, I2C, is not. I2C utilizes open-drain signaling which is both slow and consumes significant power through pull-up resistors. By switching to a hybrid model—the external RTC providing a calibrated 1 kHz reference signal to the processor RTC, we greatly reduced the power requirements.

As we have shown, the energy required by a tag depends upon both upon the power requirements of its various states (e.g., sleeping or running) and the time spent in those states. Average power requirements can be optimized by minimizing the running time. Using dynamic power information from our power measurement board, we significantly optimized the length of a wakeup event by tuning the tag software. For example, we eliminated the use of "initialized memory" including the task stack and local variables, and optimized the initialization of the tag operating system. These optimizations included modifying the linker script to create an uninitialized memory segment, restructuring the tag software to utilize this memory segment, and changing the operating system configuration to optimize startup time. Similarly, the operating system used for our tags is, by default, configured to utilize DMA for external I/O transfers; by careful measurement, we found that it requires fewer processor cycles (and hence energy) to use non-DMA I/O transfers for SPI for the short transfers required by our tags.

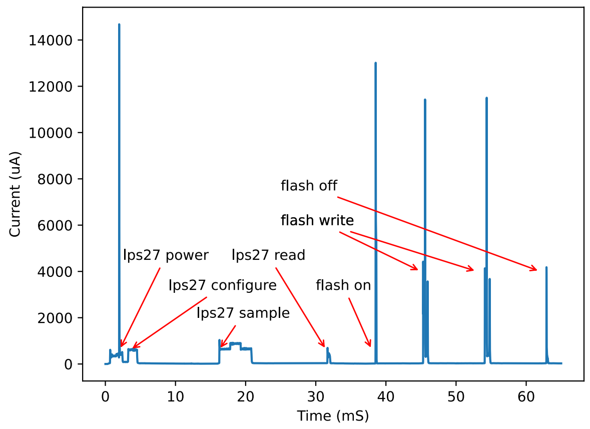


**Fig. 18** Pressure/temperature collection cycle with PresTag. A full data collection cycle with PresTag including powering the pressure sensor (LPS27), sampling pressure and temperature, and writing the data to external flash

Developing the software to configure and access a new sensor can be challenging and provides additional opportunities for optimization. For example, the "normal" way to use an LPS27 pressure sensor is to leave it always powered; however, the "power-down" state of an LPS27 requires 0.9 μA which is too high for our tag designs. Thus, we use a processor I/O pin to provide power. When the device is first powered, it undergoes a startup process in which internal trimming parameters are loaded into the sensor registers. This startup process takes 4.5 ms. The device also has several different conversion options—"low-noise", which takes approximately 13.2 ms and "low-current", which takes approximately 4.7 ms. The ususal approach to handling sensor sampling times is to poll them periodically, but polling takes energy; therefore, our designs simply wait for a suitably long period for the operation to be guaranteed to reach completion. To accommodate these relatively long periods which the tag is essentially idle, we implemented a mode for the operating system to support processor "Stop mode".

As we discussed previously, a key hardware design consideration is the sizing of load capacitors because, if too large, they waste energy through current leakage and if too small, they allow excessive voltage drop. Properly sizing load capacitors is only a piece of the design problem; after a peak power event it is necessary to provide load capacitors with sufficient time to recharge before subsequent peak power events. Providing this time is a software tuning problem. To illustrate how our system architecture enables solving this tuning problem, consider Fig. 18. There are several peak power events including powering the pressure sensor (LPS27), power up the external flash memory, and writing data to flash. In our design, we write flash in 2-byte chunks (the size of pressure or temperature data). Notice that between peak power events there are long idle periods; these are realized using the processor Stop mode which, in principle, consumes approximately 1 μA (in practice other components are powered and hence consuming energy). Choosing the length of these idle periods is yet another optimization task.

We have referred several times in this paper to excess power usage resulting from misconfigured pins. Pin configuration for the STM32 processors can be quite complex with options for pull-up and pull-down resistors, analog inputs, and connection of pins to external devices. Furthermore, during processor standby, I/O pins "float" unless explicitly connected to a pull-up or pull-down resistor through a separate configuration mechanism for the standby state. Thus pins can be misconfigured during standby, while running, and while accessing external devices. Our power monitor board has helped us find and correct misconfigurations during all three operational phases.

Our tag design system includes a novel prototype board that enables the design and testing of candidate tag architectures using readily available sensor evaluation boards. It was this prototyping board that we used to design our PresTag which, due to the pandemic semiconductor shortage, we have not been able to build. We have also used this same platform to evaluate candidate designs utilizing a variety of other sensors including a pressure sensors (LPS22, LPS27, LPS33), accelerometers (ADXL362, AIS2DW12, LIS2DTW12), magnetometers (MMC5633), and light sensors (OPT3002). It is convenient to be able to evaluate multiple sensors of the same type because there are frequently differences in the onboard processing available (e.g., activity detection) and because the datasheets rarely provide peak power data which is especially important with the batteries we use. For example, magnetometers require high peak currents, but the datasheets generally provide only average currents at a given measurement rate.

All of the hardware designs and associated software are available as open-source and all of the host software compiles on and for Windows, OS X, and Linux. All of the hardware designs utilize open-source tools (KiCad [51] for circuit boards, openSCAD [52] for 3D printed adapters) that operate on all three platforms.

The repository for our designs includes extensive documentation on building and using the hardware and software associated with this project. In addition to the designs presented, we have built prototypes for several other tags utilizing light sensors, magnetometers, and several accelerometers suitable for collecting movement data beyond simple activity. Thus, the system architecture presented is widely applicable to the creation and deployment of ultralight low-energy data loggers built with a variety of sensors.

**Acknowledgements**
The authors would like to thank the following for their feedback on BitTags used in their research: Tim Greives, Emily Elderbrock, Devraj Singh, and Heather Watts. Tim Greives also played a critical role in early field trials of BitTag and in developing a user manual.

**Author contributions**
GB: led the BitTag design and implementation effort, and manuscript preparation. AEJ: assisted in data collection and manuscript preparation. JC: contributed to the design of BitTag and its application software; contributed to the BitTag testing effort; assisted in manuscript preparation. AF: performed BitTag testing and manuscript preparation. All authors read and approved the final manuscript.

**Funding**
This material is based upon work supported by the National Science Foundation under Grant No. 1644717.

**Availability of data and materials**
All of the design files are available on github (https://github.com/geoffreymbrown/ultralight-tags) and through Zenodo (https://zenodo.org/badge/latestdoi/536749223).

## Declarations

**Ethics approval and consent to participate**
Research on American robins was approved by the Indiana University IACUC (#18-028), Federal Bird Banding Laboratory (#20261), and the Indiana Department of Natural Resources (#20-528).

**Consent for publication**
Not applicable

**Competing interests**
The authors declare that they have no competing interests.

Received: 19 October 2022 Accepted: 22 March 2023
Published online: 03 May 2023

**References**
1. Brown DD, Kays R, Wikelski M, Wilson R, Klimley AP. Observing the unwatchable through acceleration logging of animal behavior. Anim Biotelemetry. 2013;1(1):20–20. https://doi.org/10.1186/2050-3385-1-20.
2. McCafferty DJ, Gallon S, Nord A. Challenges of measuring body temperatures of free-ranging birds and mammals. Anim Biotelemetry. 2015;3(1):33–33. https://doi.org/10.1186/s40317-015-0075-2.
3. Collins PM, Green JA, Warwick-Evans V, Dodd S, Shaw PJA, Arnould JPY, Halsey LG. Interpreting behaviors from accelerometry: a method combining simplicity and objectivity. Ecol Evol. 2015;5(20):4642–54. https://doi.org/10.1002/ece3.1660.
4. Bäckman J, Andersson A, Alerstam T, Pedersen L, Sjöberg S, Thorup K, Tøttrup AP. Activity and migratory flights of individual free-flying songbirds throughout the annual cycle: method and first case study. J Avian Biol. 2017;48(2):309–19. https://doi.org/10.1111/jav.01068.
5. Shipley JR, Kapoor J, Winkler RAD. An open-source sensor-logger for recording vertical movement in free-living organisms. Methods Ecol Evol. 2018;9(3):465–71. https://doi.org/10.1111/2041-210X.12893.
6. Bridge ES, Thorup K, Bowlin MS, Chilson PB, Diehl RH, Fléron RW, Hartl P, Kays R, Kelly JF, Robinson WD, Wikelski M. Technology on the move: recent and forthcoming innovations for tracking migratory birds. BioScience. 2011;61(9):689–98. https://doi.org/10.1525/bio.2011.61.9.7.
7. Bridge ES, Kelly JF, Contina A, MacCurdy RMG, B aR, Winkler DW. Advances in tracking small migratory birds: a technical review of light-level geolocation. J Field Ornithol. 2013;84(2):121–37. https://doi.org/10.1111/jofo.12011.
8. Kays R, Crofoot MC, Jetz W, Wikelski M. Terrestrial animal tracking as an eye on life and planet. Science. 2015;348(6240):2478–2478. https://doi.org/10.1126/science.aaa2478.
9. Wilmers CC, Nickel B, Bryce CM, Smith JA, Wheat RE, Yovovich V. The golden age of bio-logging: how animal-borne sensors are advancing the frontiers of ecology. Ecology. 2015;96(7):1741–53.
10. Whitford M, Klimley AP. An overview of behavioral, physiological, and environmental sensors used in animal biotelemetry and biologging studies. Anim Biotelemetry. 2019;7(1):24. https://doi.org/10.1186/s40317-019-0189-z.
11. Williams HJ, Taylor LA, Benhamou S, Bijleveld AI, Clay TA, de Grissac S, Demšar U, English HM, Franconi N, Gómez-Laich A, Griffiths RC, Kay WP, Morales JM, Potts JR, Rogerson KF, Rutz C, Spelt A, Trevail AM, Wilson RP, Börger L. Optimizing the use of biologgers for movement ecology research. J Anim Ecol. 2020;89(1):186–206. https://doi.org/10.1111/1365-2656.13094.
12. Swiss Ornithological Institute: Tracking Devices: Miniaturized Geolocators. https://www.vogelwarte.ch/en/projects/bird-migration/tracking-devices-miniaturized-geolocators. Accessed 15 Jan 2023.
13. Hammond TT, Springthorpe D, Walsh RE, Berg-Kirkpatrick T. Using accelerometers to remotely and automatically characterize behavior in small animals. J Exp Biol. 2016;219(11):1618–24. https://doi.org/10.1242/jeb.136135.
14. Bäckman J, Andersson A, Alerstam T, Pedersen L, Sjöberg S, Thorup K, Tøttrup A. Activity and migratory flights of individual free-flying songbirds throughout the annual cycle: Method and first case study. J Avian Biol. 2016;48:309. https://doi.org/10.1111/jav.01068.
15. Bäckman J, Andersson A, Pedersen L, et al. Actogram analysis of free-flying migratory birds: new perspectives based on acceleration logging. J Comp Physiol A. 2017;203:543–64. https://doi.org/10.1007/s00359-017-1165-9.
16. Wild byte technologies: The daily diary animal movement and tracking tag. http://wildbytetechnologies.com/tags.html Accessed 15 Jan 2023.
17. Liechti F, Witvliet W, Weber R, Bächler E. First evidence of a 200-day non-stop flight in a bird. Nat Commun. 2013;4(1):2554–2554. https://doi.org/10.1038/ncomms3554.
18. Harrington KJ, Fahlbusch JA, Langrock R, Therrien J-F, Houtz JL, McDonald BI. Seasonal activity levels of a farm-island population of striated caracaras (Phalcoboenus australis) in the Falkland Islands. Anim Biotelemetry. 2020;8(1):27.
19. Meier CM, Karaardıç H, Aymí R, Peev SG, Bächler E, Weber R, Witvliet W, Liechti F. What makes Alpine swift ascend at twilight? Novel geolocators reveal year-round flight behaviour. Behav Ecol Sociobiol. 2018;72(3):45–45. https://doi.org/10.1007/s00265-017-2438-6.
20. Hedenström A, Norevik G, Warfvinge K, Andersson A, Bäckman J, Åkesson S. Annual 10-month aerial life phase in the common swift Apus apus. Curr Biol. 2016;26(22):3066–70. https://doi.org/10.1016/j.cub.2016.09.014.
21. Patterson A, Gilchrist HG, Chivers L, Hatch S, Elliott K. A comparison of techniques for classifying behavior from accelerometers for two species of seabird. Ecol Evol. 2019;9(6):3030–45. https://doi.org/10.1002/ece3.4740.
22. Lindström Å, Alerstam T, Andersson A, Bäckman J, Bahlenberg P, Bom R, Ekblom R, Klaassen RHG, Korniluk M, Sjöberg S, Weber JKM. Extreme altitude changes between night and day during marathon flights of great snipes. Curr Biol. 2021;31(15):3433–34393. https://doi.org/10.1016/j.cub.2021.05.047.
23. Sjöberg S, Pedersen L, Malmiga G, Alerstam T, Hansson B, Hasselquist D, Thorup K, Tøttrup AP, Andersson A, Bäckman J. Barometer logging reveals new dimensions of individual songbird migration. J Avian Biol. 2018;49(9):01821–01821. https://doi.org/10.1111/jav.01821.
24. Dhanjal-Adams KL, Bauer S, Emmenegger T, Hahn S, Lisovski S, Liechti F. Spatiotemporal group dynamics in a long-distance migratory bird. Curr Biol. 2018;28(17):2824–28303. https://doi.org/10.1016/j.cub.2018.06.054.
25. Dreelin RA, Shipley JR, Winkler DW. Flight behavior of individual aerial insectivores revealed by novel altitudinal dataloggers. Front Ecol Evol. 2018;6:182–182. https://doi.org/10.3389/fevo.2018.00182.
26. Nussbaumer R, Gravey M, Briedis M, Liechti F. Global positioning with animal-borne pressure sensors. Methods Ecol Evol. 2022. https://doi.org/10.1111/2041-210X.14043.

27. Kenward R. A manual for wildlife radio tracking. Cambridge: Academic Press; 2001.
28. Atema E, Van Noordwijk AJ, Boonekamp JJ, Verhulst S. Costs of long-term carrying of extra mass in a songbird. Behav Ecol. 2016;27(4):1087–96. https://doi.org/10.1093/beheco/arw019.
29. Bell SC, El Harouchi M, Hewson CM, Burgess MD. No short- or long-term effects of geolocator attachment detected in Pied Flycatchers Ficedula hypoleuca. Ibis. 2017;159(4):734–43. https://doi.org/10.1111/ibi.12493.
30. Lameris TK, Müskens GJDM, Kölzsch A, Dokter AM, Van der Jeugd HP, Nolet BA. Effects of harness-attached tracking devices on survival, migration, and reproduction in three species of migratory waterfowl. Anim Biotelemetry. 2018. https://doi.org/10.1186/s40317-018-0153-3.
31. Brlík V, Koleček J, Burgess M, Hahn S, Humple D, Krist M, Ouwehand J, Weiser EL, Adamík P, Alves JA, Arlt D, Barišić S, Becker D, Belda EJ, Beran V, Both C, Bravo SP, Briedis M, Chutný B, Ćiković D, Cooper NW, Costa JS, Cueto VR, Emmenegger T, Fraser K, Gilg O, Guerrero M, Hallworth MT, Hewson C, Jiguet F, Johnson JA, Kelly T, Kishkinev D, Leconte M, Lislevand T, Lisovski S, López C, McFarland KP, Marra PP, Matsuoka SM, Matyjasiak P, Meier CM, Metzger B, Monrós JS, Neumann R, Newman A, Norris R, Pärt T, Pavel V, Perlut N, Piha M, Reneerkens J, Rimmer CC, Roberto-Charron A, Scandolara C, Sokolova N, Takenaka M, Tolkmitt D, van Oosten H, Wellbrock AHJ, Wheeler H, van der Winden J, Witte K, Woodworth BK, Procházka P. Weak effects of geolocators on small birds: a meta-analysis controlled for phylogeny and publication bias. J Anim Ecol. 2020;89(1):207–20. https://doi.org/10.1111/1365-2656.12962.
32. Tomotani BM, Bil W, van der Jeugd HP, Pieters RPM, Muijres FT. Carrying a logger reduces escape flight speed in a passerine bird, but relative logger mass may be a misleading measure of this flight performance detriment. Methods Ecol Evol. 2019;10(1):70–9. https://doi.org/10.1111/2041-210X.13112.
33. Blackburn TM, Gaston KJ. The Distribution of Body Sizes of the World's Bird Species. Oikos. 1994;70(1):127–127. https://doi.org/10.2307/3545707.
34. Gaston KJ, Blackburn TM. The frequency distribution of bird body weights: aquatic and terrestrial species. Ibis. 1995;137(2):237–40. https://doi.org/10.1111/j.1474-919X.1995.tb03245.x.
35. Dunning JB. CRC handbook of avian body masses. 2nd ed. Baco Raton: CRC press; 2007. p. 657. https://doi.org/10.1201/9781420064452.
36. Holton MD, Wilson RP, Teilmann J, Siebert U. Animal tag technology keeps coming of age: an engineering perspective. Philos Trans Roy Soc B Biol Sci. 2021;376(1831):20200229. https://doi.org/10.1098/rstb.2020.0229.
37. Liechti F, Bauer S, Dhanjal-Adams KL, Emmenegger T, Zehtindjiev P, Hahn S. Miniaturized multi-sensor loggers provide new insight into year-round flight behaviour of small trans-Sahara avian migrants. Mov Ecol. 2018;6(1):19–19. https://doi.org/10.1186/s40462-018-0137-1.
38. Portugal SJ, White CR. Miniaturization of biologgers is not alleviating the 5% rule. Methods Ecol Evol. 2018;9(7):1662–6. https://doi.org/10.1111/2041-210X.13013.
39. Rappole JH, Tipton AR. New harness design for attachment of radio transmitters to small passerines. J Field Ornithol. 1991;62(3):335–7.
40. Analog devices. ADXL362 Micropower, 3-Axis, 2g/4g/8g Digital output MEMS accelerometer (2014).
41. Chen J, Brown G, Fudickar A. Simulation-based validation of activity logger data for animal behavior studies. Anim Biotelemetry. 2021;9(1):31. https://doi.org/10.1186/s40317-021-00254-y.
42. Arm Limited: CoreSight Architecture | Serial Wire Debug .2021. https://developer.arm.com/architectures/cpu-architecture/debug-visibility-and-trace/coresight-architecture/serial-wire-debug. Accessed 13 Mar 2021.
43. Openocd.org: Open On-Chip Debugger. 2021. http://openocd.org/. Accessed 31 May 2021.
44. stlink-org: Stlink-Org/Stlink . 2021. https://github.com/stlink-org/stlink. Accessed 31 may 2021.
45. STMicroelectronics: X-NUCLEO-LPM01A - STM32 Power Shield, Nucleo Expansion Board for Power Consumption Measurement (UM2243) - STMicroelectronics. https://www.st.com/en/evaluation-tools/x-nucleo-lpm01a.html.
46. Google: protocol buffers. Google Developers. 2016; 1–33.
47. Aimonen P. Nanopb - Protocol buffers with small code size. 2021. https://jpa.kapsi.fi/nanopb/. Accessed 21 June 2021.
48. Seiko Instruments: Seiko Instruments Inc. Micro Energy Division. 2021. http://www.sii.co.jp/en/. Accessed 25 June 2021.
49. Sjöberg S, Malmiga G, Nord A, Andersson A, Bäckman J, Tarka M, Willemoes M, Thorup K, Hansson B, Alerstam T, Hasselquist D. Extreme altitudes during diurnal flights in a nocturnal songbird migrant. Science. 2021;372(6542):646–8. https://doi.org/10.1126/science.abe7291.
50. STMicroelectronics: STM32CubeMonPwr - Graphical Tool Displaying on PC Power Data Coming from X-NUCLEO-LPM01A - STMicroelectronics. https://www.st.com/en/development-tools/stm32cubemonpwr.html. Accessed 18 Feb 2022.
51. KiCad: KiCad EDA. A cross platform and open source electronics design automation suite. https://www.kicad.org/. Accessed 3 Mar 2023.
52. openSCAD: OpenSCAD The Programmers Solid 3D CAD Modeller. https://openscad.org/. Accessed 3 Mar 2023.

## Publisher's Note