OPCIO: Optimizing Power Consumption for Embedded Devices via GPIO Configuration

XIAOYU JI, College of Electrical and Engineering, Zhejiang University, China
XINYAN ZHOU, Faculty of Electrical Engineering and Computer Science, Ningbo University, China
MIAO XU, University of South Carolina, USA
WENYUAN XU, College of Electrical and Engineering, Zhejiang University, China
YABO DONG, College of Computer Science and Technology, Zhejiang University, China

Battery lifetime is one of the main challenges that impedes the deployment of energy-constrained wireless networks, such as unattended Internet-of-Things (IoT) systems. To prolong battery lifetime, the duty-cycle mode is utilized in many IoT systems, especially in environment monitoring Wireless Sensor Networks (WSN) and Low-Power Wide-Area Networks (LPWAN). In duty-cycle mode, devices transmit packets during the active phase, which lasts for a short time, and sleeps the rest of the time. Prior research mainly focuses on energy efficiency in the active phase; energy consumption during the sleep phase, however, is always ignored, as it is assumed to have little margin to be optimized. In this work, we reveal that sleep phase can become a significant battery consumer due to the misconfiguration of General-Purpose Input/Output (GPIO) pins of micro-controllers. We propose `OPCIO`, which incorporates a genetic algorithm to obtain energy-efficient GPIO configurations automatically to squeeze the energy waste during the sleep phase. We prototype `OPCIO` on off-the-shelf devices and evaluate it on two ARM devices. Experiment results show that `OPCIO` can effectively find multiple low-power configurations that prolong the lifespans up to 10×.

CCS Concepts: • **Hardware → Wireless integrated network sensors**; **Power estimation and optimization**;

Additional Key Words and Phrases: IoT systems, automated GPIO configuration mechanism, low-power configuration

ACM Reference format:
Xiaoyu Ji, Xinyan Zhou, Miao Xu, Wenyuan Xu, and Yabo Dong. 2020. OPCIO: Optimizing Power Consumption for Embedded Devices via GPIO Configuration. *ACM Trans. Sen. Netw.* 16, 2, Article 16 (January 2020), 28 pages.
https://doi.org/10.1145/3373417

This work is supported by China NSFC Grant 61702451, ZJNSF Grant LGG19F020020, Zhejiang Provincial Natural Science Foundation of China under Grant No. LQ20F020012, and the Fundamental Research Funds for the Central Universities 2019QNA4027.
Authors' addresses: X. Ji and W. Xu (corresponding author), College of Electrical and Engineering, Zhejiang University, 38 Zheda Rd, Hangzhou, Zhejiang, 310027, China; emails: {xji, wyxu}@zju.edu.cn; X. Zhou (corresponding author), Faculty of Electrical Engineering and Computer Science, Ningbo University, 818 Fenghua Rd, Ningbo, Zhejiang, 315211, China; email: zhouxinyan@nbu.edu.cn; M. Xu, University of South Carolina, Columbia, SC; email: xum@email.sc.edu; Y. Dong, College of Computer Science and Technology, Zhejiang University, 38 Zheda Rd, Hangzhou, Zhejiang, 310027, China; email: ybdong@zju.edu.cn.

1550-4859/2020/01-ART16 $15.00
https://doi.org/10.1145/3373417

1 INTRODUCTION

With the rapid development of wireless sensing technologies and demands of modern smart lives, smart systems such as smart cities [35], smart homes [22], and smart industrial [23] become new trend applications for IoT systems. Deploying a large-scale IoT system is costly and time consuming. Not only because of deployment difficulties, users usually expect the system can last for years. For instance, large-scale IoT systems such as smart city systems [44] for smart parking [30], air quality monitoring [29], and traffic control [18] are composed of thousands of devices, which make the system expensive and hard to rebuild. Most of the aforementioned systems are deployed outdoors, and devices can only be powered by batteries. With constrained power supply, extensive research has been proposed to extend the lifespans for such systems.

There are two typical methods for energy saving: reducing the power consumption or decreasing the time period of the active phase. These two methods are based on a common understanding that devices consume much more energy in the active phase when compared with the sleep phase. As for the first method, routing protocols such as ring routing [40], EDAL [43] and SEECH [39] and MAC protocols [16, 17] such as CAD-MAC [33], E-BMA [37], and STAIRS [14] are all proposed to save energy by optimizing running mechanisms during the active phase. Besides these, designers tend to set devices to sleep mode once they finish tasks. Taking a smart parking system [30] as an example, devices in smart parking systems are only operated in the active phase when a car comes or goes, which leads to a result that over 99% of their lives are operated in the sleep phase. Other ultra-low-duty-cycle systems, like a fire-alarm system, may even be operated in the ultra-low-power mode for its entire life.

As discussed above, we find that only the active phase is considered for energy efficiency. However, our observations in some IoT systems reveal that reducing the power consumption of the sleep phase can achieve unexpected performance, especially when devices are operated in ultra-low-duty-cycle, i.e., the duty cycles of systems are less than 0.5%. In an ultra-low-duty-cycle system, the time duration of the sleep phase is a hundred times longer than the active phase, which makes the power consumption during the sleep phase impossible to ignore. However, without elaborate design, the sleep phase becomes a new candidate and leaves us a broad space for energy efficiency. Taking a duty-cycle system [25] as an example, devices are active for 0.75 second every 10 minutes (i.e., a duty cycle of 0.125%). Given a device whose current draw is 95 mA in the active phase and 180 μA in the sleep phase, decreasing the current draw from 180 μA to 100 μA in the sleep phase can extend the battery life by 36.6%. Thus, the power consumption during the sleep phase should also be taken into consideration.

A common misunderstanding is that operating a device in low-power mode guided by the datasheet can obtain an optimal power consumption during the sleep phase. However, we discover that devices can still consume a considerable amount of energy with the instruction of the datasheet, especially when peripheral modules are unknown. Essentially, an MCU contains several General-Purpose Input/Output (GPIO) pins that are used to drive and monitor peripherals, e.g., flash memories, LEDs, temperature sensors. In the sleep phase, an MCU operates in a low-power mode where it is still powered to sustain a timer for triggering the next active phase. Our observations based on smart devices in Reference [42] reveal that the current draw in sleep phases can range from about 10 μA to over 15 mA, depending on GPIOs' statuses, leading to more than a **1,500-fold** difference.

As noted above, configuring GPIOs to proper statuses seems promising to reduce a device's energy consumption in the sleep phase. However, minimizing a device's energy consumption in the sleep phase by searching the proper GPIO configuration is challenging due to various circuit schematics and external peripherals. First, different designs of devices have different peripheral components and numerous auxiliary circuits to ensure their functionality. Thus, the

Table 1. Working Currents of Typical Electronic Components in Embedded Devices

Module	Model	Current (active)	Current (sleep)
RF chips	CC2430	25 mA (TX)	0.3 μA
	CC2530	29 mA (TX)	0.3 μA
	AT86RF212	19 mA (TX)	0.2 μA
MCU	MSP430	1.32 mA	0.1 μA
DC-DC Converters	TPS8268x	7 mA	0.5 μA
	LMZ10501	6.5 mA	18 μA

configuration of GPIOs varies for different device designs. However, GPIO pins are interdependent when allocated to the same peripheral, which makes a general setting for all GPIO pins not applicable. To overcome these challenges, we propose OPCIO, which optimize power consumption of devices by configuration GPIO pins in the sleep phase automatically by utilizing the genetic algorithm. OPCIO is user-friendly with only the knowledge of the MCU type. Moreover, OPCIO requires almost no human intervention nor insightful knowledge on a node's peripherals and only needs to be executed once for each design of devices.

The contributions of our work include the following:

- We discover that for embedded systems working in ultra-low-duty-cycle, reducing the power consumption in the sleep phase is crucial to prolong their battery lives. To the best of our knowledge, this is the first work that focuses on reducing energy consumption in sleep phase. This observation provides a new angle for energy saving.
- Confirmed by our test data, we identify that configuring GPIOs to proper statuses is essential to reducing a device's energy consumption in sleep phases. Based on this observation, we propose OPCIO, an automated framework that optimizes power consumption for devices by configuring GPIO pins in sleep phases without much human intervention.
- We prototype and validate OPCIO and experiment results based on two devices, proving that OPCIO can decrease the power consumption in the sleep phase and extend system lifespans efficiently, especially for ultra-low-duty-cycle systems.

We organize the remainder of this article as follows: We first reveal the benefit of reducing the power consumption in the sleep phase and introduce the GPIO background in Section 2. We then provide an overview of OPCIO in Section 3 and describe the details in Section 4 and Section 5. We implement OPCIO in Section 6 and evaluate the performance of OPCIO in Section 7. Related works are discussed in Section 8, and we conclude our work in Section 9.

2 BACKGROUND AND MOTIVATION

In this section, we first investigate the performance of reducing energy consumption in sleep phases and then provide a brief introduction to the GPIOs of MCUs and reveal their impacts on the energy consumption in sleep phases.

2.1 Power Consumption for Ultra-low-duty-cycle Systems

Reducing power consumption is a long-term issue, as we described above. For a device in IoT systems, which is usually an embedded device, electronic components consume different magnitudes of energy when working in the active mode and the sleep mode. Here, we list the working currents of essential electrical components of embedded devices in Table 1. Three typical RF (radio

Table 2. The Power Consumption and Time Duration in Active and Sleep Phases for Typical Ultra-low-duty-cycle Systems

system	T_{active}	I_{active}	T_{sleep}	I_{sleep}	duty cycle (%)	P_{active}(%)	P_{sleep}(%)
Soil Moisture Sensing [25]	0.75 s	95 mA	600 s	180 μA	0.125%	39.75%	**60.25%**
UP-Link [21]	50 ms	13.6 mA	3600 s	1.5 μA	0.0014%	11.24%	**88.76%**
IT-WSN [42]	1 s	28.26 mA	900 s	19.8 μA	0.111%	61.31%	**38.69%**
Smart Gas Monitoring [13]	1 s	55.8 mA	10000 s	8 μA	0.01%	41.09%	**58.91%**

frequency) chips in Table 1 reveal that working in the active mode (TX or RX) consumes over 80K times more energy than in the sleep mode. Thus, operating MCUs and radios in duty cycles [13, 15, 21, 25, 42], where devices become active for a short period to finish tasks and sleep as much as possible, becomes an efficient method to extend the lifespan of the system.

The average power consumption ($Avg\_\mathcal{W}$) of a device operated in duty-cycle can be presented as Equation (1), where I_{avg} can be calculated by Equation (2):

$$Avg\_\mathcal{W} = I_{avg} \cdot V, \tag{1}$$

$$I_{avg} = I_{active} \cdot duty\_cycle + I_{sleep} \cdot (1 - duty\_cycle). \tag{2}$$

V is the working voltage, which is usually a constant, e.g., 3 V. Generally, I_{active} is in milliampere level and I_{sleep} is in microamp level. A common misunderstanding is that the power consumption during the sleep phase is so small that it can be ignored. However, this cognition is far from the truth when devices are operated in ultra-low-duty-cycle.

Nowadays, modern smart devices are often operated in ultra-low-duty-cycle, i.e., the duty cycles of systems are less than 0.5%. For example, LPWANs are required to have a maximum duty-cycle as 1% by ETSI [36]. According to the Global Market Insights [4], the LPWAN platforms held a major market share of over 70% in 2018, and the size of LPWAN may exceed USD 65B by 2025. In this case, the total power consumption during the sleep phase is considerable when compared with the active phase.

Table 2 illustrates the power consumption and time duration during active and sleep phases of four ultra-low-duty-cycle systems: soil moisture sensing system [25], UP-Link [21], IT-WSN [42], and smart gas monitoring system [13]. These four systems cover various application scenarios, including environmental monitoring, body WSN, wildlife tracking, and disaster detection. In these cases, the power consumptions during the sleep phase are all less than 1 mA. In the soil moisture monitoring system, the sleep phase consumes over 60% of the energy, which is much higher than the active phase. Thus, even a small decrease of the energy used in sleep cycles is crucial to reduce the overall consumption.

To have a deep understanding of the impacts on reducing the power consumption during the sleep phase, we further analyze the data in Table 2. We calculate the lifespan of each system with a decreased power consumption during active phase and sleep phase (70% and 50% of the original ones) individually, and the results are shown in Figure 1. We find that when decreasing the same degree of power consumption, the sleep phase can expand the lifespan even longer in most of the cases (except the IT-WSN case). For example, in the UP-Link system, the lifespan is 2.465 years with 50% power consumption in the sleep phase while this value is only 1.453 years in the active phase.

Based on the above observations, we conclude that reducing power consumption in the sleep phase is promising on extending the lifespans of systems. To the best of our knowledge, this is the first research that is proposed to reduce the power consumption during the sleep phase and may provide a new angle for energy saving.

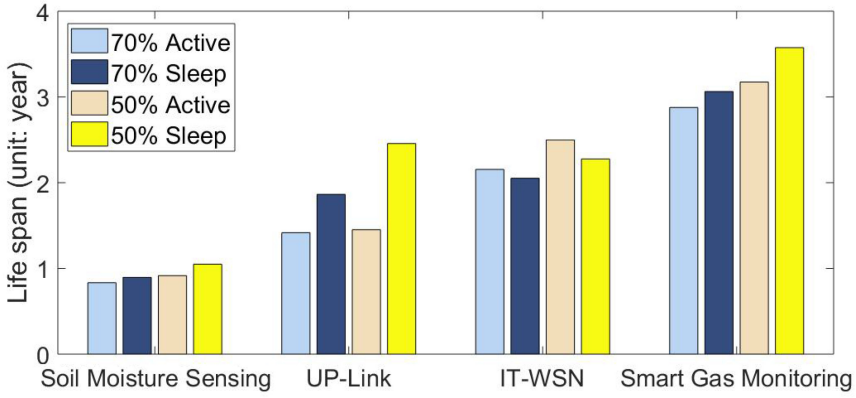


Fig. 1. The system lifespans with the decreased power consumption during active or sleep phase (Soil Moisture Sensing, UP-Link, IT-WSN, and Smart Gas Monitoring systems).

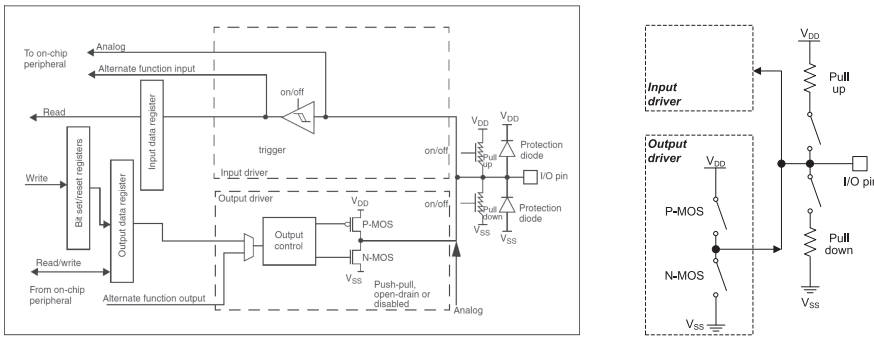
2.2 Optimizing GPIO Configurations

Modules consume far less energy when operated in the sleep mode (low-power mode) while peripheral modules are turned off and the MCU is set in low-power mode. However, we discover that devices still consume a considerable amount of energy even if all modules are turned off, due to the improper configurations of GPIOs (general purpose Input/Output). To understand the reasons behind this, we first have a close look at GPIOs on MCUs.

2.2.1 GPIOs on MCUs. Though different MCUs have a various number of pins (e.g., 48, 64, 128), they can be categorized as two types: the ones with dedicated functions (e.g., connecting to batteries) and the ones with general purpose, which are called general-purpose input/output (GPIO) pins. Unlike those pins with dedicated functions, GPIOs can be controlled by users and typically are used to drive and monitor peripherals. For example, one GPIO may be used to read measurement values from temperature sensors, while another may be used to write the values to a flash. To illustrate the property and the power consumption of GPIOs during the sleep phase, we take STM32L151C8, an MCU in the STM32 family, as an example. Over 20B STM32 chips have been embedded in smart devices during the past decade, which makes them good representative MCUs for analysis. In the following, we will provide more details.

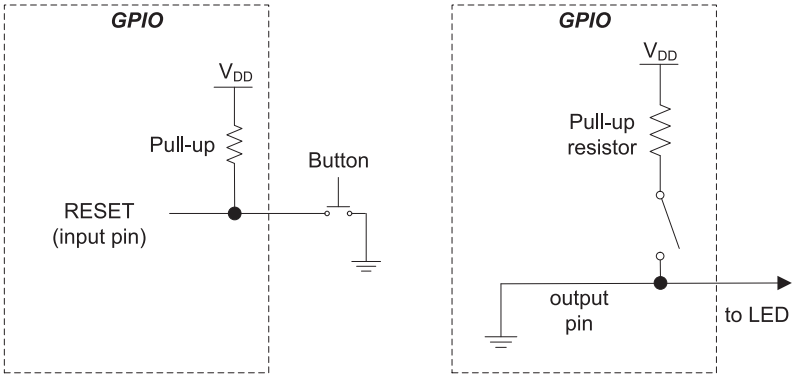
To illustrate, we show the structure of a GPIO in the 48-pin MCU in Figure 2(a) and show a simplified version in Figure 2(b). In general, a GPIO pin can be configured as either `input` or `output`. A GPIO pin has an input driver and an output driver, each of which is enabled when the pin is programmed as input or output, respectively. Once the GPIO is set as input, the voltage level (either low or high) presented on that pin is read into the MCU's input data registers. If configured as an output, it can source different logic voltages (e.g., 1 for high and 0 for low). Essentially, a GPIO pin contains two transistors (P-MOS and N-MOS) and two resistors (pull-up and pull-down resistors). In a digital circuit, a pull-up resistor refers to a resistor that is connected towards its voltage source, and a pull-down resistor is the one connected to the ground. Pull-up and pull-down resistors are used to prevent floating, i.e., an unknown state of a pin when it is not connected to any external devices.

To control the two transistors, a GPIO pin can be configured either as Push-Pull (PP) or Open-Drain (OD). PP can activate either P-MOS or N-MOS: "PP:0" activates the N-MOS, leading to a low voltage output, whereas "PP:1" activates the P-MOS, leading to a high voltage output. A pin configured as open-drain can only output a low voltage level or be in a floating status, because



(a) Basic structure (b) Simplified structure

Fig. 2. The structure of a GPIO pin in an STM32L151C8 [38].



(a) Configure a GPIO as input to read the voltage.

(b) Configure a GPIO as output low voltage to turn off the LED.

Fig. 3. The configuration examples for GPIO pins.

P-MOS is never activated: "OD:0" activates the N-MOS, leading to ground the pin, whereas "OD:1" leaves the I/O pin in floating status, which means that the I/O pin is neither driven to a high nor low level.

The built-in pull-up and pull-down resistors are enabled at the running time and can function when a pin is set both as input and output. Both resistors are controlled by Pull-Up (PU) and Pull-Down (PD) registers. Figure 3(a) illustrates an example of a reset function: The pin is set as input and configured as the pull-up mode to keep itself at a high level. Once the button is pressed, the pin is grounded, and the input pin reads a low level and the MCU is reset.

To summarize, by combining push-pull/open-drain and pull-up/pull-down, a GPIO pin of STM32L151C8 can be configured as one of the 16 modes listed in Table 3.

2.2.2 GPIO Configuration vs. Power Consumption. A GPIO's status determines the behavior of the connected peripheral. In an active phase, GPIOs must be configured to satisfy functional requirements. For instance, the pin that drives an LED must be programmed as output to turn on or turn off the LED. However, in sleep cycles, the goal of GPIO configuration is no longer to sustain peripherals' functionality but to reduce their energy consumption.

The configuration options in sleep cycles are more than the ones in active cycles, and each of them may lead to different energy consumption. We should also note that turning peripherals off

Table 3. Summary of GPIO Configuration Modes in STM32L151C8

Mode	Comments	Mode	Comments
(0) PP:0	push-pull/float, write 0	(1) PP:1	push-pull/float, write 1
(2) PP:PU:0	push-pull/pull up, write 0	(3) PP:PU:1	push-pull/pull up, write 1
(4) PP:PD:0	push-pull/pull down, write 0	(5) PP:PD:1	push-pull/pull down, write 1
(6) OD:0	open-drain/float, write 0	(7) OD:1	open-drain/float, write 1
(8) OD:PU:0	open-drain/pull up, write 0	(9) OD:PU:1	open-drain/pull up, write 1
(10) OD:PD:0	open-drain/pull down, write 0	(11) OD:PD:1	open-drain/pull down, write 1
(12) IN:FLOAT	input float	(13) IN:PU	input pull-up
(14) IN:PD	input pull-down	(15) AN	analog input

Table 4. The Power Consumption of a Device with Different GPIO Configurations (Pin 15)

Mode of pin 15	Power consumption	Mode of pin 15	Power consumption
mode 0	**9.7 μA**	mode 1	9.8 μA
mode 2	**84.4 μA**	mode 3	9.7 μA
mode 4	9.7 μA	mode 5	83.2 μA
mode 6	10.1 μA	mode 7	21.8 μA
mode 8	84.1 μA	mode 9	10.1 μA
mode 10	10.1 μA	mode 11	11.2 μA
mode 12	21.5 μA	mode 13	11.4 μA
mode 14	13.8 μA	mode 15	11.1 μA

directly does not always mean zero power consumption. Finding the right configuration is tricky, because two modes may appear to output the same voltage level but will induce different energy. For example, both mode 0 (PP:0) and mode 2 (PP:PU:0) output a low level. However, mode 2 will consume more energy, because extra current will flow between V_{DD} and ground through the pull-up resistor. This extra power is not noticeable in an active cycle when an MCU is busy. However, it could be a major source of energy waste in sleep cycles.

However, different configurations for transistors also reveal different power consumption conditions. Specifically, the power consumption for the transistor contains three components, including static power dissipation, short-circuit power dissipation, and dynamic power dissipation. The power consumption for the transistor can be calculated as follows:

$$P_{transistor} = P_{static} + P_{short-circuit} + P_{dynamic}. \tag{3}$$

P_{static} is a constant energy cost, which is the leakage current of the transistor. $P_{short-circuit}$ is introduced when both transistors are on at the same time. The switching of the transistor's gate can lead to additional power consumption $P_{dynamic}$. Therefore, different modes of GPIO pins set transistors in different conditions, which leads to various power consumption eventually.

To validate this, we conduct an experiment to reveal the current draws in sleep phases while setting the GPIO pin in Figure 3(b) in different status. The GPIO pin in Figure 3(b) is utilized for driving an LED. We change the mode of the pin from mode 0 to mode 15 (Table 3) while keeping other pins the same and measuring the current draw, respectively. The results are shown in Table 4. During the sleep phase, the current draw of mode 2 is 84.4 μA, whereas the current draw of mode 0 is 9.7 μA. Such a difference (around 75 μA) is non-negligible for a low-power device. The reason behind this is that a pull-up resistor will be connected when we set the GPIO pin in Figure 3(b) in mode 2, which generates an additional current and leads to necessary energy consumption.

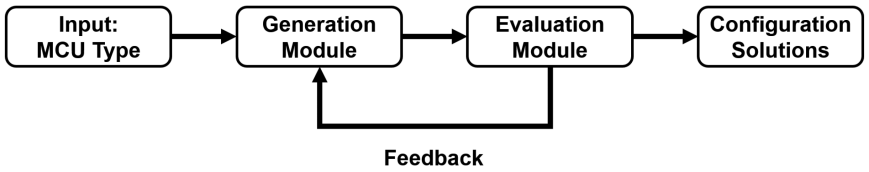


Fig. 4. The overview of OPCIO. OPCIO consists of a generation module and an evaluation module, and the output of the evaluation module can be the feedback to the generation module.

However, configuring the right GPIO status requires a deep grasp of GPIO internal structure and peripheral devices, which is exactly what this article aims to avoid. Granted that engineers with abundant experience may find the low-power GPIO configuration for their devices by carefully examining the schematics, developers with limited hardware knowledge (e.g., application developers) may not be capable to achieve the goal. Thus, we propose OPCIO, which can help software developers find the right configuration that maps to small energy consumption in the sleep phase automatically. In particular, we envision that OPCIO is useful in the following cases:

- OPCIO can help software developers, particularly those with little hardware knowledge, to quickly find out low-power configurations in sleep cycles without much intervention with hardware;
- OPCIO can help embedded device designers to review the platform, because possible design faults can make it tricky to configure GPIO pins to achieve low-power requirement;
- OPCIO can be a useful tool for hardware manufacturers to discover the default configuration for the GPIO pins in their SDKs or micro operating systems for smart devices.

3 OPCIO OVERVIEW

In this section, we first provide a brief introduction of OPCIO, which optimize power consumption of devices by configuration GPIO pins in the sleep phase automatically. Then, we describe the architecture of OPCIO and give definitions to some terms for further illustration.

3.1 An Introduction of OPCIO

OPCIO aims at providing low-power GPIO configuration solutions for devices during sleep phases, and therefore expands device lifespans eventually. Granted that engineers with abundant experience may find the low-power GPIO configuration for their devices by carefully examining the schematics, OPCIO provides developers with limited hardware knowledge (e.g., application developers) an easier way to search for the low-power GPIO configuration solutions. The users of OPCIO should at least be aware of the type of target device's MCU. This is the only requirement for users when using OPCIO.

It is challenging to design a universal GPIO configuration mechanism for diverse devices. First, OPCIO should be capable of adapting different platforms while devices in IoT systems are embedded with variant MCU chips and operating systems. Second, OPCIO should elaborately select GPIO configurations in an efficient way instead of attempting all possible solutions. Moreover, obtaining the power consumption of devices by reading register directly is infeasible, while additional function may cause extra power consumption. Thus, we need to design a module to evaluate the power consumption for different GPIO configurations.

To address the above challenges, we design OPCIO with four modules, as shown in Figure 4. OPCIO is composed of a generation module and an evaluation module. Users first input the type of MCU to the generation module. The generation module then generates candidate configuration solutions for the evaluation module. The candidate configuration solutions are the possible GPIO

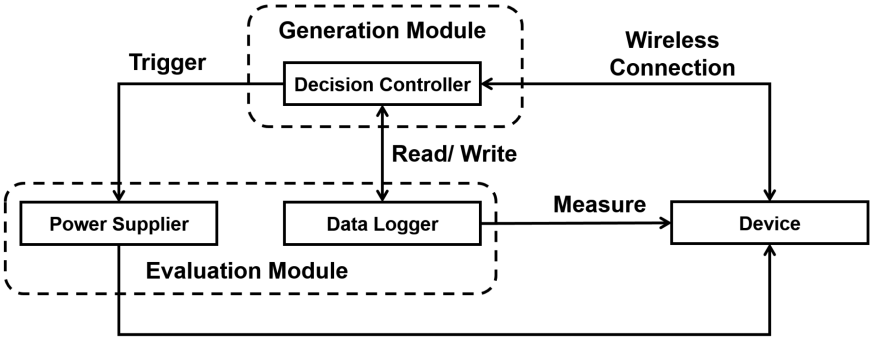


Fig. 5. The architecture of OPCIO.

configurations for devices, which we called "profiles" in the following. The evaluation module measures the power consumption of the profiles provided by the generation module. The measurement results can work as feedback to guide the generation module to generate other profiles. After evaluating all candidate profiles, OPCIO outputs the low-power GPIO configuration profiles for users.

OPCIO is designed for cross-platform devices. Different devices could be embedded with different MCU chips, and these MCU chips have different numbers of GPIO pins with variant structures. OPCIO needs the GPIO information (including the number of GPIOs and the corresponding pin sequences) as the input for the generation module. Therefore, we require users to manually provide the MCU type, and OPCIO can search for the GPIO information online. With only the GPIO information, OPCIO can be generalized to devices with variant hardware and platforms. With the knowledge of the GPIO information about devices, OPCIO generates candidate low-power configuration profiles for further evaluation. The evaluation module evaluates profiles' power consumptions during the sleep phase. The evaluation phase is time-consuming, because each evaluation requires deploying a GPIO configuration to a device and measuring its power under that configuration. Thus, reducing the number of configuration profiles is the key to increasing the efficiency of OPCIO. We select GA (genetic algorithm) as profile generation algorithm, and the details will be discussed in Section 4.

3.2 The Architecture of OPCIO

The architecture of OPCIO can be seen in Figure 5. A power supplier provides power to a target device, and a data logger measures the device's power consumption in real-time. The measurement is uploaded to a decision controller for further evaluation. The decision controller generates the candidate GPIO configuration profiles for devices. It also decides when to trigger the power supplier to power on the device and when to trigger the data logger to monitor the power consumption. It is worth mentioning that the decision controller transmits the GPIO configuration profiles to the device wirelessly. If we connect the target device with the decision controller through serial cables, we may introduce uncontrollable extra power consumption through the wire, which may lead to inaccurate measurement results. To avoid such a consequence, the decision controller communicates with the target device wirelessly.

Besides communicating with the device and controlling the power supply and the data logger, the decision controller is also responsible for the *generating* procedure: it selects top-ranked candidate profiles and derives the next generation. In our design, we choose GA to perform selection and derivation. GA is based on the evolutionary principles in natural systems where competition among individuals for scant resources results in the fittest individuals dominating the weaker

ones. They exploit historical information to direct the search into the region of better performance within the search space. More details are provided in Section 5.

3.3 Terms Definition

Here, we define the terms used throughout this article:

- An ***instruction*** c_i refers to a status of the ith GPIO. The possible statuses are denoted by $[0, \ldots, m-1]$, where m is the number of pin statuses.
- A ***profile*** p is an ordered list of instructions, i.e., $p = (c_1, \ldots, c_n)$, where n is the number of an MCU's GPIOs.
- A ***generation*** G_k is a set of candidate profiles that is evaluated by `OPCIO` in the kth round, i.e., $G_k = \{p_1, \ldots, p_N\}$, where N is the number of profiles in a generation.
- A ***solution*** p_{min} refers to the observed profile that consumes the least energy after running a search algorithm *one time*. Its power is denoted by P_{min}.

Notice that the definition of a solution p_{min} is different from the *minimized solution*, which is the profile that consumes the least energy *over all profiles*. Thus, the minimized solution is independent of search algorithms. To differentiate them, we denote the minimized solution and its power by $\hat{p}_{min}$ and $\hat{P}_{min}$. Running a search algorithm always gives us a solution, but it is not necessary to be the minimized solution. The difference between P_{min} and $\hat{P}_{min}$ plays an important role to judge the performance of a search algorithm.

4 GENERATION MODULE

The generation module provides profiles for analysis, and the efficiency of the generation module impacts the performances of the `OPCIO`. In this section, we first describe the reasons why we select GA as the profile generation algorithm and then illustrate the GA-based scheme in detail.

4.1 The Selection of Profiles Generation Algorithm

By selecting a proper profiles generation algorithm, we can search the low-power GPIO configuration in an efficient way. A naive brute force method can always find the profile with the minimum power consumption ($\hat{p}_{min}$, $\hat{P}_{min}$), but it is time-consuming. Given a 48-pin MCU that has 37 GPIO pins and each pin has 16 statuses (e.g., STM32L151C8), the search space of a brute force algorithm is 16^{37}. Thus, a greedy algorithm and heuristic algorithms become promising in our scenario.

4.1.1 Greedy Algorithm. Ideally, a greedy algorithm that adjusts GPIO pins one-by-one to find the configuration that consumes the least energy is the most efficient method, and we implement the greedy algorithm in Algorithm 1.

As shown in Algorithm 1, the greedy algorithm first generates a random initial profile by function `RandomProfile()`, which means generating a random mode for each GPIO pin. After that, the greedy algorithm then traverses all modes (from the first mode to the mth mode) for each GPIO pin (`replace`(p_{min}, c_i, j)) and measures the power consumption of each profile (`measure`(p)). For each GPIO pin, the greedy algorithm replaces the mode of GPIO pin as the one with the least power consumption. After all GPIO pins are traversed, we can get a relative minimum profile p_{min} and its power consumption P_{min}.

It is worth mentioning that the efficiency of the greedy algorithm is based on an assumption that all GPIOs are independent. Unfortunately, though GPIO pins for a specific MCU usually share the same structure, these GPIO pins are not independent in most cases. The reason is that a peripheral may be connected to multiple GPIOs at the same time, e.g., the radio chip in a temperature sensing

Table 5. The Lowest Current Draws for a Temperature Monitoring Device (MCU: STM32L151C8) Found by OPCIO with Greedy Algorithm

	Round 1	Round 2	Round 3	Round 4	Average
Greedy Algorithm	32.4 μA	403.6 μA	509.6 μA	10.7 μA	209.075 μA

ALGORITHM 1: Find a profile with greedy algorithm

Input: n: number of GPIOs; m: number of configuration modes
Output: p_{min}: the solution; P_{min}: the power consumption of p_{min}

```
p_min = RandomProfile(n,m,1);                    /* generate one profile randomly */
P_min = measure(p_min);                          /* measure its power */
for i = 1 ... n do                               /* for each GPIO */
    for j = 0 ... m − 1 do                       /* for each mode */
        p = replace(p_min, c_i, j);              /* replace c_i with j */
        P = measure(p);
        if P < P_min;
        then
            p_min = p;
            P_min = P;
        end
    end
end
return (p_min, P_min)
```

device [42] is driven by eight GPIOs. The circuit schematic is usually complex and different among peripherals, which leads to dependencies within GPIO pins.

To validate our hypothesis that the greedy algorithm cannot always find the solution with the minimum power consumption, we ran the greedy algorithm four times on a temperature monitoring device. The results of the lowest current draws found by the greedy algorithm for each round are shown in Table 5. According to the experiment results, it takes four rounds for the greedy algorithm to reach an acceptable profile, and the average power consumption of the best solutions in each round is 209.075 μA. We find that the best profile provided by the greedy algorithm is highly related to the initial profile, which is randomly generated. The experiment results confirm that the greedy algorithm does not converge consistently, and the greedy algorithm should not be selected as the profiles generation algorithm.

4.2 Heuristic Algorithms

Heuristic algorithms try to give feasible profiles with acceptable costs. In most of the cases, the feasible solution can be close enough to the optimal solution. Popular heuristic algorithms include Genetic Algorithm (GA), Artificial Neural Networks (ANN), and Simulated Annealing (SA). Among these three algorithms, we would like to choose the GA for the following reasons:

- It is hard to model the power consumption optimization problem into ANN while the corresponding relationships between the weights of the network and the GPIO modes are not clear.
- Though SA may converge to a better solution than the GA, GA has a faster convergence speed than SA with a small error to the optimal solution [34].
- GA can be adaptive to scenarios with a large number of parameters, thus it can be easily applied in our problem where the numbers of GPIO pins vary from tens to hundreds.
- GA can provide multiple solutions that are close to the optimal solution.

ALGORITHM 2: Find profiles with genetic algorithm

Input: n: number of GPIOs; m: number of configuration modes; K: number of generations; L: number of profiles in a generation
Output: p_{min}: the solution; P_{min}: the power consumption of p_{min}

```
G_0 = RandomProfile(n,m,L);          /* generate L profiles for the first generation */
for i = 0 ··· K − 1 do                                          /* for each generation */
    for j = 0 ··· L/2 do                                        /* crossover and mutation */
        G'_i = (p'_j, p'_{j+L/2}) ← (p_j, p_{j+L/2});           /* generate G'_i, an offspring set */
    end
    P_i = measure (G_i ∪ G'_i)                  /* measure the power consumption of G_i and G'_i */
    P_min(i) = min{P_i};                /* the lowest power consumption in ith generation. */
    p_min(i) = search (P_min(i));   /* search the profile with the lowest power consumption. */
    G_{i+1} = search(top(sort(P_i, L));      /* generate G_{i+1} by sorting the power consumption */
end
P_min = min{P_min(1), P_min(2), ..., P_min(K)};
p_min = search (P_min);
return (p_min, P_min)
```

Based on the aforementioned analysis, we would like to choose GA as the profile generation algorithm. We describe GA in detail in the following.

4.3 GA-based Scheme

GA is inspired by natural evolutionary theory, and it can generate high-quality profiles for optimization and search problems.

We implement OPCIO with GA algorithm in Algorithm 2. GA first randomly generates a set of candidate profiles, which is also called the first generation (G_0). Then, GA expands the search space in a genetic way and it mimics the natural evolutionary process by crossover and mutation. After this step, OPCIO obtains an offspring set of the current generation (G'_i). In general, a fitness function is required in GA to evaluate the performance of each profile. In our case, the fitness function is the power consumption under profiles. OPCIO derives the next generation by selecting the top solutions and then it repeats the steps above till the last generation.

Many genetic algorithms have been proposed, and we employ NSGA-II (Non-Dominated Sorting Genetic Algorithm II) [9] in OPCIO, because NSGA-II is known to provide a better spread of solutions and better convergence than many others.

In the following, we describe how to derive G_{k+1} from G_k in detail.

(1) **Encoding.** A profile is represented as $p = (c_1, \ldots, c_n)$, where n is the number of an MCU's GPIOs. GA encodes profiles in binary for further processing. Taking an MCU with 8 GPIO pins as an example, if each GPIO pin contains 16 optional modes, we need 4 bits to represent each mode in binary, i.e., "0000-1111." For a profile:

$$p = (15, 8, 3, 14, 3, 2, 0, 1).$$

It can be encoded as follows:

$$1111|1000|0011|1110|0011|0010|0000|0001.$$

(2) **Crossover and mutation.** During this procedure, the decision controller picks two profiles as a profile-pair from G_k. There are $L/2$ pairs of profiles in a generation, where L is the number of profiles in a generation. Each pair is considered as parents, and they produce

two offsprings by crossing over and mutating their "chromosomes," i.e.,

$$(p_i', p_j') \leftarrow (p_i, p_j),$$

where p_i' and p_j' are offsprings of p_i and p_j.

We use simulated binary crossover (SBX) [3] and polynomial mutation (PM) [8] in OPCIO, because they have been proved to be the most efficient methods that work with NSGA-II [9]. Both stimulated binary crossover and polynomial mutation are widely used in genetic algorithms, and here we give an example of the stimulated binary crossover [3] and polynomial mutation [8] in detail.

Given a pair of parents as p_1 and p_2, and their offsprings after SBX as p_1' and p_2', in SBX, we introduce a spread factor β, and β is defined as:

$$\beta = \left| \frac{p_2' - p_2'}{p_2 - p_1} \right|. \tag{4}$$

Therefore, we can obtain:

$$\begin{aligned} p_1' &= 0.5 * [(1+\beta)p_1 + (1-\beta)p_2], \\ p_2' &= 0.5 * [(1-\beta)p_1 + (1+\beta)p_2]. \end{aligned} \tag{5}$$

The probability distribution of β can be represented as follows:

$$\mathcal{P}(\beta) = \begin{cases} 0.5(\eta+1)\beta^{\eta} & \beta \le 1, \\ 0.5(\eta+1)\frac{1}{\beta(\eta+1)} & otherwise, \end{cases} \tag{6}$$

where η is any nonnegative real number that can be selected by users, and we empirically select η as 2 in OPCIO. To obtain the value of β in each crossover, we introduce $\mu \in \mathbf{U}(0,1)$, and we derive the specific β as:

$$\beta = \begin{cases} (2\mu)^{\frac{1}{\eta+1}} & \mu \le 0.5, \\ (\frac{1}{2(1-\mu)})^{\frac{1}{\eta+1}} & otherwise. \end{cases} \tag{7}$$

Based on the above procedures, we are able to derive p_1' and p_2' from p_1 and p_2 after simulated binary crossover.

In the following, we describe the process of obtaining a mutated profile p' from a parent profile p after polynomial mutation. In the polynomial mutation, a user-defined index parameter $\eta - m$ is introduced as the polynomial mutation operator. In OPCIO, we empirically select η_m as 50, and p' can be obtained by:

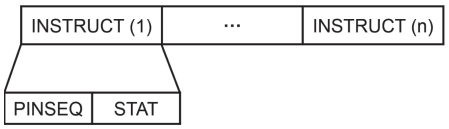
$$p' = \begin{cases} p + \overline{\delta_L}(p - p_l) & \mu \le 0.5, \\ p + \overline{\delta_R}(p_u - p) & otherwise, \end{cases} \tag{8}$$

where $\mu \in \mathbf{U}(0,1)$, p_l and p_u are profiles that all GPIO pins are configured in mode 0 or mode 15. $\overline{\delta_L}$ and $\overline{\delta_R}$ can be calculated as:

$$\begin{aligned} \overline{\delta_L} &= (2\mu)^{\frac{1}{\eta m+1}} - 1, \mu \le 0.5, \\ \overline{\delta_R} &= 1 - \frac{2(1-\mu)}{1+\eta_m}, otherwise. \end{aligned} \tag{9}$$

After crossover and mutation, we finally obtain an offspring set:

$$G_k' = \{p_1', \ldots, p_N'\}.$$



(a) The structure of a profile.

```
PINSEQ   ← 1 | 2 | ... | n
STAT     ← (in: INMODE) | (out: OUTVAL | OUTMODE)
INMODE   ← STRING
OUTVAL   ← low | high
OUTMODE  ← STRING
```

(b) The value of symbols in the profile.

Fig. 6. A profile is composed of multiple instructs for GPIO pins, and it is extensible for different microcontrollers. For each instruct, a sequence number and the state of the GPIO pins are contained.

The idea behind this crossover and mutation is to mimic the process in natural evolution: an offspring's chromosomes are inherited from her parents and may suffer occasional gene alteration.

(3) **Fitness measurement.** In `OPCIO`, we select the power consumption of profiles as the fitness function. Combining the G_k and G'_K, we then measure the power consumption of profiles in the union set $G_k \cup G'_k$, i.e.,

$$P_k = \texttt{measure}(G_k \cup G'_k).$$

(4) **Sorting and selection.** We then sort the power consumption of profiles in G_k and G'_K in ascending order, i.e.,

$$\texttt{sort}(G_k \cup G'_k).$$

Based on the sorted union set, the decision controller selects the top-ranked L winners as the next generation G_{k+1}:

$$G_{k+1} = \texttt{top}(\texttt{sort}(G_k \cup G'_k), L).$$

Once G_{k+1} is produced, each profile in G_{k+1} is deployed to the node and its power is measured by the steps in Section 5.

5 POWER CONSUMPTION MEASUREMENT OF PROFILES

Profiles with less power consumption could be saved for the next generation in GA. In this section, we first introduce the structure of a profile, and then we describe how to measure the power consumption of a profile in detail.

5.1 Structure of Profiles

To let a device know how to configure its GPIOs, we define the structure of a profile in Figure 6. As shown in Figure 6(a), a profile is an ordered list of instructions for each GPIO pin. An instruction contains the sequence number of a GPIO pin and the configuration status of the GPIO pin. The configuration status of a GPIO pin is shown in Figure 6(b), which includes input/output state and output logical value (if set in output state). For example, if the MCU's second pin is programmed as the output state sourcing high voltage value, then the instruction for that pin can be denoted as `2-out:high`. If a GPIO also has other configuration modes (e.g., pull-up mode and pull-down mode in Table 3), they can be defined in INMODE and OUTMODE. The user can customize the profile structure by defining their own INMODE and OUTMODE, depending on the MCU model that they are using. Therefore, `OPCIO` is a framework that can be extended to different MCU models. In Section 7, we will show how to extend `OPCIO` by looking at two examples.

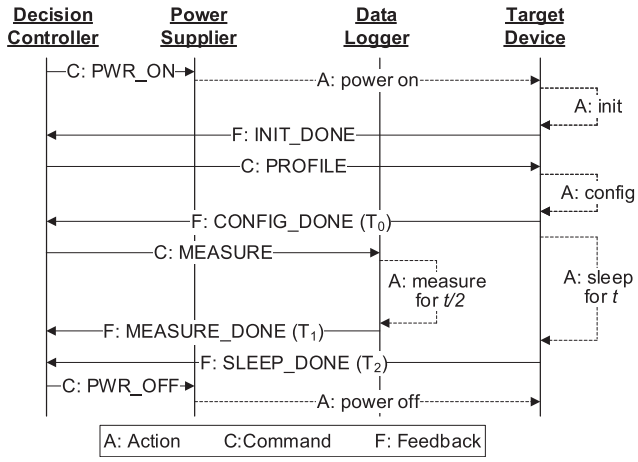


Fig. 7. Power measurement in a sleep cycle.

5.2 Power Measurement Walk-through

As we mentioned above, OPCIO contains a decision controller, a power supplier, a data logger, and a target device. Figure 7 reveals the power measurement diagram for a *single* profile. We define three primitive operations in the measurement procedure:

- **Command (C).** Triggered by the decision controller, commands are used to ask other components to take actions. For example, it can trigger the power supplier to turn on the target device.
- **Action (A).** A component must take an action once it receives a command from the decision controller. For example, the power supplier turns on the power.
- **Feedback (F).** A component must provide feedback to the decision controller that it has finished an action. For example, the data logger sends T_1 after its power measurement. Similarly, the target device sends T_2 after it wakes up.

As shown in Figure 7, the controller first sends a PWR_ON command to the power supplier to turn on the target device. After the power-on initialization, the device notifies the controller with a feedback message INIT_DONE. Then, the controller transmits a profile to the device. The device configures its GPIOs based on the received profile, sends a feedback message CONFIG_DONE to the controller, and then turns to sleep mode for t seconds. The controller sends a MEASURE command and requests the data logger to start power measurement for $t/2$ seconds. The data logger uploads the instant power in real time and sends a message MEASURE_DONE when it finishes the measurement. Once the node stays in the sleep mode for t seconds, it notifies the controller with a message SLEEP_DONE, and the controller triggers the power supplier to turn off the device. If the decision controller has any other profile that has not been deployed yet, it will repeat the same procedure for the next available profile.

During this process, we must ensure that the power measurement happens within a node's sleep mode. To achieve the goal, the controller records the timestamps of the following three messages, as shown in Figure 7:

(1) T_0: the target device notifies the decision controller that it has finished configuration and goes to sleep;
(2) T_1: the data logger notifies the decision controller that it has finished power measurement;

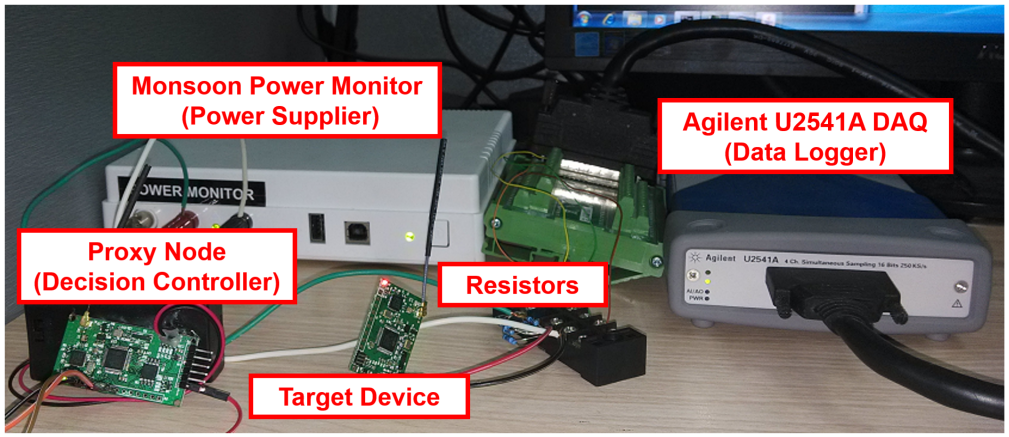


Fig. 8. Devices used for OPCIO implementation.

(3) T_2: the target device notifies the decision controller that it has woken up from a sleep cycle.

If $T_1 - T_0 > t/2$ and $T_1 < T_2$, we can confirm that the measurement happens during a sleep cycle.

6 IMPLEMENTATION

In this section, we implement OPCIO based on off-the-shelf devices, as shown in Figure 8. The system uses Monsoon Power Monitor [27] as the power supplier and Agilent U2541A DAQ [2] as a data logger. We put a resistor in serial with the device to calculate the node's power by

$$P = V_n \times \frac{V_r}{R},$$

where V_n and V_r are the voltages of the device and the resistor, respectively, and R is the resistance of the resistor.

The value of the resistor is important, because it is actually a voltage divider for the device. For example, a 50Ω resistor causes a $2V$ voltage drop when the current draw is $40\,mA$, which will lead to insufficient voltage for driving the device. In our experiment, we set the resistance of the resistor as 10Ω.

We use a PC as the decision controller to send commands to the proxy node and the target device. A direct approach to configure the GPIOs' status for a target device is connecting the device to the PC via a serial cable. However, extra power consumption through the serial cable is introduced, which impacts the result for the OPCIO. Thus, we utilize a proxy node that connects with the PC to help to deploy profiles. The proxy node receives profiles from the PC via a serial cable and forwards them to the target device wirelessly. It also picks up the device's feedback in the air and reports the feedback to the PC via the serial cable.

To generate and evaluate profiles, we implement the genetic algorithm based on jMetal [1], a multi-objective optimization tool written in Java. It provides APIs for users to define their own problems and algorithms. In OPCIO, we define a profile's fitness as its power consumption, and we create a hash table tblPwr to store all measured profiles. For a profile p that has not been measured yet, it will be deployed to the target device for measurement. If the profile has been measured before, we acquire the power consumption by checking the hash table without re-measurement.

As a result, the fitness of a profile p in OPCIO with GA is formulated as follows:

$$f(p) = \begin{cases} \texttt{look\_up}(p) & \text{if } p \in \texttt{tblPwr}, \quad (10a) \\ \texttt{measure}(p) & \text{otherwise.} \quad (10b) \end{cases}$$

Here, $f(p)$ is the profile p's fitness, look_up(p) returns the power value from the hash table, and measure(p) returns the power value by performing a measurement.

7 EVALUATION

In this section, we use two target devices as case studies to evaluate OPCIO. We first briefly summarize the architecture of the two devices, and then we introduce the experiment setup and evaluation metrics.

7.1 Target Devices

We evaluate the performance of OPCIO on two devices that are deployed in intertidal zones for environmental monitoring. Both devices are battery powered and wake up for 1 s per 15 minutes (ultra-low-duty-cycle). Devices are left in the field and are supposed to work for years. Decreasing the power consumption for such devices can prolong their lifespans significantly.

We show the two types of devices in Figure 9. To avoid confusion, we denote them as NODE-A and NODE-B hereafter. They use the same radio chip and temperature sensor. However, they differ from each other in the following major components:

(1) **MCU.** NODE-A is driven by STM32F103RC, which is a high-performance microcontroller with 64 pins. NODE-B employs STM32L151C8, an ultra-low-power microcontroller with 48 pins that consumes less power than NODE-A in the low-power mode.
(2) **External data flash.** NODE-B is embedded with an external data flash to store extra amount of data while NODE-A does not. The data flash communicates with the microcontroller through SPI (Serial Peripheral Interface), and the power source for the data flash is controlled by toggling a GPIO pin.
(3) **Power management module.** NODE-A and NODE-B have different power management mechanisms. NODE-A utilizes an LDO (low dropout) regulator to regulate the output voltage from the battery source and then uses the regulated voltage to feed the microcontroller. However, the microcontroller of the NODE-B is fed by the battery source directly.
(4) **Battery source.** With the help of an LDO regulator, NODE-A utilizes a 3.7 V battery as the power source, whereas NODE-B is powered by a 3.0 V battery.

7.2 Experiment Setup and Metrics

In the experiments, we utilize a proxy node that is the same type as NODE-B to forward command messages and feedback messages between the PC and the target device wirelessly. For all experiments, we generate 100 profiles ($L = 100$) for each generation, and the search stops after 20 generations ($K = 20$). The target device will turn to sleep mode for 10 seconds, and the data logger measures the average current draw of the target device in the middle 5 seconds. It is worth mentioning that the sampling rate of the data logger is 1000 Hz, thus the data logger records 5K samples for each profile.

We run the OPCIO with GA algorithm on both NODE-A and NODE-B. For each type of node, five experiments were performed, which are presented as R1 to R5. For the first four of them, the initial profiles in the first generation are randomly generated. For the last one, the initial profiles come from the first four experiments: 25 profiles are selected from each of the four experiments, and these 25 profiles are the top ranked ones of the first generation among the four experiments. In this way, we elaborately select profiles for the R5 experiment, which are supposed to have better

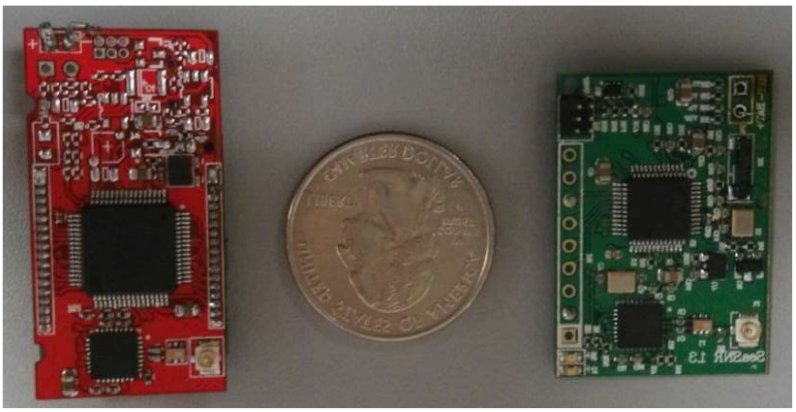


Fig. 9. Smart devices in case studies (left: NODE-A and right: NODE-B).

performance than those randomly selected ones (R1 to R4). The experiment results of R5 can help to evaluate the importance of initial profile selection in OPCIO with GA algorithm.

To evaluate the performance, we take the average fitness and the best fitness of each generation as metrics. Consider each generation's average fitness and best fitness as the metrics:

(1) The **best fitness** of kth generation refers to the lowest current draw that has been found by the end of kth generation. This metric indicates how **good** a solution OPCIO can find.
(2) The **average fitness** of kth generation refers to the average current draw of each generation. Its trend indicates how *fast* OPCIO can find low-power solutions.

7.3 Case Studies

Obtaining the absolute minimum current draw in the sleep phase is impossible, since there is an extremely large number of potential profiles for a specific device. Thus, we requested an experienced engineer to optimize the GPIO status of NODE-A and NODE-B carefully. The lowest current draws he could find for NODE-A and NODE-B were 134 μA and 9.8 μA, respectively, and we take them as ground truth for OPCIO.

By contrast, we recruited two users who major in software engineering to search for the minimum current draws for NODE-A and NODE-B according to the MCU's datasheet. As a starting point, both users downloaded STM32 sample codes from the official website as references. Because the target devices have customized peripherals such as a radio chip, they ported those peripheral drivers into the original program. The current draws in sleep phases of the original programs were 657 μA and 31 μA for NODE-A and NODE-B, respectively.

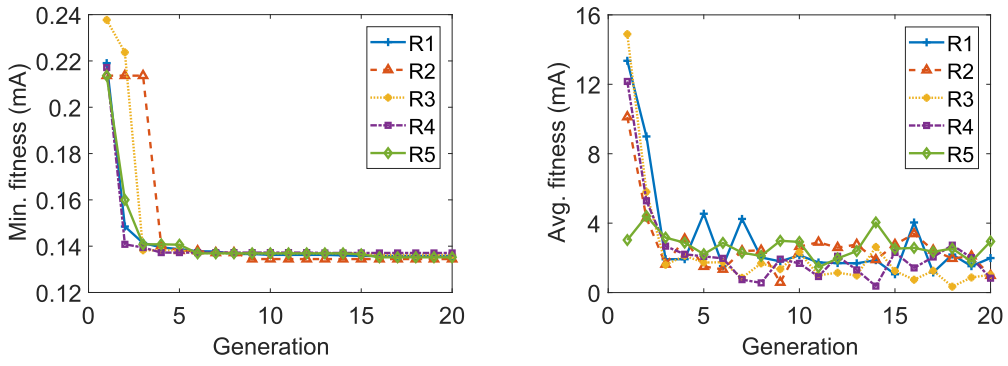
(1) NODE-A. In this section, we analyze the lowest current draw for NODE-A in the sleep phase with OPCIO. For NODE-A, each GPIO pin can be individually configured by software as follows[1]:

- Input state: analog, pull-up, pull-down, or floating mode.
- Output state: push-pull or open-drain mode.

According to the above information, we define INMODE and OUTMODE for NODE-A as follows:

$$\begin{aligned} \textbf{INMODE} &\leftarrow an|up|down|float, \\ \textbf{OUTMODE} &\leftarrow pp|od. \end{aligned}$$

[1]It also supports alternate function output state. This state is used for special features in active phases, so we do not cover this option in sleep phases.



(a) The best fitness of NODE-A in each generation.

(b) The average fitness of NODE-A in each generation.

Fig. 10. Case study: the experiment results of NODE-A. We measure the power consumption of NODE-A for five rounds. Here, R1 stands for the first experiment round, similar to other experiments.

Here, *od* and *pp* stand for open-drain mode and push-pull mode in output state; *an*, *up*, *down*, and *float* stand for analog, pull-up, pull-down, and float mode in input state, respectively. For example, if the first pin is set to open-drain output state with low output value, then we express it as `1-out:low:op`.

We show the experiment results for NODE-A in Figure 10. Though starting with different initial profiles in the first generation, all experiments (R1–R5) converge to a current draw around 134 μA finally as shown in Figure 10(a). As mentioned above, we run 20 generations in `OPCIO`, which lasts for about five hours. As shown in Figure 10(a), all experiments can obtain an acceptable profile that closes to the lowest current draw after the eighth generation, which is only about two hours. It is worth mentioning that even if the initial profiles are randomly generated, the final results of the first four experiments (R1–R4) are the same with the result where the initial profile is carefully chosen (experiment R5). This means the results of `OPCIO` with GA are not impacted by the initial profiles, which can work well for those engineers who are not familiar with the low-power deployment.

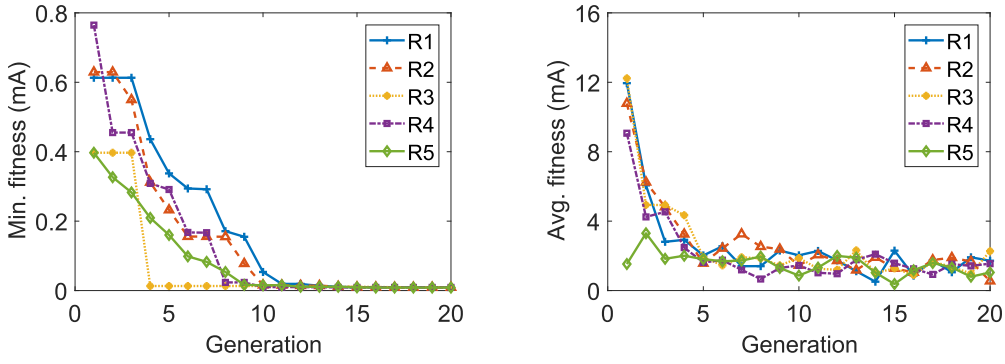
We also notice that the average fitness in Figure 10(b) fluctuates significantly. The reason behind this is few profiles may obtain opposite evolutions after the mutation process. These profiles with "outstanding" power consumption may contribute a lot when calculating average power consumption. The mutation is uncontrollable. Therefore, the average power consumption presents a state of fluctuation.

(2) NODE-B. Each GPIO pin in NODE-B can be individually configured by software as follows:

- Input state: analog, pull-up, pull-down, or floating mode.
- Output state: push-pull, push-pull with pull-up, push-pull with pull-down, open-drain, open-drain with pull-up, or open-drain with pull-down mode.

We define INMODE and OUTMODE for NODE-B as follows:

$$
\begin{array}{lcl}
\textbf{INMODE} & \leftarrow & \textit{an}|\textbf{PULL}, \\
\textbf{OUTMODE} & \leftarrow & \textbf{PP}|\textbf{OD}, \\
\textbf{PP} & \leftarrow & \textit{pp}:\textbf{PULL}, \\
\textbf{OD} & \leftarrow & \textit{od}:\textbf{PULL}, \\
\textbf{PULL} & \leftarrow & \textit{float}|\textit{up}|\textit{down}.
\end{array}
$$



(a) The best fitness of NODE-B in each generation.

(b) The average fitness of NODE-B in each generation.

Fig. 11. Case study: the experiment results of NODE-B. We measure the power consumption of NODE-B for five rounds. Here, R1 stands for the first experiment round, similar for other experiments.

Here, **PULL** stands for the mode of pull resistors, which can be pull-up, pull-down, and float. For example, if the second pin is set to push-pull output state with a pull-down resistor attached, and the pin outputs high voltage, then it can be expressed as `2-out:high:pp:down`.

Figure 11 reveals the results for NODE-B. Similarly, all experiments result in coverage to a current draw within 10 μA after 20 generations. As shown in Figure 11(a), all rounds get close to the final solution after the 11th generation, which consumes about two and a half hours. The lowest current draws found by the five experiments are 9.55 μA, 9.01 μA, 9.15 μA, 9.08 μA, and 8.64 μA, respectively, which are all smaller than the one configured by the experienced engineer (9.8 μA).

7.4 Performance Evaluation for OPCIO and Discussion

In this section, we examine how much energy we can save by applying OPCIO for embedded devices and then discuss the overhead of OPCIO.

7.4.1 Energy Consumption. As we explained above, obtaining the absolute minimum power consumption of an embedded device is impossible; therefore, we take the power consumption configured by experienced engineers as a replacement of the ground truth. For comparison, we require two users to configure the GPIO pins according to the guidance of the MCU's datasheet. We define the improvement of OPCIO as the decreasing ratio of the power consumption in the sleep phase, which can be calculated as:

$$\left(1 - \frac{P_{min}}{P_{datasheet}}\right) \times 100\%. \tag{11}$$

Here, P_{min} is the power consumption configured by OPCIO with GA, and $P_{datasheet}$ is the one that configured according to the datasheet.

We summarize the energy consumption of NODE-A and NODE-B with different configuration solutions during the sleep phase in Table 6. We first list the results configured by an experienced engineer as the ground truth, and he consumes over 10 hours to obtain an acceptable power consumption from the device. We also list the results for OPCIO with GA ($K = 10$ and $K = 20$) and OPCIO with greedy algorithm. The experiment results of OPCIO are the average current draw for the five-round experiments. When comparing the power consumption configured according to the datasheet, OPCIO with GA decreases the power consumption of NODE-A and NODE-B to 135.76 μA and 9.09 μA on average, which reduces the power in sleep phases by 79.34% and 70.68%, respectively. We also observe that OPCIO with GA ($K = 15$) can achieve similar power consumption for

Table 6. The Comparison of Configuring GPIOs with Different Methods

	$NODE-A(I_{active} = 42.1\,mA)$				$NODE-B\ (I_{active} = 28.26\,mA)$			
	Current draw(μA)	Time (hour)	Evaluation phase (hour)	Lifespan (day)	Current draw(μA)	Time (hour)	Evaluation phase (hour)	Lifespan (day)
Experienced Engineer	134	12	N/A	231	9.8	14	N/A	1011
Datasheet	657	48	N/A	59	31	63	N/A	668
OPCIO with GA (K = 20)	**135.76**	**5.56**	2.78	**230**	**9.09**	**5.56**	2.78	**1029**
OPCIO with GA (K = 15)	136.14	4.17	2.08	228	9.72	4.17	2.08	1014
OPCIO with Greedy Algorithm	/	/	/	/	209.075	1.64	0.82	173

(Battery = 1000 mA).

Table 7. The Sequences of the Generation That Can Be Stopped in OPCIO with Different User Requirements for NODE-A and NODE-B

Devices	Requirements	R1	R2	R3	R4	R5	Average[1]
NODE-A	$I < 140\,\mu A$	4	5	3	3	6	5
	$I < 138\,\mu A$	6	6	6	4	6	6
	Marginal Improvement[2]	12	11	9	6	8	10
NODE-B	$I < 12\,\mu A$	14	14	12	10	12	13
	$I < 10\,\mu A$	15	15	17	13	15	15
	Marginal Improvement[2]	17	19	19	17	17	18

[1]The average generations are rounded up to integers.
[2]The improvements are within 1% for five generations.

NODE-A and NODE-B with an experienced engineer while only costing 34.38% and 29.46% of the time, respectively. We also notice that the greedy algorithm acquires an acceptable profile after four rounds (6.56 hours). Based on this, we find that GA performs better than the greedy algorithm.

7.4.2 Life Spans for Devices. The current draws for NODE-A and NODE-B during the active phase are 42.1 *mA* and 28.26 *mA*, respectively. Given a 1000 *mA* battery and a device that is awake for 1 second per 15 minutes, we calculate the lifespans of devices in Table 6.

We observe OPCIO with GA extends the devices' lifespans significantly when compared with the one configured according to the datasheet. For example, OPCIO extends NODE-A's lifespan to 230 days, which is nearly four times more than the one configured according to the datasheet. In the NODE-B case, OPCIO also extends the lifespan for nearly one year.

7.4.3 The Number of Generations in OPCIO. We stop the OPCIO after 20 generations in our experiments. However, it is feasible for users to stop the OPCIO when the first profile that satisfies the low-power requirements is found or the improvement between generations is marginal. In Table 7, we summarize the sequences of the generation that can be stopped by users with different low-power requirements.

As shown in Table 7, with different user requirements, OPCIO can be stopped at any generation. For example, if the goal of the low-power configuration for NODE-A is within 140 μA, the user only needs to run OPCIO for five generations on average. Based on the experiment settings, it takes about 1.39 hours to finish the five-generation GA-based OPCIO. However, we choose to run OPCIO

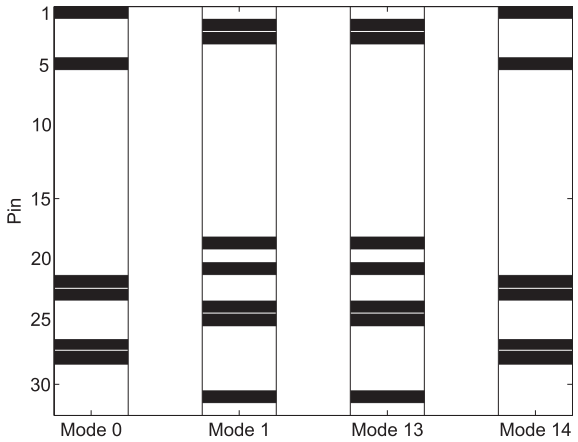


Fig. 12. Distribution of mode 0, 1, 13, and 14 in sleep phases: mode 0 and 13 exhibit the same distribution pattern, whereas mode 1 and 14 exhibit the same distribution pattern.

for 20 generations, because we want to obtain extra low-power profiles to study GPIO statuses, which will be discussed in Section 7.4.5.

7.4.4 Discussion. It is worth mentioning that configuring GPIOs' statuses does incur extra delay when switching between the active phase and the sleep phase. Taking NODE-B as an example, the switching delay from sleep to active is 2.2 *ms*, which includes waking up its MCU, waiting for a radio to be stabilized, and waiting for an external data flash to be ready. Configuring GPIOs will add 15 μs delay. Though 15 μs delay causes energy overhead, changing GPIOs' statuses can greatly reduce the energy consumption in sleep phases and prolong the battery life. Moreover, `OPCIO` only focuses on low-power configuration for single-MCU embedded devices while it will be more complicated in multi-MCU cases.

7.4.5 Explore `OPCIO` *Results.* As mentioned above, we run `OPCIO` for 20 generations and obtain extra low-power profiles to study the GPIO statuses. In the following, we take NODE-B as an example and present how to explore the low-power profiles found by `OPCIO`.

We found 340 different profiles whose current draws are less than 10 μA during the sleep phase after running `OPCIO` on NODE-B five times. Given these low-power profiles, we first define *hit* and *miss* of a mode on a pin:

For all the low-power profiles found by `OPCIO`, if a mode j appears at least once in pin i's instructions, we call the mode j *hits* the pin i; otherwise, the mode j *misses* the pin i. We represent this definition as follows:

$$H_{i,j} = \begin{cases} 1 & \text{mode } j \text{ hits pin } i, \quad (12a) \\ 0 & \text{mode } j \text{ misses pin } i. \quad (12b) \end{cases}$$

For example, $H_{1,0} = 1$, because mode 0 appears in pin 1's instructions; but $H_{2,0} = 0$, because mode 0 is not seen in pin 2's instructions among all the 340 low-power profiles.

Based on this definition, we have a matrix H whose jth column describes the distribution of mode j on all GPIO pins. Figure 12 gives an example of the distribution of mode 0 (PP:0), mode 1 (PP:1), mode 13 (IN:PU), and mode 14 (IN:PD). Each bar represents the distribution of a mode, and the y-axis represents a pin (only 32 pins are shown in the figure). If $H_{i,j} = 1$ (i.e., mode j hits pin i), pin i is marked with black in mode j's bar. For example, pin 1 is black in mode 0's bar, because $H_{1,0} = 1$; but pin 2 is not colored in that bar, because $H_{2,0} = 0$. As shown in Figure 12, mode 0 and

Table 8. Classifications of Different Modes in Sleep Cycles

Class	Mode	Class	Mode
A	0, 4, 6, 10, 11, 14	D	15
B	1, 3, 9, 13	E	7, 12
C	2, 5, 8		

mode 14 have the same distribution. They both hit the same pins: 1, 5, 22, 23, 27, and 28. That means mode 0 and mode 14 consume the same amount of energy in sleep phases. Interestingly, in terms of functional features, they are *completely* different, because mode 0 is programmed as output, whereas mode 14 is programmed as input. Similarly, mode 1 and mode 13 exhibit the same distribution pattern. This observation confirms our hypothesis: Two modes could have different functional features in active phases, but they may contribute to the same energy consumption in sleep phases.

Motivated by this observation, we categorize all the modes of node NODE-B (Table 3) by calculating the correlation based on their distribution matrix H. The result shows that these 16 modes can be grouped into five categories, which are listed in Table 8:

(1) **Class A:** The correlations of mode 0, 4, 6, 10, 11, and 14 are above 0.9, and thus they are grouped in the same category. This can be explained by looking at their internal structures: They all connect the pin to a low voltage level, regardless of their input or output status in sleep phases. In addition, no current will flow through the pull-up resistor.
(2) **Class B:** Similarly, mode 1, 3, 9, and 13 are grouped in the same category, because their correlations are also above 0.9. This is because they connect a pin to a high voltage level, and no current will flow through the pull-down resistor in sleep phases.
(3) **Class C:** Mode 2, 5, and 8 are grouped in the same class, because our result shows that these three modes never appear in the 340 low-power profiles, which means they are not energy-efficient in sleep phases. The reason is that a small amount of current constantly flows through the pull-up or pull-down resistors, which introduces extra energy consumption. This extra energy consumption is not noticeable in active phases, but it becomes a major source of energy waste in sleep phases.
(4) **Class D:** Our result shows mode 15 is the most frequently used mode for those *unoccupied* GPIO pins in the 340 low-power profiles. This is because in analog mode the pull-up and pull-down resistors are disabled, and the path to the MCU's data registers is cut off, which leads to zero consumption for unoccupied I/O pins.
(5) **Class E:** Mode 7 and mode 12 are in the same category based on our result. This can be explained by their internal structures: both of them leave a pin in the floating state.

We conclude the analysis of the platform of NODE-B as follows:

- For unoccupied GPIOs, modes in class D are recommended, because they consume the minimum energy;
- For occupied GPIOs, modes in class A are recommended for those that must be kept as a low voltage in sleep, and modes in class B are recommended for those that must be kept as a high voltage in sleep. In addition, modes in class A are interchangeable in sleep phases, and so are modes in class B;
- In any case, modes in class C are not recommended, because they can be major sources of energy consumption in sleep phases.

One may argue that the above conclusions can be included as guidelines in an MCU's datasheet or officially released sample codes so the GPIO exploration at runtime can be avoided. Unfortunately, most existing datasheets provided by the MCU manufacturers do not include such guidelines. The official sample codes are mainly developed with the purpose of illustrating *functional* features and are typically developed on evaluation boards, which may have distinct peripherals from customized device boards and the optimal GPIO configuration varies. Therefore, it is helpful and convenient to utilize OPCIO to find an optimal GPIO configuration in sleep phases for various device platforms.

7.5 OPCIO Summary

OPCIO has several advantages and can be extended to other device models:

(1) OPCIO provides another angle of view to further reduce the power consumption for embedded devices, which can be easily combined with existing low-power mechanisms that worked in the active phase.
(2) OPCIO can quickly converge to approach an optimal low-power solution and can help programmers with little hardware knowledge to find out what is the lifespan of devices. It can help hardware designers or programmers with abundant hardware expertise to find out the minimum feasible current draw in a sleep phase, because possible hardware manufacturing or hardware design faults can make it tricky to configure GPIO pins so they cannot achieve low-power requirement.
(3) OPCIO can help users to better understand their platform especially for those developers who have little experience in hardware and low-power embedded systems;
(4) OPCIO can be easily tailored to work with different wireless nodes. Although we use ZigBee-based devices as our case study, they can also be used in other networks such as BLE (Bluetooth Low Energy) as long as the decision controller in OPCIO can communicate with devices. To achieve this, one can simply put a proxy node as what we presented in our case studies.

8 RELATED WORK

OPCIO attempts to reduce the energy consumption for low-duty-cycle embedded devices during the sleep phase and extend the system lifespan eventually. The related work includes low-duty-cycle platforms, energy management techniques, and energy harvesting techniques.

Low-power low-duty-cycle wireless platforms. A typical embedded system consists of a processing module, sensing modules, and a communication module. Extensive research has shown that a microcontroller (processing module) and a radio chip (communication module) are the most power-hungry modules in a sensor node. As a result, much attention has been devoted to reducing energy consumption on microcontrollers and radio chips. For example, during the design of wireless sensor nodes [12, 31], low-power microcontrollers and radio chips are carefully selected to minimize the node's overall energy consumption. It is recognized that operating in the sleep mode consumes much less energy than the active mode, thus many battery-powered applications usually operate systems in low-duty-cycle (duty cycle $< 1\%$) or even ultra-low-duty-cycle (duty cycle $< 0.5\%$). For example, Moghaddam et al. proposed a soil moisture sensing system [25] that monitors the moisture of the underground soil periodically. Embedded with numerous peripheral modules, the current draw of the device in the active mode is up to 95 *mA*, which constrains the lifespan of the system with limited power. By setting the system in ultra-low-duty-cycle mode (duty cycle = 0.125%), the lifespan of the system is significantly extended. Other platforms such

as UP-Link [21], IT-WSN [42, 47, 48], and smart gas monitoring system [13] also face similar challenges, and they set the duty cycle to 0.0014%, 0.111%, and 0.01%, respectively.

Energy management via DPM and DVS. Dynamic Power Management (DPM) [7, 26, 46] and Dynamic Voltage Scaling (DVS) [19, 45] are two main dynamic energy management techniques that reduce the power of embedded systems. DPM saves energy by putting a node into low-power mode dynamically and prolongs the idle cycle as much as possible, whereas DVS saves energy by reducing the supply voltage and operating frequency. DPM strategies may utilize machine learning or accurate phase prediction to identify a proper idle period and a timeout value to duty cycle a device. A recent DVS strategy [24] exploits a task scheduling technique to minimize energy consumption.

Compared with DPM and DVS, which prolong battery life by *duty cycling* a node in time domains and voltage domains, our work differs from theirs in the sense that we reduce the power in *sleep cycles*.

Energy harvesting in WSNs. Energy harvesting technologies, converting ambient energy to electrical energy, are promising to solve the conflict of battery life and battery capacities [41]. Gorlatova et al. characterized the energy availability in indoor environments and developed energy allocation algorithms for energy harvesting devices [11]. Porcarelli et al. designed a two-stage energy conversion circuit to transform energy from air-flow [32]. GRAPMAN proposed a novel power management strategy for energy-harvesting WSNs and dramatically promoted the feasibility of energy-harvesting technique [5]. Lee et al. developed an oceanic sensing platform utilizing small-scale benthic microbial fuel cells [20].

Although energy-harvesting technologies are promising, not all sensing applications can take advantage of the technologies due to their deployment environments. Therefore, searching for low-power solutions is still important, and `OPCIO` can be handy in this case.

In addition to general platforms such as MICAz and TelosB, many low-power low-duty cycle wireless platforms for dedicated purposes have also been developed. For example, a low-power wireless medical device platform was designed for a wide range of medical and sports applications [10]. Using low standby current technologies, this device has a lifespan of over 10 years on a single 200 mAh lithium watch battery. Dozer, a data-gathering platform, achieves an average radio duty cycle of 0.16% by efficiently coordinating the MAC-layer and the routing protocol [6]. Koala is a reliable data-retrieval system designed for long-term environmental monitoring networks [28]. Letting the device sleep most of the time and reviving them through an efficient wake-up strategy, Koala can achieve ultra-low duty cycles at permille (0.1%).

`OPCIO` is originally proposed to optimize the energy consumption of the wireless device platform deployed in intertidal zones (IT-WSNs) where wireless channels are intermittent, and nodes operate at ultra-low duty cycles around 0.11c%. However, `OPCIO` is hardware-independent and therefore can be extended to other device platforms with minor modification by defining the structure of their own configuration options.

9 CONCLUSION

Operating on ultra-low-duty cycles, a modern embedded device's energy consumption in the sleep phase can greatly affect its lifespan. We discover that a large portion of MCU pins are GPIO pins and if not configured properly, they can lead to 1,500-fold difference in device lifetime. Finding the right configuration that minimizes the energy consumption in the sleep phase requires hardware expertise, a great burden on software developers. In this article, we designed `OPCIO`, an automated framework that searches low-power solutions using off-the-shelf devices. Our evaluation of `OPCIO` on two embedded device platforms shows that `OPCIO` can effectively find configurations inducing low energy consumption. We therefore envision that with the help of `OPCIO`, the developers can

easily extend the lifetime of embedded devices. As a direction of future work, it is worthwhile to validate OPCIO on other device platforms, and we believe that OPCIO can be used to understand the energy consumption distribution and to improve the design of embedded devices.

REFERENCES

[1] Juan J. Durillo and Antonio J. Nebro. 2011. jMetal: A Java framework for multi-objective optimization. *Adv. Eng. Softw.* 42 (2011), 760–771.

[2] Agilent Technologies. 2014. U2500A Series USB Modular Data Acquisition. Retrieved from http://www.home.agilent.com/.

[3] Ram Bhusan Agrawal, Kalyanmoy Deb, Kalyanmoy Deb, and Ram Bhushan Agrawal. 1994. Simulated binary crossover for continuous search space. *Compl. Syst.* 9, 3 (1994), 115–148.

[4] Ankita Bhutani and Preeti Wadhwani. 2010. Global LPWAN Market Size Worth over USD 65 Billion by 2025. Retrieved from https://www.gminsights.com/pressrelease/lpwan-market.

[5] Fayçal Ait Aoudia, Matthieu Gautier, and Olivier Berder. 2015. GRAPMAN: Gradual power manager for consistent throughput of energy harvesting wireless sensor nodes. In *Proceedings of the 26th IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications*. 954–959.

[6] Nicolas Burri, Pascal von Rickenbach, and Roger Wattenhofer. 2007. Dozer: Ultra-low power data gathering in sensor networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*. 450–459.

[7] Waltenegus Dargie. 2012. Dynamic power management in wireless sensor networks: State-of-the-art. *IEEE Sens. J.* 12, 5 (2012), 1518–1528.

[8] Kalyanmoy Deb and Debayan Deb. 2014. Analysing mutation schemes for real-parameter genetic algorithms. *Int. J. Artif. Intell. Soft Comput.* 4, 1 (2014), 1–28.

[9] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* 6, 2 (2002), 182–197.

[10] Arseny B. Dolgov and Regan Zane. 2006. Low-power wireless medical sensor platform. In *Proceedings of the 28th International Conference of the IEEE Engineering in Medicine and Biology Society*. 2067–2070.

[11] Maria Gorlatova, Aya Wallwater, and Gil Zussman. 2013. Networking low-power energy harvesting devices: Measurements and algorithms. *IEEE Trans. Mob. Comput.* 12, 9 (2013), 1853–1865.

[12] Jason L. Hill and David E. Culler. 2002. Mica: A wireless platform for deeply embedded networks. *IEEE Micro* 22, 6 (2002), 12–24.

[13] Vana Jeličić, Michele Magno, Giacomo Paci, Davide Brunelli, and Luca Benini. 2011. Design, characterization, and management of a wireless sensor network for smart gas monitoring. In *Proceedings of the 4th IEEE International Workshop on Advances in Sensors and Interfaces*. 115–120.

[14] Xiaoyu Ji, Yuan He, Jiliang Wang, Wei Dong, Xiaopei Wu, and Yunhao Liu. 2014. Walking down the STAIRS: Efficient collision resolution for wireless sensor networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'14)*. 961–969.

[15] Xiaoyu Ji, Yuan He, Jiliang Wang, Kaishun Wu, Daibo Liu, Ke Yi, and Yunhao Liu. 2017. On improving wireless channel utilization: A collision tolerance-based approach. *IEEE Trans. Mob. Comput.* 16, 3 (2017), 787–800.

[16] Xiaoyu Ji, Yuan He, Jiliang Wang, Kaishun Wu, Ke Yi, and Yunhao Liu. 2013. Voice over the dins: Improving wireless channel utilization with collision tolerance. In *Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP'13)*. 1–10.

[17] Xiaoyu Ji, Jiliang Wang, Mingyan Liu, Yubo Yan, Panlong Yang, and Yunhao Liu. 2014. Hitchhike: Riding control on preambles. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'14)*. 2499–2507.

[18] Vassilis Kostakos, Timo Ojala, and Tomi Juntunen. 2013. Traffic in the smart city: Exploring city-wide sensing for traffic control center augmentation. *IEEE Int. Comput.* 17, 6 (2013), 22–29.

[19] Ulf Kulau, Felix Büsching, and Lars C. Wolf. 2013. A node's life: Increasing WSN lifetime by dynamic voltage scaling. In *Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems*. 241–248.

[20] Inhee Lee, Gyouho Kim, Suyoung Bang, Adriane Wolfe, Richard Bell, Seokhyeon Jeong, Yejoong Kim, Jeffrey Kagan, Meriah Arias-Thode, Bart Chadwick, et al. 2015. System-On-Mud: Ultra-low power oceanic sensing platform powered by small-scale benthic microbial fuel cells. *IEEE Trans. Circ. Syst. I: Reg. Pap.* 62, 4 (2015), 1126–1135.

[21] Woo Suk Lee, Albert Kim, Babak Ziaie, Vijay Raghunathan, and Charles R. Powell. 2014. UP-link: An ultra-low power implantable wireless system for long-term ambulatory urodynamics. In *Proceedings of the IEEE Biomedical Circuits and Systems Conference*. 384–387.

[22] Mingfu Li and Hung-Ju Lin. 2015. Design and implementation of smart home control systems based on wireless sensor networks and power line communications. *IEEE Trans. Industr. Electron.* 62, 7 (2015), 4430–4442.

[23] Xiaomin Li, Di Li, Jiafu Wan, Athanasios V. Vasilakos, Chin-Feng Lai, and Shiyong Wang. 2017. A review of industrial wireless networks in the context of Industry 4.0. *Wirel. Netw.* 23, 1 (2017), 23–41.

[24] Xue Lin, Yanzhi Wang, Qing Xie, and Massoud Pedram. 2015. Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment. *IEEE Trans. Serv. Comput.* 8, 2 (2015), 175–186.

[25] Mahta Moghaddam, Dara Entekhabi, Yuriy Goykhman, Ke Li, Mingyan Liu, Aditya Mahajan, Ashutosh Nayyar, David Shuman, and Demosthenis Teneketzis. 2010. A wireless soil moisture smart sensor web using physics-based optimal control: Concept and initial demonstrations. *IEEE J. Select. Top. Appl. Earth Observ. Remote Sens.* 3, 4 (2010), 522–535.

[26] Olesia Mokrenko, Suzanne Lesecq, Warody Lombardi, Diego Puschini, Carolina Albea, and Olivier Debicki. 2014. Dynamic power management in a wireless sensor network using predictive control. In *Proceedings of the 40th Conference of the IEEE Industrial Electronics Society*. 4756–4761.

[27] Monsoon Solutions Inc. 2013. Power Monitor. Retrieved from http://www.msoon.com.

[28] Razvan Musaloiu-Elefteri, Chieh-Jan Mike Liang, and Andreas Terzis. 2008. Koala: Ultra-low power data retrieval in wireless sensor networks. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*. 421–432.

[29] Michele Penza, Domenico Suriano, Maria Gabriella Villani, Laurent Spinelle, and Michel Gerboles. 2014. Towards air quality indices in smart cities by calibrated low-cost sensors applied to networks. In *Proceedings of the 13th IEEE SENSORS Conference*. 2012–2017.

[30] Thanh Nam Pham, Ming-Fong Tsai, Duc Binh Nguyen, Chyi-Ren Dow, and Der-Jiunn Deng. 2015. A cloud-based smart-parking system based on Internet-of-Things technologies. *IEEE Access* 3 (2015), 1581–1591.

[31] Joseph Polastre, Robert Szewczyk, and David E. Culler. 2005. Telos: Enabling ultra-low power wireless research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*. 364–369.

[32] Danilo Porcarelli, Dora Spenza, Davide Brunelli, Alessandro Cammarano, Chiara Petrioli, and Luca Benini. 2015. Adaptive rectifier driven by power intake predictors for wind energy harvesting sensor networks. *IEEE J. Emerg. Select. Top. Power Electron.* 3, 2 (2015), 471–482.

[33] Pinyi Ren, Yichen Wang, and Qinghe Du. 2014. CAD-MAC: A channel-aggregation diversity based MAC protocol for spectrum and energy efficient cognitive ad hoc networks. *IEEE J. Select. Areas Commun.* 32, 2 (2014), 237–250.

[34] José Luis Rodríguez, Inés Garcia-Tunon, Jose Manuel Taboada, and Fernando Obelleiro Basteiro. 2007. Broadband HF antenna matching network design using a real-coded genetic algorithm. *IEEE Trans. Ant. Propag.* 55, 3 (2007), 611–618.

[35] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R. Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, et al. 2014. SmartSantander: IoT experimentation over a smart city testbed. *Comput. Netw.* 61 (2014), 217–238.

[36] Ruben M. Sandoval, Antonio-Javier Garcia-Sanchez, Joan Garcia-Haro, and Thomas M. Chen. 2018. Optimal policy derivation for transmission duty-cycle constrained LPWAN. *IEEE Int. Things J.* 5, 4 (2018), 3114–3125.

[37] G. M. Shafiullah, Salahuddin A. Azad, and A. B. M. Shawkat Ali. 2013. Energy-efficient wireless MAC protocols for railway monitoring applications. *IEEE Trans. Intell. Transport. Syst.* 14, 2 (2013), 649–659.

[38] STMicroelectronics. 2015. STM32 MCUs Reference Manual. Retrieved from http://www.st.com/.

[39] Mehdi Tarhani, Yousef S. Kavian, and Saman Siavoshi. 2014. SEECH: Scalable energy efficient clustering hierarchy protocol in wireless sensor networks. *IEEE Sens. J.* 14, 11 (2014), 3944–3954.

[40] Can Tunca, Sinan Isik, Mehmet Yunus Donmez, and Cem Ersoy. 2015. Ring routing: An energy-efficient routing protocol for wireless sensor networks with a mobile sink. *IEEE Trans. Mob. Comput.* 14, 9 (2015), 1947–1960.

[41] Sennur Ulukus, Aylin Yener, Elza Erkip, Osvaldo Simeone, Michele Zorzi, Pulkit Grover, and Kaibin Huang. 2015. Energy harvesting wireless communications: A review of recent advances. *IEEE J. Select. Areas Commun.* 33, 3 (2015), 360–381.

[42] Miao Xu, Wenyuan Xu, Tingrui Han, and Zhiyun Lin. 2016. Energy-efficient time synchronization in wireless sensor networks via temperature-aware compensation. *ACM Trans. Sens. Netw.* 12, 2 (2016), 12:1–12:29.

[43] Yanjun Yao, Qing Cao, and Athanasios V. Vasilakos. 2015. EDAL: An energy-efficient, delay-aware, and lifetime-balancing data collection protocol for heterogeneous wireless sensor networks. *IEEE/ACM Trans. Netw.* 23, 3 (2015), 810–823.

[44] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. 2014. Internet of things for smart cities. *IEEE Int. Things J.* 1, 1 (2014), 22–32.

[45] Kejiu Zhang, Shiguo Luo, Thomas X. Wu, and Issa Batarseh. 2014. New insights on dynamic voltage scaling of multiphase synchronous buck converter: A comprehensive design consideration. *IEEE Trans. Power Electron.* 29, 4 (2014), 1927–1940.

[46] Xu Zhang, Gyoung Bae Kim, and Hae-Young Bae. 2014. A reliable dynamic power management method for wireless body area network. In *Proceedings of the International Conference on Information and Communication Technology Convergence*. 59–64.

[47] Xinyan Zhou, Xiaoyu Ji, Yi-Chao Chen, Xiaopeng Li, and Wenyuan Xu. 2018. LESS: Link estimation with sparse sampling in intertidal WSNs. *Sensors* 18, 3 (2018), 747.
[48] Xinyan Zhou, Xiaoyu Ji, Bin Wang, Yushi Cheng, Zhuoran Ma, Francis Choi, Brian Helmuth, and Wenyuan Xu. 2018. Pido: Predictive delay optimization for intertidal wireless sensor networks. *Sensors* 18, 5 (2018), 1464.

Received September 2018; revised May 2019; accepted November 2019