Two Designs of Automatic Embedded System Energy Consumption Measuring Platforms Using GPIO

Huanjie Wu, Chun Chen and Kai Weng *
College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, Zhejiang, China; 3100100294@zju.edu.cn (H.W.); chenc@zju.edu.cn (C.C.)
* Correspondence: wengkai@zju.edu.cn; Tel.: +86-138-0571-1803

Received: 21 May 2020; Accepted: 13 July 2020; Published: 16 July 2020

Featured Application: The two designs proposed in this article were applied to the design of ACoreOS Lite energy saving modes.

Abstract: Energy consumption is a critical evaluation index of embedded systems, and it has impacts on battery-life, thermal design, as well as device security and reliability. Since energy is the time integral of power, power consumption should be considered, along with the impact of “time”; thus, we propose two designs of automatic energy consumption measuring platforms utilizing General Purpose Input/Output (GPIO). Using these designs, we developed software and introduced auxiliary hardware for solutions with better timing and synchronization. A series of test sets were designed to verify our designs’ capabilities and accuracy levels. Both of our designs showed an accuracy similar to that of traditional measuring methods, which can satisfy the needs of different occasions. In addition, our designs provide real-time energy consumption data, as well as unattended automated measurements.

Keywords: energy consumption; embedded system; automatic measurement; automated and unattended measurement; GPIO

1. Introduction

With the rapid development of System-on-Chips (SoC) and the demands of modern smart life, large numbers of embedded systems are being deployed everywhere in our daily life. A smart Internet of Things (IoT) system may consist of thousands of embedded devices, rendering the system expensive and hard to rebuild. Both operators and users expect these devices to last for long periods of time with little maintenance. Many embedded systems are deployed without an ease of maintenance and are self-powered with batteries or solar cells. There are also many embedded systems that require constant power with very compact designs. Energy consumption has significant impacts not only on battery life, but also on the hardware thermal design and the device’s security and reliability. To make sure that designed embedded systems are eligible for low energy consumption requirements, many methods to measure or model power consumption have been proposed. However, power consumption is not enough for hardware design and code optimization; the impact of “time” should also be carefully considered.

Herein, we propose two slightly different designs of energy consumption measuring platforms for embedded systems, one for higher precision and the other for simpler deployment. To measure the energy consumption accurately, we used General Purpose Input/Output (GPIO) in our designs for timing and synchronization. We also designed a series of test sets and verified the capabilities of our designs.
2. Related Works and Motivations

In this section, we first investigate existing power measuring methods and identify problems that these methods are unable to solve.

2.1. Existing Methods

Power saving and energy efficiency have been a consistently common research direction for embedded systems, with many power/energy consumption measuring and modeling methods having been proposed. Many battery-powered devices are equipped with a battery-monitoring unit (BMU), which continuously monitors the battery voltage, temperature, and current flowing into and out of the battery. Many researchers have proposed measuring or modeling the power consumption of embedded systems with the help of a BMU [1–9], particularly in research regarding smartphones. Mesa-Martinez et al. [10] also proposed a method of measuring the energy consumption of Android smartphones with a battery voltage characteristic curve obtained by voltmeters. However, a BMU has two major disadvantages; namely, its resolution, which is typically 0.1 mAh on Android devices [7,8], and its updating interval, which varies from several hundred milliseconds to several seconds [1,3,4,7,8]. The BMU also faces significant errors due to battery capacity degradation [3]. To overcome this shortcoming, Armen et al. [11] proposed a method of modeling smart phone power consumption using a battery simulator from National Instruments.

Some devices are equipped with data acquisition (DAQ) or a power meter connected to the power pack, which was widely used by researchers looking at power consumption measurement and the modeling of embedded systems [12–14]. McCullough et al. [15] used built-in energy counters on the Intel Calpella platform to measure actual power consumption by subsystems directly. The Running Average Power Limit (RAPL) is a Linux-based, driver-layer performance analysis tool, which can be used to measure the average power limit of an Intel CPU. David et al. [16] and Hähnel et al. [17] used RAPL to measure power consumption, with the resolution shown to be as high as 0.2 W. The sampling rate of DAQ can be up to 3 kHz, which is a much higher sampling frequency than that of the BMU. However, sampling at a high frequency is accompanied by a notable CPU time with a significant impact on the system performance, resulting in distorted measurements [12,14,18].

An infrared camera is a device that creates an image using infrared radiation. Infrared thermography can be used to intuitively show the use and distribution of energy in embedded systems by deploying an infrared camera on the surface of the board with heat transfer oil. The camera captures pictures of heat distribution and then calculates the energy consumed by the chip [10,16]. This method is visual and intuitive, but it is not easy to quantify.

Using power meters or other similar instruments is also a way to explore the power consumption of embedded systems, whereby the power supply is not directly connected to the board but is connected via a power meter. A multimeter was used in [10,19], Edip et al. [20] used a DC power supply, Shye et al. [21] used current clamps, Bircher et al. [22] used power probes, and some researchers, like Chang et al. [23] and Gilberto et al. [24], used oscilloscopes. Many commercial power meters, such as Monsoon power monitors [25–35] and LEAP power measurement devices [36], can achieve a sampling frequency of 5 kHz, which is much better than BMUs. Some researchers also used analog-to-digital (AD) modules to design their own unique instruments to measure power consumption [37,38].

Some chip vendors also developed software analyzers or profilers. Qualcomm released the first edition of Snapdragon Profiler [39] in 2016. It allows developers to analyze CPU, GPU, DSP, memory, power, thermal, and network data to find and fix performance bottlenecks. Still, it only works with Android devices powered by Qualcomm Snapdragon processors. EnergyTrace [40], developed by Texas Instruments, is a power analyzer tool that measures and displays the energy profile of an application and helps optimize it for an ultra-low-power consumption. However, it can only monitor the current for non-TI target devices. STM32CubeMonitor-Power [41], developed by STMicroelectronics, enables developers to swiftly analyze the low-power performance of target boards, but it shows no runtime status of the code.
2.2. Problems

The abovementioned works have provided a variety of power consumption measurements and modeling methods for embedded systems, but some problems remain to be solved. In many studies, researchers have focused largely on power consumption, paying more attention to changes in power consumption when applications, devices, or radios were running, in order to investigate the runtimes of operations with a high-power consumption in embedded systems and predict or estimate the overall battery life.

However, energy consumption is an accumulation, representing the time integral of power. The more detailed performance of hardware design, code optimization, and energy analysis requires an understanding of the impact of time on energy consumption. In the research mentioned in the previous subsection related to modeling and measurement, many did not consider “time”. In [19,27–29,31,33,34,36], the granularity of time was in seconds, whereas in [1,3,5,6,8,9], time was calculated per unit of battery discharge. This is no problem for most applications and radios that take tens of milliseconds or seconds to execute, but the determination of the energy consumed by small code segments that constitute the applications requires more precise timestamps to mark the execution time, which is usually only tens of microseconds. To retrieve high-precision timestamps, researchers used environment variables [11], library functions [42], or event-driven system call tracing [12,14,20,29,43–46] to log or mark precise timestamps. In general, the accuracy of a timestamp or timer provided by an operating system is in milliseconds. Millisecond accuracy may be applicable for measuring the execution time of the entire application, but it cannot measure the execution time of the functions in the application or the code segments that constitute the application. Some researchers also used library functions or environment variables to record the execution time; unfortunately, these library functions and environment variables only provide precision, not accuracy, and do not guarantee the frequency of value changes [47,48]. We conducted our own verification and the results are listed in Table 1. The results showed that the frequency of value changes was quite variable; therefore, the use of system calls, library functions, or environment variables to mark timestamps was deemed to be unacceptable.

|                      | Debian 9 on Intel Xeon E5630 | Debian 9 on Intel Celeron G3930 | Debian 9 on Intel Xeon E3-1270v6 |
|----------------------|-------------------------------|---------------------------------|-------------------------------|
| **Average (µs)**     | 23.709                        | 11.760                          | 414.760                       |
| **Maximum (µs)**     | 60.081                        | 40.054                          | 1201.630                      |
| **Minimum (µs)**     | 10.014                        | 6.676                           | 0.272                         |
| **Standard deviation** | 9.884                        | 7.073                           | 293.97                        |

It was stated in [12,14,18] that directly sampling power consumption data at a high frequency using embedded systems impacted the performance and resulted in distorted measurement results; thus, power consumption data should be sampled and collected outside the embedded systems. Since timestamps are obtained using embedded systems and power consumption data are collected by a power meter on another device, this introduces the problem of time synchronization. The connection between the code execution process and the power consumption data cannot be effectively established if the obtained timestamp and the collected power consumption data are not properly synchronized.

3. Methodology and Data Acquisition

In this section, we discuss how the problems regarding marking timestamps accurately and synchronizing the power consumption data and code execution status can be solved.

3.1. Marking the Precise Timestamp

To optimize the energy consumption of small code segments, knowledge regarding their execution times should be obtained. The execution time can be calculated using the timestamp related to when the
To optimize the energy consumption of small code segments, knowledge regarding their execution times should be obtained. The execution time can be calculated using the timestamp related to when the code segment starts and ends. In the previous section, we verified that real time cannot be guaranteed in a general-purpose operating system (GPOS). Thus, we introduced an intermediate device with a real-time operating system (RTOS) to help mark the timestamps and guarantee their accuracy. In this case, we required a method to notify the intermediate device of the runtime status of the code segment on the device being measured. Notification does not introduce much overhead, with a minimal effect on the execution process. Fortunately, most embedded systems possess GPIO interfaces; it only takes a few instructions to change the GPIO pin level within a few processor cycles, with the code execution being almost entirely unblocked and unaffected. Figure 1 shows the process of logging timestamps using GPIO.

The toggle frequency of the GPIO interface and the time required to change the pin level are provided on the datasheet by chip vendors. For example, the STM32F4 series requires only four clock cycles to change its GPIO pin level. Function calls of gpio_high() and gpio_low() are placed just before and after the code segment to produce GPIO pin level changes. The measured embedded system connects the GPIO pin(s) to those of the intermediate device. When the intermediate device senses a GPIO pin rising edge, it logs the start timestamp, followed by it logging the end timestamp when the falling edge arrives.

3.2. Obtaining Power Consumption Data

Due to the short execution time of code segments, the power consumption data must be updated at a high frequency, as otherwise some details get lost. As mentioned in Section 2.1, the sampling power consumption data at a high frequency using the built-in DAQ or power meter occupies a notable CPU time, introduces additional overhead, and leads to inaccurate power consumption data. In this case, the power consumption data should be measured and collected outside the embedded system. For devices without a built-in DAQ or power meters, the measurement and collection of the power consumption data from outside the device seems to be the only choice.

3.3. Timestamp and Data Synchronization

For the hardware design and code optimization, it is necessary to understand the energy consumption corresponding to the code, which requires the power consumption data to be synchronized with the execution status of the code segments. However, the code is executed on the embedded system and the power consumption data are measured and collected outside the embedded system; therefore, there is no common time baseline between them. GPOS like Windows or Linux do not guarantee real time. Therefore, we decided to use the intermediate device to handle the timestamp and the power consumption data. Figure 2 shows how the timestamps and power consumption data are synchronized.
Figure 2. Time sequence of the synchronized power consumption data and timestamp.

The upper half represents the embedded system being measured, wherein the code is run and the GPIO level changes are produced. The lower half represents the intermediate device, which collects the power consumption data continuously and uses interrupt service routing (ISR) to respond to GPIO level changes and to record timestamps. In this way, the power consumption data and the timestamp share a common time baseline. Because the intermediate device runs an RTOS, the accuracy of the timestamps is also guaranteed.

4. Implementation

Experiments were performed on an STM32F407 core board as the target device, in order to verify the capability of our design. STM32F407 is a widely used microcontroller based on ARM Cortex-M4, and its operating frequency can reach up to 168 MHz. In our project, a KEITHLEY 2280S-32-6 type digital power supply was used to collect the power consumption data. Its sampling rate can reach 25 kHz when the measurement resolution is set to 3½ digits, and the sampling rate is 5 kHz when the resolution is set to 4½ digits [49]. It is equipped with a General Purpose Interface Bus (GPIB, or Institute of Electrical and Electronic Engineers (IEEE) 488 bus), ethernet, and Universal Serial Bus (USB) interfaces, and it supports queries using commands like Standard Commands for Programmable Instruments (SCPI). In practical applications, we also investigated another slightly different, simplified auxiliary design.

4.1. The Host Computer Software

The host computer software is the control center of the entire automatic energy consumption measurement platform. To achieve an automatic and unattended measurement, the host computer software is designed to finish a series of jobs automatically. Figure 3 shows the workflow of energy consumption measuring. Many tasks in the process of the energy consumption measurement are completed automatically by the host computer software, including cross-compiling test cases, programming binary files to the target board, configuring the output parameters of high-precision power supplies, fault recovery, final data collection, processing, display, etc.
For the design and implementation of the host computer software, two ideas were considered. One was that the host computer software indirectly controls the operation of the entire measurement platform through an intermediate device, and the other was that the host computer is directly connected to the target board and high-precision digital power supply through a specific peripheral interface.

4.1.1. With a Standalone Device

In this design, the host computer is only connected to the intermediate device, rather than directly connecting the target board and high-precision digital power supply. In general, most personal computers do not have convenient GPIO interfaces, and general-purpose operating systems like Windows or Linux are not real-time operating systems, so an intermediate device between the host computer and the board is introduced for testing. The host computer software sends different control instructions, such as power supply configuration, measurement start/end, and target board reboot instructions, to the intermediary device according to the execution steps, and the intermediary device returns the collected data and information to the host computer software. The software draws a graph of the processed real-time voltage and current data and also records all received data for future use. The data are logged in separate files for every test case.

4.1.2. With an Auxiliary Device

According to the present idea, GPIO is an important part of the automatic energy consumption measurement platform. A peripheral device making it possible to easily perform some GPIO operations on personal computers was considered. If such a peripheral device exists, the host computer would directly connect with the target board and the high-precision digital power supply in the corresponding design. Even if some other performance indicators were to have some tradeoffs when compared to the use of independent intermediate equipment, the design of this automatic energy measurement platform may be more concise and intuitive.

The host computer software directly configures the power supply output parameters, such as the output voltage and maximum output current, before each test case starts, so that the designers and researchers do not have to reconfigure the power supply manually every time. It also queries and logs the real-time power supply output data without any agency or repeater. The software performs a cross-compilation for the source code files of each test case and then performs the programmed task. These tasks were previously performed manually and in series, but with this automatic host computer software they can be completed automatically. In our design, the host software reboots the target board into different working modes by controlling the flow control pin level of the USB to TTL Serial (USB-TTL) module (connected to the reset (RST) and boot mode (BOOT) pins of the target board), such as the in-system programming (ISP) mode and normal operation mode. The image programming
can be done through the serial port, or through the Joint Test Action Group (JTAG) by using external programs. Powering through the JTAG programmer must be avoided.

Although differences exist in the specific implementation of different hardware arrangements, the two designs still show many similarities. Large quantities of cases to be tested can exist simultaneously, and, in the past, researchers compiled executable binary files from source codes in advance. In our designs, the host computer software could undertake the cross-compilation of test cases, different algorithms, compilation optimization options, and cross-compilation toolchains, which all impact the generated instructions in binary files. Therefore, researchers would only need to write different test case source code and corresponding makefiles. For an unattended measurement, all real-time power supply output data would be captured and recorded in the logfile for each test. For a fault recovery, when the software does not receive data after the timeout, the power supply or target board could be assumed to be down. In this circumstance, a hard reset would be performed by the software, and both the power supply and the target board would be automatically restarted.

### 4.2. The Intermediate Device

In this section, we propose using GPIO as a link between the energy consumption data and running the code on the target board.

#### 4.2.1. Design A: The Standalone Design

To work with GPIOs efficiently, we used a standalone STM32F7 kit board as the intermediary device in this design, equipped with plenty of GPIO interfaces and able to respond to a GPIO level flip frequency of up to 54 MHz [50]. We ran an RTOS on this device to guarantee real time. Researchers generally program every binary file manually, so we also used an intermediary device to accomplish this task. Figure 4 shows the hardware architecture and wiring of the standalone design.

![Figure 4. The architecture and wiring of the standalone design.](image)

We developed some programs on this device to help download writing data, including booting the target board into the flash or ISP mode, erasing if necessary, and rebooting the target into the normal mode. This device followed instructions from an automatic measurement software controlling the power supply output so as to have it turn on or off in order to start or stop the testing. It also collected real-time output data during tests, synchronized with the beginning–end signal, and sent these data back to the host computer. Some GPIOs on the intermediary device board could be used as signals or interruption triggers.

#### 4.2.2. Design B: The Simplified Auxiliary Design

In practice, we found that the wiring of the intermediate device used in the standalone design was complicated and that additional programming development work was needed. In our standalone...
design, we used an STM32F7 board as the intermediate device. However, the sampling frequency of most digital power supplies/multimeters is much lower [49], and the high performance of a GPIO interface cannot be fully utilized. Millisecond accuracy is sufficient in some cases; therefore, we attempted to simplify the design by replacing the intermediate device in the standalone design.

In practice, we found that USB-TTL modules possessed many of the features that we needed. The USB-TTL module is capable of serial communication and, in addition to the pins required for data transmission and reception, it also has several pins for flow control, such as Data Terminal Ready (DTR), Data Set Ready (DSR), Request To Send (RTS), and Clear To Send (CTS). These flow control pins can be multiplexed as GPIO input and output pins.

Figure 5 shows the wiring of our simplified auxiliary measurement platform. The target board’s GPIO pins were connected to the module’s flow control pins. To program an image file to the target board, the target board’s serial port pins were connected to the module’s serial transceiver pins. Furthermore, the flow control pins could be connected to the RST and BOOT pins of the target board to guide the target board to restart and enter different working modes.

The auxiliary design has a relatively simplified architecture, but it has some limitations. In this design, timestamps are marked on the host computer; the accuracy of the timestamp is limited to the millisecond-level. Besides, there are only four flow control pins on the USB-TTL module: two for “digital out” and the other two for “digital in”; however, this issue can be “solved” by using multiple USB-TTL modules.

4.3. Rest Parts of Our Designs

In addition to the host computer software and hardware devices mentioned above, we also used some ready-made equipment in our design, such as high-precision digital power supplies, power meters, and multimeters. In addition, the GPIO signal played a very important role in our design, helping us understand the real-time running status of the board. Piling and modification in the source code were also required.

The digital power supply, power meter, or multimeter provided readings of the real-time voltage, current, and power consumption data. For all devices to collaborate effectively, the instrument was equipped with communication ports like GPIB, Ethernet, or a serial interface, as well as support queries using commands like SCPI.

To generate the GPIO level changes shown in Figure 2, the application code running on the target board also required some modification. A GPIO call was added to produce a rising edge just before the beginning of the tested code segment, as well as to produce a falling edge right after the end of

---

**Figure 5.** The architecture and wiring of the simplified auxiliary design.
the code segment. These beginning–end signals were marked as beginning–end timestamps, thereby marking out the application runtime status of the power consumption data.

5. Evaluations and Results

To verify the capability of our platform, we designed several test sets from the perspectives of power consumption, execution time, and energy consumption, respectively. We also manually conducted measurements using multimeters according to [10] as the control group.

5.1. STM32F4 in Different Power Modes

The STM32F4 was designed with three different power-saving modes, two of which also use different voltage regulators, namely, the sleep and stop modes. These different power modes obviously have different runtime power consumption levels. As we mentioned at the beginning of the previous set of tests, in addition to time, the other major factor affecting energy consumption is power. In this set of tests, we designed six groups of test cases to measure the operating current of the STM32F4 in different power states. These included the normal working mode, the sleep and stop modes using the main regulator, the sleep and stop modes using the low-power regulator, and the standby mode. The results are shown in Figure 6, with (a) showing the working current of different power modes and (b) showing the difference between the standalone design and the simplified auxiliary design.

![Figure 6. (a) Working current of different power modes measured using the standalone design and the simplified auxiliary design. (b) Difference (%) between the standalone design and the simplified auxiliary design.](image)

Although the power measurement is done using off-the-shelf instruments, we added some new content unique to our design to this set of tests. We used the intermediate device to send the changed level signal to the GPIO of the target board as an external interruption signal to wake the target board from the sleep, stop, or standby modes.

5.2. Different Types of Instructions on STM32F4

Energy consumption is based on both power rate and duration; the power rate can be calculated from the power supply output voltage and output current.

In our designs, the period was timed by the rising and falling edges. Thus, we verified the ability of our designs to measure time intervals. Because the intermediate devices used in our designs were not the same, the performance of the STM32F7 in response to GPIO level changes was much better than that of Windows/Linux using the USB-TTL module in response to the level changes of the flow control pins; the instructions in the test case were simply duplicated 1000 times without using loop statements. We also compared the difference between various STM32F4 instruction types. In this test, the clock frequency of the STM32F4 processor was set to 8 MHz. Figure 7a shows the execution time of...
different instruction types, and Figure 7b shows the difference between the standalone design and the simplified auxiliary design.

![Figure 7](image-url)  
(a) Execution time of different test groups for the STM32F4 instruction types measured using the standalone design and the simplified auxiliary design.  
(b) Difference (%) between the standalone design and the simplified auxiliary design.

5.3. STM32F4 Processor Running with Different Frequency

According to the STM32F4 datasheet and user manual, the STM32F4 processor has a maximum frequency of 168 MHz, which can be tuned by modifying the phase locked loop (PLL) parameters. We designed a series of test cases with 40 sets of different PLL parameter combinations, setting the processor to run at different frequencies. Table 1 shows the trend of the power supply output current according to different processor frequencies by setting different PLL parameters on the STM32F4 processor. Due to space limitations, we only present the test results of some frequency points here. In this test, the power rate and the execution time were also variables. We made a comprehensive comparison of the readings of both of our designs according to the measurement method described in [10].

In this test, we performed measurements in three groups, including our standalone and simplified auxiliary designs, with the measurement method in [10] as the control group. The test case is executed multiple times in a loop, and the execution time is about one second. For the standalone design and auxiliary design, the timestamp is marked using GPIO. For the control group, the execution time is calculated by the time register value before and after the execution. The results are shown in Table 2 and Figure 8. The difference between the readings of the two designs was around 1.35%, and this is deemed to be due to different accuracy values of the timestamp. Different designs showed impacts on the measurement results; however, according to all three test groups, the difference in readings was deemed acceptable. Both the standalone and simplified auxiliary designs of the automatic measuring platform exhibited a similar accuracy.

![Table 2](table-url)

Table 2. Energy consumption of the STM32F4 processor running at different clock frequencies measured using our designs and the measurement in [10].

| Test Item | Energy Consumption (μJ) |  |  |
|-----------|-------------------------|---|---|
| 1 MHz     | 8.289400                | 8.3197 | 7.3645 |
| 2 MHz     | 8.326911                | 8.5526 | 7.7057 |
| 4 MHz     | 8.877227                | 9.0001 | 8.2953 |
| 8 MHz     | 9.948424                | 10.1476 | 9.4501 |
| 16 MHz    | 12.053634               | 11.7096 | 12.8616 |
| 32 MHz    | 17.795274               | 17.7466 | 17.2399 |
| 64 MHz    | 24.290318               | 24.4644 | 24.6769 |
| 128 MHz   | 39.492187               | 39.0114 | 40.6801 |
Figure 8. Energy consumption of the STM32F4 processor running at different clock frequencies measured using our designs and the manual measurement.

We also noticed a consistent difference between our two designs and the manual measurements. Judging from the results, this difference was not related to the actual energy consumption and was deemed to be due to the bias introduced by the use of different wiring methods used in the automatic measurement platforms and manual measurement techniques.

5.4. Energy Consumption of µC/OS System Functions

µC/OS is a widely used, opensource, real-time operating system that we used in accordance with our design to measure the energy consumption of its system functions. Due to the execution time being so short, it was only measured with the standalone design. In this test, we used an STM32F407 discovery board as the target device, with its processor frequency set to 53.76 MHz. Figure 9 shows the energy consumption of different µC/OS system functions.

Figure 9. Energy consumption of the µC/OS system functions measured with the standalone design.
For most µC/OS system functions, the execution time is calculated via GPIO level changes generated by calling GPIO functions. For os_init(), the initialization function of µC/OS, the intermediate device changed the level of the RST pin on the target device and marked the start timestamp at the same time; after µC/OS was initialized, a GPIO rising edge was generated, and the intermediate device marked it as the end timestamp. The results showed that our standalone design could be used to measure the energy consumption of small code segments or functions, with even the execution time being quite short. The results also proved that the execution time is an impact factor of energy consumption.

5.5. Automation and Unattended Tests

Over 1200 test cases and 40 different hardware environment configurations were designed by our group, with over 8000 tests conducted over the course of an entire week. In light of our automatic measuring platform, all the tests listed above were executed automatically. The platform skipped to the next test case if one test case failed three times. This automated and unattended measuring platform allowed us to diverge from the repetitive compilation, programming, measuring, and recording tasks, so that we could focus more on analyzing the measurement results.

6. Conclusions

In this paper, we introduced two designs of automatic energy consumption measuring platforms for embedded systems, one using a standalone device and the other using an auxiliary peripheral device. We used the GPIO level signal as a link to establish a connection between power consumption data and runtime status using the target board. By using GPIO, we solved the problems related to obtaining high-precision timestamps and the synchronization between timestamps and power consumption data. Our designs simplified and eliminated many complicated manual steps in embedded system energy consumption measurement tasks, allowing energy consumption measurement to be unattended and carried out automatically. Our standalone design is suitable for occasions requiring a higher sampling frequency and more GPIO signals, and our auxiliary design is suitable for occasions where only a single GPIO signal is required with more concise wiring. Although the hardware and software of these two designs were different, they demonstrated similar accuracy and automated measurement capabilities. We designed groups of test sets to verify our designs’ capabilities, with the test results showing that our designs measured energy consumption automatically, in real time, and accurately. We also used our standalone design to measure the energy consumption of the µC/OS system functions. Finally, our designs are easily “upgraded”. For the standalone design, the adoption of an intermediate device with a higher performance or using a power-sensing module with a higher sampling frequency could improve its measurement precision and accuracy, and for the auxiliary design, a more compact design could be achieved by using Raspberry Pi or a similar kit-board with a high performance and plenty of GPIO.

Author Contributions: Methodology, H.W.; software, H.W.; investigation, H.W.; writing—original draft preparation, H.W.; writing—review and editing, K.W.; supervision, C.C. All authors read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. König, I.; Memon, A.Q.; David, K. Energy consumption of the sensors of smartphones. In Proceedings of the Tenth International Symposium on Wireless Communication Systems (ISWCS 2013), Ilmenau, Germany, 27–30 August 2013; pp. 1–5.
2. Krintz, C.; Gurun, S. A run-time, feedback-based energy estimation model for embedded devices. In Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS’06), Seoul, Korea, 22–25 October 2006; pp. 28–33.
3. Rattagan, E.; Chu, E.T.H.; Lin, Y.D.; Lai, Y.C. Semi-online power estimation for smartphone hardware components. In Proceedings of the 10th IEEE International Symposium on Industrial Embedded Systems (SIES), Siegen, Germany, 8–10 June 2015; pp. 1–4.

4. Jung, W.; Kang, C.; Yoon, C.; Kim, D.; Cha, H. DevScope: A nonintrusive and online power analysis tool for smartphone hardware components. In Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, Tampere, Finland, 7–12 October 2012; pp. 353–362.

5. Kim, D.; Chon, Y.; Jung, W.; Kim, Y.; Cha, H. Accurate prediction of available battery time for mobile applications. *ACM Trans. Embed. Comput. Syst.* 2016, 15, 1–17. [CrossRef]

6. Murmuria, R.; Medsger, J.; Stavrou, A.; Voas, J.M. Mobile application and device power usage measurements. In Proceedings of the 2012 IEEE Sixth International Conference on Software Security and Reliability, Gaithersburg, MD, USA, 20–22 June 2012; pp. 147–156.

7. Lee, J.; Joe, H.; Kim, H. Automated power model generation method for smartphones. *IEEE Trans. Consum. Electron.* 2014, 60, 190–197. [CrossRef]

8. Lee, J.; Joe, H.; Kim, H. Smart phone power model generation using use pattern analysis. In Proceedings of the 2012 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 12–15 January 2012; pp. 412–413.

9. Bokhari, M.A.; Bruce, B.R.; Alexander, B.; Wagner, M. Deep parameter optimisation on android smartphones for energy minimisation: A tale of woe and a proof-of-concept. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, Berlin, Germany, 15–19 July 2017; pp. 1501–1508.

10. Mesa-Martinez, F.J.; Brown, M.; Nayfach-Battilana, J.; Renau, J. Measuring performance, power, and temperature from real processors. In Proceedings of the 2007 Workshop on Experimental Computer Science, San Diego, CA, USA, 13–14 June 2007.

11. Dzhagaryan, A.; Milenkovic, A.; Milosevic, M.; Jovanov, E. An Environment for Automated Measuring of Energy Consumed by Android Mobile Devices. In Proceedings of the 6th International joint Conference on Pervasive and Embedded Computing and Communication Systems, Lisbon, Portugal, 25–27 July 2016; pp. 28–39.

12. Kim, K.; Shin, D.; Xie, Q.; Wang, Y.; Pedram, M.; Chang, N. FEPMA: Fine-grained event-driven power meter for android smartphones based on device driver layer event monitoring. In Proceedings of the 2014 Design, Automation & Test. In Europe Conference & Exhibition (DATE), Dresden, Germany, 24–28 March 2014; pp. 1–6.

13. Ardito, L.; Procaccianti, G.; Torchiano, M.; Migliore, G. Profiling power consumption on mobile devices. In Proceedings of the Energy 2013 The Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies, Lisbon, Portugal, 24–29 March 2013; pp. 101–106.

14. Dong, M.; Zhong, L. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, Bethesda, MD, USA, 28 June–1 July 2011; pp. 335–348.

15. McCullough, J.C.; Agarwal, Y.; Chandrashekar, J.; Kuppuswamy, S.; Snoeren, A.C.; Gupta, R.K. Evaluating the effectiveness of model-based power characterization. In Proceedings of the 2011 USENIX Conference On USENIX Annual Technical Conference, Portland, OR, USA, 15–17 June 2011; p. 20.

16. David, H.; Gorbatov, E.; Hanebutte, U.R.; Khanna, R.; Le, C. RAPL: Memory power estimation and capping. In Proceedings of the ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED), Austin, TX, USA, 18–20 August 2010; pp. 189–194.

17. Hähnel, M.; Döbel, B.; Völp, M.; Härtig, H. Measuring energy consumption for short code paths using RAPL. *ACM Sigmetrics Perform. Eval. Rev.* 2012, 40, 13–17. [CrossRef]

18. Walker, M.J.; Diestelhorst, S.; Hansson, A.; Das, A.K.; Yang, S.; Al-Hashimi, B.M.; Merrett, G.V. Accurate and stable run-time power modeling for mobile and embedded cpus. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 2016, 30, 106–119. [CrossRef]

19. Ahmad, R.W.; Gani, A.; Ab Hamid, S.H.; Naveed, A.; Ko, K.; Rodrigues, J.J. A case and framework for code analysis–based smartphone application energy estimation. *Int. J. Commun. Syst.* 2017, 30, 3235. [CrossRef]

20. Demirbilek, E.; Grgoire, J.C.; Vakili, A.; Reyero, L. Modelling and improving the battery performance of a mobile phone application: A methodology. In Proceedings of the 5th International Conference on Energy Aware Computing Systems & Applications, Cairo, Egypt, 24–26 March 2015; pp. 1–4.
21. Shye, A.; Scholbrock, B.; Memik, G. Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures. In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, New York, NY, USA, 12–16 December 2009; pp. 168–178.

22. Bircher, W.L.; Valluri, M.; Law, J.; John, L.K. Runtime identification of microprocessor energy saving opportunities. In Proceedings of the 2005 International Symposium on Low Power Electronics and Design, San Diego, CA, USA, 8–10 August 2005; pp. 275–280.

23. Chang, N.; Kim, K.; Lee, H.G. Cycle-accurate energy consumption measurement and analysis: Case study of ARM7TDMI. In Proceedings of the 2000 International Symposium on Low Power Electronics and Design, Rapallo, Italy, 25–27 July 2000; pp. 185–190.

24. Gilberto, C.; Margaret, M. Power prediction for intel xscale processors using performance monitoring unit events power prediction for intel xscale processors using performance monitoring unit events. In Proceedings of the ISLPED, Portland, OR, USA, 27–29 August 2007; pp. 8–10.

25. Park, J.G.; Hsieh, C.Y.; Dutt, N.; Lim, S.S. Quality-aware mobile graphics workload characterization for energy-efficient DVFS design. In Proceedings of the 2014 IEEE 12th Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia), New Delhi, India, 16–17 October 2014; pp. 70–79.

26. Lu, Z.; Cao, C.; Tao, X. Improving Screen Power Usage Model on Android Smartphones. In Proceedings of the 2015 Asia-Pacific Software Engineering Conference (APSEC), New Delhi, India, 1–4 December 2015; pp. 167–173.

27. Li, J.; Xiao, J.; Hong, J.W.K.; Boutaba, R. FSM-based Wi-Fi power estimation method for smart devices. In Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, 11–15 May 2015; pp. 147–155.

28. Altamimi, M.L.; Naik, K. A Computing Profiling Procedure for Mobile Developers to Estimate Energy Cost. In Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, Cancun, Mexico, 2–6 November 2015; pp. 301–305.

29. Yoon, C.; Kim, D.; Jung, W.; Kang, C.; Cha, H. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In Proceedings of the Presented as part of the 2012 Annual Technical Conference, Boston, MA, USA, 13–15 June 2012; pp. 387–400.

30. Shukla, N.K.; Pila, R.; Rawat, S. Utilization-based power consumption profiling in smartphones. In Proceedings of the 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I), Noida, India, 14–17 December 2016; pp. 881–886.

31. Kim, Y.G.; Kim, M.; Kim, J.M.; Sung, M.; Chung, S.W. A novel GPU power model for accurate smartphone power breakdown. ETRI J. 2015, 37, 157–164. [CrossRef]

32. Jin, T.; He, S.; Liu, Y. Towards accurate gpu power modeling for smartphones. In Proceedings of the 2nd Workshop on Mobile Gaming, Florence, Italy, 19 May 2015; pp. 7–11.

33. Sun, L.; Sheshadri, R.K.; Zheng, W.; Koutsonikolas, D. Modeling WiFi active power/energy consumption in smartphones. In Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems, Madrid, Spain, 30 June–3 July 2014; pp. 41–51.

34. Yoon, C.; Lee, S.; Choi, Y.; Ha, R.; Cha, H. Accurate power modeling of modern mobile application processors. J. Syst. Archit. 2017, 81, 27–31. [CrossRef]

35. Zhang, L.; Tiwana, B.; Qian, Z.; Wang, Z.; Dick, R.P.; Mao, Z.M.; Yang, L. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, Scottsdale, AZ, USA, 24–29 October 2010; pp. 105–114.

36. Hao, S.; Li, D.; Halfond, W.G.; Govindan, R. Estimating mobile application energy consumption using program analysis. In Proceedings of the 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 18–26 May 2013; pp. 92–101.

37. Dousti, M.J.; Ghasemi-Gol, M.; Nazemi, M.; Pedram, M. ThermTap: An online power analyzer and thermal simulator for Android devices. In Proceedings of the 2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), Rome, Italy, 22–24 July 2015; pp. 341–346.

38. Holleis, P.; Luther, M.; Broll, G.; Souville, B. A DIY power monitor to compare mobile energy consumption in situ. In Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services, Munich, Germany, 27–30 August 2013; pp. 416–421.
39. Snapdragon Profiler—Qualcomm Developer Network. Available online: https://developer.qualcomm.com/software/snapdragon-profiler (accessed on 10 June 2020).
40. EnergyTrace. Available online: https://software-dl.ti.com/ccs/esd/documents/xdsdebugprobes/emu_energytrace.html (accessed on 12 June 2020).
41. STM32CubeMonPwr - Graphical Tool Displaying on PC Power Data Coming from X-NUCLEO-LPM01A – STMicroelectronics. Available online: https://www.st.com/en/development-tools/stm32cubemonpwr.html (accessed on 18 June 2018).
42. Li, D.; Hao, S.; Gui, J.; Halfond, W.G. An empirical study of the energy consumption of android applications. In Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, 29 September 2014–3 October 2014; pp. 121–130.
43. Romansky, S.; Borle, N.C.; Chowdhury, S.; Hindle, A.; Greiner, R. Deep green: Modelling time-series of software energy consumption. In Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, 17–22 September 2017; pp. 273–283.
44. Pathak, A.; Hu, Y.C.; Zhang, M.; Bahl, P.; Wang, Y.M. Fine-grained power modeling for smartphones using system call tracing. In Proceedings of the Sixth Conference on Computer Systems, Salzburg, Austria, 10 April 2011; pp. 153–168.
45. Chowdhury, S.A.; Hindle, A. Greenoracle: Estimating software energy consumption with energy measurement corpora. In Proceedings of the 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), Austin, TX, USA, 14–15 May 2016; pp. 49–60.
46. Pathak, A.; Hu, Y.C.; Zhang, M. Where is the energy spent inside my app? Fine Grained Energy Accounting on Smartphones with Eprof. In Proceedings of the 7th ACM European Conference on Computer Systems, Bern Switzerland, 10 April 2012; pp. 29–42.
47. System (Java Platform SE 7). Available online: https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#nanoTime (accessed on 27 June 2018).
48. System (Java Platform SE 8). Available online: https://docs.oracle.com/javase/8/docs/api/java/lang/System.html#nanoTime (accessed on 27 June 2018).
49. Tektronix Series 2280S Datasheet. Available online: https://www.tek.com/datasheet/series-2280s-precision-measurement-low-noise-programmable-dc-power-supplies (accessed on 10 June 2017).
50. STMicroelectronics STM32F746ZG Datasheet—production Data. Available online: https://www.st.com/resource/en/datasheet/stm32f746zg.pdf (accessed on 10 June 2017).

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).