HardwareX 15 (2023) e00461

Contents lists available at ScienceDirect

HardwareX

journal homepage: www.elsevier.com/locate/ohx

Enerduino-pro: Smart meter led probe using Arduino

Francesco Potortì, Davide La Rosa, Filippo Palumbo [*]

Consiglio Nazionale delle Ricerche, Istituto di Scienza e Tecnologie dell'Informazione, via Giuseppe Moruzzi, 1, 56125 Pisa, Italy

ARTICLE INFO

Keywords:
Arduino
Energy meter
NILM
Reactive power

ABSTRACT

Non-intrusive load monitoring of domestic appliances has received steady interest in the last twenty years, first because of interest from energy companies interested in usage statistics for power balancing and, more recently, in order to assist users in tuning their habits for reduced power consumption. This has increased the need for accurate and economic methods of power measurement that can be efficiently implemented on cheap and easy-to-install platforms. To this end, we present a cheap and efficient device based on Arduino to monitor the usage of domestic appliances in real-time: Enerduino-pro. The design uses low-cost easy-to-assemble open-source electronic components and consists of four main parts: an Arduino UNO microcontroller, one photoresistor to measure instantaneous power absorption plus one optional additional one to measure reactive power, a WiFi shield, and an LED (for debugging purposes only). We describe the device, complete with open software and hardware specifications, and different use cases with proof-of-concept solutions.

Specifications table

Hardware name	Enerduino-pro
Subject area	• *Engineering and material science* • *Educational tools and open source alternatives to existing infrastructure* • *General*
Hardware type	• *Measuring physical properties and in-lab sensors* • *Field measurements and sensors* • *Electrical engineering and computer science*
Open source license	GNU General Public License (GPL) v3.0
Cost of hardware	< 50 $
Source file repository	10.5281/zenodo.6903052

1. Hardware in context

A household smart meter is an electronic device that records electrical energy consumption in regular intervals and makes that information available to the provider by means of a network interface. Aside from measuring the total energy consumption for billing purpose, it has the potential for load monitoring of domestic appliances, a field that has received steady interest in the last twenty years, both due to the attention by energy companies interested in using statistics for power balancing and, more recently, in assisting users in tuning their habits for reduced power consumption.

There are many do-it-yourself (DIY) projects and commercial products out there which are meant to produce a real-time measurement of a household power consumption. Existing solutions often require a common communication infrastructure among the household appliances and, whereas new appliances could be manufactured with the necessary communication and control

[*] Corresponding author.
E-mail address: Potorti@isti.cnr.it (F. Potortì).

https://doi.org/10.1016/j.ohx.2023.e00461
Received 25 July 2022; Received in revised form 19 April 2023; Accepted 30 July 2023
Available online 9 August 2023

Table 1
Comparison of existing solutions based on their key features: type of system (commercial or DIY), sensor used (current transformer or blinks reading), accuracy, reading rate, data display method, open hardware, open software, API.

	Type	Sensor	Accuracy	Reading rate	Data display	Open HW	Open SW	Open API
TED	Commercial	CT	±2% over 3 A, N/A below	N/A	External Energy Control Center	N	N	N
EnergyCloud	Commercial	Blinks	~1%	15 s	Web Portal	N	N	N
ECM-1240	Commercial	CT/Blinks	N/A/~1%	1 Wh	Executable dashboard	N	N	Y
Wattvision	Commercial	Blinks	~1%	7.5 s	Web Portal	N	N	Y
thediylife	DIY	CT	apparent power	2 s	LCD Screen	Y	Y	Y
Enerduino	DIY	Blinks	~1%	1 Wh	SD card or XBee module	Y	Y	Y
Enerduino-pro	DIY	Blinks	~1%	1 Wh	Embedded HTTP server	Y	Y	Y

systems, existing appliances usually do not provide this kind of capability. This aspect leads to the need for a non-intrusive, low cost and easy-to-install electrical end-use appliance load monitoring system for buildings.

To this end, commercially available off-the-shelf sensors exist on the market, such as The Energy Detective (TED) [1], EnergyCloud by Blue Line Innovations [2], Brultech ECM-1240 Energy Monitor [3], and Wattvision [4]. These solutions use two main hardware approaches. The first one is based on current transformers (CTs) clamped around the main incoming wires from the utility service provider inside the breaker panel. The second one is based on a smart meter front-mounted LED that reads electricity usage in real-time via optical readers and transmits via a cellular network or on a powerline to a paired receiving device in the home. These devices typically provide information at a sampling rate of up to 1 Hz and a sensitivity of up to 1 Watt [5].

Following the latter approach, we present an inexpensive hardware solution based on optical sensors applied to the smart meter. It is called Enerduino-pro [6] and its use has been presented in [7] from a software point of view, as a non-intrusive load monitoring (NILM) application.

We chose to implement a smart meter LED probe also because using such an inexpensive device makes both the installation and maintenance easier. The proposed solution is based on an open source and open hardware system, built upon the Arduino platform. We embrace the open source and hardware principles in order to offer a system easily modifiable to suit the user needs and to be used as the basis for new products. Many authors and developers in the makers community based their work on the Arduino platform to perform appliance load monitoring by means of standalone power meter plugs [8,9], wireless sensor networks [10], and dedicated microcontrollers [11], offering different levels of connectivity [12]. We focused on the non-intrusiveness of the system developing a single device easy to install and embedding a light web server in order to avoid dedicated software infrastructures to get real-time information about the energy consumption and the usage of domestic appliances.

In order to better contextualise and compare our solution to the state of the art both in commercial products and in the maker/scientific community, we analyze different key features provided by the most promising systems in the field. We only consider systems that can be used with a sensor put at the energy meter, like the ones proposed by thediylife [13] and Enerduino [14] in the makers community and we exclude multi-sensor systems using a number of radio-equipped plugs.

In Table 1, we list some representative systems and distinguish them by a list of several features, namely: type of system (commercial or DIY), sensor used (current transformer or blinks reading), accuracy, reading rate, data visualisation, open hardware, software and API.

From Table 1 we can observe the following key features of the considered systems with respect to Enerduino-pro:

- **The power sensor**: The main difference among systems is the sensor used to measure the power. Most systems rely on measuring the current flowing through the energy meter using a current transformer. This solution is simple and independent of the specific energy meter model, the only difficulty being to dealing with the live wire running out of the energy meter. Sampling the current allows one to compute the effective current flowing through the wire and obtain a rough estimate of the power by multiplying it by 230 V (in Europe). There is a basic flaw to this simple system: it does not account for non-resistive loads, so in general it overestimates the used power, and the error may be significant for all appliances using some sort of electric engines such as washing machines, dishwashers, heat pumps (including refrigerators and air conditioners), and all appliances including a transformer, the main example being microwave ovens. Moreover, small electronic appliances produce big errors because of their highly non linear current absorption with very high harmonic content. This means that the power collectively used by all small devices (power adapters, non-incandescent light bulbs and generally all electronic devices) is significantly overestimated. Add to this that some devices only measure peak current absorption, rather than the effective current.
  To overcome this limitation, some systems sense not only current, but voltage too. However, this requires turning off the main power and connecting to both the neuter and live wires, so installation becomes slightly more difficult. By sampling

both voltage and current, it is in principle possible to obtain a very accurate power reading if the phase error of the CT is known. In practice, an Arduino has two significant limitations: sampling frequency and sampling resolution. At minimum (8 bit) resolution, an Arduino Uno can get many thousands samples per second, but this requires almost the whole available processing power. In practice, the systems that we have analyzed read 100–300 samples per second and use maximum resolution (10 bits). The reason for using max resolution and slowing down the sampling rate is that the CT has a usually higher-than-necessary range (typical ranges are from 50 A to 400 A) meaning that most of the resolution is lost and the quantisation error is significant. Add the digital-to-analog converter inner error, the CT measurement error, the usually unknown CT phase error and the error on the voltage transformer, and you easily get few percent errors on the higher part of the scale, typically above 1500 W. What happens on the lower part is unknown and errors easily exceed 10% and more for low power absorption.
Sure, if you are mostly interested in the overall power, accuracy in the low range is not much important. But if you are curious about how different appliances behave, or what is the effect of removing an appliance or using it in a different way, you will want a much higher accuracy. Some systems overcome these limitations completely by reading the energy meter LEDs, which blink every 1 Wh. This means one blink per second at 3600 W absorption, with an excellent accuracy (the same as the energy counter's, which is typically below 2%). One downside is that the measurement is slow at low absorption, for example one blink every 18 s at 200 W, but the excellent accuracy remains the same.
- **Active and reactive power**: Systems reading both voltage and current and systems reading the blink on power meters equipped with two LEDs can in principle measure both active and reactive power. In practice, we have found no system other than Enerduino-pro which reads reactive power.
- **Standard real-time external reading**: Enerduino-pro incorporates a minuscule HTTP server which can connect over the Internet and answer standard Ajax queries, so an external system can be used to get all the raw readings, which can be stored offline. Other systems provide a limited interface for reading some specific readings, others only speak with a proprietary server, others only show the current reading on a display.
- **Reading resolution**: Data provided by Enerduino-pro have a resolution of 1 Wh, which is dictated by the rate of the energy meter's blinks; this is equivalent to a reading every 15 s at 240 W absorption, or a reading every second at 3600 W. Most other systems give a reading at a fixed rate, for example every 15 s.

As summarised in Table 1, Enerduino-pro exhibits a unique combination of features that makes it stand out from several points of view. We consider three: researcher/hacker, maker, and commercial producer.

From the researcher/hacker point of view, Enerduino-pro exploits the high measuring accuracy of an energy meter to produce highly reliable and detailed data which can be read and stored offline with minimal effort. The possibility of storing both active and reactive power measurements opens the view on an interesting scenario from the hacker's point of view. From the scientist's point of view, it opens the possibility of storing high-quality, complete data for NILM research.

From the maker's point of view, Furthermore, even the least experienced maker will appreciate how easy it is to connect the single, unpolarised sensor used by Enerduino-pro in the simplest case of reading just the active power. Photoresistors are cheap, widely available and error-proof.

From the commercial producer's point of view, Enerduino-pro can be turned into a commercial product thanks to its free software license, ease of installation, and cheap hardware. It can export an open and easily-accessible interface for the sophisticated enthusiast. Creating a stand for the photoresistor (or photoresistors) is comparatively easy for many flavors of energy meters. A unique feature is that installation is easy and not critical even if the stand is badly aligned with the LED or subject to strong and variable ambient light noise.

As a final note, the Arduino UNO hardware platform has been chosen some years ago, but we think it is still relevant both for historical reasons and because there is not such a big difference between the presented system and more recent ones using updated hardware.

2. Hardware description

The smart meter developed by Enel is currently deployed in about 32 million households in Italy (Fig. 1(a)). It has also been chosen by ENDESA, who has deployed it in 13 million households as of 2018, 1 million in Romania and started pilots in Chile, Brasil, Colombia, Peru, Argentina starting in 2015 [15]. It provides real-time information regarding the active and reactive power consumption by means of two LEDs blinking one time per Wh and one time per Varh consumed respectively. When there is no consumption in the active or reactive power for more than twenty minutes, the corresponding LED remains on until the next activation. Enerduino-pro station gathers this information by means of two photoresistors (Fig. 1(b)).

The two photoresistors are connected to the analog inputs of an Arduino board in order to detect the LEDs blinking (Fig. 2). The system allows the use of different kinds of photoresistors with resistance ranging from 2 kΩ to 20 MΩ in full light and full dark conditions and response time ranging from 5 ms to 30 ms, depending on the model.

Regarding the connectivity capabilities, the Arduino WiFi shield with an external WiFi antenna was the only way to add WiFi, back when this code was developed. Unfortunately, the library supporting it and apparently the hardware itself are not reliable. There are some software tricks in the code to make it more stable, like not writing more than a given number of bytes a time and the like, which were found by trial and error, as shown in the comments. And only one TCP connection at a time could be used, even if the library claims to allow for more. The software tricks increased the stability enough for the board to be usable for establishing an outgoing HTTP connection to read the network clock and for listening to incoming connection on the web server. However, a

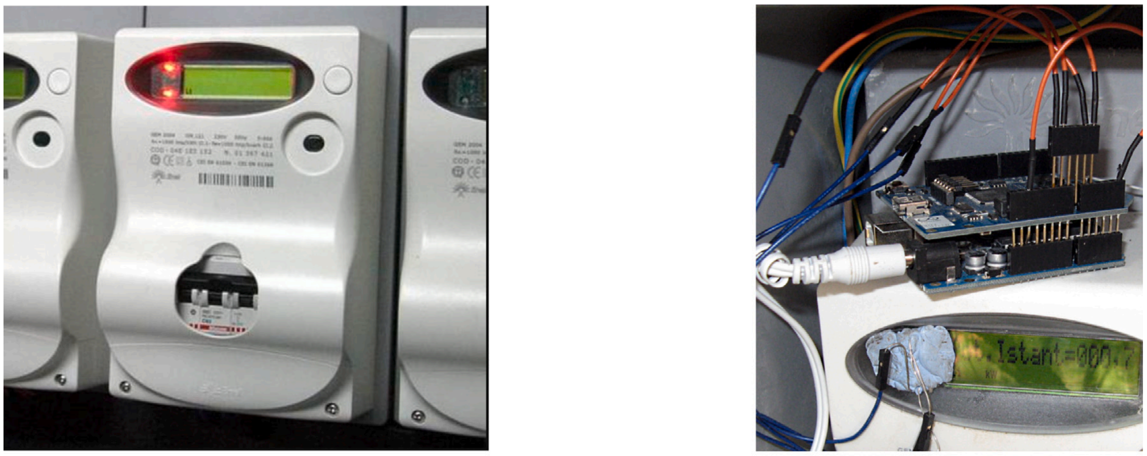

(a) The Enel smart meter. (b) The Enerduino-pro station.

Fig. 1. The proposed solution.

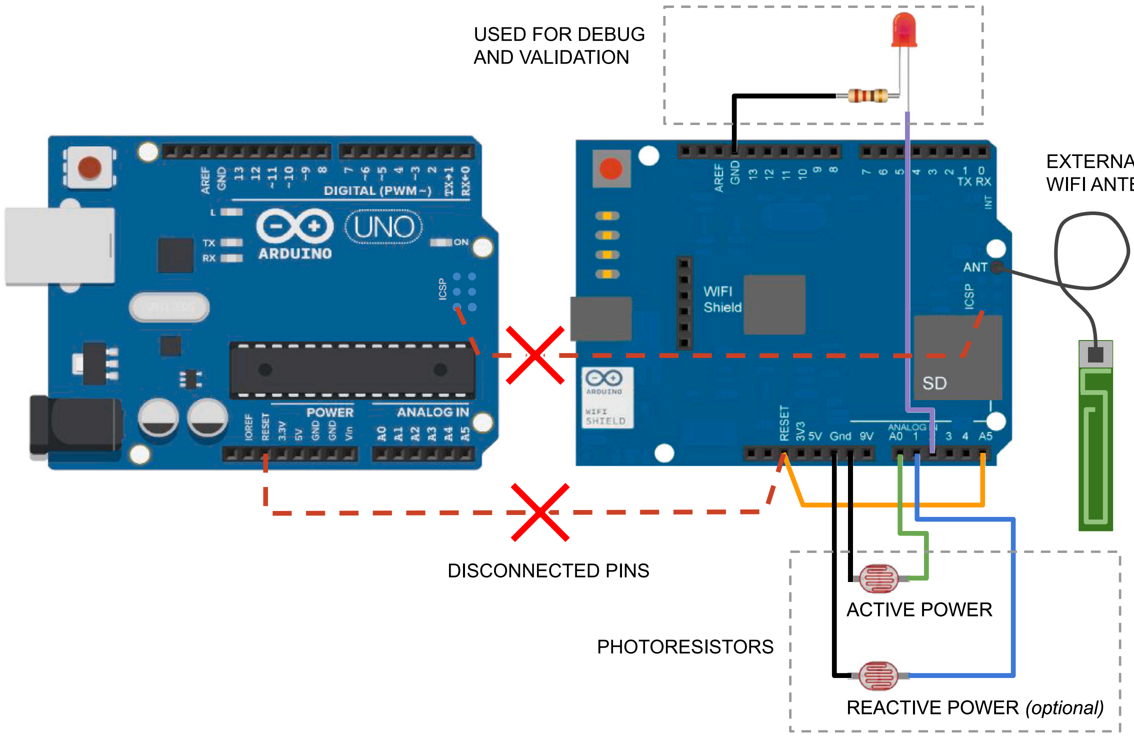


Fig. 2. Diagram of the electrical connections.

few times a day the WiFi board freezes without any apparent reason. Fortunately, this situation can be detected by software. Once the web server task starts, it calls a WiFi.begin() to connect to the local WiFi network. Every time the server task loops, it checks that the board is still connected and, if it has disconnected, resets it. Resetting the WiFi board requires a small hardware trick. In order to speed up the process and most importantly to retain the last readings which are in the readings ring buffer, the main board should not be reset, only the WiFi board. But since the reset pins of both boards are connected together, resetting one and not the other requires, before mounting the shield on the main board, its reset pin to be bent so it does not connect to the main board reset (see Fig. 2). Instead, it is connected to the X output of the main board, which can then reset it via software.

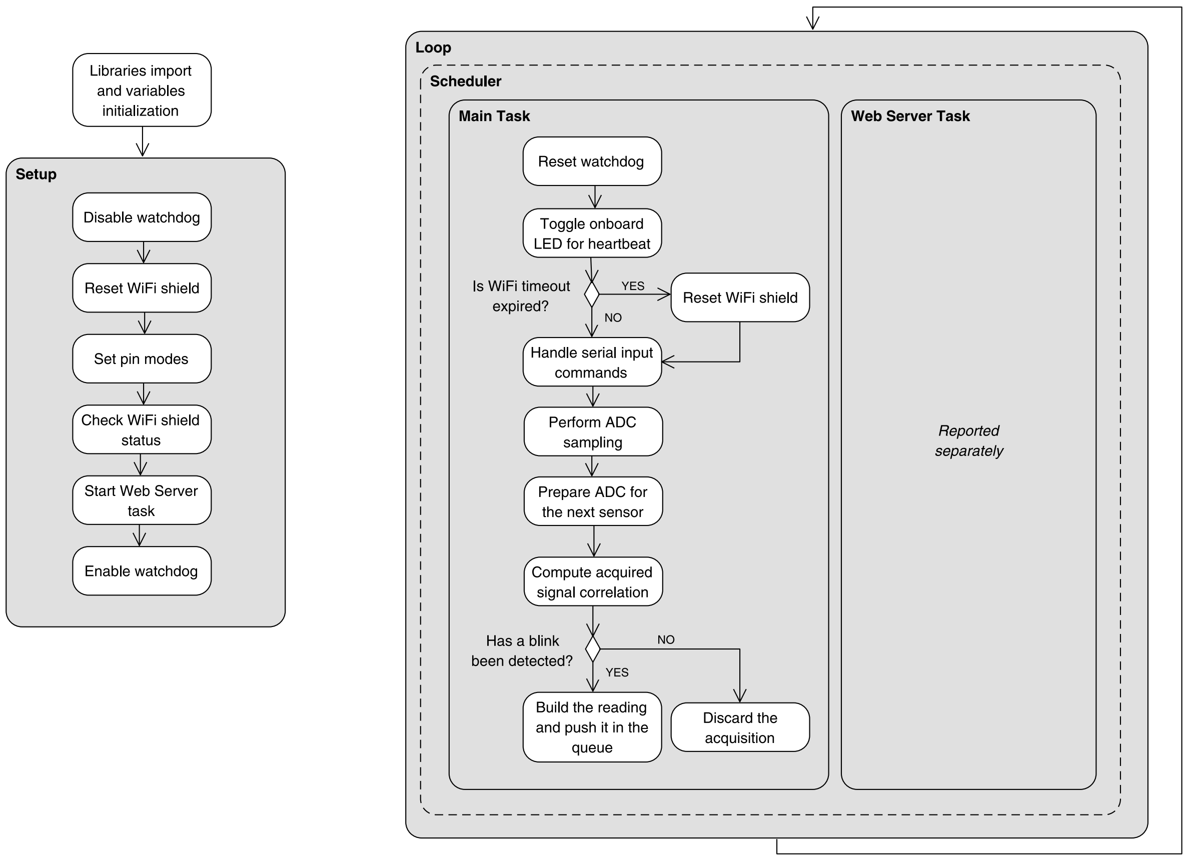


Fig. 3. Overview of the main software operations.

The last piece of hardware composing the Enerduino-pro station is the LED used for debugging purposes. It has a typical forward voltage of 2.0 V and a rated forward current of 20 mA. It simulates the smart meter blinking in order to develop and test smart home application without a real power consumption.

3. Software description

The Enerduino-pro software was developed with the Arduino IDE [16] and consists of three source files:

- *enerduino.h:* header file containing constants initialisation, data types and the definition of the support classes
- *enerduino.ino:* primary source file containing all the sketch logic, including the setup and main loop functions
- *yield.patch:* patch file containing the modifications needed to allow the used libraries to work with the cooperative scheduler

The code uses the Streaming [17], WiFi [18] and SchedulerARMAVR [19] libraries, the latter being a porting on the AVR processor of the standard Scheduler library. The overall software logic is shown in Fig. 3. In the setup phase, while the watchdog is disabled, the WiFi shield is reset, the used pins are set to the correct modes (input for the sensor pins and output for the reset and LED pins), the WiFi shield is then queried to check the status and eventually the web server task is initialised in the scheduler. The main loop splits the execution among two tasks, which are very different from a real-time point of view, by means of a cooperative multitasking scheduler. One is the sensor reading task, which has strict deadlines and relatively long uninterruptible stretches of code. The other one is the web server task, whose deadlines are much longer and can generally be interrupted very frequently.

Managing two such tasks in the simple main loop of Arduino would have been a nightmare, leading to code both unreadable and very difficult to debug, with an enormous number of nested states. On the other hand, resorting to a true real-time multitasking kernel would have meant to give up most of the convenient features of the Arduino development environment. Fortunately, here comes the Arduino Scheduler library, which implements a trivial but small and efficient cooperative scheduler, and Fabrice Oudert who ported it on the AVR processor [20]. The general idea is that the sensor task does its important work without interruptions and, if the next deadline is not too close, it yields (by calling the yield() scheduler function) to the web server task. The web server task, on the other hand, yields to the scheduler task very frequently and in a transparent way: very few yield() calls are to be seen in the code, because the great majority is hidden inside the patched core libraries. In fact, the provided *yield.patch* file

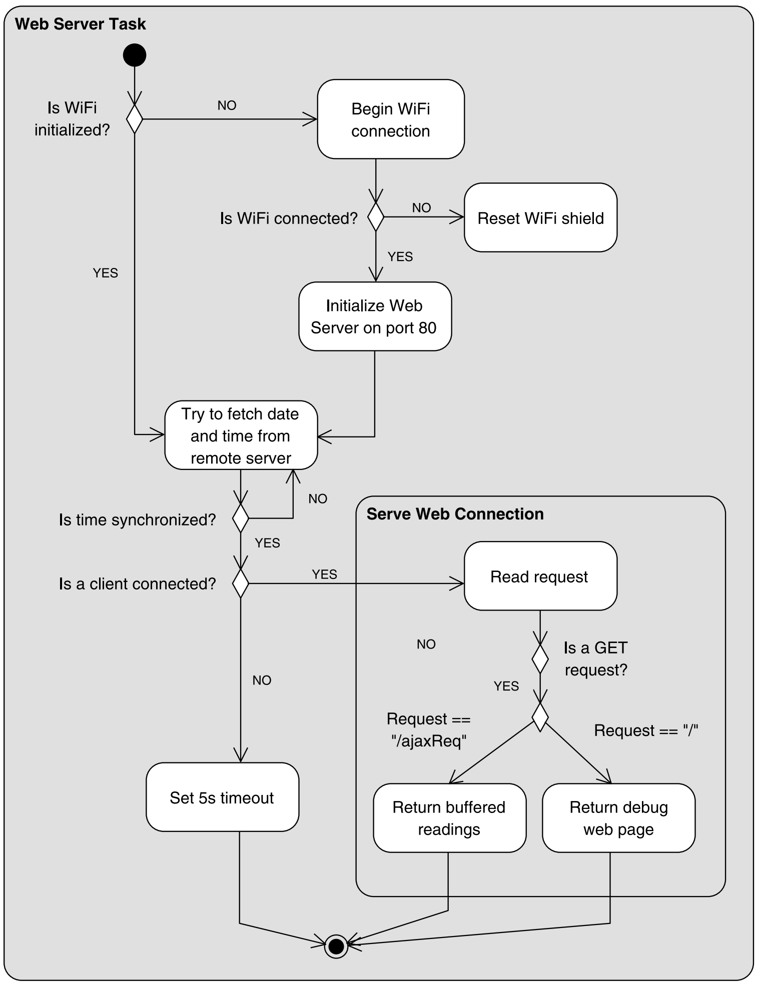


Fig. 4. Operations of the Web Server task.

simply adds a yield() call inside all the polling loops that are involved into blocking calls, like for example `Stream::timedRead`, `analogRead`, `UARTClass::flush`, `HardwareSerial::write`. Validation and tuning of this approach is done through a series of debug controls that check delayed deadlines, lost readings and such, using a procedure detailed in the "Scheduling policy" section of the code.

The web server, whose logic is shown in Fig. 4, can answer two requests: `‘‘GET /’’` and `‘‘GET /ajaxReq’’`; it discards any other request. The first one serves a 1912-bytes page embedded into the source code as a string that displays a dynamically updated table, only useful for debugging, as the server cannot serve simultaneous requests due to bugs in the WiFi shield library. The second one is used by an external requester for gathering the readings. The normal request rate is once every 3 to 6 s, which is enough to keep the 64-entries FIFO reading buffer from overflowing, even accounting for the occasional network problem and high blinking rates. An interesting aspect concerns the stream operator "<<" used in the code. It is overloaded to manage buffering on the data sent by the web server to the TCP stream managed by the WiFi shield library. The main purpose of this overload is to provide buffering, which is necessary for chunked HTTP output. The chunked output allows for greater efficiency on the client side, but here there is a much more compelling reason: sending a limited and well-known number of bytes at a time to the WiFi library, which greatly improves stability. Overloading also allows for some more facilities: unsigned long is converted to ASCII decimal with fixed width; unsigned long long is converted to ASCII hex representation, "endl" is CR LF according to HTTP, and the special const "clear" flushes the output buffer. The XML returned from the `‘‘GET /ajaxReq’’` contains all the readings stored in the buffer since the last request. Every reading is composed of several elements as reported in Table 2.

Depending on the actual power consumption, the number and type of readings packed in the reply message can vary; to highlight this difference we reported two samples of server responses: one during low power consumption of 435 W (Fig. 5) and another during high power consumption of 3 kW (Fig. 6). In the first reply, the second and third blocks contain no readings (no P0/T0 or P1/T1 tags) because the absorbed power is so low that no blinking happens between subsequent Ajax requests. In the second case, all the blocks report multiple P0/T0 power readings: this happens because power absorption is relatively high. In this situation we have

Table 2
Types and meaning of the possible XML elements returned by the Ajax request, as shown in Figs. 5 and 6.

Tag	Description
`<ST>`	First ajax request reading after boot, uptime and debug statistics are reset
`<_>`	Readings block
`<P0>`	Active power in watt
`<T0>`	Time of active power reading in milliseconds since the Unix epoch
`<P1>`	Reactive power in watt
`<T1>`	Time of reactive power reading in milliseconds since the Unix epoch
`<T>`	Time of reading
`<UT>`	Time in seconds
`<WS>`	Debug: unused stack of the web server task in bytes
`<DC>`	Debug: number of late readings while sensor task was working
`<DY>`	Debug: number of late readings while sensor task was yielding
`<RL>`	Debug: cumulative sensor readings discarded because the reading ring overflowed

about 3 kW, which means almost one blink per second for the active power LED (A0). The last block contains also a P1/T1 reading which is related to the reactive power.

4. Design files

For this paper, all the required source files are available and maintained in the file repository of Zenodo: https://doi.org/10.5281/zenodo.6903052.

Design files summary

Design filename	File type	Open source license	Location of the file
Diagram of the electrical connections	SVG figure	Creative Commons Attribution 4.0	Available with article (Fig. 2)
enerduino.h	Header source	GPLv3	Source File Repository
enerduino.ino	INO source	GPLv3	Source File Repository
yield.patch	Patch	GPLv3	Source File Repository

5. Bill of materials

The costs of the electronic system components listed in Section 2 was calculated considering the price difference between vendors.

Bill of materials summary

Component	Qty.	Unit cost	Total cost	Source of materials	Material type
Arduino Uno	1	$ 27.95	$ 27.95	https://www.sparkfun.com/products/11021	Electronics
Photoresistor	2	$ 1.60	$ 3.20	https://www.sparkfun.com/products/9088	Electronics
WiFi Shield	1	$ 15	$ 15	https://www.sparkfun.com/products/retired/11287	Electronics
WiFi Antenna	1	$ 2.95	$ 2.95	https://www.sparkfun.com/products/15877	Electronics
LED	1	$ 0.45	$ 0.45	https://www.sparkfun.com/products/9590	Electronics

6. Build instructions

Building the system is quite an easy task since it requires to perform the few electrical connections shown in Fig. 2. Soldering is not required. Before plugging the WiFi shield on top of the Arduino Uno board, one should ensure the two pins related to the RESET signal are disconnected (by simply bending or cutting them) among the two boards. This is necessary to prevent the reset signal sent to the WiFi shield from resetting also the Arduino board. To connect the electrical components (photoresistors, resistor and LED) to the board, jumper wires are used, while the external WiFi antenna is connected to the WiFi shield through an U.FL miniature RF connector. The final assembled system for normal usage (no debug/validation) and before being mounted on the energy meter, is shown in Fig. 7. The external LED and resistor are required only if the system is used for test or validation. To prepare the code for the uploading to the Arduino board, one should first apply the patch file by running this command from the arduino base folder:

```
patch -s -p0 < yield.patch
```

Next, one should connect the Arduino Uno to the PC with a USB cable, open the source code with the Arduino IDE and under *Tools >Boards* select *Arduino Uno*. Under *Tools >Port* make sure the correct COM is selected. Set the correct WiFi network SSID in the code by editing the variable `ssid`. If the network also requires a password, one can add it in the `WiFi.begin()` call. The program can then be uploaded by pressing the corresponding button on the IDE.

During the assembly operations, the only safety concern regards the possibility of short-circuiting the boards due to mistakes in the pin connections. That might cause damages to the main board or to the shield when the power supply is provided.

```
<?xml version='1.0'?>
<_>
    <ST></ST>
    <P0>435</P0>
    <T0>0x0143E3131D61</T0>
    <T>0x0143E3133EC0</T>
    <UT>13</UT>
    <WS>35</WS>
    <DC>36</DC>
    <DY>4</DY>
    <RL>0</RL>
</_>
<?xml version='1.0'?>
<_>
    <T>0x0143E3134C2E</T>
    <UT>16</UT>
    <WS>35</WS>
    <DC>36</DC>
    <DY>8</DY>
    <RL>0</RL>
</_>
<?xml version='1.0'?>
<_>
    <T>0x0143E31353BF</T>
    <UT>18</UT>
    <WS>35</WS>
    <DC>36</DC>
    <DY>8</DY>
    <RL>0</RL>
</_>
<?xml version='1.0'?>
<_>
    <P0>435</P0>
    <T0>0x0143E3133DBB</T0>
    <T>0x0143E3135FFC</T>
    <UT>22</UT>
    <WS>19</WS>
    <DC>36</DC>
    <DY>8</DY>
    <RL>0</RL>
</_>
<?xml version='1.0'?>
<_>
    <T>0x0143E3136790</T>
    <UT>24</UT>
    <WS>19</WS>
    <DC>36</DC>
    <DY>8</DY>
    <RL>0</RL>
</_>
<?xml version='1.0'?>
<_>
    <T>0x0143E31373EF</T>
    <UT>27</UT>
    <WS>19</WS>
    <DC>36</DC>
    <DY>8</DY>
    <RL>0</RL>
</_>
```

Fig. 5. Example of a reply from the web server in a low power consumption (435 W) situation. The tags used in the XML code are described in Table 2.

7. Operation instructions

There are two ways in which the system can be operated: normal or in debug/validation configuration.

In the normal configuration, the system is installed in the proximity of the energy meter and the photoresistors are fixed in front of the energy meter LEDs by applying a putty-like adhesive (Fig. 1(b)). Once installed, plug the power supply and wait until the Arduino connects to the WiFi network. When it successfully connects to the network, the onboard LED goes from a slow blinking pattern (2 s) to a fast blinking pattern (0.5 s). At this point the web server is reachable at the IP address assigned to the Arduino by the WiFi router and the queries described in Section 3 can be performed from any device connected to the same network.

```xml
<?xml version='1.0'?>
<_>
    <ST></ST>
    <P0>2922</P0>
    <T0>0x0143E049D8E2</T0>
    <P0>2927</P0>
    <T0>0x0143E049DDB2</T0>
    <T>0x0143E049E611</T>
    <UT>15</UT>
    <WS>19</WS>
    <DC>36</DC>
    <DY>8</DY>
    <RL>0</RL>
</_>
<?xml version='1.0'?>
<_>
    <P0>2946</P0>
    <T0>0x0143E049E280</T0>
    <P0>2944</P0>
    <T0>0x0143E049E746</T0>
    <P0>2927</P0>
    <T0>0x0143E049EC0D</T0>
    <T>0x0143E049F37C</T>
    <UT>18</UT>
    <WS>19</WS>
    <DC>36</DC>
    <DY>8</DY>
    <RL>0</RL>
</_>
<?xml version='1.0'?>
<_>
    <P0>2910</P0>
    <T0>0x0143E049F0DB</T0>
    <P0>2915</P0>
    <T0>0x0143E049F5B0</T0>
    <T>0x0143E049FB10</T>
    <UT>20</UT>
    <WS>19</WS>
    <DC>36</DC>
    <DY>8</DY>
    <RL>0</RL>
</_>
<?xml version='1.0'?>
<_>
    <P0>2908</P0>
    <T0>0x0143E049FA83</T0>
    <P0>2922</P0>
    <T0>0x0143E049FF59</T0>
    <T>0x0143E04A0775</T>
    <UT>23</UT>
    <WS>19</WS>
    <DC>36</DC>
    <DY>8</DY>
    <RL>0</RL>
</_>
<?xml version='1.0'?>
<_>
    <P0>2903</P0>
    <T0>0x0143E04A0429</T0>
    <P0>2899</P0>
    <T0>0x0143E04A0901</T0>
    <T>0x0143E04A0F0C</T>
    <UT>25</UT>
    <WS>19</WS>
    <DC>36</DC>
    <DY>8</DY>
    <RL>0</RL>
</_>
<?xml version='1.0'?>
<_>
    <P0>2882</P0>
    <T0>0x0143E04A0DDB</T0>
    <P0>2899</P0>
    <T0>0x0143E04A12BC</T0>
    <T>0x0143E04A1B4A</T>
    <UT>28</UT>
    <WS>19</WS>
    <DC>36</DC>
    <DY>8</DY>
    <RL>0</RL>
</_>
<?xml version='1.0'?>
<_>
    <P1>247</P1>
    <T1>0x0143E049E346</T1>
    <P0>2920</P0>
    <T0>0x0143E04A1796</T0>
    <P0>2915</P0>
    <T0>0x0143E04A1C67</T0>
    <T>0x0143E04A22EB</T>
    <UT>30</UT>
    <WS>19</WS>
    <DC>36</DC>
    <DY>8</DY>
    <RL>0</RL>
</_>
```

Fig. 6. Example of a reply from the web server in a high power consumption (3 kW) situation. The tags used in the XML code are described in Table 2.

In the debug/validation configuration the board is not installed on the energy meter, it is instead connected to a host PC through a USB cable and the external debug LED is closely pointed towards one of the photoresistors. When the board is connected to a PC, at startup it detects the presence of a USB serial interface and enters debug mode. In this configuration the external debug LED blinks every 1.8 s, simulating a power consumption of 2 kW. In debug mode, the serial interface is initialised as well, messages are occasionally printed on the console and some commands, triggered by pressing a key on the keyboard, are accepted. The commands are:

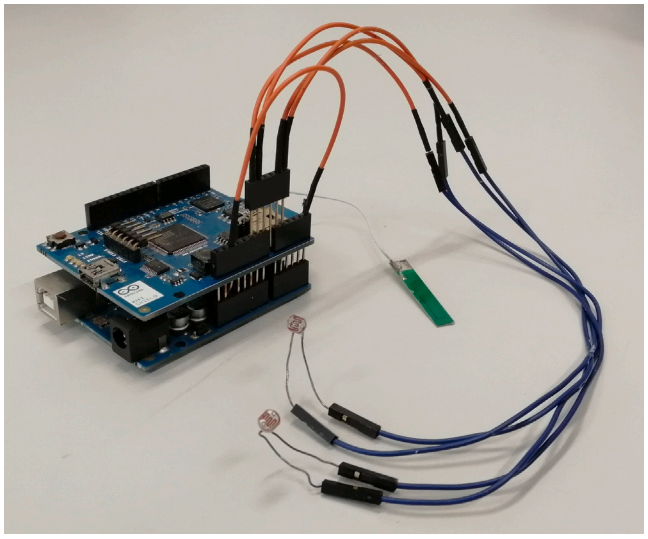


Fig. 7. Fully assembled system ready to be installed on the energy meter.

- **'m'** reset readings and statistics
- **'r'** go into reading state and proceed to normal loop
- **'i'** stop the reading task, do not yield to the serve task
- **'f'** blink the external debug LED for 20 ms, if the debug LED option is active
- **'F'** switch the external debug LED on, if the debug LED development option is active
- **'d'** print statistics
- **'D'** as above, plus print correlator prototypes for all correlators
- **'y'** force yield to the server task
- **'R'** reset the board, restart from scratch

These commands allow to test different functionalities of the software and simulate several power consumption conditions; for instance by pressing **'f'** once per second, we can simulate 3600 W of power absorption.

8. Validation and characterisation

The core functionality of Enerduino-pro system is its capability of detecting the blinks of a generic smart meter installed in private buildings. The blink detection allows us to have a fine-grained information about the overall power consumption of the building and from it the consumption of specific appliances in use.

8.1. Blink detection

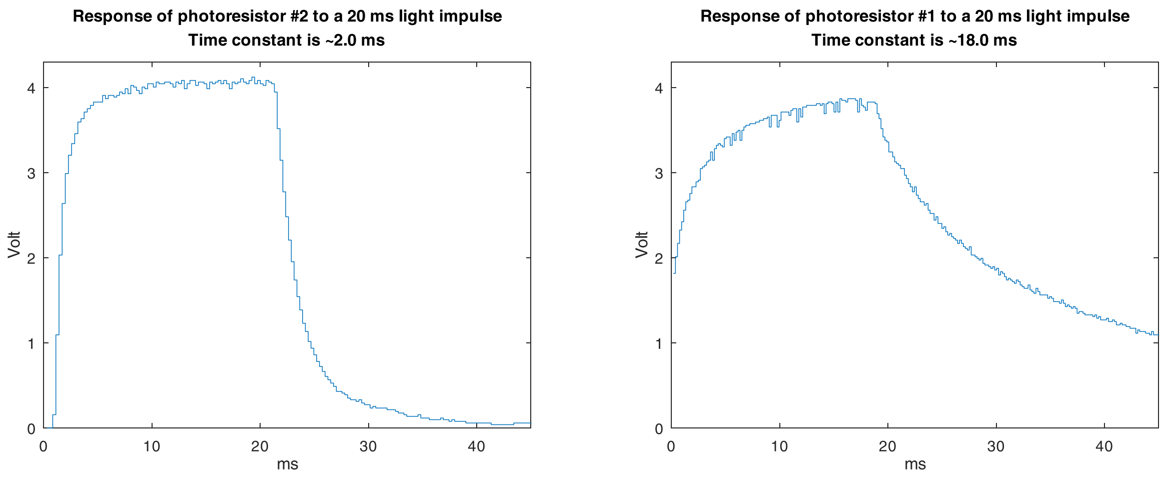
Blink detection passes through three phases. In the first phase, every 2.5 ms, if the power of the signal recorded by the sensor in a sliding window of 225 ms (equal to 90 periods of 2.5 ms) exceeds a small threshold (10% of the maximum ADC output), covariance is computed respect to three prototype responses. Each response is a 20 ms rectangular blink passed through a one-pole filter with time constants of 5, 12 and 30 ms in order to account for different photoresistors found on the market. The three prototypes allow for a good match of photoresistors having response times in the range from 0 to 50 ms, and a reasonable match for possibly slower devices.

In the second phase, for each prototype response, if covariance turns out to be positive, then correlation is computed. If correlation exceeds a threshold of 0.5, then a blink has been detected.

In the third phase, the correlation is tracked and its maximum value is stored. An envelope detector is applied to guard against cross-interference form the other blinking led. The envelope detector decays with a time constant of 128 blinks.

One key purpose of this project has been to make it very easy to use. In practice, this means mainly two things. First, it should be easy to assemble, from which the choice of using a photoresistor as a sensor over a photodiode or a photoresistor. Second, it should be robust in the face of photoresistor positioning.

A photoresistor is cheaper than a photodiode or a phototransistor, even if it can be argued that the price difference pales when compared with the price of an Arduino board, so this is one reason for preferring a photoresistor, but not the main one. The main reason is that photoresistors are not polarised, so they can be mounted in either direction: this makes it easier for non-expert makers and completely removes one reason for bad assembling for anyone.



(a) Sample photoresistor with response time less than 2 ms. (b) Sample photoresistor with response time of 18 ms.

Fig. 8. Response of two sample resistors perfectly aligned with a LED blinking a 20 ms pulse. Measures are taken with the ADC set to 256-bit resolution and a resulting rate of about 5 kHz.

A perfect positioning of the light sensors would be to stick them just on front of the LEDs on the energy meter. This would guarantee that the full light of the led hits the sensor and that ambient light has minimum influence on the reading, an ideal situation which would require just a threshold reading in software, with a sampling rate twice as fast as the light pulse width. But the idea was that one could build a simple wooden or plastic L-shaped stand to be put on the top of the energy meter, one leg of the L on the top and the lower leg carrying two sensors to stand in front of the LEDs. This arrangement would allow for very simple installation, with nothing sticking to the energy meter. The problem with this arrangement is that ambient light would disturb the reading. Moreover, if the meter (as it happens in a condominium) is behind a glass door in the building entrance, the ambient light could significantly and abruptly change whenever people turn on the lights.

To get rid of this problem, a threshold is not enough. One needs something that recognises the shape of the led light pulse: a rectangle of a given length. Here is where the need for a correlator arises. However, once the shape of the pulse is measured to get the length of the pulse, it turned out that the speed of the photoresistor is low, sometimes of the same order of magnitude of the pulse length, depending on the photoresistor.

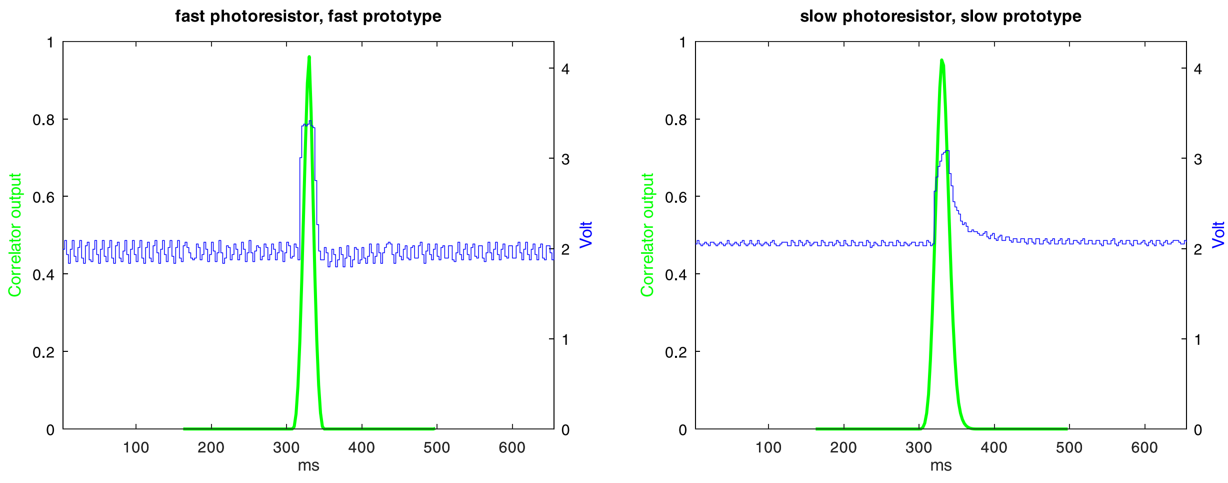
The data sheets of the PGM series, which are the most common on the market, indicate "response times" in the range of 20 to 40 ms. However, photoresistors are known to exhibit "light history" effects, and data sheets response times are apparently measured after a long period of darkness [21], so probably the numbers on the data sheets are not the best way to characterise their behavior when relatively frequent and strong pulses of light hit them.

Tests with photoresistors bought from different distributors showed that they behave like a low-pass one-pole filter with constants up to 25 ms (see Figs. 8(a) and 8(b)) when reading light pulses. The measurements were taken at 5 kHz sample rate with 8-bit resolution. The conclusion was that the correlator needs to account for the "smeared" rectangular shape, and be robust in the front of sensors with different response times. The solution was to implement a rake correlator featuring three correlators with different time constants set to 5, 12 and 30 ms and to consider a hit whenever one of the three ones gives a hit. At startup, the system initialises three "prototypes", that is three wave forms representing a 20 ms rectangular pulse passed through a one-pole low-pass filter with constant α set to 5, 12 and 30 ms respectively.

The rake correlator uses fixed-point arithmetics. Using floating point is slower and requires more memory due to the floating point library. While the effort required to write a correlator in fixed-point arithmetic was not negligible, it was necessary to fit inside the small Arduino Uno memory and for more predictable run time. Figs. 9(a) and 9(b) show the output of a single correlator shifted back in time to be superimposed over the LED readings with respectively a fast and a slow photoresistor. Note that the ambient light was set to be intentionally far from ideal, as the max and min illuminations are quite far from the values of about 0 and 4 V which are visible in Figs. 8(a) and 8(b), whose measurement were taken in ideal conditions. The luminosity pollution however has little effect on the correlators, as expected. In fact, their output peaks to values greater than 0.9, while the threshold for detecting a blink is seto to 0.5, meaning that the rake correlator has a very high noise immunity, as intended.

8.2. Use case: Home monitoring of power usage and non-intrusive appliance load monitoring

Home monitoring of the overall power usage was used for a couple years in a pilot site to keep power usage under control, for curiosity, for avoiding wasting power and for being warned when the current power was exceeding the provider's limits and power was going to be cut off, thanks to a beeping sound proportional to the probability of the power going off in the next minute.



(a) Time constants 5 ms (prototype) and 2 ms (photoresistor).

(b) Time constants 12 ms (prototype) and 18 ms (photoresistor).

Fig. 9. Correlator output superimposed over photoresistor's response in realistic conditions with luminous noise and sloppy LED-photoresistor alignment for two sample photoresistors and correlator prototypes of matching time constant.

The interface in Fig. 10 looks good on a tablet in the kitchen, when most of the high-power appliances were used (dish washing machine, electrical oven, microwave oven). The gauge is updated in real time (in practice, every few seconds) and is easy to be seen from afar. Tapping on it hides the graph and leaves only a big gauge on screen for even easier reading.

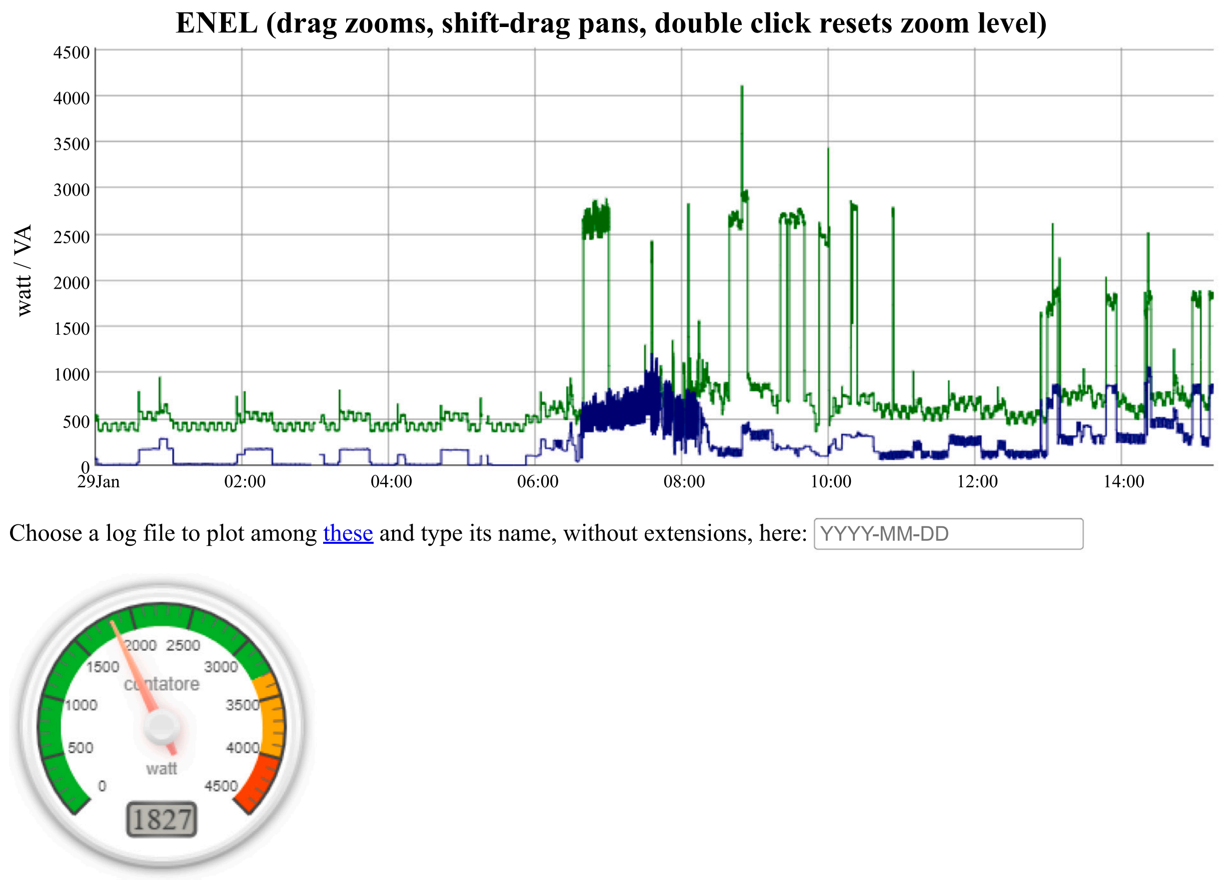
The core functionality of monitoring the power usage of building in such a fine-grained scale can be exploited in the more complex scenario of NILM. Our solution infers the domestic electric consumption from the readings of the smart meter's LED flashes and, using a Finite State Machine (FSM), it recognises one common appliances, a microwave oven, used in domestic activities. Smart meters are currently deployed on national scales, thus constituting an ideal data collection gateway for NILM solutions. The capability of recognising in real-time the usage of domestic appliances becomes relevant in the more complex scenario of activity recognition of people in smart environments. Domestic appliances, like ovens, washing machines, or hair dryers, are used in typical Instrumental Activities of Daily Living (AIDL), like feeding, doing laundry, or personal care. Monitoring these tasks is key in Active and Healthy Ageing scenarios to determine the level of independence of people, in particular older people living alone. The results obtained with the proposed system both from a qualitative and quantitative point of view are detailed in [7]. As a proof-of-concept, we tested the Enerduino-pro system to infer when the electrical microwave oven was on. The graphical interface puts an orange marker M when the microwave oven is turned on, and a green M when it is turned off. Fig. 11 shows how the electric power consumption changes when the microwave oven is turned on.

The FSM relies on the behavior of a standard microwave oven which tunes the emitted microwave power by using an on–off switch with a period of 20 s and a duty cycle variable from about 5% to 100%. The FSM signals when the oven is switched on and off by detecting a square wave with the said timing characteristics and showing amplitudes of 1500 W active power and 160 VAR reactive power. The FSM is not flexible: 1500 W, 160 VAR and 20 s are hardwired constants, so while it can detect common 900–1000 W commercial microwaves based on mechanical switches, it is not of general use. In fact, it is intended as a proof of concept showing that the detailed information provided by Enerduino-pro can in principle be used for NILM purposes, without the need for the expensive, high-resolution measurements usually associated with NILM [7].

9. Portability

Since the original Arduino WiFi shield has been discontinued, we considered the alternative boards currently available on the market for a potential porting of our solution. The Arduino Uno WiFi Rev2 and the Arduino Yún Rev2 are the platforms which look most suitable for a future hardware upgrade of our system. Both of the boards are equipped with an onboard WiFi chipset which means that no additional shields are required to connect to the network and no shield reset wiring and logic is needed either. The pinout of both boards are the same as the one used for Enerduino-pro, so the A0 and A1 inputs and the A2 output electrical connections remain unchanged. Regarding the source code, the two main aspects that should be evaluated for a porting concern the libraries related to the WiFi and web server management and the cooperative scheduler.

The Arduino Uno WiFi Rev2 (currently available for less than $ 55) is an ATmega4809 8-bit based board with a dedicated NINA-W102 WiFi module from u-Blox and the ATECC608 crypto chip accelerator enabling SSL and OAuth functionalities. The ATmega4809 has a completely new architecture with respect to the ATmega328P but thanks to the compatibility layer included in the core, it is possible to run all the sketches made for the Arduino Uno's ATmega328P microcontroller on this chip. To handle the WiFi and the web server operations, a dedicated library called WiFiNINA [22] is available. Although the library is different, the


ENEL (drag zooms, shift-drag pans, double click resets zoom level)

Choose a log file to plot among these and type its name, without extensions, here: YYYY-MM-DD

Fig. 10. Web page showing actual power consumption.

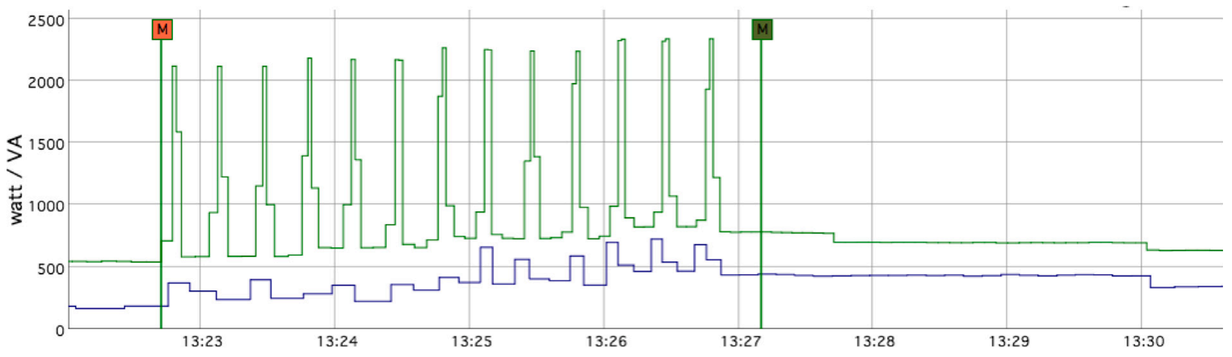


Fig. 11. The detection of the microwave oven turned on and off.

functions provided to the client sketch are the same, making code adaptation unnecessary. Concerning the cooperative multitasking library that we have used, in principle it should work on this board too since it is advertised to run on both ARM and AVR platforms. Other more recent libraries might be used as an alternative, such as TaskScheduler [23] or CoopTask [24] but this would inevitably require to rewrite the parts of the code related to the instantiation and scheduling of the tasks.

The Arduino Yún Rev2 (currently available for less than $ 60) is an ATmega32U4 8-bit based board with and additional Atheros AR9331 processor operating at 400 MHz which runs an embedded distribution of Linux called OpenWrt-Yún, based on OpenWrt. It includes a full installation of Python 2.7 and additional modules can be installed with the opkg package manager. The system configuration can be modified both through the command line or via a web page. Due to these features, this board is targeted to satisfy more advanced requirements and computationally intensive applications, thus enabling a full-fledged solution, such as the one illustrated in Section 8.2, where a web server with detailed power consumption charting and historical database is provided.

Regarding the code, the web server task would be replaced by calls to the Bridge library for the Yún devices [25], and a web server would run on the Linux OS, thus providing a self-contained solution.

10. Conclusion

Enerduino-pro is a versatile project addressed to a wide audience, including researchers, advanced hackers, novice makers and also commercial producers. Its advantages with respect to current solutions include being based on both open-source hardware and software, providing a low cost solution that is easy to assemble even for those without specialised expertise, being very responsive and accurate and optionally providing reactive power measurement. The software running on the board includes a compact and optimised web server able to expose an HTTP endpoint through which a client can read power absorption. Enerduino-pro features a rake correlator employing three correlators with time constants of 5, 12 and 30 ms to accommodate a wide range of photoresistors on the market and is able to detect the 20 ms light pulses of smart power meters even in difficult light conditions. Reading the light pulses provides active and (optionally) reactive power with the accuracy provided by the power meter every 1 Wh (or 1 VAR for reactive power). While originally designed for the Arduino Wi-Fi shield, which is now out of production, it can be ported to current hardware and still provide its unique features.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] TED - The Energy Detective, https://www.theenergydetective.com.
[2] EnergyCloud - Blue Line Innovations, https://www.bluelineinnovations.com/.
[3] Brultech - ECM-1240 Energy Monitor, https://www.brultech.com/ecm-1240/.
[4] Wattvision, https://wattvision.readme.io/docs.
[5] M. Baranski, J. Voss, Nonintrusive appliance load monitoring based on an optical sensor, in: IEEE Bologna Power Tech Conference Proceedings, 2003.
[6] Enerduino-pro, http://wnet.isti.cnr.it/software/enerduino-pro/.
[7] P. Barsocchi, E. Ferro, F. Palumbo, F. Potortı, Smart meter led probe for real-time appliance load monitoring, SENSORS (2014) 2014 IEEE.
[8] K. Gomez, R. Riggio, T. Rasheed, D. Miorandi, F. Granelli, Energino: A hardware and software solution for energy consumption monitoring, in: Modeling and Optimization in Mobile Ad Hoc and Wireless Networks (WiOpt) 2012 10th International Symposium on, IEEE, 2012.
[9] A.H. Shajahan, A. Anand, Data acquisition and control using arduino-android platform: Smart plug, in: Energy Efficient Technologies for Sustainability (ICEETS) 2013 International Conference on, IEEE, 2013.
[10] M.F.B. Anbya, M. Salehuddin, S. Hadisupadmo, E. Leksono, Wireless sensor network for single phase electricity monitoring system via Zigbee protocol, in: Control Systems & Industrial Informatics (ICCSII) 2012 IEEE Conference on, 2012.
[11] R. Fransiska, E. Septia, W. Vessabhu, W. Frans, W. Abednego, et al., Electrical power measurement using arduino uno micro controller and labview, in: Instrumentation Communications Information Technology and Biomedical Engineering (ICICI-BME) 2013 3rd International Conference on, IEEE, 2013.
[12] M. Soliman, T. Abiodun, T. Hamouda, J. Zhou, C.-H. Lung, Smart home: Integrating internet of things with web services and cloud computing, in: Cloud Computing Technology and Science (CloudCom) 2013 IEEE 5th International Conference on, 2013.
[13] thediylife - Simple Arduino Home Energy Meter, https://www.instructables.com/Simple-Arduino-Home-Energy-Meter/.
[14] Enerduino, http://enerduino.blogspot.com/2012/04/enerduino-20-english.html.
[15] 9th September 2021, Interview by Climate Action to ENEL representative published on https://www.climateaction.org/news/enel-open-meter-reality.
[16] Arduino IDE, https://www.arduino.cc/en/software.
[17] Arduino Streaming Library, https://github.com/janelia-arduino/Streaming.
[18] Arduino WiFi Library, https://github.com/arduino-libraries/WiFi.
[19] Arduino SchedulerARMAVR Library, https://code.google.com/archive/p/arduino-scoop-cooperative-scheduler-arm-avr/downloads.
[20] Fabrice Ouder - Arduino Scheduler library on AVR processors, http://forum.arduino.cc/index.php?topic=142101.0.
[21] LEDnique - Light dependent resistor (LDR), http://lednique.com/opto-isolators-2/light-dependent-resistor-ldr/.
[22] Arduino WiFININA Library, https://www.arduino.cc/reference/en/libraries/wifinina.
[23] Arduino TaskScheduler Library, https://www.arduino.cc/reference/en/libraries/taskscheduler.
[24] Arduino CoopTask Library, https://www.arduino.cc/reference/en/libraries/cooptask.
[25] Arduino Bridge library for Yún devices, https://docs.arduino.cc/retired/archived-libraries/YunBridgeLibrary.

Francesco Potortì (Member, IEEE) has been working in satellite and terrestrial communications with the Information Science and Technologies Institute, National Research Council, Pisa, Italy, since 1989, where he is a Senior Researcher. He has organised the 2011–2013 EvAAL competitions; defined the EvAAL framework; organised the IPIN competitions from 2014 to 2017; and chaired the tenth edition of the IPIN conference and the sixth edition of the IPIN Competition in 2019. He has coauthored more than 80 peer-reviewed scientific articles. His current research interests include RSS-based indoor localisation, interoperability, and evaluation of indoor localisation systems. He is a member of the IPIN Steering Board.

Davide La Rosa received the M.Sc. degree in Computer Science and Networking from the University of Pisa and the Scuola Superiore Sant'Anna, Pisa in 2012. In 2013 joined the Institute of Information Science and Technologies of the Italian National Research Council, where he currently works as a technologist. His research interests include pervasive wireless sensor networks, IoT architectures, distributed platforms for data acquisition and processing.

Filippo Palumbo received the M.Sc. (Hons.) degree in computer science engineering from the Polytechnic University of Bari, Italy, in 2010, and the Ph.D. degree in computer science from the University of Pisa, Italy, in 2016. He is with the Information Science and Technologies Institute, National Research Council. He has participated in several EU- and national-funded research actions in the areas of ambient intelligence. His research interests include the application of AI to wireless sensor networks for intelligent system design and software development in distributed systems.