eLABS PROJECT

GoNotify, a Flexible IoT Sensor Interface

Join the bubble

By **Gino De Cock** (Belgium)

IoT; the Internet of Things; IIoT, connected devices, fridges… we keep hearing a lot about it. Industry watchers predict a huge market and billions of connected devices in a few years' time, but you still have to walk over to a wall-mounted switch to turn on a light in your home. Now those dark days are over, because with GoNotify you too can connect whatever you like to the Internet.

Do you ever wonder what happens in and around your home when you are not there? Are the children inside or outside? Did they set the house on fire? Is the heating on? Did you leave a tap open or is water leaking somewhere else? Is someone ringing the doorbell? Or *vice versa*: you are at home, but is everything going fine at the factory? Maybe a machine is overheating? Is that cooler not cooling enough? Is water flowing where it should be? Would you like to follow your dog on Facebook? All of this is possible simply by putting the right sensor in the right place on the object of interest, connect it to GoNotify and you're done. It's that easy.

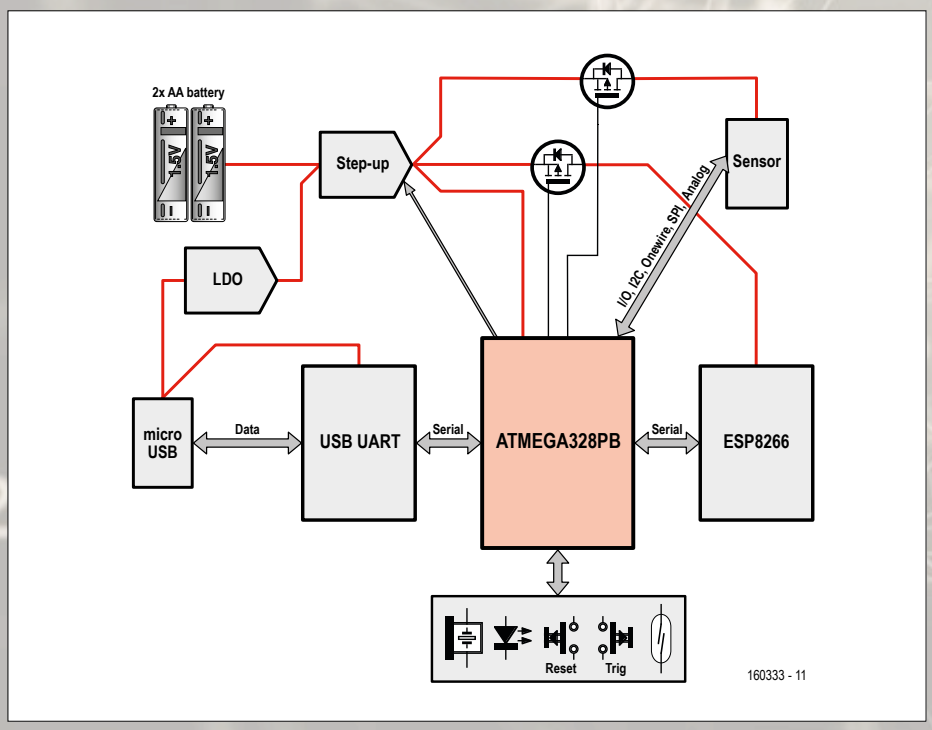


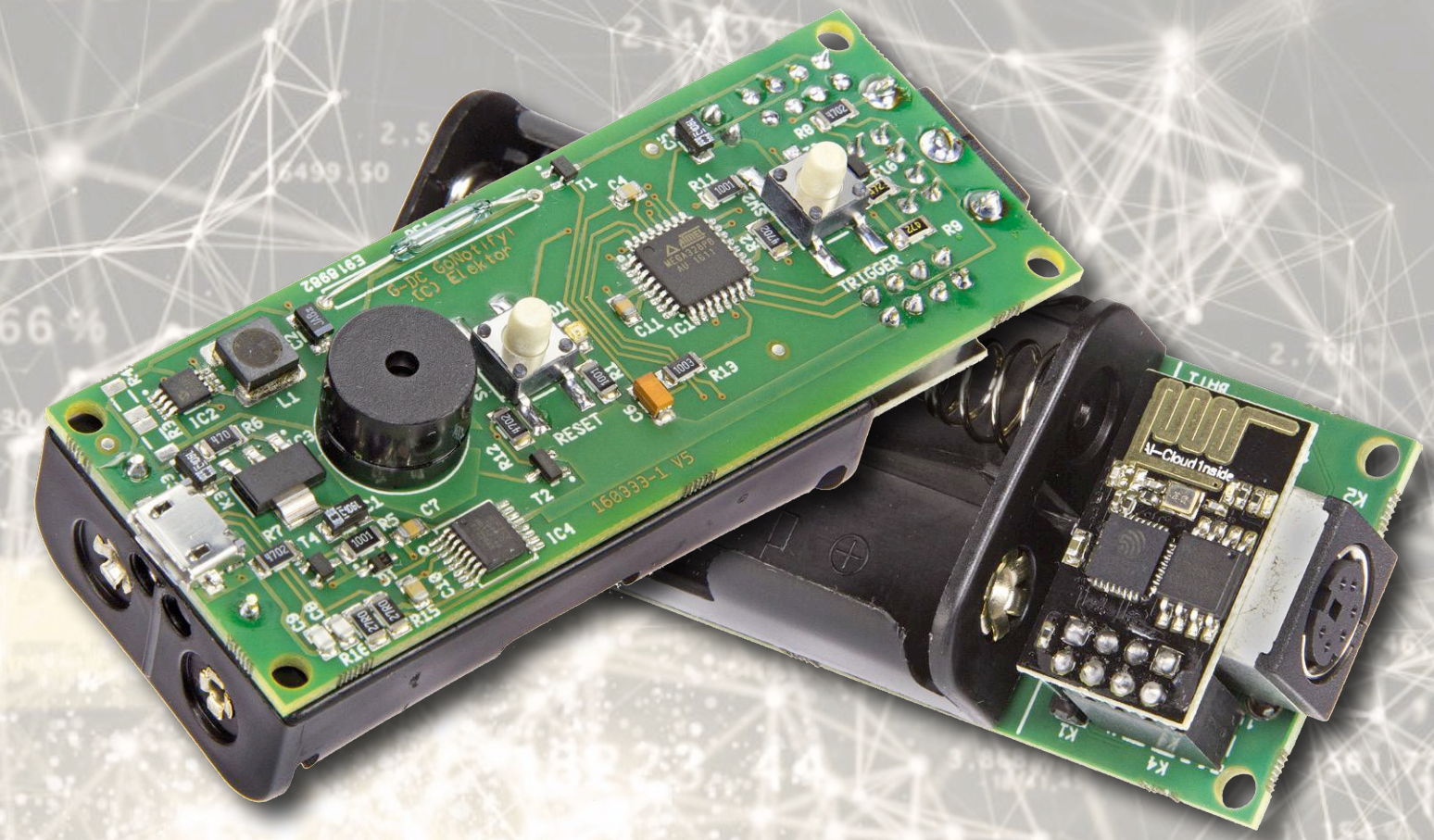
Figure 1. GoNotify's functional overview.

The beginning

Some time ago I wondered what the requirements for a connected device would be. First off, it had to be wireless, meaning it's battery- or solar-powered. This in turn implies that it should consume as little energy as possible; hence it must be a low-power design all over. Furthermore, the system should be simple and low cost, and there should be no connection costs. Finally, the device has to be flexible and easy to develop with. Wi-Fi fits almost perfectly what I was looking for and the popular Wi-Fi chip ESP8266 makes this technology accessible at low cost. The low-power side of things, on the other hand, might be a challenge. With all this in mind I set out to work.

The hardware

The design that I came up with consists of four major blocks (**Figure 1**):



- for Internet connectivity a cheap ESP-01S module containing the ESP8266 Wi-Fi microcontroller;
- an ATmega328PB to handle the sensors;
- a USB serial port for programming and debugging;
- a power module consisting of an LDO and a step-up regulator.

The schematic is shown in **Figure 2**. The microcontroller (MCU) used in this design is the ATmega328PB, an upgraded version of the popular ATmega328P with many new features. One of them is its reduced power consumption, but it also has a more accurate internal RC oscillator eliminating the need for an external quartz crystal, and — most importantly — it has two serial ports. Serial0 is used in this design for uploading firmware and for debugging it. Serial1 is reserved for communicating with the ESP-01S module. The main task of the ATmega328PB is monitoring the sensor while consuming as little power as possible.

In this design the ESP8266 is used in its module form factor, easily available on the Internet under the name ESP-01S. Plugged on K4 it is connected to serial port 1 (Serial1) of the MCU and to two of its GPIO pins. Its power supply is switched by T1 under control of the MCU. For a reliable connection to the Internet it is important to enable the 3.3 V power rail before switching on T1.

R13 and C6 have been provided to allow Over-the-Air (OTA) update of the ATmega328PB's firmware. When the ESP-01S receives the new firmware and pushes it to the MCU a reset of the latter is required to make it enter bootloader mode. While the MCU is rebooting, releasing the control of T1, the ESP-01S must remain powered on; R13 and C6 ensure this is indeed the case.

IC4, an FT230XS, is a USB-to-UART bridge that provides a convenient programming and debugging port on the MCU's serial port 0 (Serial0). The falling edge of its RTS signal, extracted by C4 and R11, is used as a reset signal for the MCU, allowing it to receive firmware from the Arduino IDE.

IC3 transforms the 5 V from the USB bus into 3.3 V. Actually, due to R6, IC3's output is slightly higher than 3.3 V in order to overcome the threshold voltage of Schottky diode D1.

The circuit is in all cases powered through IC2, independently of its state (i.e. enabled or not) because when its

PROJECT INFORMATION

ATmega328PB | ESP8266 | IoT | AllThingsTalk | IFTTT | FFA 2016

entry level
➡ intermediate level
expert level

4 hours approx.

SMD soldering, PC, Arduino IDE

£25 / €30 / $35 approx.

Features

- Universal wireless sensor interface
- Supports Wi-Fi and ESP-Now
- Ultra low-power
- Arduino compatible
- Grove compatible

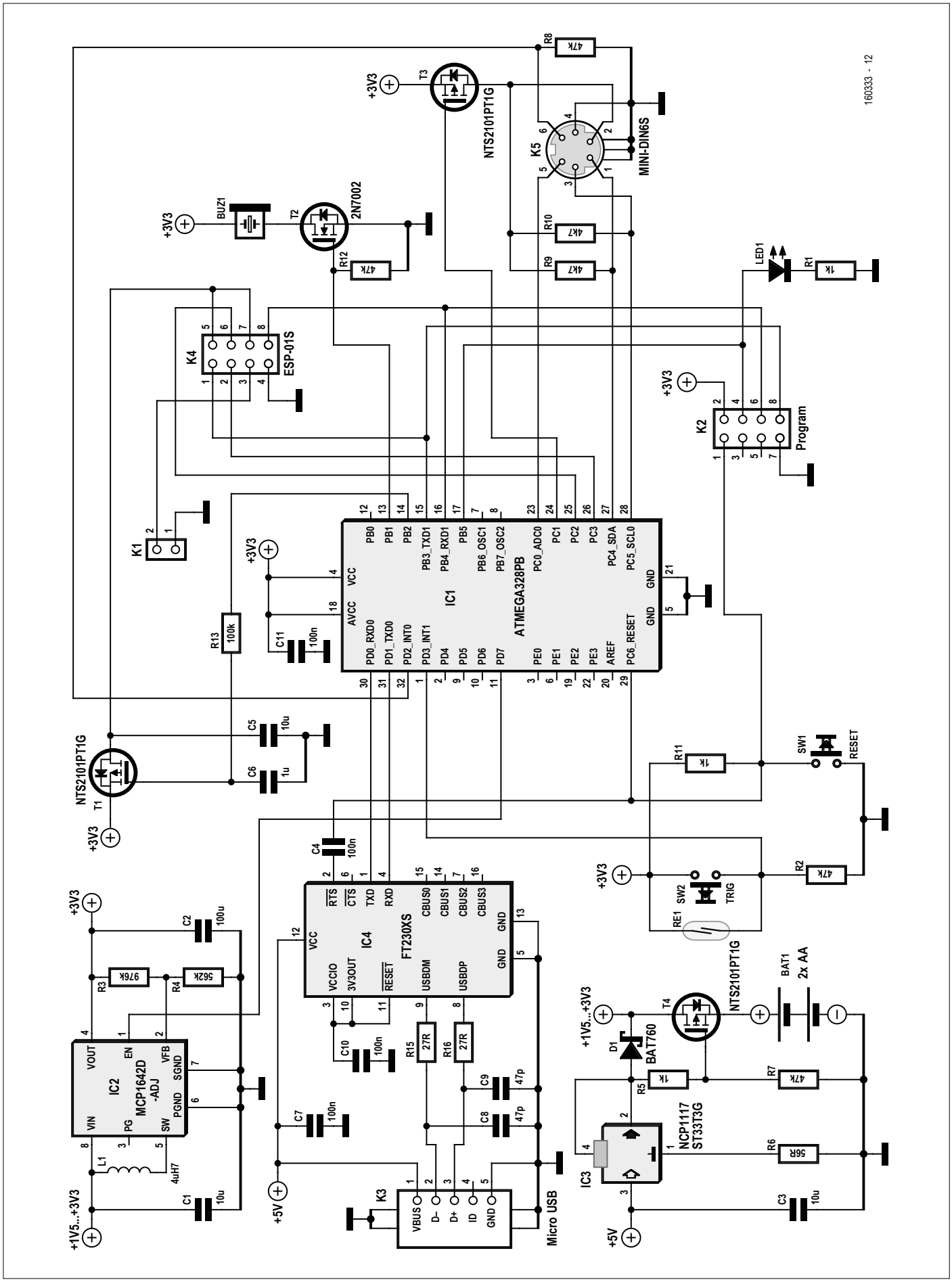


Figure 2. Full schematic of the device. The Wi-Fi module is connected to K4.

enable input is held Low, the output voltage will follow the input voltage thanks to its bypass mode. IC2 is available in several versions, here the adjustable type is used for more flexibility. An MCP1642D-33I/MS 3.3 V fixed-output model may be used also. If you do, do not mount R3 and R4. Do not use a 'B' version (MCP1642B-…) as it does not have the input bypass mode.

Sensors are connected to mini DIN connector K5. The wiring of this connector is such that you can either mount a Seeed-Studio 4-pin Grove connector or, for extra connectivity, use a regular 6-way mini-DIN connector instead.

When it is time to do a sensor reading the micro enables power to the sensor by activating T3. If the sensor requires 3.3 V then the step-up convertor IC2 must be enabled too (this depends, of course, on the sensor and has to be done by the application programmer, i.e. you). Reed switch RE1 and pushbutton SW2 ('Trigger') can be used for testing the device in the absence of an external sensor. They use external interrupt INT1 to wake up the MCU.

K2 is available for in-circuit programing of the MCU with for instance the custom bootloader. SW1 allows resetting the MCU with a button press.

K1 allows access to the second serial port of the ESP8266 microcontroller. This can be useful for debugging your Wi-Fi code without breaking communication with the MCU.

LED1 is a general purpose "Arduino Pin 13" LED. It can have any function you like.

Power management

When powered from two AA batteries it is possible to consume as little as 10 µA in "guarding mode". In this low-power mode as much circuitry as possible is disabled without stopping sensor monitoring. The ESP-01S is switched off, the step-up converter is in bypass mode and the MCU is sleeping; only the MCU's watchdog timer is running to periodically wake up the MCU to do a sensor reading. An external interrupt can also be used for this.

In guarding mode the ESP-01S module is powered off via T1 and the step-up regulator IC2 is disabled by pulling its enable pin low. In this situation IC2 is actually in bypass mode, meaning that its output is connected to its input; its internal circuitry is switched off making it consume hardly anything. Now only the ATmega328PB (IC1) is powered as it is connected to either the batteries (through T4) or to the low dropout device (LDO) IC3 (through D1) if 5 V is present at pin 1 of the micro USB connector K3. With reduced RF transmit power and Internet connection periods kept as short as possible GoNotify consumes about 1.5 mAh on average per message even when a REST protocol over a secure HTTPS connection is used. Battery life can be extended even further by utilizing the ESP-01S modules with the (proprietary) ESP-Now technology. In this case one device acts as a master/bridge while the other acts as a remote sensor slave. Communication between the master and slave is done without any overhead from the TCP/IP stack (more on this later).

When powered from two 2500 mAh batteries, 1.5 mAh per message means that 3,000 to 4,000 messages can be sent with a single set of fresh batteries. This corresponds to about five months' usage at a rate of one message per hour. Programing some intelligence into the MCU to reduce the number of Internet connections is therefore useful. Furthermore, when a condition is detected that requires an alarm, GoNotify can still inform the user via buzzer BUZ1 (switched by T2) without requiring an Internet connection. Using the buzzer should be done as a last resort.

If an application requires an 'Always On' Internet connection, for example when it is an MQTT client, it is best to power the device via the micro USB connector (K3). In this case the batteries will act as a back-up power supply when the USB power is disconnected.

To maximize the number of messages on one battery charge the following fine-tune options are available (besides writing cleverer software):

- Reduce transmit power: by default the ESP-01S operates on a rather long range and draws a lot of power when transmitting. Depending on the situation the range may be reduced, saving power in the process. This can be done through software, but also with the ESP8266 Download Tool (V3.4.4), on the 'RFConfig' tab (the tool is available from the Espressif website).
- Fix the Wi-Fi channel: scanning Wi-Fi channels consumes power. Scanning can be avoided by telling the ESP-01S module to use the same Wi-Fi channel as the access point.
- Avoid DHCP: DHCP negotiations at startup can be avoided by fixing the device's IP address (at the expense of reduced flexibility, of course).
- Avoid DNS lookup: using the IP address of the destination (Cloud or other) avoids DNS lookup time, and saves power.
- Optimize the ESP-01S RF initialization process: normally the ESP-01S executes an RF alignment when it starts up, which consumes quite some current. This can be done with the API function `system_phy_set_powerup_option(2)`. The ESP8266 Download Tool also can change this behavior, but the best option (no. 2) is not available.
- Reduce the MCU's clock frequency by setting the DIV2 fuse. Now the MCU will run at 4 MHz instead of 8 MHz, allowing operation until the battery voltage drops below 1.8 V.

Enclosure

Besides a circuit, a PCB and a lot of software I also created a 3D-printable enclosure for GoNotify (**Figure 3**). You can print it at home or online (find a service near you with www.3dhubs.com); it will only cost you a few euros. The design files for the enclosure, as all other GoNotify files, BTW, can be downloaded from [2].

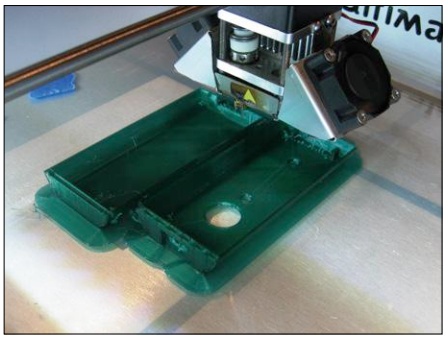
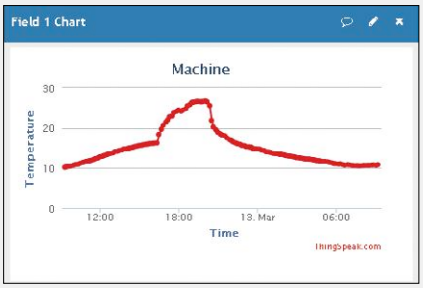


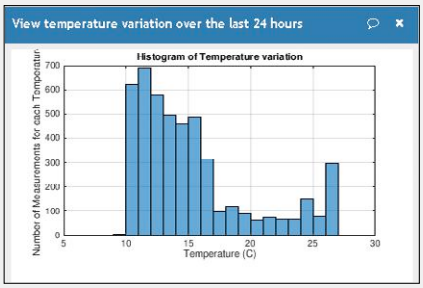
Figure 3. A design for a 3D-printed enclosure is available too.

Updating the firmware

The Arduino IDE can be used to update the firmware of the MCU over USB thanks to a compatible bootloader. Furthermore, K2 is available for in-circuit programming with an AVR programmer. However, to program the ESP-01S module over the USB connection a more elabo-

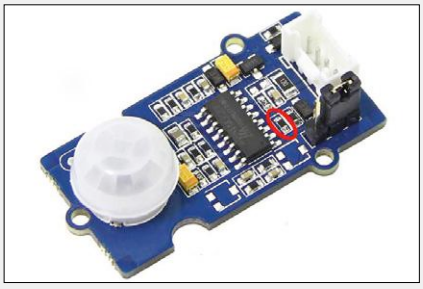
Online statistical analysis with ThingSpeak





Monitor the temperature of some machine with a suitable sensor (a thermocouple for instance). Connect GoNotify to the online service ThingSpeak and analyze the captured data with Matlab.
Source: https://goo.gl/tthgeJ
ThingSpeak: https://thingspeak.com

Movement detection with IFTTT

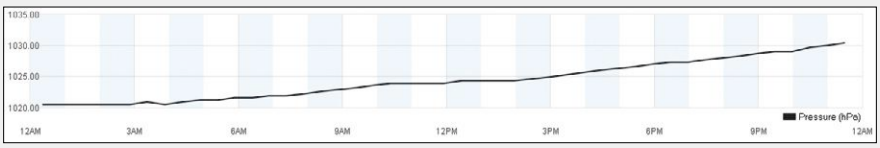


Short the encircled resistor.

Monitor movement and get alerted via the popular online 'If This Then That' (IFTTT) service. A Grove PIR motion sensor from SeeedStudio can be connected to GoNotify. To make this work the sensor's 10-kΩ resistor (see photo) in the output signal must be bypassed because it will interfere with GoNotify's 4.7-kΩ I2C pull-up resistors. Connect your IFTTT applet to a Google spreadsheet and start logging movements.
Source: https://goo.gl/shSWe6
IFTTT: https://maker.ifttt.com

Join the Weather Underground Network

Use GoNotify to connect a BMP180 sensor to the Weather Underground network and make quality weather information available to every person on this planet.
Source: https://goo.gl/A3QByX
Weather Underground: www.wunderground.com



rated mechanism is needed. To achieve this, the MCU must put the ESP-01S in bootloader mode first and then create a bridge between its two serial ports to pass the new firmware to the ESP-01S. This has been accomplished by customizing the MCU's bootloader.
The programing data rate for the ESP-01S is currently limited to 57600 bits/s until a better bridge algorithm for the MCU is found. Faster programming is possible if you include Over-the-Air (OTA) code into your sensor application. The ESP-01S module has 8 Mbit of Flash memory, the minimum needed for OTA updates.
Network R13-C6 is intended for OTA firmware updates of the ATmega328PB. When the ESP-01S receives the new firmware and pushes it to the ATmega328PB a reset of the latter is required to make it enter bootloader mode. During this process the ESP-01S must keep on running, which is accomplished by C6 together with R13.

Application development

For practical applications it is best to divide the tasks over the two microcontrollers. The ATmega328PB monitors the sensor and, if needed, triggers an alarm, and activates the ESP-01S module. The latter then takes care of connecting securely to the Internet and handles the communication using the protocol of your choice (e.g. HTML, REST, MQTT). The ESP-01S can also take care of the user interface (UI) for configuring or monitoring via a standard Internet browser on your computer or smartphone.
For comfortable application development it is important that programing the device is as easy as possible. Furthermore, debugging of the sensor application must be possible too. For GoNotify the Arduino IDE was elected to do all this. To set up your Arduino IDE the URLs of two board definition files need to be added to the 'Additional Board Manager URLs' field of the 'Preferences' dialog window (under the 'File' menu). You can add multiple URLs, separating them with commas (there are no spaces in the following URLs):

- For the ATmega328PB: https://raw.githubusercontent.com/ginodecock/V3GoNotify/master/ArduinoBoard/package_gonotify_v3_index.json
- For the ESP-01S: http://arduino.esp8266.com/versions/2.3.0/package_esp8266com_index.json

About the Author

Gino De Cock (1977) has been fascinated with electronics from his youth. He studied electronics in Ghent, Belgium where he graduated in 1999. Tormented by the existential question "*Isn't there a better way?*" Gino keeps pursuing his ideas with projects like GoNotify! as a result.

Fast Forward Award 2016

GoNotify was one of the entries for the Fast Forward Award organized by Elektor in collaboration with the governors of the 2016 electronica tradeshow in Munich. It was a great opportunity for hobbyists and professionals alike to share and present projects, products and startups.

Programing is quite easy when using the Arduino IDE:

- Connect GoNotify to your PC. The first time you do this a virtual COM port will be installed.
- Install the ATmega328PB and ESP-01S boards packages in the Arduino IDE:
  - Open the Boards Manager (from the 'Tools' ➞ 'Board' menu), select 'Contributed' and install 'GoNotify-V3 ATMEGA'.
  - Do the same for the ESP-01S by installing the 'esp8266 by ESP8266 Community'.
- From the 'Tools' ➞ 'Port' menu select the virtual COM port that corresponds to your device.
- To program:
  - ATmega328PB: select the board 'GoNotify @ 4 MHz (internal RC)'. Clicking the 'Upload' button is enough to start programming.
  - ESP-01S: select the board 'Generic ESP8266 Module'. Press and hold trigger pushbutton SW2 before clicking the IDE's 'Upload' button. When programming has started the pushbutton can be released.
- Open the 'Serial Monitor' to debug.

When new firmware has been programed into the device, and all memory has been erased it must be (re)configured before it can be used (configuration data is stored in the MCU's EEPROM, Wi-Fi configuration data is stored in the ESP-01S module). Start by pressing Reset button SW1. GoNotify will now act as an access point (AP) by default at 192.168.4.1 (but the user can change this in the software), see **Figure 4**. After configuration GoNotify expects a trigger, either by pressing SW2, closing RE1 or by making the sensor pass a certain threshold value (to be defined in the software first). The buzzer produces a notification beep and a test message is sent; its contents and destination depend on how you configured the device. The example in Figure 4 sends a message to a PushBullet service (www.pushbullet.com). From this point on GoNotify will enter normal operation in its low-power guard mode.

Virtual machine

Now that you know how GoNotify works, how to build one and how to program it, it is time to look at some practical applications. Note that the shortened URLs all lead to my GitHub repository at [2]. Useful to know also is that I have prepared a virtual machine (VM) with all the tools preinstalled so you can get started quickly. It is available in the 'GoNotify-Development-Env' folder of [2].

Californian water meter with alert in the Cloud

In this example GoNotify monitors the water meter via a reed switch mounted on it producing a pulse for every 500 ml of water used. When a leak is detected or a tap runs too long GoNotify sends a notification. It also reports the water consumption every hour to help to discover when the most water is used and why. To save power, only when there is a problem the ESP-01S module is powered up to send an alert message. In this case an alert is sent when, measured over a period of 24 hours, there are no 2-hour periods where the total water consumption is less than 0.5 l. In other words, when water consumption is continuous GoNotify considers that there is a leak. Also, when water consumption remains high for 30 minutes or more, GoNotify will assume that a tap has been left open and it will send an alert too.

In case of a problem with the Internet connection, GoNotify will alert the user by activating its buzzer.

For this application the MCU's integrated temperature sensor must be calibrated. Press and hold the trigger button SW2 and reset the device. While keeping SW2 pressed, cool de processor to 0° Celsius. GoNotify will beep when the temperature changes, when it stops beeping it has reached the reference temperature. You can now release SW2.

- URL: https://wma-gonotify.rhcloud.com
- User: demo
- Password: demo
- Source code: https://goo.gl/UIkaXJ

In this sample the ATmega328PB is programed from within the Arduino IDE, the ESP-01S module is programed using the native IoT SDK2.0 (available from the Espressif website). The Cloud solution is created using Redhat's OpenShift (https://www.openshift.com/) and is programed with Nodejs and Mongodb.

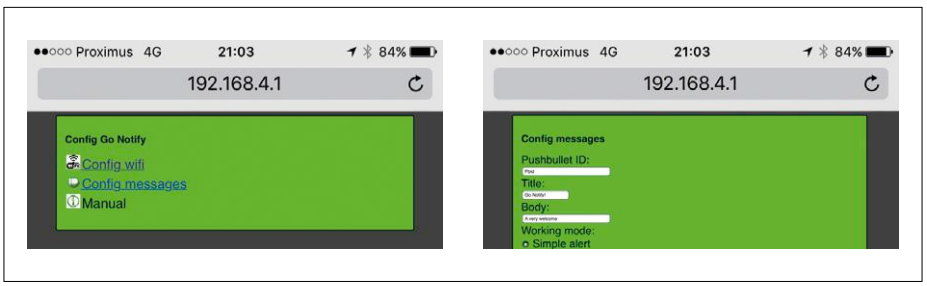


Figure 4. Connect to the GoNotify access point to configure the device for your Wi-Fi network.

There are several advantages to using a private Cloud approach:

- You own the data: data is valuable and private.
- Notifications: GoNotify has the intelligence to detect that a problem has occurred and to create an alert for it. These alerts are reported to the Cloud where they are relayed to a notification system. Also, when an expected reading does not arrive (in time), the Cloud will flag the absence of the sensor.
- Time synchronized readings: the Cloud can act as a clock for the sensor. With each reported reading the Cloud replies in the REST header when the next reading is expected.

MQTT client with AllThingsTalk

Being compatible with Arduino and using popular microcontrollers like the ATmega328PB and the ESP8266 has the advantage that many IoT Cloud platforms have an API that is compatible too. This application shows how to interface GoNotify to the AllThingsTalk Maker API (https://maker.allthingstalk.com) as an MQTT client. This is accomplished with a few lines of code. The example illustrates two-way communication between the AllThingsTalk server and a GoNotify client. It allows you to control GoNotify remotely and receive status information in return.

In this setup GoNotify is always on and connected, meaning that it must be powered over USB. The batteries act as a back-up power supply in case USB power is lost for some reason.

In this sample the ATmega328PB is programed from within the Arduino IDE, the ESP-01S module is programed using the native IoT SDK2.0.

URL: https://maker.allthingstalk.com/device/Kx7vo0kSYI5P6e9PrJIOKWS7.

In order to see this page you must first create your own AllThingsTalk account and connect to it.

Source code: https://goo.gl/OIF6Rw

Monitoring a greenhouse with ESP-Now

ESP-Now is a proprietary communication protocol developed by Espressif, which enables devices to talk to each other, peer-to-peer (P2P), without using Wi-Fi

COMPONENT LIST

Resistors
Default: 1206
R1,R5,R11 = 1kΩ
R2,R7,R8,R12 = 47kΩ
R3 = 976kΩ*
R4 = 562kΩ*
R6 = 56Ω
R9,R10 = 4.7kΩ
R13 = 100kΩ
R15,R16 = 27Ω

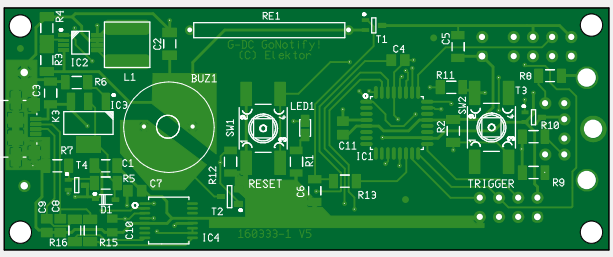
Capacitors
C1,C3,C5 = 10µF 16V, tantalum, 1206
C2 = 100µF 6.3V, tantalum, 1206
C4,C7,C10,C11 = 100nF, 0805
C6 = 1µF, 16V, tantalum, 1206
C8,C9 = 47pF, 0805

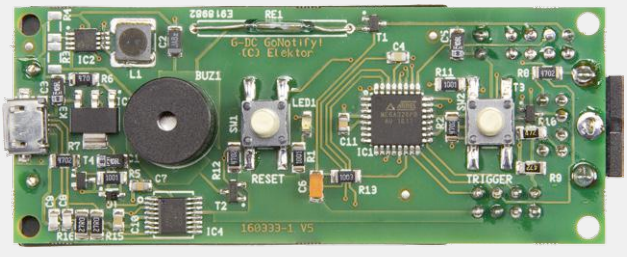
Inductors
L1 = 4.7µH, I_{rms}=1.72A, 0.082Ω

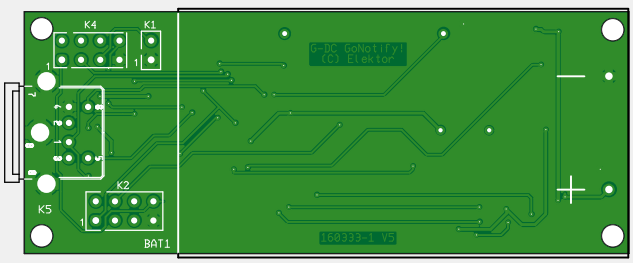
Semiconductors
D1 = BAT760
LED1 = green, 2012
T1,T3,T4 = NTS2101PT1G
T2 = 2N7002
IC1 = ATmega328PB-AU
IC2 = MCP1642D-ADJ*
IC3 = NCP1117ST33T3G
IC4 = FT230XS

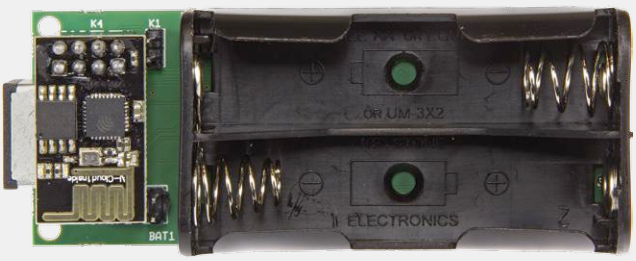
Miscellaneous
BUZ1 = buzzer, 12mm diameter, 6.5mm pitch
K1 = 2-pin pinheader, 0.1" pitch
K2 = 8-pin (2x4) pinheader, 0.1" pitch
K3 = Micro USB Type AB connector
K4 = 8-way (2x4) pinheader socket, 0.1" pitch
K5 = 6-way miniature DIN socket or 4-way Grove connector*
RE1 = reed switch, glass length 15mm max.
SW1,SW2 = tactile switch, SMT, 6.2x6.2mm
Battery holder, 2x AA (Keystone 2462)
ESP-01S ESP8366 Wi-Fi module
PCB # 160333-1

* see text









or handshaking. It is intended for remote sensors that connect to a bridge (another ESP8266/ESP-01S) providing an Internet connection (see **Figure 5**).
In this application one GoNotify device acts as a bridge to the Internet and as a secure MQTT client to the AllThingsTalk server, the other will read the sensor. The bridge device is an ESP-Now slave and is always on. To make sure the MQTT client remains active (persists) ping packets are sent periodically to the AllThingsTalk server to check connectivity.
The sensor GoNotify plays the role of ESP-Now controller. When it wants to communicate something, an event for example or an alert, it wakes up and connects to the MAC address of the bridge to send and verify a few bytes.
In this example ESP-Now is used to monitor a greenhouse. The remote sensor GoNotify is equipped with a DHT12 sensor to measure relative humidity of the air and the temperature. Another sensor checks the humidity of the soil to find out if the plants are in need of water (**Figure 6**). Powered from Lithium batteries this setup can send 400,000 sensor readings, with regular cheap alkaline batteries approximately 200,000 sensor readings will be possible.
In this application the bridge and remote sensor have both been programed as Arduino sketches.

- Source code for humidity sensor: https://goo.gl/9mqLos
- Source code for ESP-Now: https://goo.gl/18Qhd5

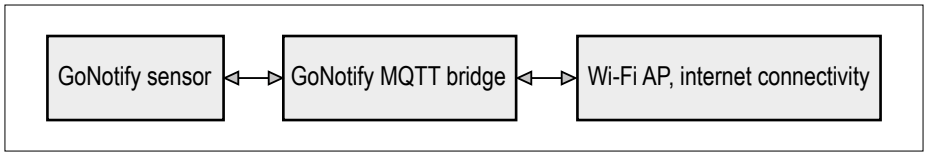


Figure 5. The ESP-Now communication protocol allows peer-to-peer connections between ESP8266 devices.

Figure 6. GoNotify busy monitoring air humidity and temperature and soil humidity in a greenhouse.

From idea to IoT

The GoNotify device presented in this article can be used as a connected device to capture data from a sensor and monitor it online or it can be used as a simple remote-controlled system. It is your application that can transform it into a smart device capable of producing alerts and taking actions.
I really hope that GoNotify will get used as a true Internet product and becomes part of an ecosystem where the recorded data can be processed to extract useful information. This is where the additional value is of any IoT idea.
Finally, a word of thanks to all of you open source enthusiasts. Many open source projects are available for both the ESP8266 and ATmega328PB microcontrollers without which doing this project would have been much more difficult. Reusing these projects in your own applications will speed up prototyping tremendously. Therefore, hats off to the maker community!

(160333)

Web Links

[1] www.elektormagazine.com/160333
[2] https://github.com/ginodecock/V3GoNotify/
[3] www.elektormagazine.com/labs/flexible-iot-sensor-interface-gonotify-1