enOcean

enOcean is a pretty cool energy harvesting technology focused on smart home applications. enOcean is best known for the wireless light switches that don't require a battery to operate. additionally, they come in looks compatible to Feller edizio due and hager callysto, which are used in almost every building in switzerland.

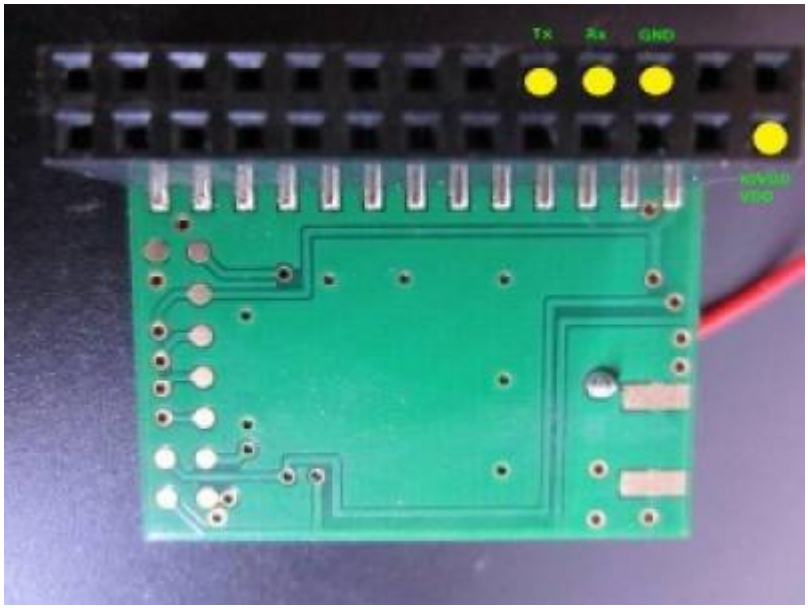
the light switch has a piezo module built in which uses the energy of the button-push to create enough electricity to send a datagram to the enOcean receiver to tell it that the switch has been pushed.

another nice thing about enOcean is the fact, that they sell a pretty affordable (15$ range) module called TCM310 which can be used to both send or receive enOcean data packages over the air. The protocol is also openly documented.

my goal is to use a TCM310 in the form of an enOcean Pi module together with ideally a sonoff or some other ESP8266 based board to receive push button presses and send them to an MQTT broaker. As the “enOcean PI” product name says, the module is acutally made for a raspberry pi and I am sure, that my goal could be achieved simpler by using a raspberry Pi. But i want this to be a set-and-forget system, and a full blown linux server to me is just not such a thing. however, we will see how it works out.. maybe i'll revert to the easy path in case it gets too complicated to do it all on an ESP8266 :)

enOcean PI pinout

there are only four pins which are used on the enOcean PI module:



thanks to Kerry D. Wong who published this Information. He writes:

> The IOVdd and Vdd pins are connected directly to pin 1 (3.3V) on Raspberry Pi. Besides the power and ground pins, the only connections are to the hardware UART pins (header pin 8 and pin 10).

esp8266

I plan on using a Wemos D1 mini module (or a clone of it i guess) which can be purchased on aliexpress for a few $. i will then use the arduino core on it to program it via the arduino IDE.

arduino-esp8266 documentation

based on this documentation, the serial port will be available on GPIO1 and GPIO3.

here are a few libraries i have found beforehand and intend on trying out:

- Async MQTT client for mqtt obviously
- ESPAsyncTCP to plot debug messages to a netcat server while working with the enOcean module on the serial port.. this is to avoid using SoftSerial.
- WiFiManager seems to do all we need to make the final device easy to get configured without a serial connection (has automatic fall back to AP mode with captive portal config page).
- arduino enocean library with a description from the developer

setup and development

first setup_arduino_ide_for_esp8266 and install the WiFiManager via the `Tools→Include Library→Manage Libraries`. Search for and install `WiFiManager`. This also installs all the dependencies which is nice. However i had to manually download and install the latest development release from the git, in order to have the configuration page available even when the device was connected to the wifi.. this should become part of the stable release any time soon, so this step might not be necessary anymore when you read this:

1. navigate to `<arduino directory>/portable/sketchbook/libraries/`
2. ```
   rm -rf WiFiManager
   git clone https://github.com/tzapu/WiFiManager.git
   cd WiFiManager/
   git branch -a
   git checkout remotes/origin/development
   ```

get wifimanager to work

that's actually fairly simple.. just include it and put `autoConnect()` in your startup. the tricky part was to get the config webpage to run using `startWebPortal()`. the trick i've missed was, that the `process()` method needs to be called in the `loop()` or else it won't display any webpages. so here is the working sketch:

```
#include <ESP8266WiFi.h>          //ESP8266 Core WiFi Library (you most
likely already have this in your sketch)
#include <DNSServer.h>            //Local DNS Server used for redirecting
all requests to the configuration portal
#include <ESP8266WebServer.h>     //Local WebServer used to serve the
```
```

```
configuration portal
#include <WiFiManager.h>          //https://github.com/tzapu/WiFiManager
WiFi Configuration Magic

WiFiManager wifiManager;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  wifiManager.autoConnect();
  wifiManager.startWebPortal();
}

void loop() {
  // put your main code here, to run repeatedly:
  wifiManager.process();
}
```

add the async printer

we use ESPAsyncTCP's AsyncPrinter to print debug messages to a tcp socket at our development machine as we will use the serial port to connect to our enOcean board.

first we need to install the library:

```
cd <arduino directory>/portable/sketchbook/libraries/
git clone https://github.com/me-no-dev/ESPAsyncTCP.git
```

for my development machine to receive those messages i need to open a socket with `netcat` in listening mode. this is for linux but i am sure there are tools like that in windows too, i just don't know them. so on your develpment machine run:

```
nc -l 3333
```

and leave the terminal open.

now we could integrate it as is into our code as you can see in the example I posted on Github, but the problem with AsyncPrinter is, that it blocks forever waiting for a connection. This means, that our sketch won't make it past the setup process if our debug host is not online. that's not what we want. I therefore modified the AsyncPrinter source slightly by removing these two lines from both `AsnycPrinter::connect()` functions:

```
    while(_client->state() < 4)
      delay(1);
```

let's add `AsyncPrinter` and a `debug()` function which handles sending debug messages if we have a connection to a debug host:

```
#include <ESP8266WiFi.h>          //ESP8266 Core WiFi Library (you most
```

```
likely already have this in your sketch)
#include <DNSServer.h>            //Local DNS Server used for redirecting
all requests to the configuration portal
#include <ESP8266WebServer.h>     //Local WebServer used to serve the
configuration portal
#include <WiFiManager.h>          //https://github.com/tzapu/WiFiManager
WiFi Configuration Magic
#include <AsyncPrinter.h>

bool enableOnlineDebugging=true;
const char *debuggerHost="192.168.168.48";

AsyncPrinter ap;
WiFiManager wifiManager;
bool debugOnline=false;
bool firstround=true;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  wifiManager.autoConnect();
  wifiManager.startWebPortal();
  ap.connect(debuggerHost,3333);
  if(enableOnlineDebugging){
    //block for maximum 5 seconds trying to reach the debugger
    for ( int i = 0; i < 10 ; i++){
      if(ap.connected()){
        debugOnline=true;
        ap.println("setup done");
        break;
      } else {
        delay(500);
      }
    }
  }
}

void debug(const char *message){
  if(debugOnline){
    ap.println(message);
  } else {
    Serial.println("debugger is not online");
  }
}
void loop() {
  // put your main code here, to run repeatedly:
  wifiManager.process();
  if(firstround){
    debug("hello world");
    firstround=false;
  }
```

```
}
```

you should now see “setup done” and “hello world” on your terminal with `nc` running.

add enOcean

now we can finally add the enocean receiver to our setup. we therefore connect the +3.3V to the +3.3V of the Wemos D1, the GND to the GND and we cross connect the RX of the enocean Module to the TX of the Wemos D1 and the same for the TX fo the enOcean that goes to the RX of the Wemos

next we need the enOceanMsg library.. however, i have modified it slightly (removed all serial debugging and changed the enOcean communication part from `Serial1` to `Serial` as the ESP8266's `Serial1` only supports transmitting data but not receiving. So either you do these modifications yourself or you get [[

my Version

of it.

extract the contents of this ZIP to the `sketchbook/libraries` folder. I then also modified the Sketch to both include the enOcean stuff and to disable serial debugging of the WiFiManager when online-debugging is requested.

```
#include <ESP8266WiFi.h>          //ESP8266 Core WiFi Library (you most likely already have this in your sketch)
#include <DNSServer.h>            //Local DNS Server used for redirecting all requests to the configuration portal
#include <ESP8266WebServer.h>     //Local WebServer used to serve the configuration portal
#include <WiFiManager.h>          //https://github.com/tzapu/WiFiManager WiFi Configuration Magic
#include <AsyncPrinter.h>
#include <EnOceanMsg.h>           //http://djynet.net/?p=635

bool enableOnlineDebugging=true;
const char *debuggerHost="192.168.168.48";

EnOceanMsg _Msg;
AsyncPrinter ap;
WiFiManager wifiManager;
bool debugOnline=false;
bool firstround=true;
int lastPayload=0;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(57600);
  if(enableOnlineDebugging){
    wifiManager.setDebugOutput(false);
  }
  wifiManager.autoConnect();
  wifiManager.startWebPortal();
```

```
  ap.connect(debuggerHost,3333);
  if(enableOnlineDebugging){
    //block for maximum 5 seconds trying to reach the debugger
    for ( int i = 0; i < 10 ; i++){
      if(ap.connected()){
        debugOnline=true;
        ap.println("connected");
        break;
      } else {
        delay(500);
      }
    }
  }
}

void debug(const char *message){
  if(debugOnline){
    ap.println(message);
  } else {
    Serial.println("debugger is not online");
  }
  _Msg.decode();
  if (_Msg.dataAvailable() == true){

  }
}

void debugHex(int payload){
  if(debugOnline){
    ap.println(payload,HEX);
  }
}


void loop() {
  // put your main code here, to run repeatedly:
  int payload=0;
  wifiManager.process();
  if(firstround){
    debug("hello world");
    firstround=false;
  }
  _Msg.decode();
  if(_Msg.dataAvailable() == true){
    payload=_Msg.getPayload();
    if(payload!=lastPayload){
      debug("new payload received:");
      debugHex(payload);
      lastPayload=payload;
      debugHex(_Msg.getSenderId());
    }
```

```
  }
  delay(50);
}
```

now you should see output like this on your netcat console after you started the sketch and then pressed a button:

```
connected
hello world
new payload received:
70
3102F7
new payload received:
0
3102F7
```

`70` is the hex code for one of the buttons and `3102f7` is actually `0x00 0x31 0x02 0xf7` which is the enOcean device's address which can also be found on the lable on the device itself.