plant_care Documentation

Release 1.0

plant_care

Jan 30, 2018

Contents

CHAPTER 1

Plant Care

- Docs: http://plant-care.rtfd.io/
- Git: https://gitlab.com/ThomasDevoogdt/plant_care.git

Plant Monitoring System

1.1 1. Plant Care ESP8266

1.1.1 1.1. Powering:

- Solar power
- Small space

Battery charging: Power Cell - LiPo Charger/Booster

- MCP73831 Single Cell LiPo charger met 500mA
- TPS61200 Boost Converter
- Selecteerbaar output voltage 3.3 or 5V
- 5V @ 600mA max
- 3.3V @ 200mA max
- JST connector voor LiPo batterij
- micro-USB connector voor charge power
- Temp. beveiliging

Power Supply: 4 x 5V - 30mA Solar Panel 53x30

- 53x30 mm
- Typical voltage: 5V
- Typical current: 30mA

Battery: LiPo accupack 3.7 V 600 mAh

Belgium Solar Hours in Januari = 1.9h/day → Energy = 0.95Wh/day → 12mA continuous current (3.3V)

1.1.2 1.2. Microcontroller:

- Low Power Consumption
- WiFi and TCP/IP stack available

Model

Wemos D1 mini **/ ESP 12e (ESP8266)**

The ESP8266 is a low-cost Wi-Fi chip with full TCP/IP stack and MCU (microcontroller unit) capability produced by Shanghai-based Chinese manufacturer, Espressif Systems. [ESP8266 Overview]

- 802.11 b / g / n
- Wi-Fi Direct (P2P), soft-AP
- Built-in TCP / IP protocol stack
- Built-in TR switch, balun, LNA, power amplifier and matching network
- Built-in PLL, voltage regulator and power management components
- 802.11b mode + 19.5dBm output power
- Built-in temperature sensor
- Support antenna diversity
- off leakage current is less than 10uA
- Built-in low-power 32-bit CPU: can double as an application processor
- SDIO 2.0, SPI, UART
- STBC, 1×1 MIMO, 2×1 MIMO
- A-MPDU, A-MSDU aggregation and the 0.4 Within wake
- 2ms, connect and transfer data packets
- standby power consumption of less than 1.0mW (DTIM3)

Power Consumption

Parameters	Min	Typical	Max	Unit
Tx 802.11b, CCK 11Mbps, P OUT=+17dBm		170		mA
Tx 802.11g, OFDM 54Mbps, P OUT =+15dBm		140		mA
Tx 802.11n, MCS7, P OUT =+13dBm		120		mA
Rx 802.11b, 1024 bytes packet length, -80dBm		50		mA
Rx 802.11g, 1024 bytes packet length, -70dBm		56		mA
Rx 802.11n, 1024 bytes packet length, -65dBm		56		mA
Modem-Sleep		15		mA
Light-Sleep		0,9		mA
Deep-Sleep		10		uA

Modem-Sleep requires the CPU to be working, as in PWM or I2S applications. According to 802.11 standards (like U-APSD), it saves power to shut down the Wi-Fi Modem circuit while maintaining a Wi-Fi connection with no data transmission. E.g. in DTIM3, to maintain a sleep 300mswake 3ms cycle to receive AP's Beacon packages, the current is about 15mA.

During Light-Sleep, the CPU may be suspended in applications like Wi-Fi switch. Without data transmission, the Wi-Fi Modem circuit can be turned off and CPU suspended to save power according to the 802.11 standard (U-APSD). E.g. in DTIM3, to maintain a sleep 300ms-wake 3ms cycle to receive AP's Beacon packages, the current is about 0.9mA.

Deep-Sleep does not require Wi-Fi connection to be maintained. For application with long time lags between data transmission, e.g. a temperature sensor that checks the temperature every 100s, sleep 300s and waking up to connect to the AP (taking about 0.3~1s), the overall average current is less than 1mA.

1.1.3 1.3. Communication:

I²C: Internal Sensor - ESP

It is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication. (I²C)

IEEE 802.11 b/g/n Wi-Fi - MQTT: Data publishing

It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. (MQTT)

1.1.4 Sensors:

I²C

- Lichtsensor
- Barometric Pressure Sensor

Analog

- Bodemvochtigheidssensor

- [STRIKEOUT:Capacitieve Bodem Sensor]

Costum

- Temp. en vochtigheid - DHT22

1.1.5 1.4. Data analysis / Algorithm

1.2 2. Plant Care Login Portal

Initially the system needs to be connected to the internet. Therefore the system creates a WiFi portal where you can connect on. Then you need to configure your local WiFi connection. If the config portal is not displayed, go to http://10.0.1.1/.

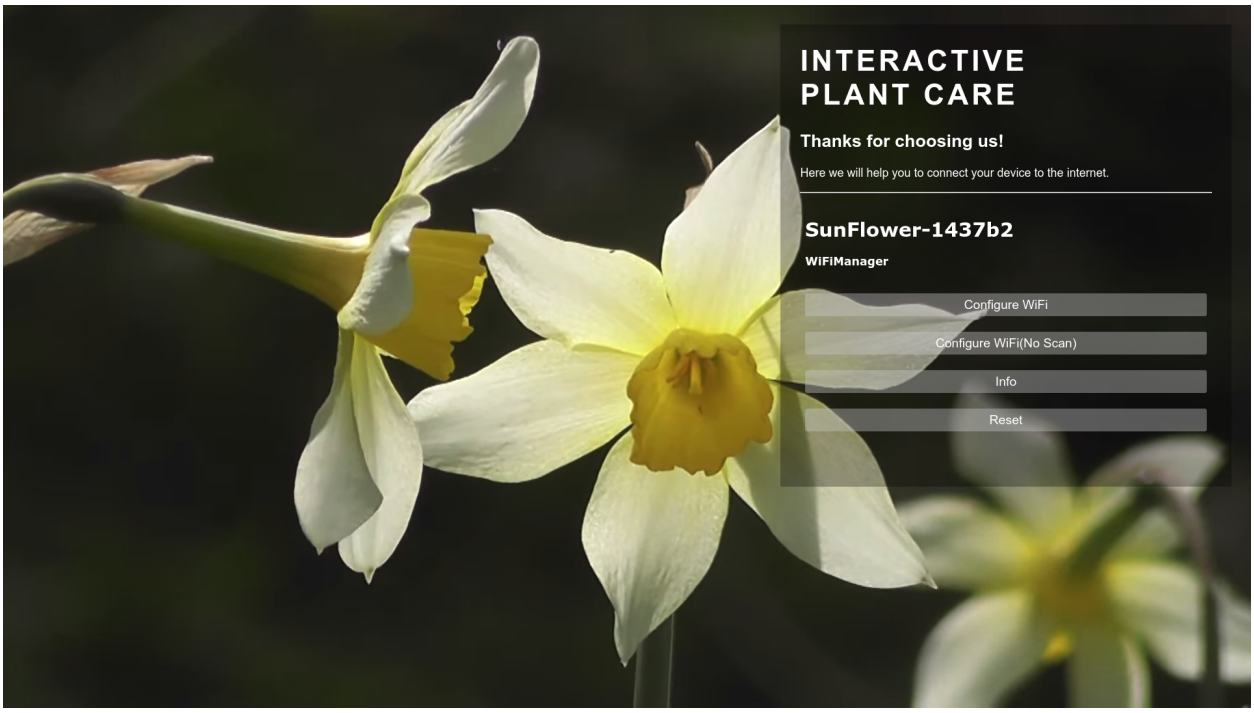


Fig. 1.1: Login Portal

1.3 3. Plant Care Monitorings Portal

1.4 4. Intern Protocol

1.4.1 Sensor communication

MQTT **Topic**: `sunflower_project/<ID>/Sensor/<Sensor_Metric>/<Unix_Timestamp>`

- ID: (e.g.)‘‘1437B2‘‘

- System_Metric: (e.g.) `sensor_light`
- Unix_Timestamp: (e.g.) `1510255785`

MQTT **Message**: `value`

1.4.2 System info

MQTT **Topic**: `sunflower_project/<ID>/System/<System_Metric>/<Unix_Timestamp>`

- ID: (e.g.)‘‘1437B2‘‘
- System_Metric: (e.g.) `system_battery`
- Unix_Timestamp: (e.g.) `1510255785`

MQTT **Message**: `value`

1.5 5. Export this document to pdf

```
pandoc -V links-as-notes=true --pdf-engine=xelatex README.md -o docs/README.pdf
```

CHAPTER 2

ESP Plant Care

This program collect and publishes plant sensor data. A login portal is used for WiFi credential settings.

2.1 ESP Plant Care - Class Documentation

2.1.1 General

MQTTHelper

class MQTTHelper
MQTT helper for fast sensor and system data uploading.

Public Functions

MQTTHelper(PubSubClient *pubSubClient, TimeProvider *timeProvider)

Parameters

- `pubSubClient`: provided by https://github.com/knolleary/pubsubclient.git
- `timeProvider`:

void **setup**()
Setup the MQTT connection.

void **update**()
Update the MQTT connection.

void **connect**()
Connect/Reconnect to the MQTT broker.

void **disconnect**()
Disconnect from the MQTT broker.

template <typename Generic**>**
void **logSensorMetric**(String *sensor*, Generic *payload*, bool *retained* = false)
Send generic sensor message with automatic time providing.

Template Parameters

- `Generic`:

Parameters

- `sensor`: name used in MQTT topic.
- `payload`: to send over MQTT. (Parsed as String(payload);)
- `retained`: MQTT message. (See: https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages)

template <typename Generic**>**
void **logSystemMetric**(String *system*, Generic *payload*, bool *retained* = false)
Send generic system message with automatic time providing.

Template Parameters

- `Generic`:

Parameters

- `system`: name used in MQTT topic.
- `payload`: to send over MQTT. (Parsed as String(payload);)
- `retained`: MQTT message. (See: https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages)

template <typename Generic**>**
void **logSensorMetric**(String *sensor*, Generic *payload*, unsigned long *time*, bool *retained* = false)
Send generic sensor message.

Template Parameters

- `Generic`:

Parameters

- `sensor`: name used in MQTT topic.
- `payload`: to send over MQTT. (Parsed as String(payload);)
- `time`:
- `retained`: MQTT message. (See: https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages)

template <typename Generic**>**
void **logSystemMetric**(String *system*, Generic *payload*, unsigned long *time*, bool *retained* = false)
Send generic sensor message.

Template Parameters

- `Generic`:

Parameters

- `system`: name used in MQTT topic.

- `payload`: to send over MQTT. (Parsed as String(payload);)
- `time`:
- `retained`: MQTT message. (See: https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages)

TimeProvider

class TimeProvider

Epoch time keeper, for accurate measurement times.

Inherits from ITimeProvider

Public Functions

TimeProvider(UDP *udp)

Parameters

- `udp`:

unsigned long **getTime**()

Return epoch time

void **update**(bool *force* = false)

update time

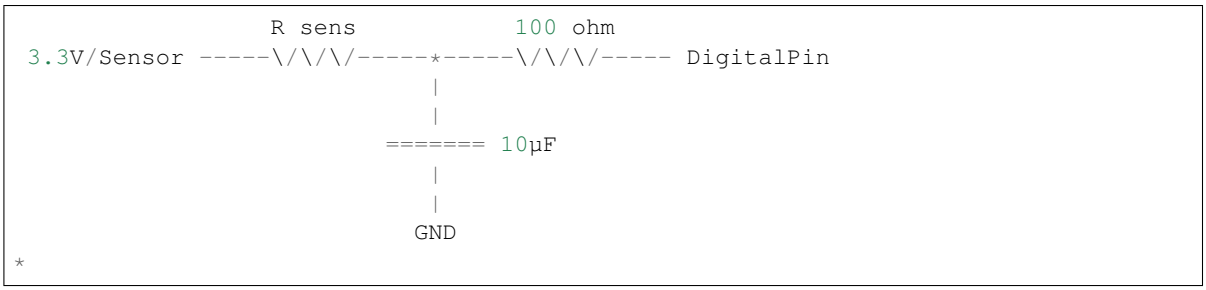
Parameters

- `force`: - if true, ignore update interval time

2.1.2 Sensors

AnalogReadCap

class AnalogReadCap



Analog sensor reader:

1. Discharge Capacitor
2. Wait until charged and time it
3. RC = time

Inherits from Sensor

Public Functions

AnalogReadCap(String *name*, *ITimeProvider* *timeProvider, uint8_t *ioPin*)

Parameters

- `name`:
- `timeProvider`:
- `ioPin`: digital read pin

BH1750

class BH1750light

BH1750 Sensor.

Inherits from Sensor

Public Functions

BH1750light(String *name*, *ITimeProvider* *timeProvider, int *sda*, int *scl*, BH1750::Mode = BH1750::CONTINUOUS_HIGH_RES_MODE)

Parameters

- `name`: - used as MQTT sensor topic name
- `timeProvider`:
- `sda`: - i2c
- `scl`: - i2c

BMP180

class BMPholder

BMP180 Sensor.

Inherits from SensorSet

Public Functions

void **registerPressureSensor**(*Sensor* *pressureSensor)

Register sensor to hold the data.

Parameters

- `pressureSensor`:

void **registerTemperatureSensor**(*Sensor* *temperatureSensor)

Register sensor to hold the data.

Parameters

- `temperatureSensor`:

void **registerAltitudeSensor**(*Sensor* *altitudeSensor)
Register sensor to hold the data.

Parameters

- altitudeSensor:

DHT22

class DHTholder
DHT22 Sensor.

Inherits from SensorSet

CHAPTER 3

Sensor Facade

This library provides a facade for sensor data. It provides a way to attach, initialize, update and read the data.

3.1 Sensor Facade - Class Documentation

- Link to github: https://github.com/ThomasDevoogdt/SensorFacade.git
- Link to docs: http://sensorfacade.readthedocs.io

This library provides a facade for sensor data. It provides a way to attach, initialize, update and read the data.

3.1.1 SensorFacade

class SensorFacade

The sensor facade holds a set of sensors and/or sensorSets.

Add all the sensors you'll keep track on and call the *begin()* function. Thereafter call periodically the *update()* function. As last provide a callback and call *ItrSensor()* to read them out.

Inherits from *ISensor*

Public Functions

SensorFacade()

creates a sensor facade instance

SensorFacade(*ITimeProvider* **timeProvider*)

creates a sensor facade instance with a time provider

Parameters

- `timeProvider`:

SensorFacade (void (**sensorItr*)) String, *Data*
creates a sensor facade instance with a iteration callback

Parameters

- `sensorItr`:

SensorFacade (*ITimeProvider* **timeProvider*, void (**sensorItr*)) String, *Data*
creates a sensor facade with a time provider and a iteration callback

Parameters

- `timeProvider`:
- `sensorItr`:

void **addSensor** (*Sensor* **sensor*)
add a sensor

Parameters

- `sensor`:

void **addSensorSet** (*SensorSet* **sensorSet*)
add a sensor set

Parameters

- `sensorSet`:

Sensor ***getSensor** (int *index*)
get a sensor by index

Return sensor

Parameters

- `index`:

void **setSensorItr** (void (**sensorItr*)) String, *Data*
set a callback for the sensor iteration

Parameters

- `sensorItr`:

void **ItrSensor** ()
iterate over all the sensors using the predefined callback

int **size** ()
get number of sensors

Return

void **update**()
call update function of all sensors

void **begin**()
call begin function of all sensors

3.1.2 SensorSet

class SensorSet
A sensor set helps you to add sensor metrics to one container.

e.g. a sensor that can reads the temperature as well the pressure.

Inherits from *ISensor*

Public Functions

SensorSet()
creates a sensor set

SensorSet(*ITimeProvider* *timeProvider)
creates a sensor set with a time provider

Parameters

- `timeProvider`:

void **begin**()
call begin function of all sensors

void **update**()
call update function of all sensors

SensorLinkedList<*Sensor* *> **getSensors**()

Return all the sensors of the container

3.1.3 Sensor

class Sensor
Each metric of each sensor should have an sensor implementation.

Implement the the update method or override the getData method.

Inherits from *ISensor*

Public Functions

Sensor(String *name*, *ITimeProvider* *timeProvider)

Parameters

- `name`: of the sensor instance
- `timeProvider`:

Sensor(String *name*)

Parameters

- `name`: of the sensor instance

Data **getData**()

Return last captured data

String **getName**()

Return name of the sensor instance - used for e.g. MQTT topic

3.1.4 ISensor

class ISensor

an abstract sensor class

Subclassed by *Sensor*, *SensorFacade*, *SensorSet*

Public Functions

virtual void **update**()

use for sensors that needs periodic attention

virtual void **begin**()

use for sensors that needs initialization

3.1.5 ITimeProvider

class ITimeProvider

an abstract time provider class

Public Functions

virtual unsigned long **getTime**()

Return abstract version returns always NAN

3.1.6 Data

struct Data

Public Functions

Data()

default struct constructor

Data(float *value*, unsigned long *time*)

Parameters

- `value`:
- `time`:

Public Members

float **`value`** = NAN
data enum value

unsigned long **`time`** = NAN
data enum time

Index