Mailbox Notifier E32

Content

[Mailbox Notifier 3](#_Toc188265827)

[PCB 3](#_Toc188265828)

[Programming ATtiny1614 3](#_Toc188265829)

[Gateway Home Assistant 4](#_Toc188265830)

[Pinout 4](#_Toc188265831)

[ESPhome YAML 4](#_Toc188265832)

[Additional files in esphome directory 5](#_Toc188265833)

[/homeassistant/esphome/uart\_Mailbox\_Notifier.h 5](#_Toc188265834)

[/homeassistant/esphome/uart\_read\_line\_sensor.h 5](#_Toc188265835)

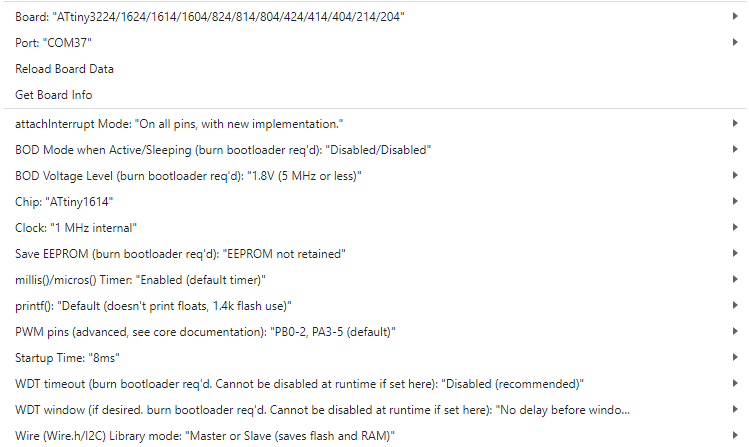
# Mailbox Notifier

## PCB

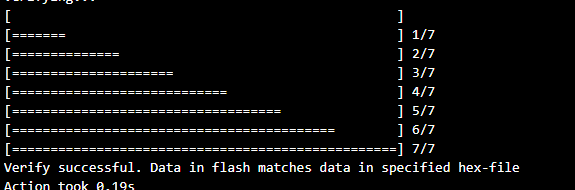
[Mailbox Notifier - OSHWLab](https://oshwlab.com/spiessa/mailbox-notifier)

## Programming ATtiny1614

### Arduino Boards setup



### After flashing:



# Gateway Home Assistant

### Pinout



## ESPhome YAML

substitutions:

  name: liliane-mailbox

  friendly\_name: Liliane-Mailbox

esphome:

  name: ${name}

  friendly\_name: ${friendly\_name}

  min\_version: 2024.6.0

  name\_add\_mac\_suffix: false

  project:

    name: esphome.web

    version: dev

  # Initialization of LoRa Board

  on\_boot:

    priority: 100

    then:

      - output.turn\_on: m0\_pin

      - output.turn\_on: m1\_pin

      - delay: 7ms

      - uart.write: [0xC0, 0x00, 0x01, 0x1A, 0x17, 0x47]

      - delay: 500ms

      - output.turn\_off: m0\_pin

      - output.turn\_off: m1\_pin

  includes:

  - uart\_Mailbox\_Notifier.h

esp32:

  board: esp32dev

  framework:

    type: esp-idf

# Enable logging

logger:

# Enable Home Assistant API

api:

# Allow Over-The-Air updates

ota:

- platform: esphome

wifi:

  ssid: !secret wifi\_ssid

  password: !secret wifi\_password

  fast\_connect: true

  # Set up a wifi access point

  ap:

    ssid: mailNotifier

    password: !secret ap\_password

# In combination with the `ap` this allows the user

# to provision wifi credentials to the device via WiFi AP.

captive\_portal:

# To have a "next url" for improv serial

web\_server:

# Restart & Safe Mode Buttons

button:

  - platform: restart

    name: "Restart Device"

  - platform: safe\_mode

    name: "Restart in Safe Mode"

  - platform: factory\_reset

    name: Restart with Factory Default Settings

#-----------------------------------------

# UART Configuration for LoRa Module

uart:

  id: uart\_bus

  tx\_pin: GPIO17

  rx\_pin: GPIO16

  baud\_rate: 9600

# GPIO Pins for LoRa Module

output:

  - platform: gpio

    pin: GPIO21

    id: m0\_pin

  - platform: gpio

    pin: GPIO19

    id: m1\_pin

text\_sensor:

  - platform: custom

    lambda: |-

      auto my\_custom\_sensor = new Uart\_Mailbox\_Notifier(id(uart\_bus));

      App.register\_component(my\_custom\_sensor);

      return {my\_custom\_sensor};

    text\_sensors:

      id: uart\_readline

binary\_sensor:

  - platform: template

    id: mailbox\_state

    name: "Mailbox State"

    lambda: |-

      // Check for full (0x55) and empty (0xAA) states from uart\_readline

      if (id(uart\_readline).state == "true") {

        return true;  // Mailbox Full

      } else if (id(uart\_readline).state == "false") {

        return false;  // Mailbox Empty

      }

      return {};  // Return no state if data is invalid

    on\_state:

      then:

        - logger.log: "Mailbox state changed"

        - uart.write: [0x25]  # Send 0x25 via UART

## Additional files in esphome directory

### /homeassistant/esphome/uart\_Mailbox\_Notifier.h

#include "esphome.h"

class Uart\_Mailbox\_Notifier : public Component, public UARTDevice, public TextSensor {

public:

Uart\_Mailbox\_Notifier(UARTComponent \*parent) : UARTDevice(parent) {}

void setup() override {

// nothing to do here

}

void loop() override {

while (available()) {

char readch = read();

if (readch == 0x55) {

//ESP\_LOGI("Uart\_Mailbox\_Notifier", "Publishing state: true");

publish\_state("true"); // Publish "full" when the byte is 0x55

} else if (readch == 0xAA) {

//ESP\_LOGI("Uart\_Mailbox\_Notifier", "Publishing state: false");

publish\_state("false"); // Publish "empty" when the byte is 0xAA

}

}

}

};

### /homeassistant/esphome/uart\_read\_line\_sensor.h

#include "esphome.h"

class UartReadLineSensor : public Component, public UARTDevice, public TextSensor {

public:

UartReadLineSensor(UARTComponent \*parent) : UARTDevice(parent) {}

void setup() override {

// nothing to do here

}

int readline(int readch, char \*buffer, int len)

{

static int pos = 0;

int rpos;

if (readch > 0) {

switch (readch) {

case '\n':

case '\r': // Return on CR or newline

buffer[pos] = 0; // Just to be sure, set last character 0

rpos = pos;

pos = 0; // Reset position index ready for next time

return rpos;

default:

if ((pos < len-1) && ( readch < 127 )) { // Filter on <127 to make sure it is a character

buffer[pos++] = readch;

buffer[pos] = 0;

}

else

{

buffer[pos] = 0; // Just to be sure, set last character 0

rpos = pos;

pos = 0; // Reset position index ready for next time

return rpos;

}

}

}

// No end of line has been found, so return -1.

return -1;

}

void loop() override {

const int max\_line\_length = 80;

static char buffer[max\_line\_length];

while (available()) {

if(readline(read(), buffer, max\_line\_length) > 0) {

ESP\_LOGI("UartReadLineSensor", "Publishing state: %s", buffer);

publish\_state(buffer);

}

}

}

};