DETAILED DESIGN DOCUMENT

for

WiSocket

Version 1.0

Prepared by HgN

CRETA Solution Company

September 22, 2017

Contents

1 Introduction and Goals

1.1 Introduction

This document introduce the key idea of WiSocket. Its intended audience comprises Developers and Testers who need a rough understanding on the concepts of the module.

The document describes the component and its internal logic, lists any dependencies to external components, internal constraints and defines its APIs.

The document explains the components design in detail.

1.2 Goals

The main goals of this product is:

1. WiSocket can control one device via both button or app.
2. WiSocket's WiFi can be setup easily using SmartConfig.

1.3 Definitions, Acronyms and Abbreviations

MQTT	Message Queuing Telemetry Transport
MAC	Media Access Controll Address

1.4 Referenced Document

HW Schematic	1_Input/WiSocket_Schematic.pdf
Detailed requirement	1_Input/WiSocket_Requirement.pdf

2 Design Constrains

1. Product must have a status led that show product status clearly, easy.
2. Control/Config method should be simple and convinient for non-tech user.
3. All devices and leds must be active-low.

3 Design Overview

3.1 Overview

1. General:
    - This product has:
        - 1 device which can be controlled ON/OFF via digital sysnal.
        - 1 config button (push button), use polling method.
        - 1 control button (switch), use interrupt.
        - 1 status led.
    - All devices and leds must be active-low.
    - Status led can show: WiFi Connecting, WiFi Connected and WiFi Configuring.
    - Firmware has two state: Config and Control.
    - User can go in and out state Config by holding config button for about 3 seconds.
    - User can control device via control button at anytime, even without internet.
2. State Config:
    - Use SmartConfig to setup device's WiFi.
    - An config flag stored in EEPROM at address 0x10. This flag must be read each time start up to check WiFi configuration.
        - Value 0x05: Device is already configured. Try to connect using saved SSID and password.
        - Other value: Deive has no SSID and password to connect.
3. State Control:

- Device try to connect to WiFi and MQTT server. Always check for proper connection. Should have connection failure hanlder.
- Topic name:

Topic to subcribe	ESP+<MAC>/master
Topic to publish	ESP+<MAC>/slave

- Use protocol as in Requirement Documentation (5.3.2 Data Format) to communicate with user's app. There's 2 function:
  - Function Control: Control 1 device ON/OFF.
  - Function Data: Get status of device.

3.2 Configuration

These parameters should be setup manually in firmware:

MQTT server	"iot.eclipse.org"

3.3 File structure

```
WiSocket.ino
|___state.h
    |___protocol.h
    |   |___mqtt.h
    |___device.h
    |___button.h
```

3.4 Critical timing constrains

- All function must have execute time within 1s to prevent ESP watchdog timer reset.

3.5 Diagram

3.5.1 State machine

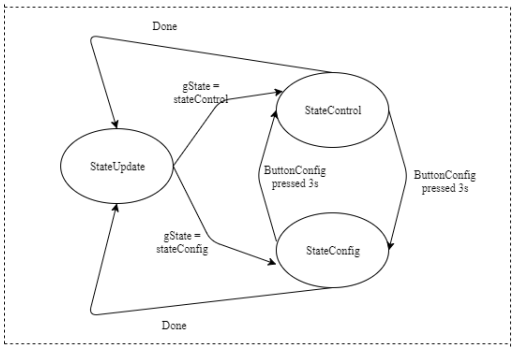
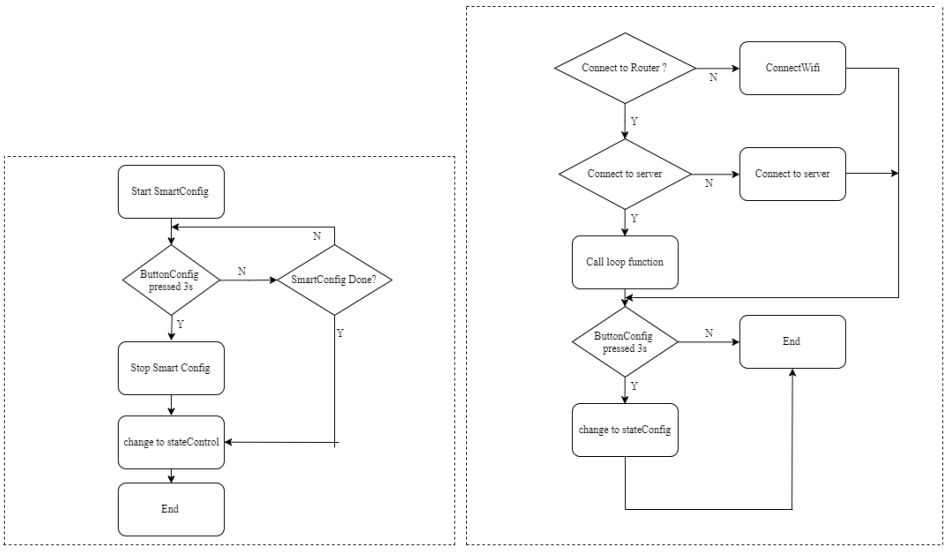


Figure 3.1: WiSocket's state machine

3.5.2 State Config and Control



(a) State Config (b) State Control

Figure 3.2: WiSocket's state config and control

3.6 Failure handling strategy

- All conditional loops must have time out condition to prevent hanging.

4 Environment and Dependencies

4.1 Evironment

This project is built using Arduino IDE 1.8.4.

4.2 Dependencies

This project need these dependencies:

- Board packet: esp8266 by ESP8266 Community version 2.2.0.
- Library: ArduinoJson by Benoit Blanchon version 5.5.1.
- Library: PubSubClient by Nick O'Leary version 2.6.0.

5 API specification

5.1 Device

```
void deviceInit(void);
void deviceOn(void);
void deviceOff(void);
void deviceToggle(void);
void ledDeviceOn(void);
void ledDeviceOff(void);
void ledWifiOn(void);
void ledWifiOff(void);
void ledWifiToggle(void);
int deviceStatus(void);
```

5.2 Button

```
void buttonInit(void);                                  [ESP_PJ 018]
void buttonConfigISRHandler(void);                      [ESP_PJ 020]
bool buttonConfigCheck(void);                           [ESP_PJ 019, 008]
bool buttonControlCheck(void);                          [ESP_PJ 020]
```

5.3 MQTT

```
void callback(char* topic, byte* payload, unsigned int length);
                                                        [ESP_PJ 014-017]
void mqttCreateTopic(void);                             [ESP_PJ 022-023]
void mqttSubscribe(void);                               [ESP_PJ 021]
int mqttConnect (void);                                 [ESP_PJ 021]
int mqttConnected (void);                               [ESP_PJ 021]
void mqttPublish (String jsonOut);                      [ESP_PJ 021]
void mqttLoop (void);                                   [ESP_PJ 013]
```

5.4 Protocol

```
String protocolCreateJson (String pFunc, String pAddr, String pData)
                                                              [ESP_PJ 024]
void protocolDataProcess (String recv_json);                  [ESP_PJ 026-032]
void protocolButtonProcess (void);
```

5.5 State

```
void stateSetup(void);                                        [ESP_PJ 006-007]
void stateConfig(void);                                       [ESP_PJ 005, 033-037]
void stateControl(void);                                      [ESP_PJ 009-12]
```

6 Unit test