ECE 4999 Electromechanical Lock

1.2

Generated by Doxygen 1.8.20

Chapter 1

ECE 4999 Electromechanical Lock

ESP32 based lock that features RFID and keypad to unlock

Author

Matthew Hait

Remington Davids

Version

1.2

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

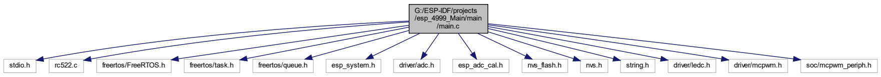
File Documentation

3.1 G:/ESP-IDF/projects/esp_4999_Main/main/CMakeLists.txt File Reference

3.2 G:/ESP-IDF/projects/esp_4999_Main/main/main.c File Reference

```
#include <stdio.h>
#include "rc522.c"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "esp_system.h"
#include "driver/adc.h"
#include "esp_adc_cal.h"
#include "nvs_flash.h"
#include "nvs.h"
#include <string.h>
#include "driver/ledc.h"
#include "driver/mcpwm.h"
#include "soc/mcpwm_periph.h"
```

Include dependency graph for main.c:



Macros

- #define ESP_INTR_FLAG_LEVEL 0

  Interupt vector level ref: `https://github.com/anoochit/esp32-example/blob/master/2080_GPIO_Interrupt/main/gpio_intr.c`.

- #define MULTISAMPLING (50)

  ADC samples to collect for keypad

- #define defaultKEY 1234

  Default key pin from factory. Randomize on every product.

- #define serialLen (5)

  Length of RFID serial

- #define GPIO_KEYPAD (GPIO_SEL_34)

  Keypad IO Pin

- #define GPIO_KEYPAD_IO 34

  Keypad IO Pin

- #define GPIO_BATV (GPIO_SEL_35)

  Battery Monitor IO Pin

- #define GPIO_BATV_IO 35

  Battery Monitor IO Pin

- #define KEYPAD_BUF_SZ (17)

  Max pin size is KEYPAD_BUF_SZ - 1.
- #define LEDC_HS_TIMER LEDC_TIMER_0

  HW RTC for RGB control.
- #define LEDC_HS_MODE LEDC_HIGH_SPEED_MODE

  RGB PWM timing mode.
- #define LEDC_HS_RED_GPIO (18)

  Status RGB RED output pin.
- #define LEDC_HS_RED_CHANNEL LEDC_CHANNEL_0

  Status RGB RED LEDC channel.
- #define RED 0
- #define LEDC_HS_GREEN_GPIO (19)

  Status RGB GREEN output pin.
- #define LEDC_HS_GREEN_CHANNEL LEDC_CHANNEL_1

  Status RGB GREEN LEDC channel.
- #define GREEN 1
- #define LEDC_HS_BLUE_GPIO (20)

  Status RGB BLUE output pin.
- #define LEDC_HS_BLUE_CHANNEL LEDC_CHANNEL_2

  Status RGB BLUE LEDC channel.
- #define Blue 2
- #define LEDC_TEST_CH_NUM 3

  Status RGB channel quantity.
- #define LEDC_TEST_DUTY (4000)

  Duty cycle for quick LED flash.
- #define MTR_PWM0A_OUT 15

  PWM H-Bridge Input A.
- #define MTR_PWM0B_OUT 16

  PWM H-Bridge Input B.
- #define MTR_Sleep (GPIO_SEL_33)

  PWM H-Bridge Sleep Control.
- #define MTR_SleepC (GPIO_NUM_33)

  PWM H-Bridge Sleep Control.
- #define MTR_PWM_FREQ 1000

  PWM H-Bridge PWM Frequency.
- #define MTR_PERIOD 3000

  PWM H-Bridge lock cycle time.
- #define MTR_duty 100

  PWM H-Bridge duty cycle.

Functions

- void pairTagHandler (uint8_t ∗serial_no)

  Callback function for rc522 when pairing new NFC tags.
- void tag_handler (uint8_t ∗serial_no)

  Callback function for rc522 library under normal operation.
- void toggle_Lock ()

  Handles motor control and position logic.
- void unpairNFC ()

  Removes NFC keys from heap and NVS.
- void writePinNVS (uint64_t secret)

  Stores keypad pin into non-voltile storage.
- unsigned long runTime ()

  Grabs FreeRTOS runtime to determine timeouts.
- void batCheck ()

  Flashes LED if battery is low.
- static void IRAM_ATTR gpio_isr_handler (void ∗arg)

  Keypad GPIO xQueue translate.
- static void keypadCallback (void ∗arg)

  Keypad GPIO interupt handler.
- static void keypadPINCallback (void ∗arg)

  Keypad GPIO interupt handler for setting new pin.
- void initGPIO ()

  Initialize GPIO pins. Ref: `https://github.com/espressif/esp-idf/blob/master/examples/peripherals/gpio/` `_gpio/main/gpio_example_main.c`.
- void init_NVS ()

  Initialize NVS; pulls key qty and pin from storage into memory. Ref: `https://github.com/espressif/esp-idf/tree/1d7068` `_rw_value`.
- void grabKeyList (uint8_t ∗buf, int count)

  Grabs NFC keylist from NVS.
- void writeNFCKey (uint8_t ∗buf, int count)

  Writes NFC key to NVS. Only writes one key, not all NFC keys at once. (For use in pairing new NFC Tag)
- void writeKeyQtyNVS (uint8_t qty)

  Stores NFC key qty into non-voltile storage.
- void app_main (void)

Variables

- uint8_t storedKey_QTY = 0

  QTY of stored NFC keys

- uint64_t keypadPin = 0

  Secret pin, shhhhhhhh

- bool keyPinValid = false

  Logic check to insure password cannot be entered from corrupted memory

- esp_adc_cal_characteristics_t characteristics
- static xQueueHandle gpio_evt_queue = NULL

  FreeRTOS queue for keypad events.
- bool mtrForward = true

  Motor direction control. Default Forward

- int thresholds [12] = {195, 305, 375, 405, 447, 477, 491, 512, 530, 538, 551, 563}

  ADC values for keypad reference.
- char keypad [12] = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '∗', '0', '#'}

  Translated keypad values.
- char keypadInputBuffer [KEYPAD_BUF_SZ]

  Buffer storage for keypad input.
- int keypadInputCnt = 0

  Keypad buffer index.
- unsigned long keypadLastUpdate = 0

  Last keypad input tick [ms].
- const unsigned long keypadInvalidAfter = 5000

  Keypad timeout period [ms].
- const unsigned long pairInvalidAfter = 5000

  NFC pair timeout period [ms].
- bool nfcEnable = true

  NFC enabler.
- uint8_t ∗ keyList [serialLen]

  NFC key list.
- ledc_channel_config_t ledc_channel [LEDC_TEST_CH_NUM]

  Status RGB LEDC config.
- const rc522_start_args_t pairNFC_args

  rc522 config when pairing NFC tags.
- const rc522_start_args_t running_args

  rc522 config when under normal operation.
- mcpwm_config_t pwm_config

  H-Bridge Motor config.
- bool noPairDect = true

  Poorly designed timeout bypass for NFC pair.
- TaskHandle_t xPinHandle

  FreeRTOS task handle for changing xqueue on keypad.
- TaskHandle_t xKeypadHandle

  FreeRTOS task handle for normal xqueue on keypad.

3.2.1 Macro Definition Documentation

3.2.1.1 Blue

```
#define Blue 2
```

Definition at line 54 of file main.c.

3.2.1.2 defaultKEY

```
#define defaultKEY 1234
```

Default key pin from factory. Randomize on every product.

Definition at line 22 of file main.c.

3.2.1.3 ESP_INTR_FLAG_LEVEL

```
#define ESP_INTR_FLAG_LEVEL 0
```

Interupt vector level ref: https://github.com/anoochit/esp32-example/blob/master/2080_↩GPIO_Interrupt/main/gpio_intr.c.

Definition at line 18 of file main.c.

3.2.1.4 GPIO_BATV

```
#define GPIO_BATV (GPIO_SEL_35)
```

Battery Monitor IO Pin

Definition at line 31 of file main.c.

3.2.1.5 GPIO_BATV_IO

```
#define GPIO_BATV_IO 35
```

Battery Monitor IO Pin

Definition at line 33 of file main.c.

3.2.1.6 GPIO_KEYPAD

```
#define GPIO_KEYPAD (GPIO_SEL_34)
```

Keypad IO Pin

Definition at line 26 of file main.c.

3.2.1.7 GPIO_KEYPAD_IO

```
#define GPIO_KEYPAD_IO 34
```

Keypad IO Pin

Definition at line 28 of file main.c.

3.2.1.8 GREEN

```
#define GREEN 1
```

Definition at line 49 of file main.c.

3.2.1.9 KEYPAD_BUF_SZ

```
#define KEYPAD_BUF_SZ (17)
```

Max pin size is KEYPAD_BUF_SZ - 1.

Definition at line 35 of file main.c.

3.2.1.10 LEDC_HS_BLUE_CHANNEL

```
#define LEDC_HS_BLUE_CHANNEL LEDC_CHANNEL_2
```

Status RGB BLUE LEDC channel.

Definition at line 53 of file main.c.

3.2.1.11 LEDC_HS_BLUE_GPIO

```
#define LEDC_HS_BLUE_GPIO (20)
```

Status RGB BLUE output pin.

Definition at line 51 of file main.c.

3.2.1.12 LEDC_HS_GREEN_CHANNEL

```
#define LEDC_HS_GREEN_CHANNEL LEDC_CHANNEL_1
```

Status RGB GREEN LEDC channel.

Definition at line 48 of file main.c.

3.2.1.13 LEDC_HS_GREEN_GPIO

```
#define LEDC_HS_GREEN_GPIO (19)
```

Status RGB GREEN output pin.

Definition at line 46 of file main.c.

3.2.1.14 LEDC_HS_MODE

```
#define LEDC_HS_MODE LEDC_HIGH_SPEED_MODE
```

RGB PWM timing mode.

Definition at line 39 of file main.c.

3.2.1.15 LEDC_HS_RED_CHANNEL

```
#define LEDC_HS_RED_CHANNEL LEDC_CHANNEL_0
```

Status RGB RED LEDC channel.

Definition at line 43 of file main.c.

3.2.1.16 LEDC_HS_RED_GPIO

```
#define LEDC_HS_RED_GPIO (18)
```

Status RGB RED output pin.

Definition at line 41 of file main.c.

3.2.1.17 LEDC_HS_TIMER

```
#define LEDC_HS_TIMER LEDC_TIMER_0
```

HW RTC for RGB control.

Definition at line 37 of file main.c.

3.2.1.18 LEDC_TEST_CH_NUM

```
#define LEDC_TEST_CH_NUM 3
```

Status RGB channel quantity.

Definition at line 56 of file main.c.

3.2.1.19 LEDC_TEST_DUTY

```
#define LEDC_TEST_DUTY (4000)
```

Duty cycle for quick LED flash.

Definition at line 58 of file main.c.

3.2.1.20 MTR_duty

```
#define MTR_duty 100
```

PWM H-Bridge duty cycle.

Definition at line 72 of file main.c.

3.2.1.21 MTR_PERIOD

```
#define MTR_PERIOD 3000
```

PWM H-Bridge lock cycle time.

Definition at line 70 of file main.c.

3.2.1.22 MTR_PWM0A_OUT

```
#define MTR_PWM0A_OUT 15
```

PWM H-Bridge Input A.

Definition at line 60 of file main.c.

3.2.1.23 MTR_PWM0B_OUT

```
#define MTR_PWM0B_OUT 16
```

PWM H-Bridge Input B.

Definition at line 62 of file main.c.

3.2.1.24 MTR_PWM_FREQ

```
#define MTR_PWM_FREQ 1000
```

PWM H-Bridge PWM Frequency.

Definition at line 68 of file main.c.

3.2.1.25 MTR_Sleep

```
#define MTR_Sleep (GPIO_SEL_33)
```

PWM H-Bridge Sleep Control.

Definition at line 64 of file main.c.

3.2.1.26 MTR_SleepC

```
#define MTR_SleepC (GPIO_NUM_33)
```

PWM H-Bridge Sleep Control.

Definition at line 66 of file main.c.

3.2.1.27 MULTISAMPLING

```
#define MULTISAMPLING (50)
```

ADC samples to collect for keypad

Definition at line 20 of file main.c.

3.2.1.28 RED

```
#define RED 0
```

Definition at line 44 of file main.c.

3.2.1.29 serialLen

```
#define serialLen (5)
```

Length of RFID serial

Definition at line 24 of file main.c.

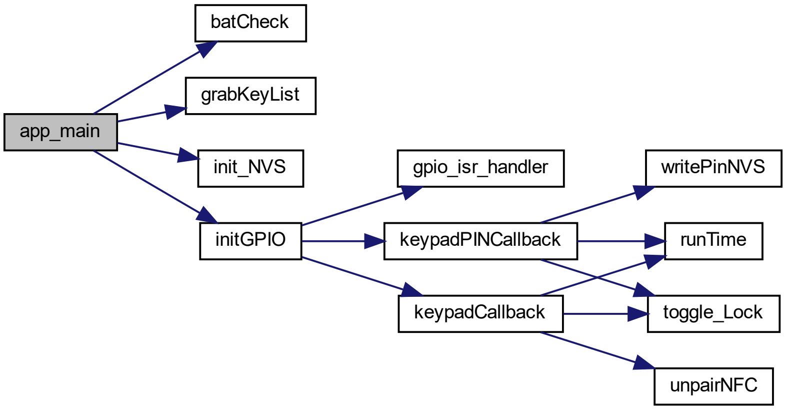
3.2.2 Function Documentation

3.2.2.1 app_main()

```
void app_main (
            void  )
```

Definition at line 846 of file main.c.

Here is the call graph for this function:



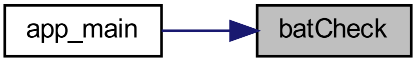
3.2.2.2 batCheck()

```
void batCheck ( )
```

Flashes LED if battery is low.

Definition at line 196 of file main.c.

Here is the caller graph for this function:



3.2.2.3 gpio_isr_handler()

```
static void IRAM_ATTR gpio_isr_handler (
             void * arg )  [static]
```

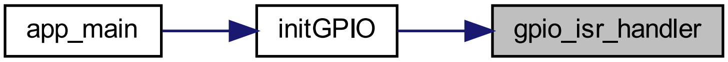
Keypad GPIO xQueue translate.

Parameters

∗*arg*	Optional switch on GPIO pin.

Definition at line 222 of file main.c.

Here is the caller graph for this function:



3.2.2.4 grabKeyList()

```
void grabKeyList (
             uint8_t * buf,
             int count )
```

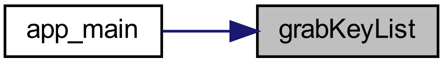
Grabs NFC keylist from NVS.

Parameters

∗*buf*	Key Output
count	Key Index

Definition at line 627 of file main.c.

Here is the caller graph for this function:



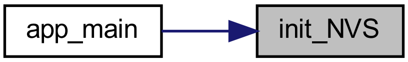
3.2.2.5 init_NVS()

```
void init_NVS ( )
```

Initialize NVS; pulls key qty and pin from storage into memory. Ref: https://github.com/espressif/esp-idf/tree/1d7 _rw_value.

Definition at line 543 of file main.c.

Here is the caller graph for this function:



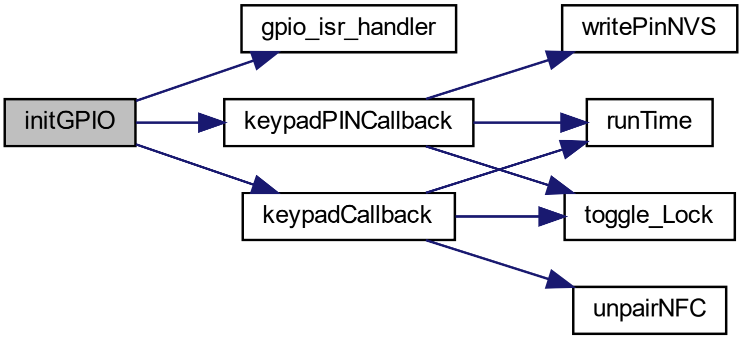
3.2.2.6 initGPIO()

```
void initGPIO ( )
```

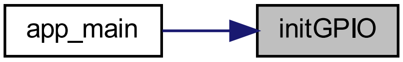
Initialize GPIO pins. Ref: https://github.com/espressif/esp-idf/blob/master/examples/peripherals/gp _gpio/main/gpio_example_main.c.

Definition at line 466 of file main.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.7 keypadCallback()

```
static void keypadCallback (
            void * arg )  [static]
```

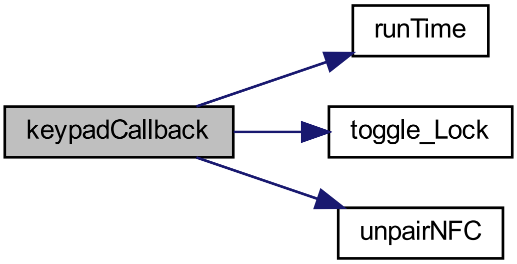
Keypad GPIO interupt handler.

Parameters

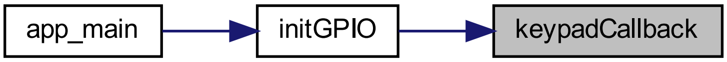
∗*arg*	Optional switch on GPIO pin.

Definition at line 232 of file main.c.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.8 keypadPINCallback()

```
static void keypadPINCallback (
             void * arg )  [static]
```

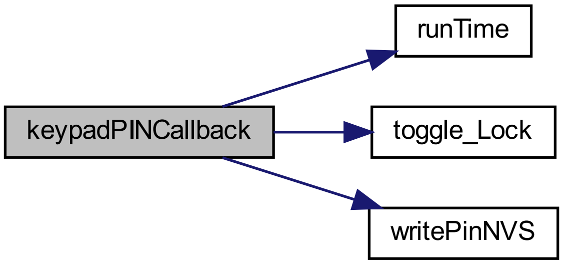
Keypad GPIO interupt handler for setting new pin.

Parameters

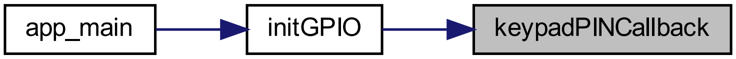
∗arg	Optional switch on GPIO pin.

Definition at line 398 of file main.c.

Here is the call graph for this function:



Here is the caller graph for this function:



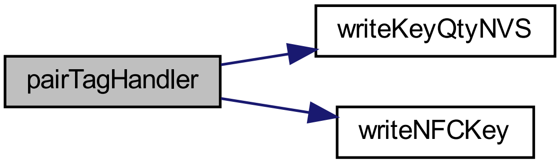
3.2.2.9 pairTagHandler()

```
void pairTagHandler (
            uint8_t * serial_no )
```

Callback function for rc522 when pairing new NFC tags.

Definition at line 811 of file main.c.

Here is the call graph for this function:



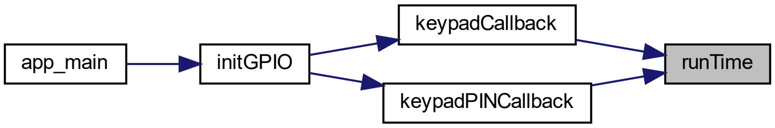
3.2.2.10 runTime()

```
unsigned long runTime ( )
```

Grabs FreeRTOS runtime to determine timeouts.

Definition at line 191 of file main.c.

Here is the caller graph for this function:



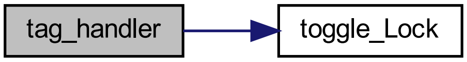
3.2.2.11 tag_handler()

```
void tag_handler (
            uint8_t * serial_no )
```

Callback function for rc522 library under normal operation.

Definition at line 781 of file main.c.

Here is the call graph for this function:



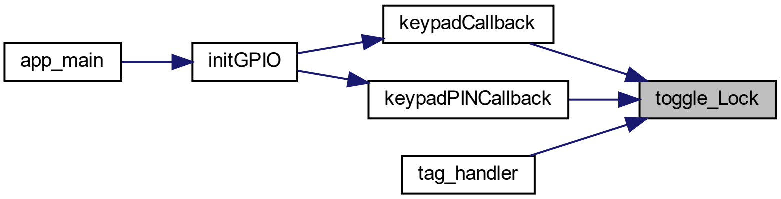
3.2.2.12 toggle_Lock()

```
void toggle_Lock ( )
```

Handles motor control and position logic.

Definition at line 825 of file main.c.

Here is the caller graph for this function:



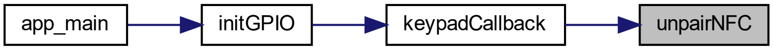
3.2.2.13 unpairNFC()

```
void unpairNFC ( )
```

Removes NFC keys from heap and NVS.

Definition at line 714 of file main.c.

Here is the caller graph for this function:



3.2.2.14 writeKeyQtyNVS()

```
void writeKeyQtyNVS (
            uint8_t qty )
```

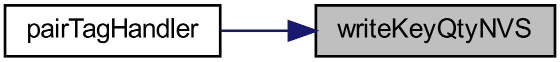
Stores NFC key qty into non-voltile storage.

Parameters

qty	Value to be stored

Definition at line 758 of file main.c.

Here is the caller graph for this function:



3.2.2.15 writeNFCKey()

```
void writeNFCKey (
            uint8_t * buf,
            int count )
```

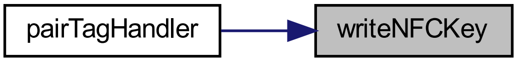
Writes NFC key to NVS. Only writes one key, not all NFC keys at once. (For use in pairing new NFC Tag)

Parameters

∗*buf*	Key Input
count	Key Index

Definition at line 677 of file main.c.

Here is the caller graph for this function:



3.2.2.16 writePinNVS()

```
void writePinNVS (
             uint64_t secret )
```

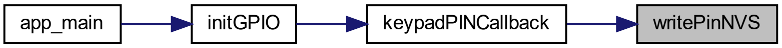
Stores keypad pin into non-voltile storage.

Parameters

secret	Input pin to be stored

Definition at line 601 of file main.c.

Here is the caller graph for this function:



3.2.3 Variable Documentation

3.2.3.1 characteristics

```
esp_adc_cal_characteristics_t characteristics
```

Definition at line 81 of file main.c.

3.2.3.2 gpio_evt_queue

```
xQueueHandle gpio_evt_queue = NULL  [static]
```

FreeRTOS queue for keypad events.

Definition at line 83 of file main.c.

3.2.3.3 keyList

```
uint8_t* keyList[serialLen]
```

NFC key list.

Definition at line 103 of file main.c.

3.2.3.4 keypad

```
char keypad[12] = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '*', '0', '#'}
```

Translated keypad values.

Definition at line 89 of file main.c.

3.2.3.5 keypadInputBuffer

```
char keypadInputBuffer[KEYPAD_BUF_SZ]
```

Buffer storage for keypad input.

Definition at line 91 of file main.c.

3.2.3.6 keypadInputCnt

```
int keypadInputCnt = 0
```

Keypad buffer index.

Definition at line 93 of file main.c.

3.2.3.7 keypadInvalidAfter

```
const unsigned long keypadInvalidAfter = 5000
```

Keypad timeout period [ms].

Definition at line 97 of file main.c.

3.2.3.8 keypadLastUpdate

```
unsigned long keypadLastUpdate = 0
```

Last keypad input tick [ms].

Definition at line 95 of file main.c.

3.2.3.9 keypadPin

```
uint64_t keypadPin = 0
```

Secret pin, shhhhhhhh

Definition at line 78 of file main.c.

3.2.3.10 keyPinValid

```
bool keyPinValid = false
```

Logic check to insure password cannot be entered from corrupted memory

Definition at line 80 of file main.c.

3.2.3.11 ledc_channel

```
ledc_channel_config_t ledc_channel[LEDC_TEST_CH_NUM]
```

Initial value:

```
= {
    {
        .channel    = LEDC_HS_RED_CHANNEL,
        .duty       = 0,
        .gpio_num   = LEDC_HS_RED_GPIO,
        .speed_mode = LEDC_HS_MODE,
        .hpoint     = 0,
        .timer_sel  = LEDC_HS_TIMER
    },
    {
        .channel    = LEDC_HS_GREEN_CHANNEL,
        .duty       = 0,
        .gpio_num   = LEDC_HS_GREEN_GPIO,
        .speed_mode = LEDC_HS_MODE,
        .hpoint     = 0,
        .timer_sel  = LEDC_HS_TIMER
    },
    {
        .channel    = LEDC_HS_BLUE_CHANNEL,
        .duty       = 0,
        .gpio_num   = LEDC_HS_BLUE_GPIO,
        .speed_mode = LEDC_HS_MODE,
        .hpoint     = 0,
        .timer_sel  = LEDC_HS_TIMER
    }
}
```

Status RGB LEDC config.

Definition at line 105 of file main.c.

3.2.3.12 mtrForward

```
bool mtrForward = true
```

Motor direction control. Default Forward

Definition at line 85 of file main.c.

3.2.3.13 nfcEnable

```
bool nfcEnable = true
```

NFC enabler.

Definition at line 101 of file main.c.

3.2.3.14 noPairDect

```
bool noPairDect = true
```

Poorly designed timeout bypass for NFC pair.

Definition at line 159 of file main.c.

3.2.3.15 pairInvalidAfter

```
const unsigned long pairInvalidAfter = 5000
```

NFC pair timeout period [ms].

Definition at line 99 of file main.c.

3.2.3.16 pairNFC_args

```
const rc522_start_args_t pairNFC_args
```

Initial value:

```
= {
    .miso_io = 25,
    .mosi_io = 23,
    .sck_io = 19,
    .sda_io = 22,
    .callback = &pairTagHandler
}
```

rc522 config when pairing NFC tags.

Definition at line 134 of file main.c.

3.2.3.17 pwm_config

```
mcpwm_config_t pwm_config
```

Initial value:

```
= {
    .frequency = MTR_PWM_FREQ,
    .cmpr_a = 0,
    .cmpr_b = 0,
    .counter_mode = MCPWM_UP_COUNTER,
    .duty_mode = MCPWM_DUTY_MODE_0
}
```

H-Bridge Motor config.

Definition at line 151 of file main.c.

3.2.3.18 running_args

```
const rc522_start_args_t running_args
```

Initial value:

```
= {
    .miso_io = 25,
    .mosi_io = 23,
    .sck_io = 19,
    .sda_io = 22,
    .callback = &tag_handler
}
```

rc522 config when under normal operation.

Definition at line 143 of file main.c.

3.2.3.19 storedKey_QTY

```
uint8_t storedKey_QTY = 0
```

QTY of stored NFC keys

Definition at line 76 of file main.c.

3.2.3.20 thresholds

```
int thresholds[12] = {195, 305, 375, 405, 447, 477, 491, 512, 530, 538, 551, 563}
```

ADC values for keypad reference.

Definition at line 87 of file main.c.

3.2.3.21 xKeypadHandle

```
TaskHandle_t xKeypadHandle
```

FreeRTOS task handle for normal xqueue on keypad.

Definition at line 163 of file main.c.

3.2.3.22 xPinHandle

`TaskHandle_t xPinHandle`

FreeRTOS task handle for changing xqueue on keypad.

Definition at line 161 of file main.c.

3.3 main.c

```
00001 #include <stdio.h>
00002 #include "rc522.c"
00003 #include "freertos/FreeRTOS.h"
00004 #include "freertos/task.h"
00005 #include "freertos/queue.h"
00006 #include "esp_system.h"
00007 #include "driver/adc.h"
00008 #include "esp_adc_cal.h"
00009 #include "nvs_flash.h"
00010 #include "nvs.h"
00011 #include <string.h>
00012 #include "driver/ledc.h"
00013 #include "driver/mcpwm.h"
00014 #include "soc/mcpwm_periph.h"
00015 
00016 
00018 #define ESP_INTR_FLAG_LEVEL 0
00019 
00020 #define MULTISAMPLING       (50)
00021 
00022 #define defaultKEY  1234
00023 
00024 #define serialLen   (5)
00025 
00026 #define GPIO_KEYPAD (GPIO_SEL_34)
00027 
00028 #define GPIO_KEYPAD_IO  34
00029 
00030 //Might need to move to ADC 2 if confict with Keypad trigger
00031 #define GPIO_BATV   (GPIO_SEL_35)
00032 
00033 #define GPIO_BATV_IO    35
00034 
00035 #define KEYPAD_BUF_SZ   (17)
00036 
00037 #define LEDC_HS_TIMER          LEDC_TIMER_0
00038 
00039 #define LEDC_HS_MODE           LEDC_HIGH_SPEED_MODE
00040 
00041 #define LEDC_HS_RED_GPIO       (18)
00042 
00043 #define LEDC_HS_RED_CHANNEL    LEDC_CHANNEL_0
00044 #define RED                     0
00045 
00046 #define LEDC_HS_GREEN_GPIO     (19)
00047 
00048 #define LEDC_HS_GREEN_CHANNEL  LEDC_CHANNEL_1
00049 #define GREEN                   1
00050 
00051 #define LEDC_HS_BLUE_GPIO      (20)
00052 
00053 #define LEDC_HS_BLUE_CHANNEL   LEDC_CHANNEL_2
00054 #define Blue                    2
00055 
00056 #define LEDC_TEST_CH_NUM       3
00057 
00058 #define LEDC_TEST_DUTY         (4000)
00059 
00060 #define MTR_PWM0A_OUT 15
00061 
00062 #define MTR_PWM0B_OUT 16
00063 
00064 #define MTR_Sleep (GPIO_SEL_33)
```

```

#define MTR_SleepC (GPIO_NUM_33)

#define MTR_PWM_FREQ 1000

#define MTR_PERIOD 3000

#define MTR_duty 100
    //If falling trigger doesn’t work with ADC, attach pin to keypad circuit.
    //#define GPIO_SENSE    (GPIO_SEL_39)
uint8_t storedKey_QTY = 0;
uint64_t keypadPin = 0;
bool keyPinValid = false;
esp_adc_cal_characteristics_t characteristics;
static xQueueHandle gpio_evt_queue = NULL;
bool mtrForward = true;
int thresholds[12] = {195, 305, 375, 405, 447, 477, 491, 512, 530, 538, 551, 563};
char keypad[12] = {’1’, ’2’, ’3’, ’4’, ’5’, ’6’, ’7’, ’8’, ’9’, ’*’, ’0’, ’#’};
char keypadInputBuffer[KEYPAD_BUF_SZ];
int keypadInputCnt = 0;
unsigned long keypadLastUpdate = 0;
const unsigned long keypadInvalidAfter = 5000;
const unsigned long pairInvalidAfter = 5000;
bool nfcEnable = true;
uint8_t *keyList[serialLen];
ledc_channel_config_t ledc_channel[LEDC_TEST_CH_NUM] = {
    {
        .channel    = LEDC_HS_RED_CHANNEL,
        .duty       = 0,
        .gpio_num   = LEDC_HS_RED_GPIO,
        .speed_mode = LEDC_HS_MODE,
        .hpoint     = 0,
        .timer_sel  = LEDC_HS_TIMER
    },
    {
        .channel    = LEDC_HS_GREEN_CHANNEL,
        .duty       = 0,
        .gpio_num   = LEDC_HS_GREEN_GPIO,
        .speed_mode = LEDC_HS_MODE,
        .hpoint     = 0,
        .timer_sel  = LEDC_HS_TIMER
    },
    {
        .channel    = LEDC_HS_BLUE_CHANNEL,
        .duty       = 0,
        .gpio_num   = LEDC_HS_BLUE_GPIO,
        .speed_mode = LEDC_HS_MODE,
        .hpoint     = 0,
        .timer_sel  = LEDC_HS_TIMER
    }
};

void pairTagHandler(uint8_t* serial_no);
const rc522_start_args_t pairNFC_args = {
    .miso_io = 25,
    .mosi_io = 23,
    .sck_io = 19,
    .sda_io = 22,
    .callback = &pairTagHandler
};
void tag_handler(uint8_t* serial_no);
const rc522_start_args_t running_args = {
    .miso_io = 25,
    .mosi_io = 23,
    .sck_io = 19,
    .sda_io = 22,
    .callback = &tag_handler
};
mcpwm_config_t pwm_config = {
    .frequency = MTR_PWM_FREQ,
    .cmpr_a = 0,
    .cmpr_b = 0,
    .counter_mode = MCPWM_UP_COUNTER,
    .duty_mode = MCPWM_DUTY_MODE_0
};
bool noPairDect = true;
TaskHandle_t xPinHandle;
TaskHandle_t xKeypadHandle;


void toggle_Lock();
void unpairNFC();

void writePinNVS(uint64_t secret);

unsigned long runTime() {
    return xTaskGetTickCount() * portTICK_PERIOD_MS;
}

void batCheck() {
    TickType_t xLastWakeTime;
    const TickType_t xFrequency = 5000 / portTICK_PERIOD_MS;
    xLastWakeTime = xTaskGetTickCount ();
    uint32_t adc1_gpio35 = 0;
    while(true) {
        vTaskDelayUntil( &xLastWakeTime, xFrequency );
        for(int i = 0; i < MULTISAMPLING; i++) {
            adc1_gpio35 = adc1_get_raw(ADC1_CHANNEL_7);
        }
        adc1_gpio35 /= MULTISAMPLING;
        // Read ADC as mV
        uint32_t rightDampVoltage = esp_adc_cal_raw_to_voltage(adc1_gpio35, &characteristics);
        if (rightDampVoltage < 3000) {
            ledc_set_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel, LEDC_TEST_DUTY);
            ledc_update_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel);
            vTaskDelay(125 / portTICK_PERIOD_MS);
            ledc_set_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel, 0);
            ledc_update_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel);
        }
    }
}

static void IRAM_ATTR gpio_isr_handler(void* arg)
{
    uint32_t gpio_num = (uint32_t) arg;
    xQueueSendFromISR(gpio_evt_queue, &gpio_num, NULL);
}

static void keypadCallback(void* arg)
{
    uint32_t adc1_gpio34 = 0;
    uint32_t io_num;
    uint64_t bufTranslate;
    for(;;) {
        if(xQueueReceive(gpio_evt_queue, &io_num, portMAX_DELAY)) {
            if(keypadLastUpdate != 0 && (keypadLastUpdate + keypadInvalidAfter) > runTime()) {
                //Timeout period reached; reset buffer before input
                memset(keypadInputBuffer, 0, KEYPAD_BUF_SZ);
                keypadInputCnt = 0;
            }
            // Wait 5ms for input to settle
            vTaskDelay(5 / portTICK_PERIOD_MS);
            keypadLastUpdate = runTime();
            adc1_gpio34 = 0;
            for(int i = 0; i < MULTISAMPLING; i++) {
                adc1_gpio34 = adc1_get_raw(ADC1_CHANNEL_6);
            }
            adc1_gpio34 /= MULTISAMPLING;
            for (int i=0; i<12; i++) {
                if(adc1_gpio34 < thresholds[i]) {
                    keypadInputBuffer[keypadInputCnt] = keypad[i];
                }
            }
            if(adc1_gpio34 > thresholds[11]) {
                keypadInputBuffer[keypadInputCnt] = keypad[11];
            }
            keypadInputCnt++;
            // EX: "1#1234*" with "1234" being the pin code would disable the NFC.
            if (keypadInputBuffer[keypadInputCnt - 1] == '*' && keypadInputBuffer[1] == '#') {
                //Menu Mode
                //Check Pin
                    //Cut out * & keypadInputCnt[0] through '#
                    //Double check "-3" and not "-2"
                    int numBytes = sizeof(char) * (keypadInputCnt - 3);
                    char *pinParse = malloc(numBytes);
                    memcpy(pinParse, keypadInputBuffer + 2, numBytes);
                    bufTranslate = atoi(pinParse);
                    free(pinParse);
                if (bufTranslate == keypadPin && keyPinValid) {
                    switch(keypadInputBuffer[0]) {
                        case '1':
                            //NFC Disable
                            nfcEnable = false;
                            break;
                        case '2':
                            //NFC Enable
                            nfcEnable = true;
                            break;
                        case '3':
                            //Unpare All NFC
                            unpairNFC();
                            break;
                        case '4':
                            //Pair NFC Tag
                            rc522_destroy();            //Remove running callback
                            rc522_start(pairNFC_args);  //Attach pairing callback
                            //Setup pair timeout. Reuse keypadLastUpdate because...
                            keypadLastUpdate = runTime();
                            //Wait until pair or NFC timeout
                            while (runTime() < (keypadLastUpdate + keypadInvalidAfter) && noPairDect) {
                                vTaskDelay(250 / portTICK_PERIOD_MS);
                                //Check every 1/4 sec for result
                            }
                            if (noPairDect) {
                                //No tag was detected
                                //Blink Red thrice
                                for (int i = 0; i < 6; i++) {
                                    if ( i % 2) {
                                        ledc_set_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel, 0);
                                    } else {
                                        ledc_set_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel, LEDC_TEST_DUTY);
                                    }
                                    ledc_update_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel);
                                    vTaskDelay(250 / portTICK_PERIOD_MS);
                                }
                            }
                            noPairDect = true;            //Reset for next time.
                            rc522_destroy();            //Remove pairing callback
                            rc522_start(running_args);  //Attach running callback
                            break;
                        case '5':
                            //Change Pin

                            //Clear buffer for new process
                            memset(keypadInputBuffer, 0, KEYPAD_BUF_SZ);
                            keypadInputCnt = 0;
                            //Resume change pin task and suspend this task
                            vTaskResume( xPinHandle );
                            vTaskSuspend( NULL );
                            break;
                        default:
                            //A proper menu code was not selected.
                            //Blink Red thrice
                            for (int i = 0; i < 6; i++) {
                                if ( i % 2) {
                                    ledc_set_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel, 0);
                                } else {
                                    ledc_set_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel, LEDC_TEST_DUTY);
                                }
                                ledc_update_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel);
                                vTaskDelay(250 / portTICK_PERIOD_MS);
                            }
                            break;
                    }
                    //Flash Green
                    ledc_set_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel, LEDC_TEST_DUTY);
                    ledc_update_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel);
                    vTaskDelay(250 / portTICK_PERIOD_MS);
                    ledc_set_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel, 0);
                    ledc_update_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel);
                } else {
                    //Blink Red twice
                    for (int i = 0; i < 4; i++) {
                        if ( i % 2) {
                            ledc_set_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel, 0);
                        } else {
                            ledc_set_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel,
 LEDC_TEST_DUTY);
                        }
                        ledc_update_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel);
                        vTaskDelay(125 / portTICK_PERIOD_MS);
                    }
                }
                memset(keypadInputBuffer, 0, KEYPAD_BUF_SZ);
                keypadInputCnt = 0;
            }
            if (keypadInputBuffer[keypadInputCnt - 1] == '*') {
                //Check Pin
                keypadInputBuffer[keypadInputCnt - 1] = 0;
                bufTranslate = atoi(keypadInputBuffer);
                if (bufTranslate == keypadPin && keyPinValid) {
                    ledc_set_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel,
 LEDC_TEST_DUTY);
                    ledc_update_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel);
                    toggle_Lock();
                    ledc_set_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel, 0);
                    ledc_update_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel);
                } else {
                    //Blink Red twice
                    for (int i = 0; i < 4; i++) {
                        if ( i % 2) {
                            ledc_set_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel, 0);
                        } else {
                            ledc_set_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel,
 LEDC_TEST_DUTY);
                        }
                        ledc_update_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel);
                        vTaskDelay(125 / portTICK_PERIOD_MS);
                    }
                }
                memset(keypadInputBuffer, 0, KEYPAD_BUF_SZ);
                keypadInputCnt = 0;
            }
            if (keypadInputCnt == KEYPAD_BUF_SZ) {
                //buffer limit reached; wrong code or menu option entered.
                memset(keypadInputBuffer, 0, KEYPAD_BUF_SZ);
                keypadInputCnt = 0;
                //Do error message or something
            }
        }
    }
}

static void keypadPINCallback(void* arg)
{
    uint32_t adc1_gpio34 = 0;
    uint32_t io_num;
    uint64_t bufTranslate;
    for(;;) {
        if(xQueueReceive(gpio_evt_queue, &io_num, portMAX_DELAY)) {
            if(keypadLastUpdate != 0 && (keypadLastUpdate + keypadInvalidAfter) > runTime()) {
                //Timeout period reached; reset buffer before input
                memset(keypadInputBuffer, 0, KEYPAD_BUF_SZ);
                keypadInputCnt = 0;
            }
            // Wait 5ms for input to settle
            vTaskDelay(5 / portTICK_PERIOD_MS);
            keypadLastUpdate = runTime();
            adc1_gpio34 = 0;
            for(int i = 0; i < MULTISAMPLING; i++) {
                adc1_gpio34 = adc1_get_raw(ADC1_CHANNEL_6);
            }
            adc1_gpio34 /= MULTISAMPLING;
            for (int i=0; i<12; i++) {
                if(adc1_gpio34 < thresholds[i]) {
                    keypadInputBuffer[keypadInputCnt] = keypad[i];
                }
            }
            if(adc1_gpio34 > thresholds[11]) {
                keypadInputBuffer[keypadInputCnt] = keypad[11];
            }
            keypadInputCnt++;
            if (keypadInputBuffer[keypadInputCnt - 1] == '*') {
                //Check Pin
                keypadInputBuffer[keypadInputCnt - 1] = 0;
                bufTranslate = atoi(keypadInputBuffer);
                //Update pin in RAM
                keypadPin = bufTranslate;
```

```c
                //Update pin in NVS
                writePinNVS(keypadPin);
                //Flash Blue
                ledc_set_duty(ledc_channel[Blue].speed_mode, ledc_channel[Blue].channel, LEDC_TEST_DUTY);
                ledc_update_duty(ledc_channel[Blue].speed_mode, ledc_channel[Blue].channel);
                toggle_Lock();
                ledc_set_duty(ledc_channel[Blue].speed_mode, ledc_channel[Blue].channel, 0);
                ledc_update_duty(ledc_channel[Blue].speed_mode, ledc_channel[Blue].channel);
                //Clear buffer for other process
                memset(keypadInputBuffer, 0, KEYPAD_BUF_SZ);
                keypadInputCnt = 0;
                //Resume other task and suspend self
                vTaskResume( xKeypadHandle );
                vTaskSuspend( NULL );
            }
            if (keypadInputCnt == KEYPAD_BUF_SZ) {
                //buffer limit reached;
                memset(keypadInputBuffer, 0, KEYPAD_BUF_SZ);
                keypadInputCnt = 0;
                //Resume other task and suspend self
                vTaskResume( xKeypadHandle );
                vTaskSuspend( NULL );
            }
        }
    }
}



void initGPIO() {
    //Start ADC setup
        gpio_config_t io_conf;

        //Keypad Input
        io_conf.mode = GPIO_MODE_INPUT;
        io_conf.intr_type = GPIO_INTR_NEGEDGE;
        io_conf.pin_bit_mask = GPIO_KEYPAD;
        io_conf.pull_down_en = 0;
        io_conf.pull_up_en = 0;
        if(gpio_config(&io_conf) != ESP_OK)
            printf("Error setting up GPIO 34 | Keypad Input");

        //Start Battery check ADC
        io_conf.mode = GPIO_MODE_INPUT;
        io_conf.intr_type = GPIO_INTR_NEGEDGE;
        io_conf.pin_bit_mask = GPIO_BATV;
        io_conf.pull_down_en = 0;
        io_conf.pull_up_en = 0;
        if(gpio_config(&io_conf) != ESP_OK)
            printf("Error setting up GPIO 35 | BatV Input");

        adc_power_on();
        adc1_config_width(ADC_WIDTH_BIT_12);
        adc1_config_channel_atten(ADC1_CHANNEL_6,ADC_ATTEN_DB_11);  //and attetntuation | Channel 6 = gpio34 | 11 db = 0 to 3.9v attentuation
        adc1_config_channel_atten(ADC1_CHANNEL_7,ADC_ATTEN_DB_11);  //and attetntuation | Channel 7 = gpio35 | 11 db = 0 to 3.9v attentuation
        esp_adc_cal_characterize(ADC_UNIT_1, ADC_ATTEN_DB_11, ADC_WIDTH_BIT_12, 1101, &characteristics);
        gpio_evt_queue = xQueueCreate(10, sizeof(uint32_t));
        //Create FreeRTOS task here for keypad events. Maybe move to MAIN later.
        xTaskCreate(keypadCallback, "keypadCallback_task", 2048, NULL, 10, &xKeypadHandle);
        //Create FreeRTOS task for changing pin. Immediately suspend.
        xTaskCreate(keypadPINCallback, "keypadPINCallback_task", 2048, NULL, 10, &xPinHandle);
        vTaskSuspend( xPinHandle );
        gpio_install_isr_service(ESP_INTR_FLAG_LEVEL);
        gpio_isr_handler_add(GPIO_KEYPAD_IO, gpio_isr_handler, (void*) GPIO_KEYPAD_IO);
    //End Keypad Setup
    //Start Status RGB Setup
        ledc_timer_config_t ledc_timer = {
            .duty_resolution = LEDC_TIMER_13_BIT,
            .freq_hz = 5000,
            .speed_mode = LEDC_HS_MODE,
            .timer_num = LEDC_HS_TIMER,
            .clk_cfg = LEDC_AUTO_CLK,
        };
        ledc_timer_config(&ledc_timer);
        for (int ch = 0; ch < LEDC_TEST_CH_NUM; ch++) {
            ledc_channel_config(&ledc_channel[ch]);
        }
        //Dance and say hello.
        for (int i = 0; i < 6; i++) {
            if ( i % 2) {
                ledc_set_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel, 0);
            } else {
                ledc_set_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel, LEDC_TEST_DUTY);
            }
            ledc_update_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel);
            vTaskDelay(167 / portTICK_PERIOD_MS);
        }
    //End Status RGB Setup
    //Start Motor Control Setup
        mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0A, MTR_PWM0A_OUT);
        mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0B, MTR_PWM0B_OUT);
        io_conf.mode = GPIO_MODE_OUTPUT;
        io_conf.pin_bit_mask = MTR_Sleep;
        io_conf.pull_down_en = true;
        io_conf.pull_up_en = false;
        if(gpio_config(&io_conf) != ESP_OK)
            printf("Error setting up GPIO 33 | Motor Sleep");
        gpio_set_level(MTR_SleepC, true);
        mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config);
    //End Motor Control Setup
}

void init_NVS() {
    esp_err_t err = nvs_flash_init();
    //Error handling as recommended by example. May be able to remove later.
    if (err == ESP_ERR_NVS_NO_FREE_PAGES || err == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        err = nvs_flash_init();
    }
    ESP_ERROR_CHECK( err );
    nvs_handle_t my_handle;
    err = nvs_open("storage", NVS_READWRITE, &my_handle);
    if (err != ESP_OK) {
        //Error condition if NVS cannot be opened. Turn into LED status.
        printf("Error (%s) opening NVS handle!\n", esp_err_to_name(err));
    } else {
        //NVS handle open, withdraw data.
        uint8_t tmpStoredKey_QTY = 0;
        err = nvs_get_u8(my_handle, "storedKey_QTY", &tmpStoredKey_QTY);
        switch (err) {
            case ESP_OK:
                storedKey_QTY = tmpStoredKey_QTY;
                printf("Done\n");
                break;
            case ESP_ERR_NVS_NOT_FOUND:
                //Must be first time device is being powered up. YAY!!!
                printf("The value is not initialized yet!\n");
                break;
            default :
                //Soda machine broke. Run LED alert and reboot.
                printf("Error (%s) reading!\n", esp_err_to_name(err));
        }
        uint64_t tmpkeypadPin = 0;
        err = nvs_get_u64(my_handle, "keypadPin", &tmpkeypadPin);
        switch (err) {
            case ESP_OK:
                keypadPin = tmpkeypadPin;
                keyPinValid = true;
                printf("Done\n");
                break;
            case ESP_ERR_NVS_NOT_FOUND:
                //Must be first time device is being powered up. YAY!!!
                printf("The value is not initialized yet!\n");
                //Default Key Pin
                keypadPin = defaultKEY;
                keyPinValid = true;
                break;
            default :
                //Soda machine broke. Run LED alert and reboot.
                printf("Error (%s) reading!\n", esp_err_to_name(err));
        }
        nvs_close(my_handle);
    }
}

void writePinNVS(uint64_t secret) {
    esp_err_t err = nvs_flash_init();
    if (err == ESP_ERR_NVS_NO_FREE_PAGES || err == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        err = nvs_flash_init();
    }
    ESP_ERROR_CHECK( err );
    nvs_handle_t place_handle;
    err = nvs_open("storage", NVS_READWRITE, &place_handle);
    if (err != ESP_OK) {
        //Error condition if NVS cannot be opened. Turn into LED status.
        printf("Error (%s) opening NVS handle!\n", esp_err_to_name(err));
    } else {
        err = nvs_set_u64(place_handle, "keypadPin", secret);
        err = nvs_commit(place_handle);
        printf((err != ESP_OK) ? "Failed!\n" : "Done\n");
        nvs_close(place_handle);
    }
}

void grabKeyList(uint8_t *buf, int count) {
    esp_err_t err = nvs_flash_init();
    //Error handling as recommended by example. May be able to remove later.
    if (err == ESP_ERR_NVS_NO_FREE_PAGES || err == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        err = nvs_flash_init();
    }
    ESP_ERROR_CHECK( err );
    nvs_handle_t my_handle;
    err = nvs_open("storage", NVS_READWRITE, &my_handle);
    if (err != ESP_OK) {
        //Error condition if NVS cannot be opened. Turn into LED status.
        printf("Error (%s) opening NVS handle!\n", esp_err_to_name(err));
    } else {
        uint8_t tmp_key[serialLen];
        size_t size = sizeof(tmp_key);
        char* tmpNvsKey;
        tmpNvsKey = "NFCKEY";
        strcat(tmpNvsKey, (char*)count);
        err = nvs_get_blob(my_handle, tmpNvsKey, tmp_key, &size);
        switch (err) {
            case ESP_OK:
                *buf = *tmp_key; //double check this line, thx
                printf("Grabbed Key: %u\n",count);
                break;
            case ESP_ERR_NVS_NOT_FOUND:
                //Stored keys do not equal qty reported. Bad.
                *buf = 0;
                printf("The key %u is not initialized yet!\n",count);
                break;
            default :
                //Soda machine broke. Run LED alert and reboot.
                *buf = 0;
                printf("Error (%s) reading!\n", esp_err_to_name(err));
        }
        tmpNvsKey[0] = ’\0’;
        if (err != ESP_OK) {
            printf("Error (%s) grabbing key!\n", esp_err_to_name(err));
        }
        nvs_close(my_handle);
    }
}

void writeNFCKey(uint8_t *buf, int count) {
    esp_err_t err = nvs_flash_init();
    if (err == ESP_ERR_NVS_NO_FREE_PAGES || err == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        err = nvs_flash_init();
    }
    ESP_ERROR_CHECK( err );
    nvs_handle_t blobHandle;
    err = nvs_open("storage", NVS_READWRITE, &blobHandle);
    if (err != ESP_OK) {
        //Error condition if NVS cannot be opened. Turn into LED status.
        printf("Error (%s) opening NVS handle!\n", esp_err_to_name(err));
    } else {
        uint8_t tmp_key[serialLen];
        //Replace line below with known size
        size_t size = sizeof(tmp_key);
        char* tmpNvsKey;
        tmpNvsKey = "NFCKEY";
        strcat(tmpNvsKey, (char*)count);
        err = nvs_set_blob(blobHandle, tmpNvsKey, buf, size);
        err = nvs_commit(blobHandle);
        switch (err) {
            case ESP_OK:
                //Successful Write
                printf("Wrote Key: %u\n",count);
                break;
            default :
                //Soda machine broke. Run LED alert and reboot.
                printf("Error (%s) Writing!\n", esp_err_to_name(err));
        }
        nvs_close(blobHandle);
    }
}

void unpairNFC() {
    //First, free from heap
    for (int i=0; i<storedKey_QTY-1; i++) {
        free(keyList[i]);
    }
    esp_err_t err = nvs_flash_init();
    if (err == ESP_ERR_NVS_NO_FREE_PAGES || err == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        err = nvs_flash_init();
    }
    ESP_ERROR_CHECK( err );
    nvs_handle_t unpairHandle;
    err = nvs_open("storage", NVS_READWRITE, &unpairHandle);
    if (err != ESP_OK) {
        //Error condition if NVS cannot be opened. Turn into LED status.
        printf("Error (%s) opening NVS handle!\n", esp_err_to_name(err));
    } else {
        //Replace lines below with known size
        char* tmpNvsKey;
        for (int i=0; i<storedKey_QTY-1; i++) {
            tmpNvsKey = "NFCKEY";
            strcat(tmpNvsKey, (char*)i);
            nvs_erase_key(unpairHandle, tmpNvsKey);
        }
        err = nvs_commit(unpairHandle);
        switch (err) {
            case ESP_OK:
                //Successful Write
                printf("Successful Clear: \n");
                break;
            default :
                //Soda machine broke. Run LED alert and reboot.
                printf("Error (%s) Writing!\n", esp_err_to_name(err));
        }
        nvs_close(unpairHandle);
    }
    storedKey_QTY = 0;
}

void writeKeyQtyNVS(uint8_t qty) {
    esp_err_t err = nvs_flash_init();
    if (err == ESP_ERR_NVS_NO_FREE_PAGES || err == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        err = nvs_flash_init();
    }
    ESP_ERROR_CHECK( err );
    nvs_handle_t placeqtyHand;
    err = nvs_open("storage", NVS_READWRITE, &placeqtyHand);
    if (err != ESP_OK) {
        //Error condition if NVS cannot be opened. Turn into LED status.
        printf("Error (%s) opening NVS handle!\n", esp_err_to_name(err));
    } else {
        err = nvs_set_u8(placeqtyHand, "storedKey_QTY", qty);
        err = nvs_commit(placeqtyHand);
        printf((err != ESP_OK) ? "Failed!\n" : "Done\n");
        nvs_close(placeqtyHand);
    }
}

void tag_handler(uint8_t* serial_no) {
    bool thisIsNotTheKeyYouSeek = true;
    //Check every stored NFC key
    for(int i = 0; i < storedKey_QTY-1; i++) {
        if (serial_no == keyList[i]) {
            ledc_set_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel, LEDC_TEST_DUTY);
            ledc_update_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel);
            toggle_Lock();
            ledc_set_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel, 0);
            ledc_update_duty(ledc_channel[GREEN].speed_mode, ledc_channel[GREEN].channel);
            thisIsNotTheKeyYouSeek = false;
        }
    }
    if (thisIsNotTheKeyYouSeek) {
        //Blink Red thrice
        for (int i = 0; i < 6; i++) {
            if ( i % 2) {
                ledc_set_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel, 0);
            } else {
                ledc_set_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel, LEDC_TEST_DUTY);
            }
            ledc_update_duty(ledc_channel[RED].speed_mode, ledc_channel[RED].channel);
            vTaskDelay(250 / portTICK_PERIOD_MS);
        }
    }
}

void pairTagHandler(uint8_t* serial_no) {
    //Allocate heap and update
    keyList[storedKey_QTY] = (uint8_t *)malloc(serialLen * sizeof(uint8_t));
    keyList[storedKey_QTY] = serial_no;
    //Update NVS
    writeNFCKey(keyList[storedKey_QTY], storedKey_QTY);
    storedKey_QTY++;
    writeKeyQtyNVS(storedKey_QTY);
    noPairDect = false;
}

void toggle_Lock() {
    gpio_set_level(MTR_SleepC, false);
    vTaskDelay(5 / portTICK_PERIOD_MS);
    if (mtrForward) {
        mcpwm_set_signal_low(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_B);
        mcpwm_set_duty(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_A, MTR_duty);
        mcpwm_set_duty_type(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_A, MCPWM_DUTY_MODE_0);
        mtrForward = false;
    } else {
        mcpwm_set_signal_low(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_A);
        mcpwm_set_duty(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_B, MTR_duty);
        mcpwm_set_duty_type(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_B, MCPWM_DUTY_MODE_0);
        mtrForward = true;
    }
    vTaskDelay(MTR_PERIOD / portTICK_PERIOD_MS);
    mcpwm_set_signal_low(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_A);
    mcpwm_set_signal_low(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_B);
    vTaskDelay(5 / portTICK_PERIOD_MS);
    gpio_set_level(MTR_SleepC, true);
}

void app_main(void) {
    // Runs rc522_init or handles esp32 error.
    // rc522_init runs startup, tests, and starts "rc522_task";
    // Maybe adjust "rc522_task" to run on FreeRTOS HW interupt and not timer.
    init_NVS();
    if(storedKey_QTY != 0) {
        //Dynamicly cast keylist into heap. Total possible keys: 2^8 = 256
        for (int i=0; i<storedKey_QTY-1; i++) {
            keyList[i] = (uint8_t *)malloc(serialLen * sizeof(uint8_t));
            grabKeyList(keyList[i],i);
        }
    }
    initGPIO();
    rc522_start(running_args);
    xTaskCreate(batCheck, "Battery_Check_Task", 1024, NULL, 3, NULL);
}
```

Index