# File: GarageRTC.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File: GarageRTC.h

\* Description: Implements a real time IoT Garage Door controller. See

\* https://github.com/jharmer95/Garage-RTC/ for details on the

\* Open GarageRTC project.

\* Authors: Daniel Zajac, danzajac@umich.edu

\* Jackson Harmer, jharmer@umich.edu

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef GARAGERTC\_H\_INCLUDE

#define GARAGERTC\_H\_INCLUDE

#if CONFIG\_FREERTOS\_UNICORE

# define ARDUINO\_RUNNING\_CORE 0

#else

# define ARDUINO\_RUNNING\_CORE 1

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Include libraries and references

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Library: LiquidCrystal\_PCF8574.h

\* Author: Matthias Hertel www.mathertel.de

\* Source: http://www.mathertel.de/Arduino/LiquidCrystal\_PCF8574.aspx

\* Version: 1.1.0

\* Description: A library for driving LiquidCrystal displays (LCD) by using the

\* I2C bus and an PCF8574 I2C adapter. This library is derived

\* from the original Arduino LiquidCrystal library and uses the

\* original Wire library for communication.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <LiquidCrystal\_PCF8574.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Library: WiFi.h

\* Author: Hristo Gochkov <hristo@espressif.com>

\* Source: https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi/src

\* Version: 1.0

\* Description: With this library you can instantiate Servers, Clients and

\* send/receive UDP packets through WiFi. The shield can connect

\* either to open or encrypted networks (WEP, WPA). The IP

\* address can be assigned statically or through a DHCP. The

\* library can also manage DNS.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "WiFi.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Library: AsyncUDP.h

\* Author: Me-No-Dev

\* Source: https://github.com/me-no-dev/ESPAsyncUDP

\* Version: 1.0.0

\* Description: Async UDP Library for ESP32

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "AsyncUDP.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Library: esp\_system.h

\* Author: Hristo Gochkov, Ivan Grokhtkov <hristo@espressif.com>

\* Source: https://github.com/espressif/arduino-esp32

\* Version: 1.0.2

\* Description: ESP32 Board Support, includes FreeRTOS for ESP32. Reference

\* below.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "esp\_system.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Library: FreeRTOS Copyright (C) 2015

\* Author: Real Time Engineers Ltd.

\* Source: http://www.FreeRTOS.org

\* Version: FreeRTOS V8.2.0

\* Description: FreeRTOS provides completely free yet professionally developed,

\* robust, strictly quality controlled, supported, and cross

\* platform software that is more than just the market leader, it

\* s the industry's de facto standard.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Library: WIFI\_AP.h

\* Author: Daniel Zajac, Jackson Harmer

\* Source: https://github.com/jharmer95/Garage-RTC/

\* Version: 1.0.0

\* Description: Contains WIFI credentials

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "WIFI\_AP.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Defines to make addressing Pins easier

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Defines to make the code more readable

#define PIN\_TEMP A7

#define PIN\_CO A6

#define PIN\_CO\_DIG 13

#define BUTTON\_ALARM 5

#define BUTTON\_DOOR 4

#define BUTTON\_STOP 0

#define BUTTON\_LIGHT 15

#define LIMSW\_UP 27

#define LIMSW\_DOWN 12

#define LIMSW\_OBS 14

#define RELAY\_DOOR 32

#define RELAY\_ALARM 33

#define RELAY\_LIGHT 25

#define RELAY\_AUX 26

// Flagging bits for timing or other debug

#define DEBUG\_T1 23

#define DEBUG\_T2 19

#define DEBUG\_T3 18

#define DEBUG\_T4 2

#define DP\_OPEN 0

#define DP\_CLOSE 1

#define DP\_STOP 2

#define DP\_MOVE 3

int outputPins**[]** **=** **{** RELAY\_DOOR**,** RELAY\_ALARM**,** RELAY\_LIGHT**,** RELAY\_AUX**,** DEBUG\_T1**,** DEBUG\_T2**,** DEBUG\_T3**,** DEBUG\_T4 **};**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Static system parameters

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define LOW\_PRI\_TASK\_DELAY 1000 // Low priority loop time in mS

#define MED\_PRI\_TASK\_DELAY 100 // Medium Priority Task Loop in mS

#define HIGH\_PRI\_TASK\_DELAY 10 // High Prioirty Task Loop in mS

#define CO\_WARMUP 90000 // worst case CO sensor warm-up time

#define DEBOUNCEMS 10 // debounce time in milliseconds

#define MAXSWS 7 // Max number of switches in the system

#define MAXIO 8 // Max number of IO pins

#define BTNHOLDMIL 1000 // Hold time for the garage door button

#define HIGHTEMP 95 // High temperature alarm limit

#define LOWTEMP 50 // Low Temperature alarm limit

#define TEMPHYST 5 // Temp Hysteresis

#define HIGHCO 20 // High CO Limit

#define MEDCO 10 // High CO Limit

#define COHYST 5 // CO Hysteresis

#define SERIALSPEED 115200 // Serial Baud Rate

#define WDTIMEOUT 5000 // Watch Dog Timeout

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function prototypes

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Tasks

void TaskReadSensors**(**void**\*** pvParameters**);**

void TaskUpdateDisplay**(**void**\*** pvParameters**);**

void TaskPriorityMachines**(**void**\*** pvParameters**);**

void TaskNetwork**(**void**\*** pvParameters**);**

void TaskWatchdog**(**void**\*** pvParameters**);**

// general

void initWatchdog**();**

void debounce**(**int pinIndex**,** bool bouncing**[],** int buttonState**[],** int stopTime**[]);**

void initNetwork**(**AsyncUDP**&** udp**);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Global Variables

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// custom char since '\' does not exist on the display

static byte customBackslash**[**8**]** **=** **{** 0b00000**,** 0b10000**,** 0b01000**,** 0b00100**,** 0b00010**,** 0b00001**,** 0b00000**,** 0b00000 **};**

static int g\_switches**[**MAXSWS**]** **=** **{** BUTTON\_ALARM**,** BUTTON\_DOOR**,** BUTTON\_STOP**,** BUTTON\_LIGHT**,** LIMSW\_UP**,** LIMSW\_DOWN**,** LIMSW\_OBS **};**

// Needs to match order above.

enum SWITCH\_INDEX

**{**

ALARM**,**

DOOR**,**

STOP**,**

LIGHT**,**

UP**,**

DOWN**,**

OBS

**};**

byte g\_WatchDogBowl **=** 0**;** // Watchdog Feed bowl

portMUX\_TYPE g\_wdMutex**;** // Mux to protect the watchdog bowl

portMUX\_TYPE g\_sharedMemMutex**;** // mutex to protect the shared globals

portMUX\_TYPE g\_serialMutex**;** // Mux to protect the serial device

byte g\_buttonState**[**MAXSWS**]** **=** **{** 1**,** 1**,** 1**,** 1**,** 1**,** 1**,** 1 **};**

bool g\_alarmState **=** **false;**

bool g\_doorMovingState **=** **false;**

bool g\_lightOnState **=** **false;**

bool g\_Connected **=** **false;**

byte g\_doorPosition **=** 0**;**

// Shared Variables

float g\_temp **=** 23.4**;**

float g\_co **=** 0.0**;**

byte g\_webCmd **=** 0**;**

byte g\_coState **=** 0**;**

// converting the g\_coState to a string

const char**\*** g\_coStateStr**[**3**]** **=** **{** "LOW"**,** "WARN"**,** "HIGH" **};**

bool g\_firstRun **=** **true;** // Indicating the system is just starting up

bool g\_heating **=** **true;** // Indicates the CO sensor is still heating

#endif

# File: GarageRTC.INO

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

File: GarageRTC.ino

Description: Implements a real time IoT Garage Door controller. See

https://github.com/jharmer95/Garage-RTC/ for details on the

Open GarageRTC project.

Authors: Daniel Zajac, danzajac@umich.edu

Jackson Harmer, jharmer@umich.edu

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "GarageRTC.h"

// configure global for the LCD port

LiquidCrystal\_PCF8574 lcd**(**0x27**);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: void setup( void )

Description: This function is run once at startup after reset. It sets up

hardware and initializes the system.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setup**()**

**{**

// initialize serial communication

Serial**.**begin**(**SERIALSPEED**);**

// Setup the serial mutex

g\_serialMutex **=** portMUX\_INITIALIZER\_UNLOCKED**;**

// initalize digital output pins

**for** **(**int i **=** 0**;** i **<** MAXIO**;** **++**i**)**

**{**

digitalWrite**(**outputPins**[**i**],** HIGH**);** // set the pins for inital pullup

pinMode**(**outputPins**[**i**],** OUTPUT**);** // Assign them the output state

**}**

// initalize digital input pins

**for** **(**int i **=** 0**;** i **<** MAXSWS**;** **++**i**)**

**{**

pinMode**(**g\_switches**[**i**],** INPUT\_PULLUP**);** // Setup as input with pull up

**}**

// setup the display

lcd**.**begin**(**20**,** 4**);** // setup the LCD dims

lcd**.**setBacklight**(**128**);** // set the backlighting

lcd**.**home**();** // send the cursor home

lcd**.**clear**();** // clear the display

// create a new character, display missing backslash

lcd**.**createChar**(**1**,** customBackslash**);**

// set the static display elements

lcd**.**setCursor**(**0**,** 0**);**

lcd**.**print**(**"T: NET: "**);**

lcd**.**setCursor**(**0**,** 1**);**

lcd**.**print**(**"C: SYSTEM: "**);**

lcd**.**setCursor**(**0**,** 2**);**

lcd**.**print**(**" LIGHT: "**);**

lcd**.**setCursor**(**0**,** 3**);**

lcd**.**print**(**" DOOR: "**);**

// Share Memory mutex

g\_sharedMemMutex **=** portMUX\_INITIALIZER\_UNLOCKED**;**

// g\_WatchDogBowl mutex

g\_wdMutex **=** portMUX\_INITIALIZER\_UNLOCKED**;**

xTaskCreatePinnedToCore**(**TaskReadSensors**,** "TaskReadSensors"**,** 1024**,** **NULL,** 3**,** // Priority 3 is the highest

**NULL,** ARDUINO\_RUNNING\_CORE**);**

xTaskCreatePinnedToCore**(**TaskUpdateDisplay**,** "TaskUpdateDisplay"**,** 2048**,** // Stack size

**NULL,** 1**,** // Priority

**NULL,** ARDUINO\_RUNNING\_CORE**);**

xTaskCreatePinnedToCore**(**TaskPriorityMachines**,** "TaskPriorityMachines"**,** 1024**,** // Stack size

**NULL,** 2**,** // Priority

**NULL,** ARDUINO\_RUNNING\_CORE**);**

xTaskCreatePinnedToCore**(**TaskNetwork**,** "TaskNetwork"**,** 4096**,** // Stack size

**NULL,** 1**,** // Priority

**NULL,** ARDUINO\_RUNNING\_CORE**);**

xTaskCreatePinnedToCore**(**TaskWatchdog**,** "TaskWatchdog"**,** 1024**,** // Stack size

**NULL,** 1**,** // Priority

**NULL,** ARDUINO\_RUNNING\_CORE**);**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: void loop( void )

Description: This function is run during idle cycles but does no useful

work since everything is done in tasks.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void loop**()**

**{**

// Empty. Things are done in Tasks.

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: void initNetwork( AsyncUDP &udp )

AsyncUDP &udp - handle to the UDP connection

Description: This function is run once at startup after reset. It sets up

wifi network.

Referecnes: Modifed from example code provided with the WiFi Library.

See GarageRTC.h for the complete reference.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void initNetwork**(**AsyncUDP**&** udp**)**

**{**

WiFi**.**mode**(**WIFI\_STA**);** // Setup the wifi Mode

WiFi**.**begin**(**g\_ssid**,** g\_password**);** // Setup the SSID and Password

bool connected **=** **false;** // local for connected

// Wait for aconnection

**if** **(**WiFi**.**waitForConnectResult**()** **!=** WL\_CONNECTED**)**

**{**

// if we do not get a connection, let the serial know

taskENTER\_CRITICAL**(&**g\_serialMutex**);**

Serial**.**println**(**"WiFi Did not connect"**);**

taskEXIT\_CRITICAL**(&**g\_serialMutex**);**

connected **=** **false;**

**}**

**else**

**{**

// otherwise we connected

connected **=** **true;**

// setup the port to listen to

**if** **(**udp**.**listen**(**1234**))**

**{**

// print out the IP we obtained.

taskENTER\_CRITICAL**(&**g\_serialMutex**);**

Serial**.**print**(**"UDP Listening on IP: "**);**

Serial**.**println**(**WiFi**.**localIP**());**

taskEXIT\_CRITICAL**(&**g\_serialMutex**);**

**}**

**}**

// write the connection status to the shared variable

taskENTER\_CRITICAL**(&**g\_sharedMemMutex**);**

g\_Connected **=** connected**;**

taskEXIT\_CRITICAL**(&**g\_sharedMemMutex**);**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: void TaskReadSensors( void\* pvParameters )

void\* pvParameters - Paramaters passed in as part of the

scheduler configuration for this task.

Description: Sensor Processing Tasks - This task is responsible for fetching,

converting values from the sensors and making the results

available other tasks. In general, this task will perform the

following functions:

• Fetch values from sensors

• Scale/Convert values

• Read/debounce switches

• Store values to memory

• Update shared variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TaskReadSensors**(**void**\*** pvParameters**)**

**{**

**(**void**)**pvParameters**;**

TickType\_t xLastWakeTime**;**

bool bouncing**[**MAXSWS**];**

byte buttonState**[**MAXSWS**];**

byte stopTime**[**MAXSWS**];**

**for** **(**int i **=** 0**;** i **<** MAXSWS**;** i**++)**

bouncing**[**i**]** **=** **false;**

**while** **(true)**

**{**

// Macro that expands to schedule the next wake-up time and then

// sleep the task until the next schedule wake-up.

vTaskDelayUntil**(&**xLastWakeTime**,** 10**);**

// read Analogs - this is the only task that reads so no need to

// provide exclusive access.

int sensorValueT **=** analogRead**(**PIN\_TEMP**);**

int sensorValueCO **=** analogRead**(**PIN\_CO**);**

// read digitals:

**for** **(**int i **=** 0**;** i **<** MAXSWS**;** **++**i**)**

**{**

debounce**(**i**,** bouncing**,** buttonState**,** stopTime**);**

**}**

// print out the value you read:

float temp\_buf **=** 0.0637 **\*** sensorValueT **-** 40.116**;**

float co\_buf **=** 0.0527 **\*** sensorValueCO **-** 72.728**;**

// write back variables to shared globals. Protect with

// mutex to avoid race condition with another thread

taskENTER\_CRITICAL**(&**g\_sharedMemMutex**);**

g\_temp **=** temp\_buf**;**

g\_co **=** co\_buf**;**

**for** **(**int i **=** 0**;** i **<** MAXSWS**;** i**++)**

g\_buttonState**[**i**]** **=** buttonState**[**i**];**

taskEXIT\_CRITICAL**(&**g\_sharedMemMutex**);**

// set my bit in the watchdog

taskENTER\_CRITICAL**(&**g\_wdMutex**);**

g\_WatchDogBowl **=** g\_WatchDogBowl **|** 0x1**;**

taskEXIT\_CRITICAL**(&**g\_wdMutex**);**

**}**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: void debounce( int pinIndex )

int pinIndex - The index of the pin to debounce.

bool bouncing[] - Array of bools indicating if that switch is

currently debouncing.

int buttonState[] - Array of current button states.

int stopTime[] - array of times to stop debouncing.

Description: Debounces the pin for DEBOUNCEMS milliseconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void debounce**(**int pinIndex**,** bool bouncing**[],** byte buttonState**[],** byte stopTime**[])**

**{**

// only read if we are in a debounce

**if** **(!**bouncing**[**pinIndex**])**

**{**

int current **=** digitalRead**(**g\_switches**[**pinIndex**]);**

// have we have seen an edge?

**if** **(**buttonState**[**pinIndex**]** **!=** current**)**

**{**

//startTimer and signal debouncing

stopTime**[**pinIndex**]** **=** esp\_log\_timestamp**()** **+** DEBOUNCEMS**;**

bouncing**[**pinIndex**]** **=** **true;**

**}**

**}**

**if** **(**bouncing **&&** **(**esp\_log\_timestamp**()** **>** stopTime**[**pinIndex**]))** // see if we are past the bounce window

**{**

//save the value

buttonState**[**pinIndex**]** **=** digitalRead**(**g\_switches**[**pinIndex**]);**

//reset the bouncing

bouncing**[**pinIndex**]** **=** **false;**

**}**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: void TaskUpdateDisplay( void\* pvParameters )

void\* pvParameters - Paramaters passed in as part of the

scheduler configuration for this task.

Description: LCD Display Task - This task is responsible for updating the

local display. In general, this task will perform the

following functions:

• Fetch values from memory

• Post readings or status to the display

• Update the displays including:

o Temperature

o CO Level

o Connection Status

o System state

Note: The LCD selected is very slow to update so we do not

redraw the entire display each cycle, only updating the chars

that have changed.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TaskUpdateDisplay**(**void**\*** pvParameters**)**

**{**

**(**void**)**pvParameters**;**

TickType\_t xLastWakeTime**;**

char TEMPmsg**[**6**];**

char DOORmsg**[**5**];**

int twiddle **=** 0**;** // the running indicator on the display

// local buffers for shared memory

float temp**;**

float co**;**

bool doorMovingState**;**

bool lightOnState**;**

byte upButtonState**;**

byte downButtonState**;**

bool alarmState**;**

bool connected**;**

byte doorPosition**;**

byte coState**;**

**while** **(true)**

**{**

// Macro that expands to schedule the next wake-up time and then

// sleep the task until the next schedule wake-up.

vTaskDelayUntil**(&**xLastWakeTime**,** 500**);**

// buffer in the shared memory

taskENTER\_CRITICAL**(&**g\_sharedMemMutex**);**

temp **=** g\_temp**;**

doorPosition **=** g\_doorPosition**;**

lightOnState **=** g\_lightOnState**;**

upButtonState **=** g\_buttonState**[**UP**];**

downButtonState **=** g\_buttonState**[**DOWN**];**

alarmState **=** g\_alarmState**;**

connected **=** g\_Connected**;**

doorMovingState **=** g\_doorMovingState**;**

taskEXIT\_CRITICAL**(&**g\_sharedMemMutex**);**

lcd**.**setCursor**(**0**,** 3**);**

**switch** **(**twiddle**)**

**{**

**case** 0**:**

lcd**.**write**(**'|'**);**

**break;**

**case** 1**:**

lcd**.**write**(**'/'**);**

**break;**

**case** 2**:**

lcd**.**write**(**'-'**);**

**break;**

**case** 3**:**

lcd**.**write**(**byte**(**1**));**

**break;**

**}**

twiddle **=** **(**twiddle **+** 1**)** **%** 4**;**

// Create temprate string from float

dtostrf**(**temp**,** 5**,** 1**,** TEMPmsg**);**

lcd**.**setCursor**(**2**,** 0**);**

lcd**.**print**(**TEMPmsg**);**

// update the CO content

**if** **(**co **>** HIGHCO**)**

**{**

lcd**.**setCursor**(**2**,** 1**);**

lcd**.**print**(**"HIGH"**);**

coState **=** 2**;**

**}**

**else** **if** **(**co **>** MEDCO**)**

**{**

lcd**.**setCursor**(**2**,** 1**);**

lcd**.**print**(**"WARN"**);**

coState **=** 1**;**

**}**

**else**

**{**

lcd**.**setCursor**(**2**,** 1**);**

lcd**.**print**(**"LOW "**);**

coState **=** 0**;**

**}**

// Update the door Position

**if** **(**doorMovingState**)**

**{**

lcd**.**setCursor**(**15**,** 3**);**

lcd**.**print**(**"MOVE "**);**

doorPosition **=** DP\_MOVE**;**

**}**

**else** **if** **(**upButtonState **==** LOW**)**

**{**

lcd**.**setCursor**(**15**,** 3**);**

lcd**.**print**(**"OPEN "**);**

doorPosition **=** DP\_OPEN**;**

**}**

**else** **if** **(**downButtonState **==** LOW**)**

**{**

lcd**.**setCursor**(**15**,** 3**);**

lcd**.**print**(**"CLOSE"**);**

doorPosition **=** DP\_CLOSE**;**

**}**

**else**

**{**

lcd**.**setCursor**(**15**,** 3**);**

lcd**.**print**(**"STOP "**);**

doorPosition **=** DP\_STOP**;**

**}**

// Update the light state

**if** **(**lightOnState**)**

**{**

lcd**.**setCursor**(**15**,** 2**);**

lcd**.**print**(**"ON "**);**

**}**

**else**

**{**

lcd**.**setCursor**(**15**,** 2**);**

lcd**.**print**(**"OFF"**);**

**}**

// update the alarm state

**if** **(**alarmState**)**

**{**

lcd**.**setCursor**(**15**,** 1**);**

lcd**.**print**(**"ALARM"**);**

**}**

**else**

**{**

lcd**.**setCursor**(**15**,** 1**);**

lcd**.**print**(**"OK "**);**

**}**

// update the network state

**if** **(**connected**)**

**{**

lcd**.**setCursor**(**15**,** 0**);**

lcd**.**print**(**"OK "**);**

**}**

**else**

**{**

lcd**.**setCursor**(**15**,** 1**);**

lcd**.**print**(**"NONE"**);**

**}**

//write back global door state and co state for web

taskENTER\_CRITICAL**(&**g\_sharedMemMutex**);**

g\_doorPosition **=** doorPosition**;**

g\_coState **=** coState**;**

taskEXIT\_CRITICAL**(&**g\_sharedMemMutex**);**

// set my bit in the watchdog

taskENTER\_CRITICAL**(&**g\_wdMutex**);**

g\_coState **=** coState**;**

g\_WatchDogBowl **=** g\_WatchDogBowl **|** 0x2**;**

taskEXIT\_CRITICAL**(&**g\_wdMutex**);**

**}**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: void TaskNetwork( void\* pvParameters )

void\* pvParameters - Paramaters passed in as part of the

scheduler configuration for this task.

Description: Handles network communication tasks

Referecnes: Modifed from example code provided with the esp32 Library.

See GarageRTC.h for the complete reference.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TaskNetwork**(**void**\*** pvParameters**)**

**{**

**(**void**)**pvParameters**;**

t TickType\_t xLastWakeTime**;**

char wifiBuff**[**255**]** **=** ""**;**

char tempBuff**[**6**];**

char coBuff**[**6**];**

char doorPosStr**[**6**];**

AsyncUDP udp**;**

// local buffers for shared memory

float temp**;**

float co**;**

bool lightOnState**;**

bool connected**;**

int doorPosition**;**

bool alarmState**;**

byte coState**;**

xLastWakeTime **=** xTaskGetTickCount**();**

**while** **(true)**

**{**

// Macro that expands to schedule the next wake-up time and then

// sleep the task until the next schedule wake-up.

vTaskDelayUntil**(&**xLastWakeTime**,** 500**);**

// buffer in the shared memory

taskENTER\_CRITICAL**(&**g\_sharedMemMutex**);**

temp **=** g\_temp**;**

co **=** g\_co**;**

lightOnState **=** g\_lightOnState**;**

doorPosition **=** g\_doorPosition**;**

connected **=** g\_Connected**;**

alarmState **=** g\_alarmState**;**

coState **=** g\_coState**;**

taskEXIT\_CRITICAL**(&**g\_sharedMemMutex**);**

**if** **(!**connected**)**

initNetwork**(**udp**);**

**if** **(**connected**)**

**{**

// have to load via dtosttrf due to sprintf bug

dtostrf**(**temp**,** 5**,** 1**,** tempBuff**);**

dtostrf**(**co**,** 5**,** 1**,** coBuff**);**

sprintf**(**wifiBuff**,**

"[{\"name\": \"alarmStatus\", \"value\": \"%s\"}, {\"name\": \"doorStatus\", \"value\": \"%d\"}, {\"name\": "

"\"lightStatus\", \"value\": \"%s\"}, {\"name\": \"tempStatus\", \"value\": \"%s\"}, {\"name\": \"coStatus\", \"value\": "

"\"%s\"}]"**,**

alarmState **?** "True" **:** "False"**,** doorPosition**,** lightOnState **?** "ON" **:** "OFF"**,** tempBuff**,** g\_coStateStr**[**coState**]);**

udp**.**broadcast**(**wifiBuff**);**

**}**

udp**.**onPacket**([](**AsyncUDPPacket packet**)** **{**

**if** **(**packet**.**length**()** **==** 7**)**

**{**

byte value **=** **(**packet**.**data**()[**5**]);**

#ifdef DEBBUG

Serial**.**print**(**"value = "**);**

Serial**.**print**(**value**,** HEX**);**

Serial**.**print**(**"\n"**);**

#endif

taskENTER\_CRITICAL**(&**g\_sharedMemMutex**);**

g\_webCmd **=** value**;**

taskEXIT\_CRITICAL**(&**g\_sharedMemMutex**);**

**}**

#ifdef DEBBUG

Serial**.**print**(**"UDP Packet Type: "**);**

Serial**.**print**(**packet**.**isBroadcast**()** **?** "Broadcast" **:** packet**.**isMulticast**()** **?** "Multicast" **:** "Unicast"**);**

Serial**.**print**(**", From: "**);**

Serial**.**print**(**packet**.**remoteIP**());**

Serial**.**print**(**":"**);**

Serial**.**print**(**packet**.**remotePort**());**

Serial**.**print**(**", To: "**);**

Serial**.**print**(**packet**.**localIP**());**

Serial**.**print**(**":"**);**

Serial**.**print**(**packet**.**localPort**());**

Serial**.**print**(**", Length: "**);**

Serial**.**print**(**packet**.**length**());**

Serial**.**print**(**", Data: "**);**

Serial**.**write**(**packet**.**data**(),** packet**.**length**());**

Serial**.**println**();**

//reply to the client

packet**.**printf**(**"Got %u bytes of data"**,** packet**.**length**());**

#endif

**});**

// set my bit in the watchdog

taskENTER\_CRITICAL**(&**g\_wdMutex**);**

g\_WatchDogBowl **=** g\_WatchDogBowl **|** 0x4**;**

taskEXIT\_CRITICAL**(&**g\_wdMutex**);**

**}**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: void TaskPriorityMachines( void\* pvParameters )

void\* pvParameters - Paramaters passed in as part of the

scheduler configuration for this task.

Description: This task is responsible monitoring door position, obstacle

detection, and starting or stopping movement. In general,

this task will perform the following functions:

• Maintain Door state

• Maintain direction state

• Start/Stop door movement

• Monitor obstacle detection

Referecnes: Modifed from example code provided with the WiFi Library.

See GarageRTC.h for the complete reference.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TaskPriorityMachines**(**void**\*** pvParameters**)**

**{**

**(**void**)**pvParameters**;**

TickType\_t xLastWakeTime**;**

// state machine state vars

byte lightState **=** 0**;**

byte doorState **=** 0**;**

byte alarmState **=** 0**;**

int buttonHoldTime**;**

xLastWakeTime **=** xTaskGetTickCount**();**

// local buffers for global variables

float temp**;**

float co**;**

bool doorMovingState**;**

byte lightButtonState**;**

byte alarmButtonState**;**

byte doorButtonState**;**

byte upButtonState**;**

byte downButtonState**;**

byte stopButtonState**;**

byte obsButtonState**;**

bool lightOnState**;**

**while** **(true)**

**{**

// Macro that expands to schedule the next wake-up time and then

// sleep the task until the next schedule wake-up.

vTaskDelayUntil**(&**xLastWakeTime**,** MED\_PRI\_TASK\_DELAY**);**

// buffer in the shared memory

taskENTER\_CRITICAL**(&**g\_sharedMemMutex**);**

temp **=** g\_temp**;**

co **=** g\_co**;**

lightButtonState **=** g\_buttonState**[**LIGHT**]** **&** **~(**g\_webCmd **>>** 2**);**

alarmButtonState **=** g\_buttonState**[**ALARM**]** **&** **~(**g\_webCmd **&** 0x1**);**

doorButtonState **=** g\_buttonState**[**DOOR**]** **&** **~(**g\_webCmd **>>** 1**);**

upButtonState **=** g\_buttonState**[**UP**];**

downButtonState **=** g\_buttonState**[**DOWN**];**

stopButtonState **=** g\_buttonState**[**STOP**]** **&** **~(**g\_webCmd **>>** 1**);**

obsButtonState **=** g\_buttonState**[**OBS**];**

taskEXIT\_CRITICAL**(&**g\_sharedMemMutex**);**

// light State machine (on, off)

**switch** **(**lightState**)**

**{**

**case** 0**:**

**if** **(**lightButtonState **==** LOW**)**

**{** // switch relay ON

// switch LED ON

digitalWrite**(**RELAY\_LIGHT**,** LOW**);**

lightOnState **=** **true;**

lightState **=** 1**;**

**}**

**break;**

**case** 1**:**

**if** **(**lightButtonState **==** HIGH**)**

**{**

lightState **=** 2**;**

**}**

**break;**

**case** 2**:**

**if** **(**lightButtonState **==** LOW**)**

**{** // switch relay OFF

// switch LED OFF

digitalWrite**(**RELAY\_LIGHT**,** HIGH**);**

lightOnState **=** **false;**

lightState **=** 3**;**

**}**

**break;**

**case** 3**:**

**if** **(**lightButtonState **==** HIGH**)**

**{**

lightState **=** 0**;**

**}**

**break;**

**}**

// Door Control State Machine

**switch** **(**doorState**)**

**{**

**case** 0**:**

**if** **(**doorButtonState **==** LOW**)** // someone pressed the button

**{**

buttonHoldTime **=** esp\_log\_timestamp**()** **+** BTNHOLDMIL**;**

digitalWrite**(**RELAY\_DOOR**,** LOW**);**

doorMovingState **=** **true;**

doorState **=** 1**;**

**}**

**break;**

**case** 1**:** // hold the button for some time to start moving

**if** **(**esp\_log\_timestamp**()** **>** buttonHoldTime**)**

**{**

digitalWrite**(**RELAY\_DOOR**,** HIGH**);**

doorState **=** 2**;**

**}**

**break;**

**case** 2**:** // clear the limit switch

**if** **(((**upButtonState **==** HIGH**)** **&&** **(**downButtonState **==** HIGH**))** **||** **(**stopButtonState **==** LOW**)** **||** **(**obsButtonState **==** LOW**))**

**{**

doorState **=** 3**;**

**}**

**break;**

**case** 3**:** // moving

**if** **((**upButtonState **==** LOW**)** **||** **(**downButtonState **==** LOW**)** **||** **(**stopButtonState **==** LOW**)**

**||** **(**obsButtonState **==** LOW**))** // Limit switch Hit

**{**

buttonHoldTime **=** esp\_log\_timestamp**()** **+** BTNHOLDMIL**;**

digitalWrite**(**RELAY\_DOOR**,** LOW**);**

doorState **=** 4**;**

**}**

**break;**

**case** 4**:** // stop moving

**if** **(**esp\_log\_timestamp**()** **>** buttonHoldTime**)**

**{**

digitalWrite**(**RELAY\_DOOR**,** HIGH**);**

doorMovingState **=** **false;**

doorState **=** 0**;**

**}**

**break;**

**}**

// alarm state machine

**switch** **(**alarmState**)**

**{**

**case** 0**:** // idle

**if** **((**temp **>** HIGHTEMP**)** **||** **(**temp **<** LOWTEMP**)** **||** **(**co **>** HIGHCO**))**

**{** // ALARM condition

digitalWrite**(**RELAY\_ALARM**,** LOW**);**

alarmState **=** 1**;**

**}**

**break;**

**case** 1**:** // ALARM

**if** **((**temp **<** HIGHTEMP **-** TEMPHYST**)** **&&** **(**temp **>** LOWTEMP **+** TEMPHYST**)** **&&** **(**co **<** HIGHCO **-** COHYST**))**

**{** // ALARM condition clearing

digitalWrite**(**RELAY\_ALARM**,** HIGH**);**

alarmState **=** 0**;**

**}**

**else** **if** **(**alarmButtonState **==** LOW**)**

**{** // Someone manually cleared the alarm

digitalWrite**(**RELAY\_ALARM**,** HIGH**);**

**}**

**break;**

**}**

// write back global variables

taskENTER\_CRITICAL**(&**g\_sharedMemMutex**);**

g\_doorMovingState **=** doorMovingState**;**

g\_lightOnState **=** lightOnState**;**

g\_alarmState **=** alarmState**;**

g\_webCmd **=** 0**;** // clear the web command

taskEXIT\_CRITICAL**(&**g\_sharedMemMutex**);**

// set my bit in the watchdog

taskENTER\_CRITICAL**(&**g\_wdMutex**);**

g\_WatchDogBowl **=** g\_WatchDogBowl **|** 0x8**;**

taskEXIT\_CRITICAL**(&**g\_wdMutex**);**

**}**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: void IRAM\_ATTR resetModule( void )

Description: Interrupt service routine for watchdog timer

Referecnes: Modifed from example code provided with the ESP32 library.

See GarageRTC.h for the complete reference.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void IRAM\_ATTR resetModule**()**

**{**

esp\_restart**();** // if called, trigger a systems reset

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: void TaskWatchdog( void\* pvParameters )

void\* pvParameters - Paramaters passed in as part of the

scheduler configuration for this task.

Description: This task is responsible monitoring the watchdog bowl and

ensuring all tasks are running. If a task fails to update

its bit in the bowl, the timer will eventually elapse and

reset the device.

Referecnes: Modifed from example code provided with the ESP32 library.

See GarageRTC.h for the complete reference.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TaskWatchdog**(**void**\*** pvParameters**)**

**{**

**(**void**)**pvParameters**;**

TickType\_t xLastWakeTime**;**

byte g\_WatchDogBowl\_buff **=** 0**;**

// configure the watchdog timer

const int wdtTimeout **=** WDTIMEOUT**;** //time in ms to trigger the watchdog

hw\_timer\_t**\*** timer **=** **NULL;**

timer **=** timerBegin**(**0**,** 80**,** **true);** //timer 0, div 80

timerAttachInterrupt**(**timer**,** **&**resetModule**,** **true);** //attach callback

timerAlarmWrite**(**timer**,** wdtTimeout **\*** 1000**,** **false);** //set time in us

timerAlarmEnable**(**timer**);** //enable interrupt

xLastWakeTime **=** xTaskGetTickCount**();**

// Check to see that the watchdog is being fed by all tasks

// no task is slower than the idle task

**while** **(true)**

**{**

// Macro that expands to schedule the next wake-up time and then

// sleep the task until the next schedule wake-up.

vTaskDelayUntil**(&**xLastWakeTime**,** 1500**);**

taskENTER\_CRITICAL**(&**g\_wdMutex**);**

**if** **(**g\_WatchDogBowl **==** 0xF**)**

**{**

timerWrite**(**timer**,** 0**);** //reset timer (feed watchdog)

**}**

g\_WatchDogBowl **=** 0**;** // clear the bowl

taskEXIT\_CRITICAL**(&**g\_wdMutex**);**

**}**

**}**

# File: WIFI\_AP.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File: WIFI\_AP.ino

\* Description: Configures the AP globals.

\* Authors: Daniel Zajac, danzajac@umich.edu

\* Jackson Harmer, jharmer@umich.edu

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef WIFI\_AP\_H\_INCLUDE

#define WIFI\_AP\_H\_INCLUDE

const char**\*** g\_ssid **=** "ssid"**;**

const char**\*** g\_password **=** "password"**;**

#endif