Remote Radio Aux Control

Steve Patterson WA3RTC

One of the techniques Ham Radio folks are exploring today is the use of Remote radio Operations. While this is not a new undertaking, many new tech devices have made it common. My entry into Remote Operations came with quite a few questions. These questions are both technical and operational in nature. I still have many technical issues to be explored, but one operational concern is that of a locked ON transmitter. This concern was significant enough to result in the Aux Control described here and shown in fig 1a.

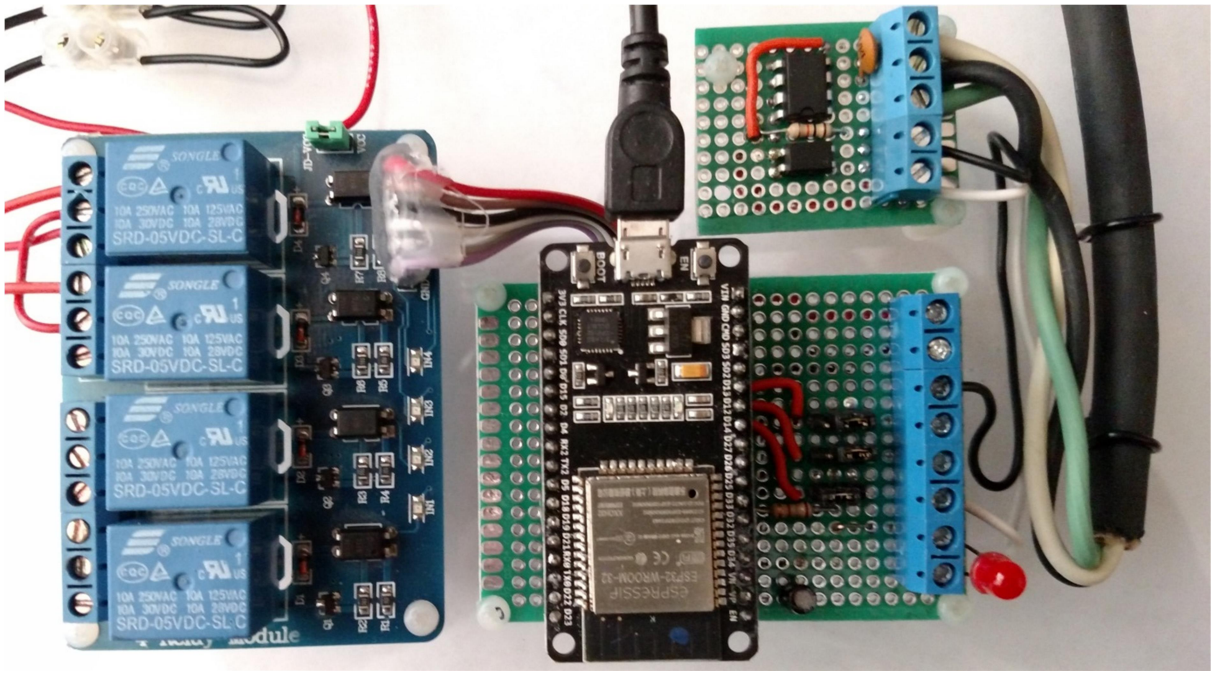


Fig 1a Aux Ctl

Main (center) with Relay (left) and PTT detector (upper right) boards

This operational issue can be boiled down to “what happens when your technically suffocated system fails”. I spent too many years troubleshooting computer controlled machines to think that won’t happen. Further refinement of this failure can be broken down to three failure modes

1. Your transmitter locks in transmit mode and you lose the normal method of shutting down. This could be a control computer failure, local network failure, or even loss of the Internet connection to the control computer.
2. Need to remotely disconnect and reconnect your antennas from your radio. May not be a failure mode unless lightning is involved.
3. Reset a hung up computer (in my case a Raspberry Pi) used to control the system?

While addressing these main failure modes, several added issues came up and were addressed in the Aux Ctl logic. These included:

- Access of the Aux Ctl remotely
- Operation of Aux Ctl when remote link (internet) fails
- Auto reset of Aux Ctl if it is the Aux Ctl device which fails
- Mitigating damage of transmitting with the antenna switched off (operator fails sometimes)

The Aux Ctl, like many of today's homebrew projects, can be split into the Hardware design and the Software implementation to perform its desired tasks.

HARDWARE REQUIREMENTS:

1. **Simple to program and troubleshoot**

   Having some experience with the popular Arduino micro devices, the search started there. New to me, was the Espressif System's ESP32 device. With plenty of Input/output pins, it can be programmed using the Arduino IDE (integrated development environment). Being familiar with this IDE, a somewhat shortened learning curve could be expected.

2. **Accessible remotely**

   The ESP32 has a WiFi and a Web Server Library available which checked off the accessibility requirement. The programming of the ESP32 was aided with information obtained from an e-book and course from Rui Santo's Random Nerd Tutorial's (RNT) found at: http://randomnerdtutorials.com .

3. **Inexpensive** (we are Hams)

   As of mid-2020, the cost of the DOIT ESP32 DEV V1 development board, which I chose to use, ranged less the $6 US from China to just under $10 US from Amazon. Note to Ham's, using Amazon Prime; use of "Smile.Amazon.com" does not add additional cost to your order, but Amazon will send a percentage of your payment to your favorite charity with each purchase. Mine goes to my local Ham Radio Club.

4. **Reliability**

   Reliability is a major issue, but only time can tell. Several reliability issues where uncovered using the WiFi Library which required troubleshooting and testing. The end result has been a system which has been online several weeks as of this writing without any major glitches.

HARDWARE CONTROLS:

The control of three major elements of the remote station was done using three relays under the ESP32 control. The Aux Ctl commands come from both its Web Server, which is accessed remotely through an internet browser, and its internal logic and timing functions. The use of a "5V 4 Channel Relay Module that work with official Arduino boards" (as listed in its product description) made the interface connections to the ESP32 straightforward.

Radio Power: The power to the radio is controlled using a 12vdc coil power relay with contacts rated 30A at 12vdc. This relay is energized through the use of Relay 4 on the 4 channel Relay Module. The Power Relay was purchased at a local auto store (O'Reillys is the closest here).

Antenna Connection: Using Relay 3, the antennas are controlled with two additional 12 volt dc relays which connect the radio to the antenna when energized and when de-energized, connect the antenna to ground and the radio to a 50 ohm resistor. This resistor cannot take a lot of power, so adding a

feature to detect the antenna OFF state with the radio in transmit mode causes the power to the radio to be shut off after 20 seconds. This added shutdown is an example of project scope creep and is why these projects are never done.

The radio used for this remote control project is an ICOM IC-746. Modules for the antenna control (fig 2) and the detection of the radio's PTT status (fig 4) were also constructed to work with this radio. The IC-746 is setup to use separate antennas for HF and VHF, so two antenna switch modules were constructed.

Reset of Server: Using Relay 2, the power to the Raspberry Pi controller can be interrupted to reset the Pi. The normal program shutdown should be initiated first, the shutdown procedure permitted to finish, and then cycle the 5 volt power for a hard reset. Memory card corruption is what we are trying to avoid so a warning notice is presented on the remote browser before the off function is activated. The controller I am using is the MFJ-1234 RigPi Station Server which was developed by Howard Nurse, W6HN. There are, however, several other Pi based station controls for which this Aux Ctl can be used. One of the 5V Channel Relays can handle the current directly for this reset function and the use of the normally closed (NC) contact minimizes the chances of an accidental interrupt of the Pi Server power due to Aux Ctl resetting.

Detection of Transmit Mode (RF): It was found that even with the PTT function from the control computer via a serial CI-V interface, the PTT pin in the IC-746 mic connector went to zero volts. This feature was used to develop an input to the ESP32 to indicate transmit mode. The use of a voltage follower Op-Amp provided a high impedance input to prevent loading the radio's PTT line. The use of an optocoupler provided further isolation to the IC-746 at this detector. Other methods of RF detection can be added and a couple of additional inputs are provided if you decide to implement additional methods.

The overall schematic for the Auxiliary Control is included (fig 3). If you wish to build your own Aux Ctl, please experiment with the modules you have available. That is what makes projects like this fun and your own.

SOFTWARE - NETWORK:

A Random Nerd Tutorial's (RNT) module was the basis for the Network connection element of the Aux Ctl. The key network tasks however took place in the local router and the setting up of a dynamic domain name service.

1. To give the ESP32 a fixed IP address, the local router was configured to assign a specific IP address to the Aux Ctl using the mac address of the ESP32. To provide its mac address, the "void setup()" section of the ESP32 program prints its mac address and the assigned IP address upon startup or reset (fig 5).
2. The network port of the ESP32 web server was programmatically set to :8888. The local router was configured to forward the incoming port used to connect to this device (:18183) to port :8888 of the static IP which it had assigned (fig 6). Note that the :18183 port is an arbitrary port selected to provide access to the ESP32 from the internet in the form of "FARC.ddns.net:18183". (This domain name and port are NOT the ones I used, but were chosen for this document only.)
3. To be able to access the Aux Ctl from the internet, the local IP address assigned by your ISP must be known. Since my service is with Spectrum, they can change my IP address depending what is available at the time my router is rebooted. To work around this, a dynamic domain name service can be setup to keep track of these changes and provide a stable name/address with

which to access the Aux Ctl. The dynamic domain name service chosen was https://www.noip.com/. This is a free service but requires a simple conformation via e-mail that you wish to continue every month. This conformation procedure can be avoided if you pay a yearly fee, but we are Hams! Noip.com is where the domain name FARC.ddns.net was setup.

4. Finally, the local router is configured to automatically update noip of changes to the WAN (internet) IP which is currently assigned by the internet service provider.

SOFTWARE - LOGIC:

Access of Aux Ctl via a web browser:
While RNT directed how to setup a basic web server control via a remote web browser, many changes were required to match it to the functions desired. Each of the controls was assigned to a row in a table consisting of a title and a status. The status is color coded and labeled with Red = OFF and Green = ON. Pressing the color status button will cause the feature to change status. Both the Radio and RigPi power shut off must be verified before the action is actually performed.

Added to the control and status of the three devices is a display to indicate if the radio is in transmit mode (RF) using the signal from the IC-746 PTT line. The RF block is Gray if not in transmit mode and Blue if in transmit mode. The other feature is the Refresh Screen button at the bottom. The Aux Ctl was constructed to be able to control a remote radio site, not to continually monitor its mode. To that end, you must refresh the screen when you wish to see the current state of the parameters.

The color of the buttons is set in the program section which creates the “CSS styles” and can be modified to accommodate your visual requirements.

Screen shots of the Aux Ctl are shown here, with comments on the function of each screen, as seen using a web browser either from a computer or a smart phone.

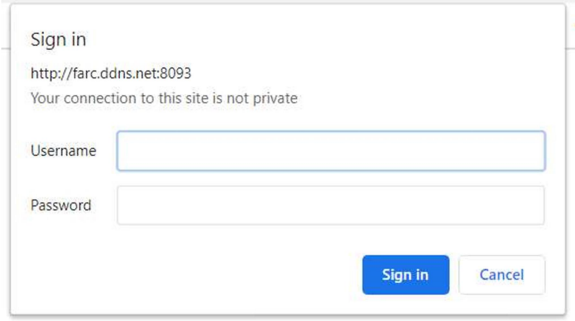
Operation of Aux Ctl functions when the internet fails:
The ESP32 device has the control ability of today’s micro controllers even without access to the internet. The functions of shutting down a locked ON transmitter as well as removing antenna connections after a transmitter power shut down are all internal to the ESP32’s logic functions.

Automatic reset of Aux Ctl if its software locks up:
The use of a software watchdog was tested and found reliable in monitoring and resetting the ESP32 in case of a logic processing crash. The software Watchdog Timer by Jack Rickard was chosen for the ESP32. Details can be found at https://www.youtube.com/watch?v=7kLy2iwIvy8.

Mitigating damage of transmitting with the antenna switched off (operator fails sometimes):
Other monitoring and control were included which have no requirement for the internet being available. Including 1) transmitting without the antennas being active, 2) loss of the WiFi connection results in re-connection attempts every two seconds until connection is back, and 3) activates an automatic reset of the ESP32 to avoid any concerns with an internal timing value named millis() from rolling over every 49.7 days.

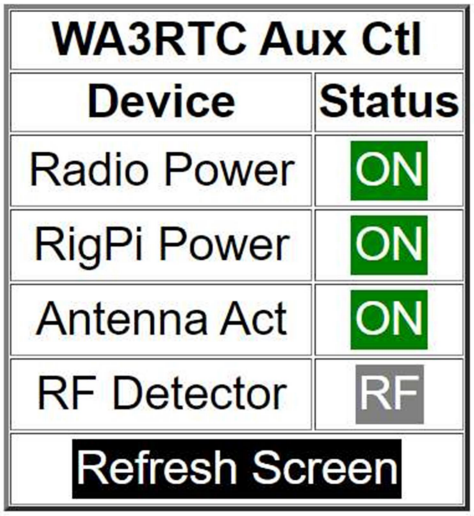
For more details of the software functionality, review the included flow diagram. A flow diagram is to the software as a schematic is to the hardware.

SCREEN SHOTS:



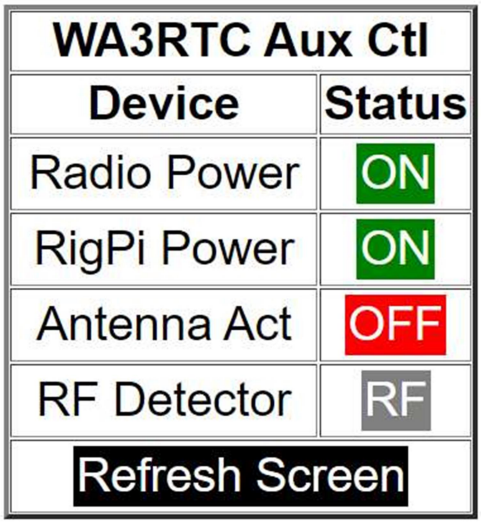
Screen 1 Sign in:
Not wanting just anyone being able to control your remote equipment, a User Name and Password is included in the ESP32 generated web page. You can generate your own unique user/password using: https://www.base64encode.org/.

This feature and the main body of the web page generation can be found in a RNT tutorial.



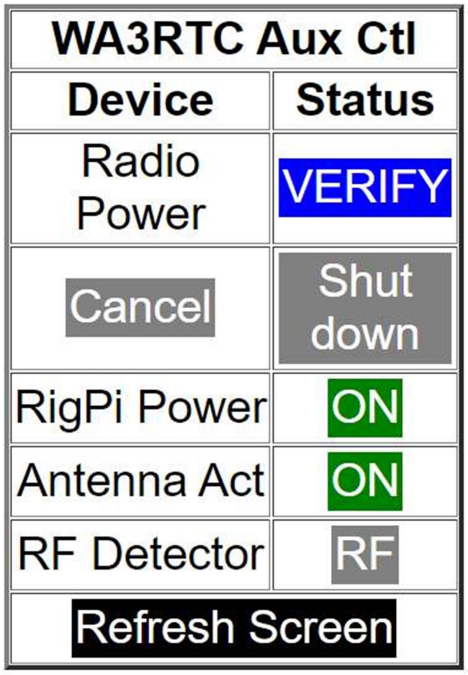
Screen 2 All controls on for normal operation:
A simple table form was chosen for the web page display. Shown here, with all conditions set for normal operation. The RF indication would be blue if the PTT were active when this screen was refreshed.

Pressing the "ON" of the Antenna Act will immediately disconnect the antennas for both the HF and VHF from the radio.

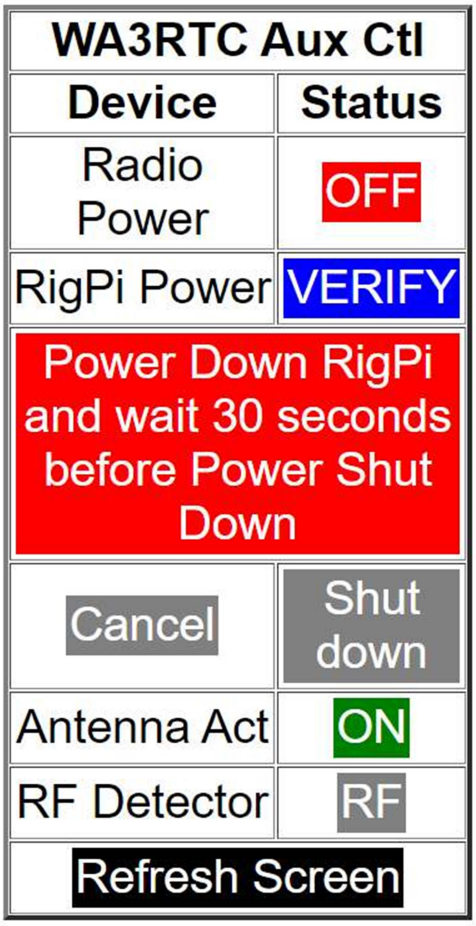


Screen 3 Turnning the Antenna Act OFF:
To turn off, or actually to change the state of any of the three controls, one just presses the current status as a button. The function of the web page is to send a GET (HTML stuff) command with a request to the ESP32 web server. The server determines which request has been made, acts on the request, changes the state of the control and replies with an updated button color.

Good thing all that is behind the scenes and that ESP32 takes care of the details. If interested however, see the details in RNT's Learn ESP32 With Arduino IDE or in the actual ESP32 code.



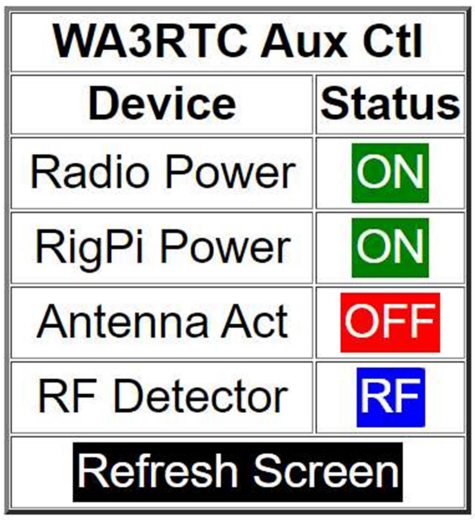
Screen 4 Now for some of the warnings:
Always like to "VERIFY" that the operator meant to push the power off button. Choices are given to Cancel which will return with the Radio Power still ON, or to go ahead and Shut down.



Screen 5 Another VERIFY:
Requesting to turn the power off to the Raspberry Pi.
The Pi does not like just pulling its plug. It likes its program to be shut down then wait for an orderly computer shut down, which takes less than 30 seconds, before proceeding.

Thus, the warning in RED, or you can cancel if you pressed the button by mistake.

The Aux Control has been designed to energize a relay to remove power from the Pi which avoids sudden interruptions during Aux Ctl resets.



Screen 6 Antenna OFF with PTT detected:
If the RF is detected while the Antenna Act is "OFF", Radio Power will be shut down within 20 seconds.

This is not a normal condition, but I have done it. I used very small 50 ohm resistors across the radio's antenna connections when the antenna switch is off, so the time this situation can exist is monitored and limited by Aux Ctl.

Remember that this is not a self-refreshing page, so the Refresh Screen button should be used to bring in current status of all features including the RF Detector.

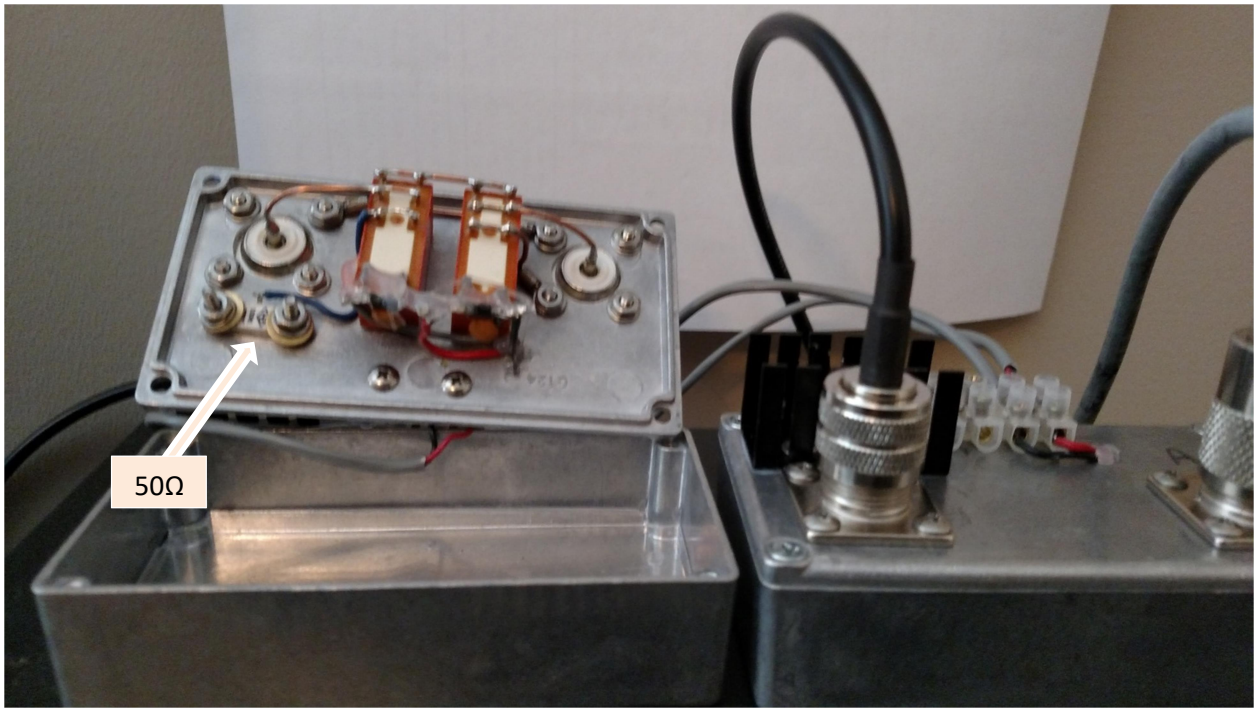
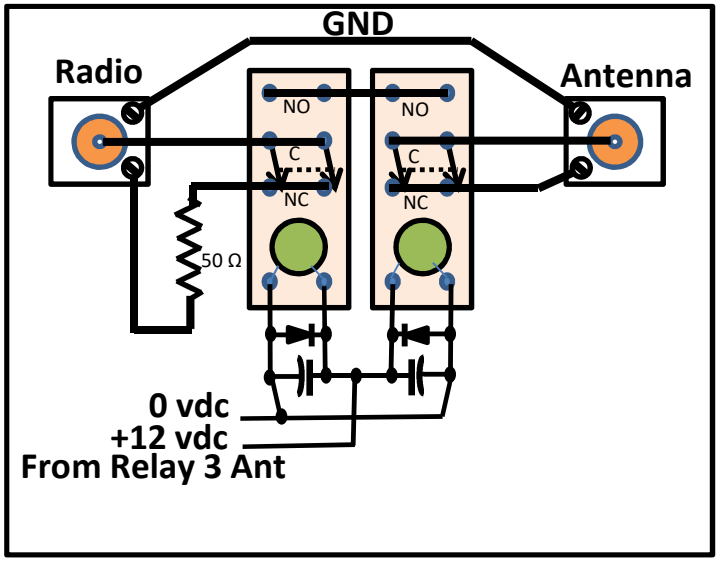


Fig 1b Antenna Switch

One required per antenna connected to the remote radio. Center components are double pole, double throw (DPDT) 12 vdc coil relays with their poles connected in parallel for max current.



When you find "5Pcs 100w 50ohm dummy load RF resistor" on ebay for $5.25 you know it's too good to be true, but... I did melt one with less than 20 watts of dc, without a heatsink. So I put on a heatsink and programmed a max of 20 seconds transmit time before having the Aux Ctl pull the plug to the radio. Note in fig 1a below the SO-239 in the open unit, these resistors are tiny.

Fig 2 Antenna Switch 1 of 2

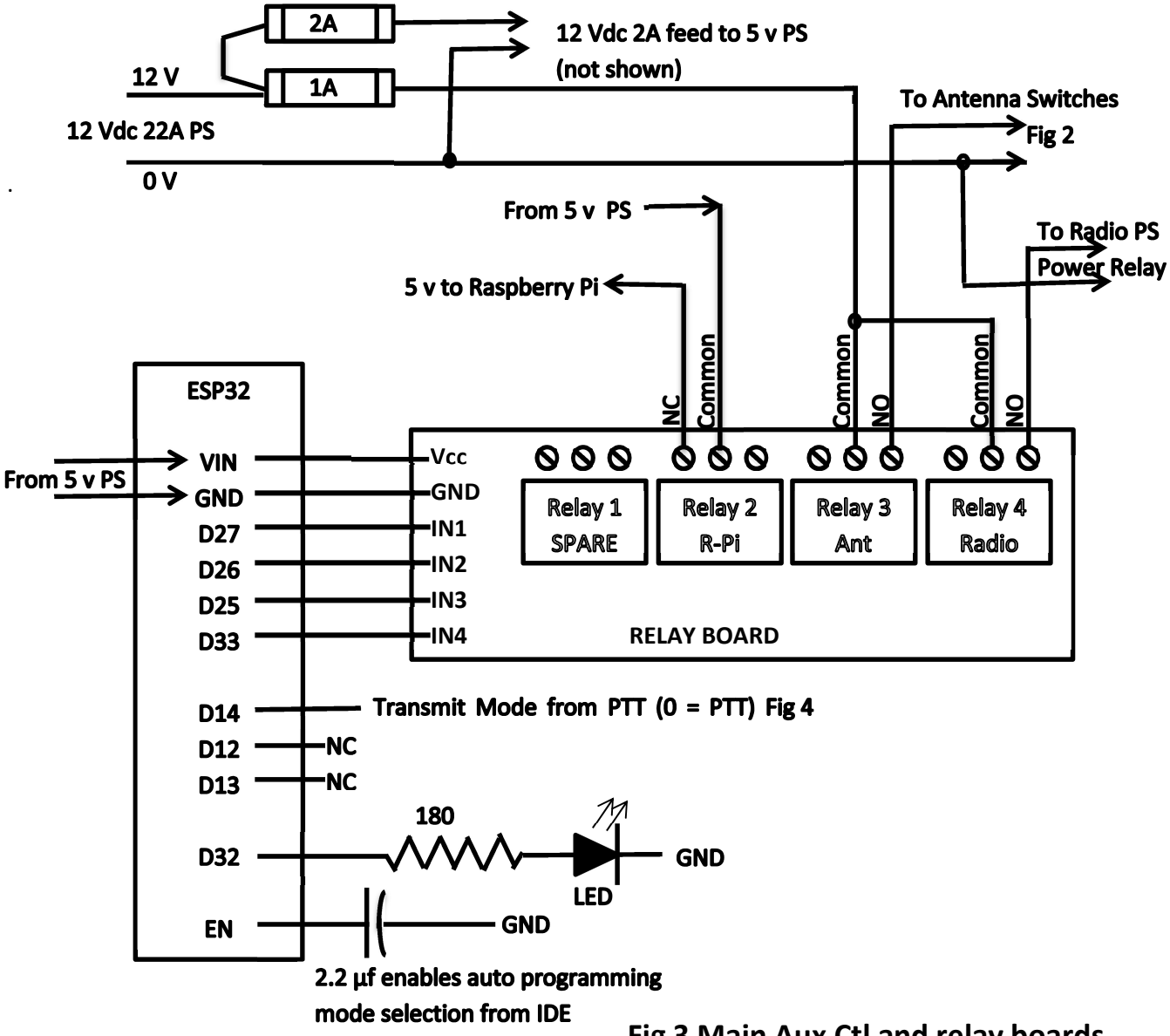


Fig 3 Main Aux Ctl and relay boards

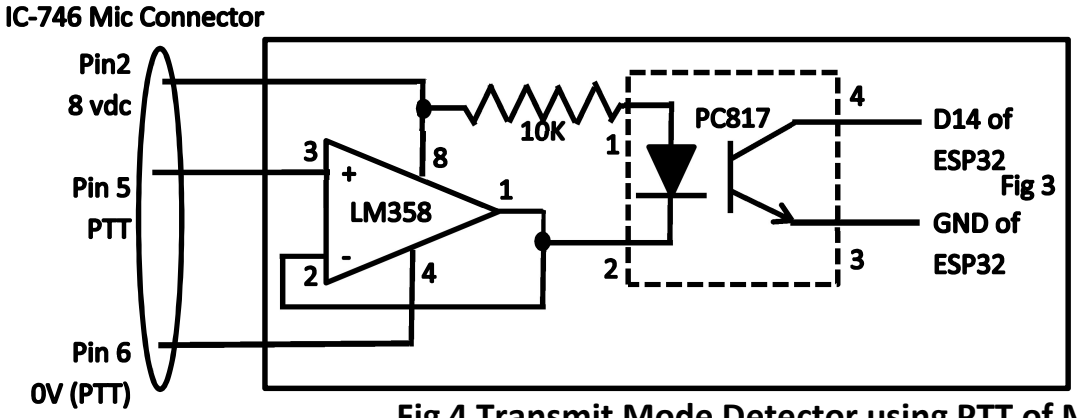


Fig 4 Transmit Mode Detector using PTT of Mic Connector

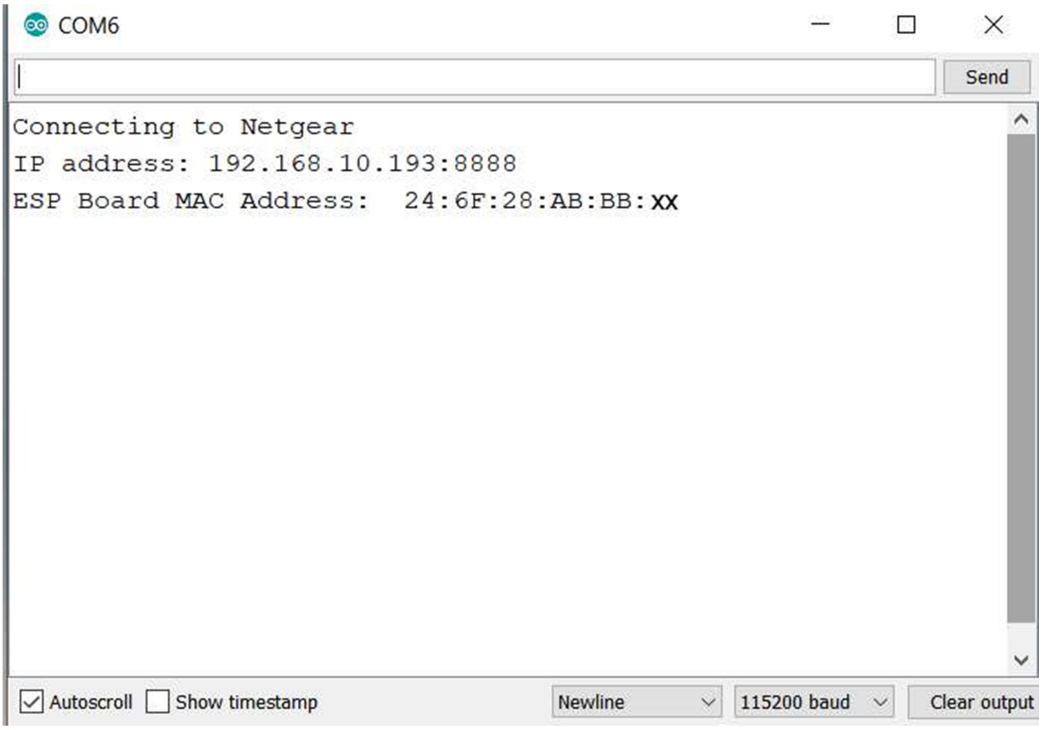


Fig 5 ESP32 Startup screen displaying its IP and Mac addresses

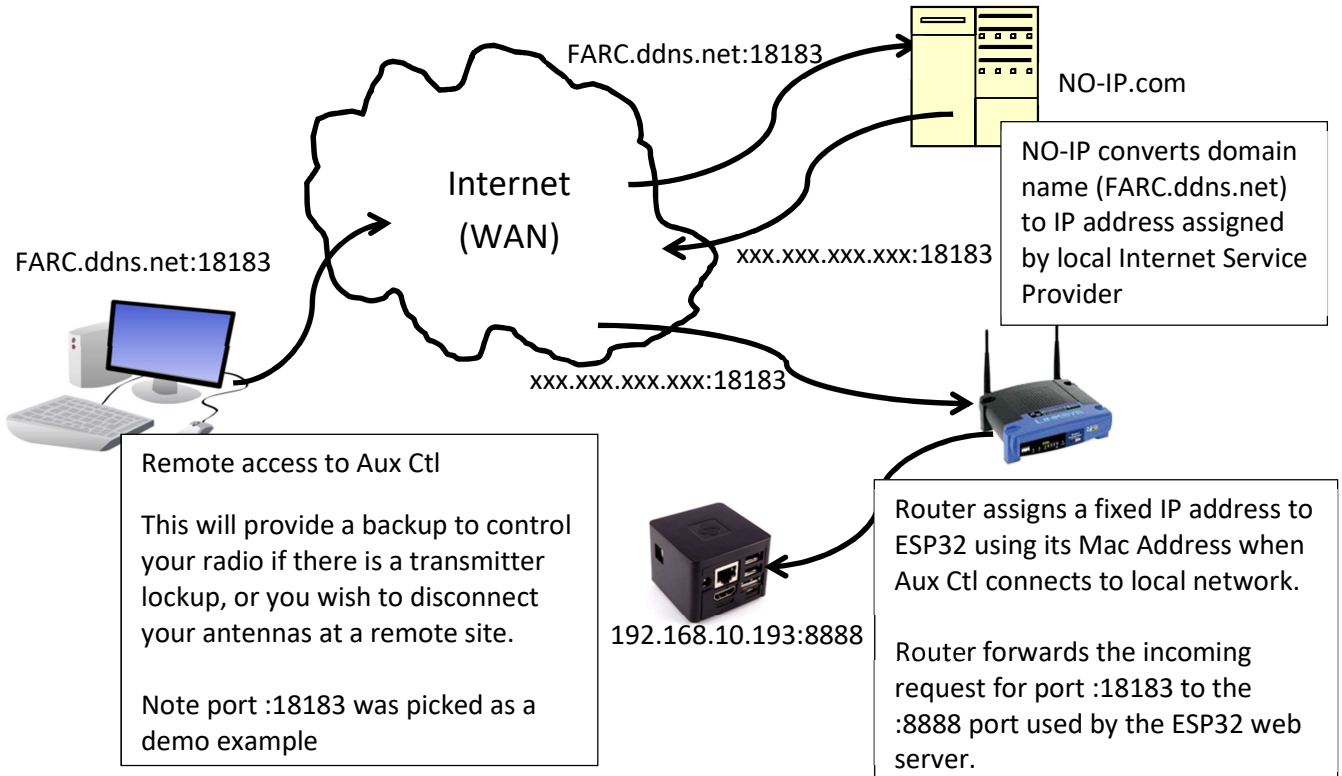


Fig 6 Remote Access of Control using the internet

Remote Radio Aux Ctl Program Flow

Steve Patterson WA3RTC

Aux Ctl is an independent monitor of a remote radio controlled by a RigPi (tm of W6HN)
Aux Ctl by WA3RTC 2020 makes use of: Rui Santo's RNT Learn ESP32 with Arduino IDE as a reference. https://rntlab.com/
The flow chart uses a block reference for composing the string (header) from the requesting data from the ESP32's WiFi connection as routine is maintained in the general configuration presented by RNT. For more detail it is recommend subscribing to RNT as I have found it an extremely helpful tool.

Global Definitions and parameters for compiler

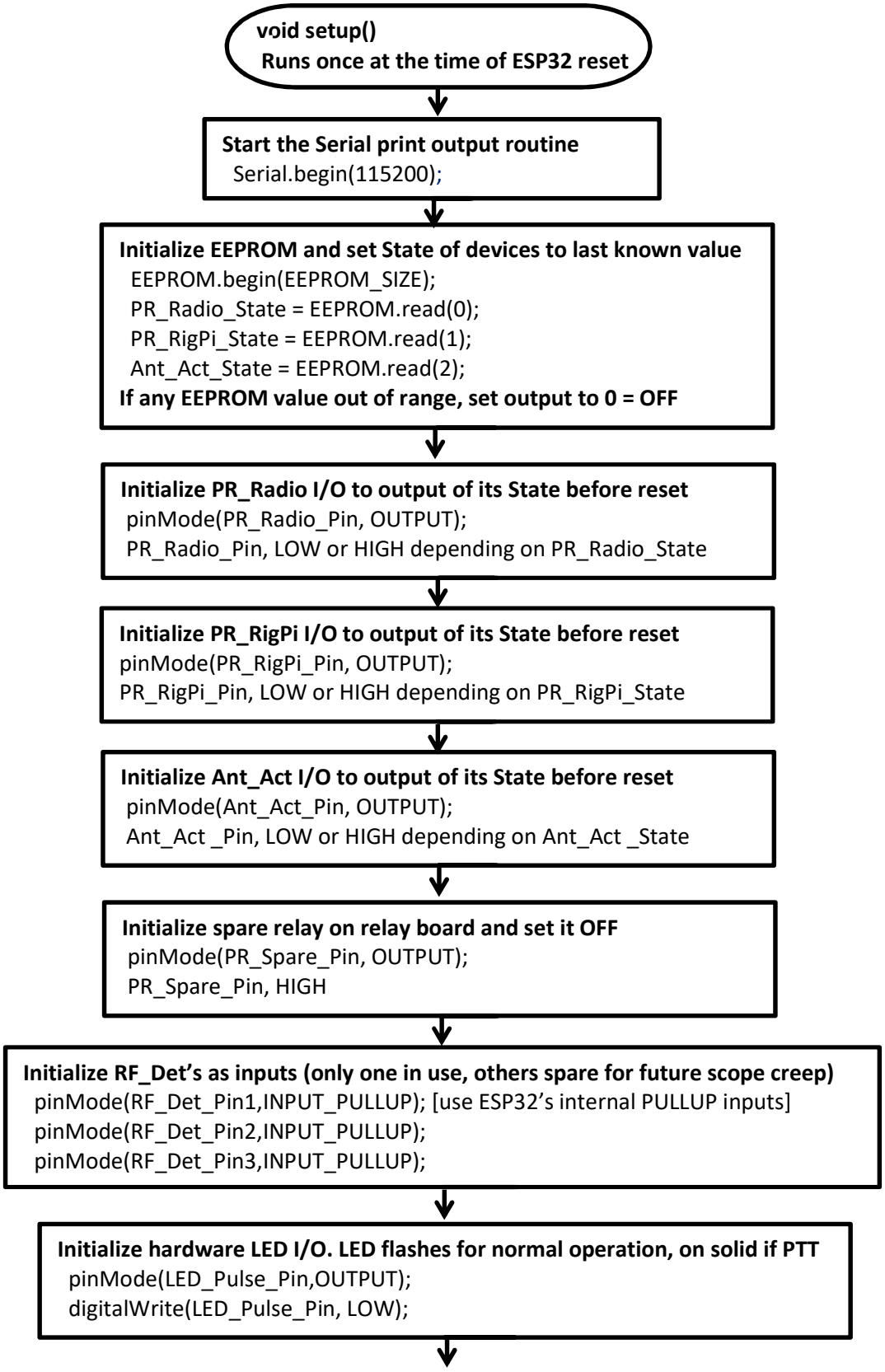
Load libraries
WiFi, EEPROM (size = 3)

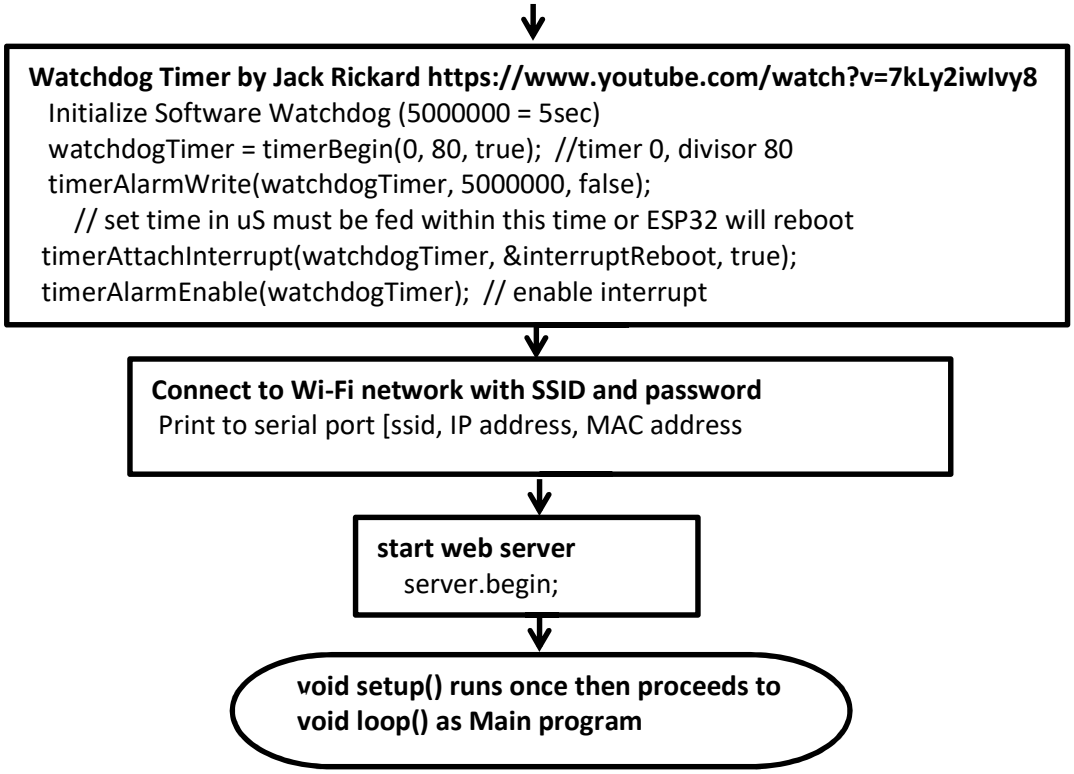
Define network credentials
Your, ssid, password
check_wifi = 2 seconds to recheck WiFi if it is down
WiFi Server to use port 8888

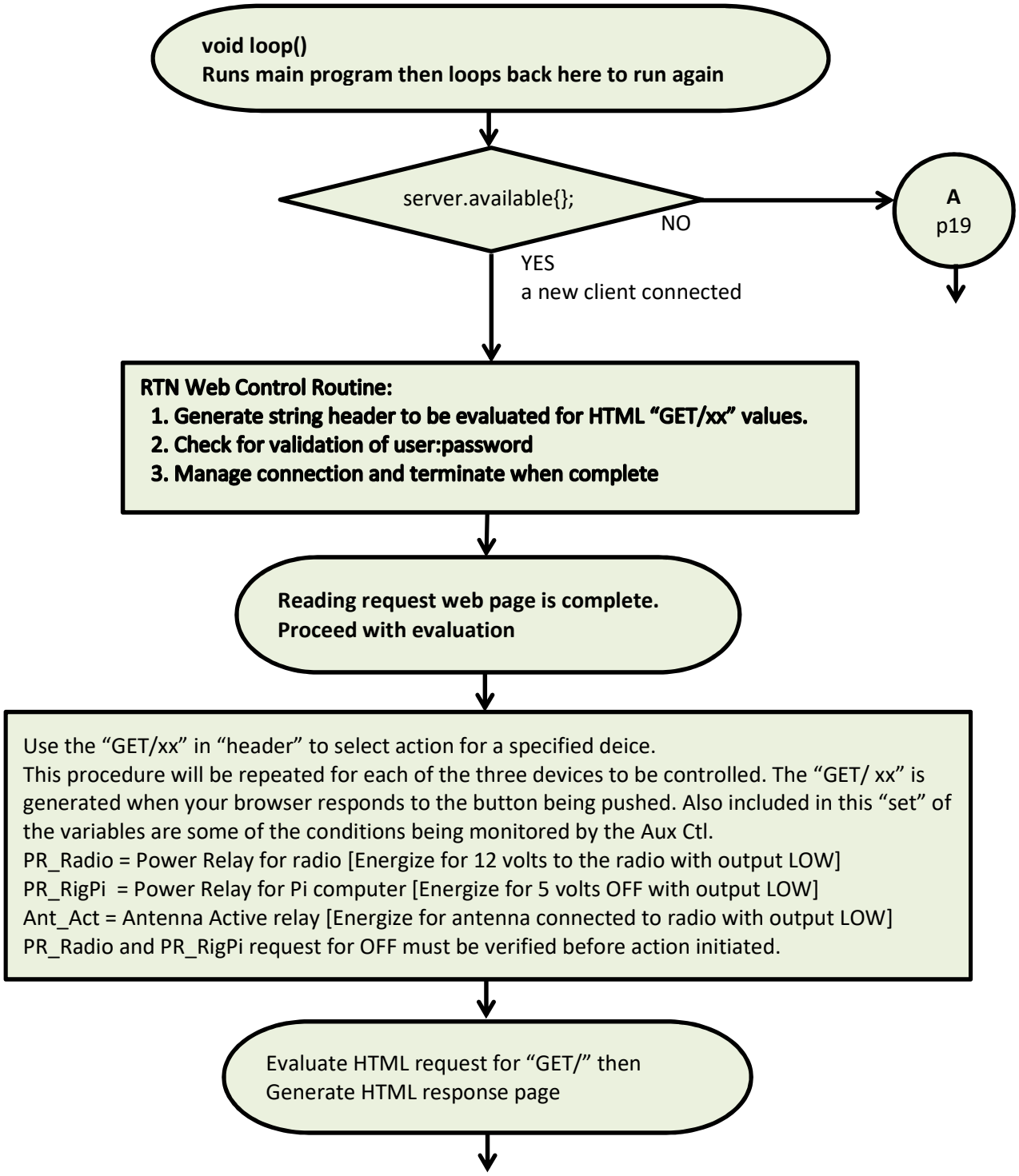
Define Program Constants
#define PR_Spare_Pin 27
#define PR_Radio_Pin 33
#define PR_RigPi_Pin 26
#define Ant_Act_Pin 25
#define RF_Det_Pin1 13
#define RF_Det_Pin2 12
#define RF_Det_Pin3 14
#define LED_Pulse_Pin 32

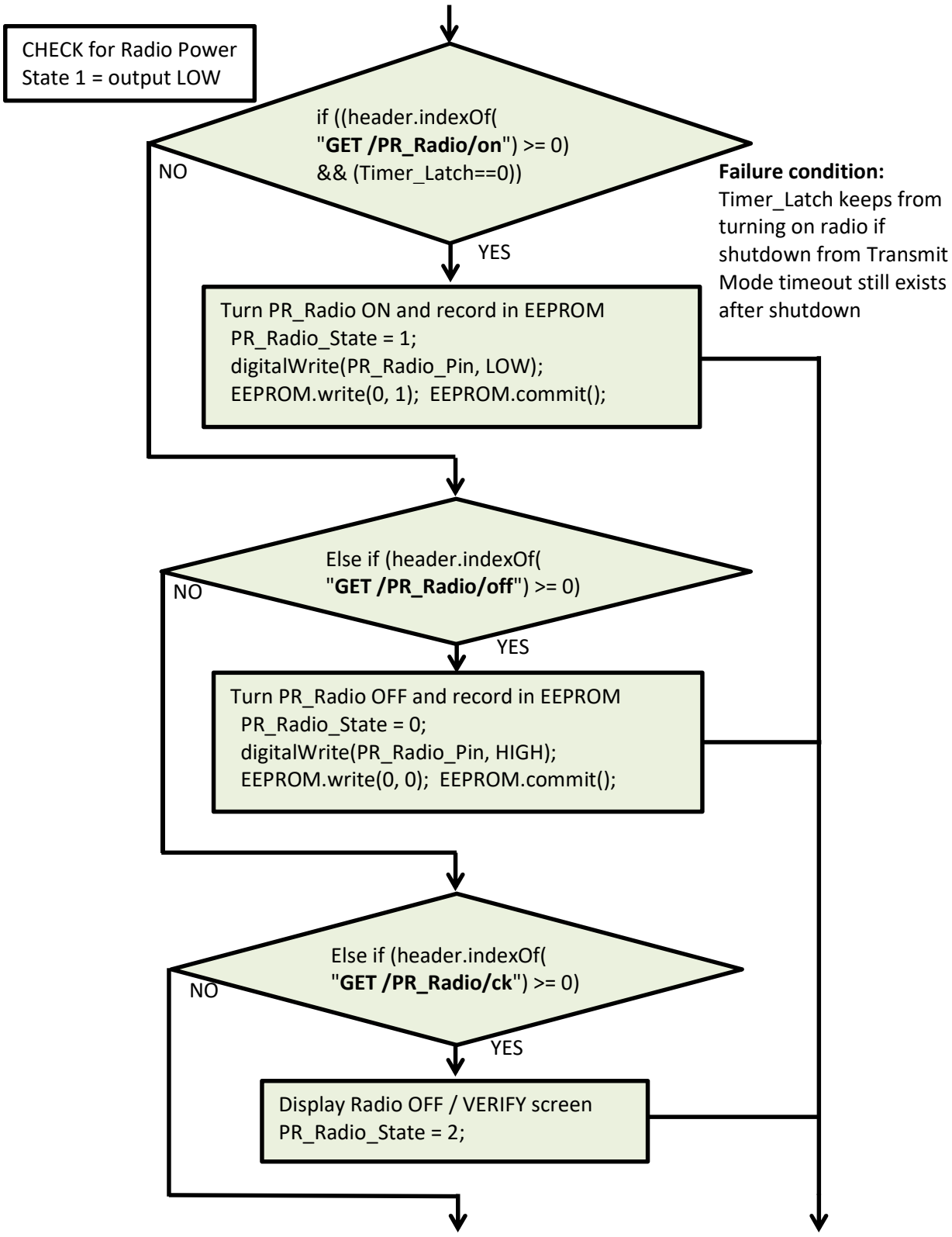
Define Program variables
HTTP request string [header]
Fix for Chrome not dropping connection [currentTime, previousTime, timeoutTime]
State variables [PR_Radio_State, PR_RigPi_State, PR_RigPi_State, Ant_Act_State, RF_Det_State]

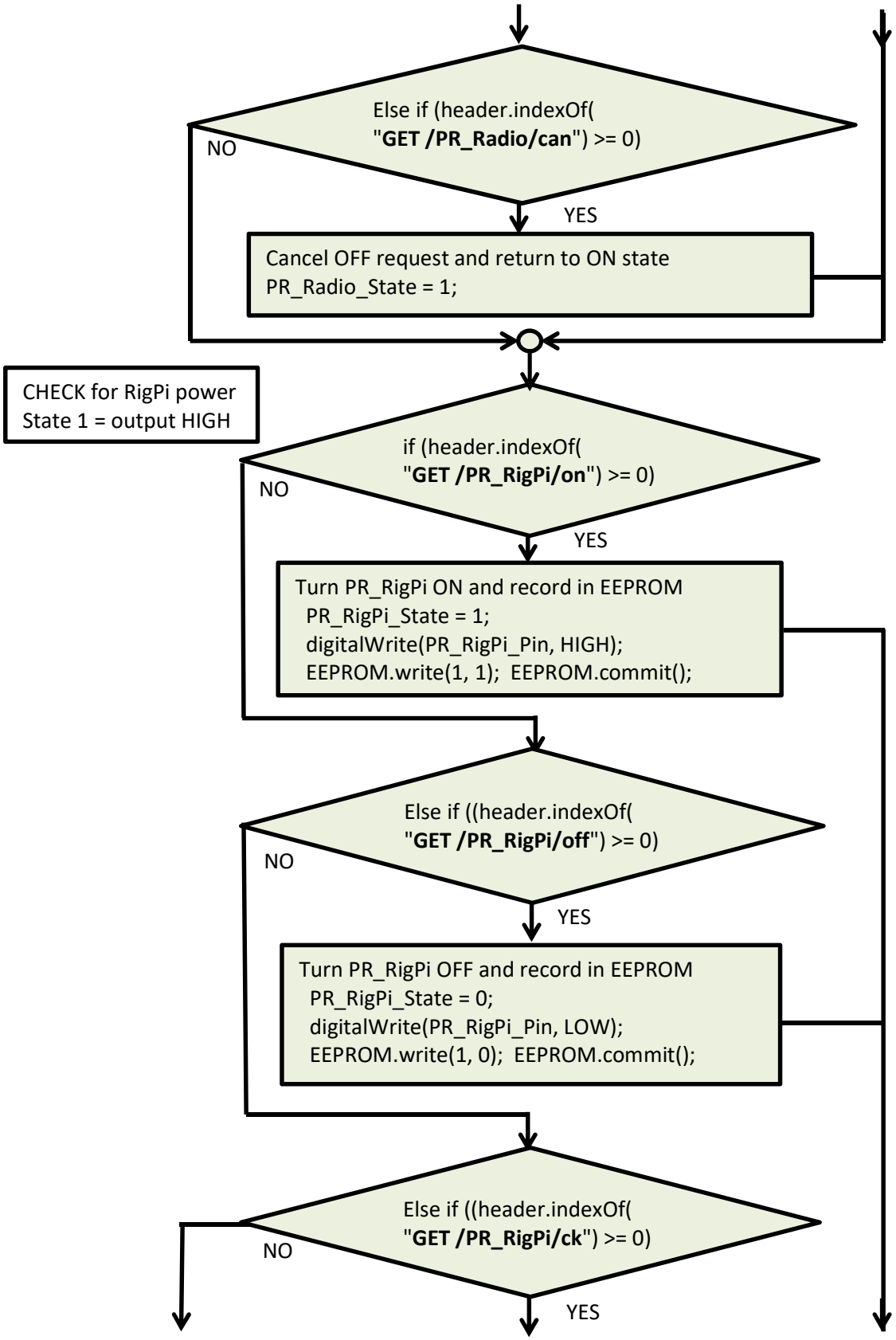
Define Timer Parameters & Flags
Antenna OFF if Radio power OFF [Radio_Ant_Act, Radio_Ant_Time, Radio_Ant_Start]
RF Fail ON detect [Timer_Cnt, Timer_Act, Timer_Latch, Timer_min, Timer_ms, Timer_Time]
RF on with Ant Off detect [Radio_AntNot_RF, Radio_AntNot_RF_Start, Radio_AntNot_RF_Time]
Hardware LED flash rate [LED_Time, LED_previousTime, LED_timeoutTime, LED_f]
Software Watchdog [hw_timer_t *watchdogTimer = NULL, looptime]

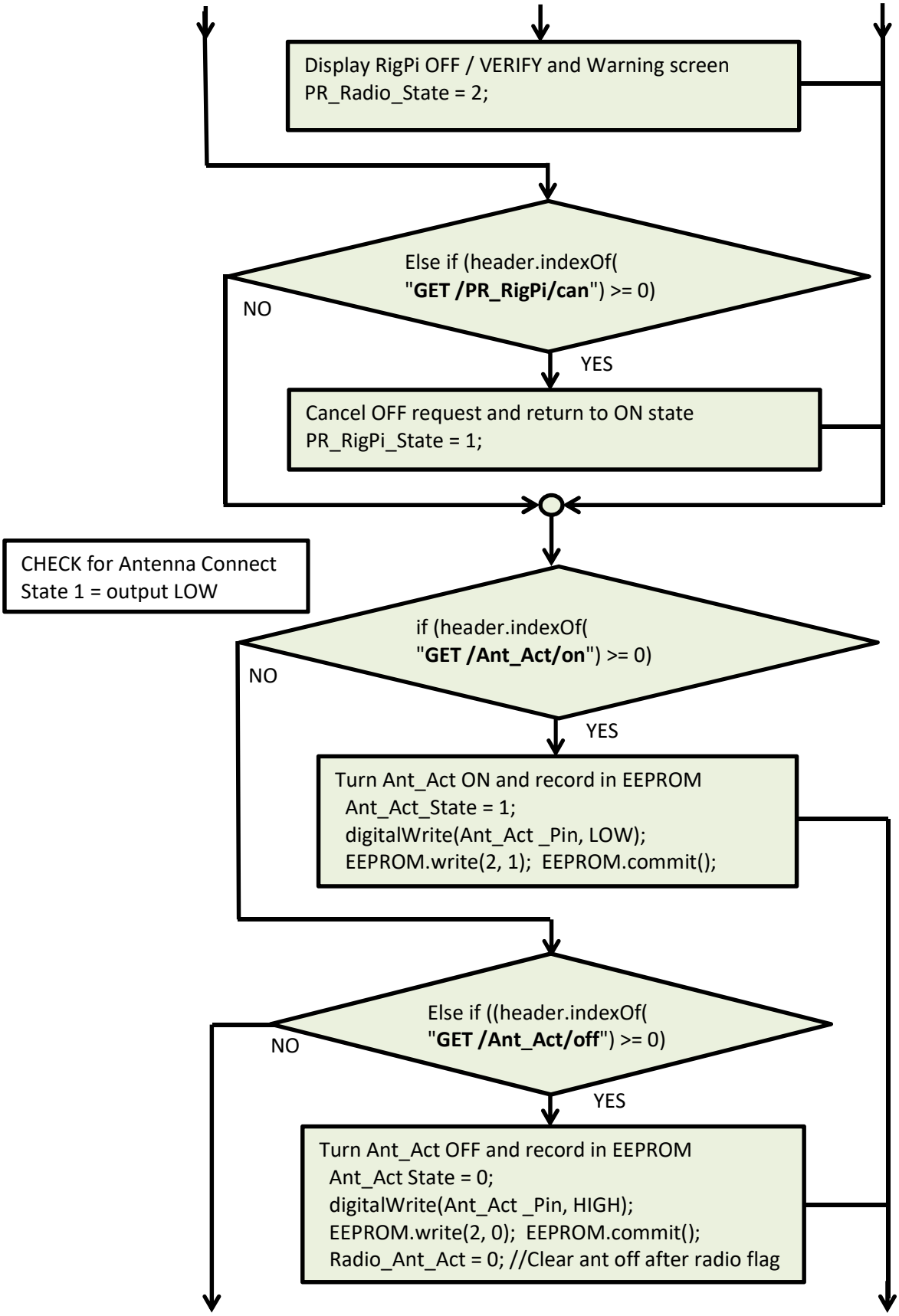

void setup()
Runs once at the time of ESP32 reset
Start the Serial print output routine
Serial.begin(115200);
Initialize EEPROM and set State of devices to last known value
EEPROM.begin(EEPROM_SIZE);
PR_Radio_State = EEPROM.read(0);
PR_RigPi_State = EEPROM.read(1);
Ant_Act_State = EEPROM.read(2);
If any EEPROM value out of range, set output to 0 = OFF
Initialize PR_Radio I/O to output of its State before reset
pinMode(PR_Radio_Pin, OUTPUT);
PR_Radio_Pin, LOW or HIGH depending on PR_Radio_State
Initialize PR_RigPi I/O to output of its State before reset
pinMode(PR_RigPi_Pin, OUTPUT);
PR_RigPi_Pin, LOW or HIGH depending on PR_RigPi_State
Initialize Ant_Act I/O to output of its State before reset
pinMode(Ant_Act_Pin, OUTPUT);
Ant_Act _Pin, LOW or HIGH depending on Ant_Act _State
Initialize spare relay on relay board and set it OFF
pinMode(PR_Spare_Pin, OUTPUT);
PR_Spare_Pin, HIGH
Initialize RF_Det's as inputs (only one in use, others spare for future scope creep)
pinMode(RF_Det_Pin1,INPUT_PULLUP); [use ESP32's internal PULLUP inputs]
pinMode(RF_Det_Pin2,INPUT_PULLUP);
pinMode(RF_Det_Pin3,INPUT_PULLUP);
Initialize hardware LED I/O. LED flashes for normal operation, on solid if PTT
pinMode(LED_Pulse_Pin,OUTPUT);
digitalWrite(LED_Pulse_Pin, LOW);

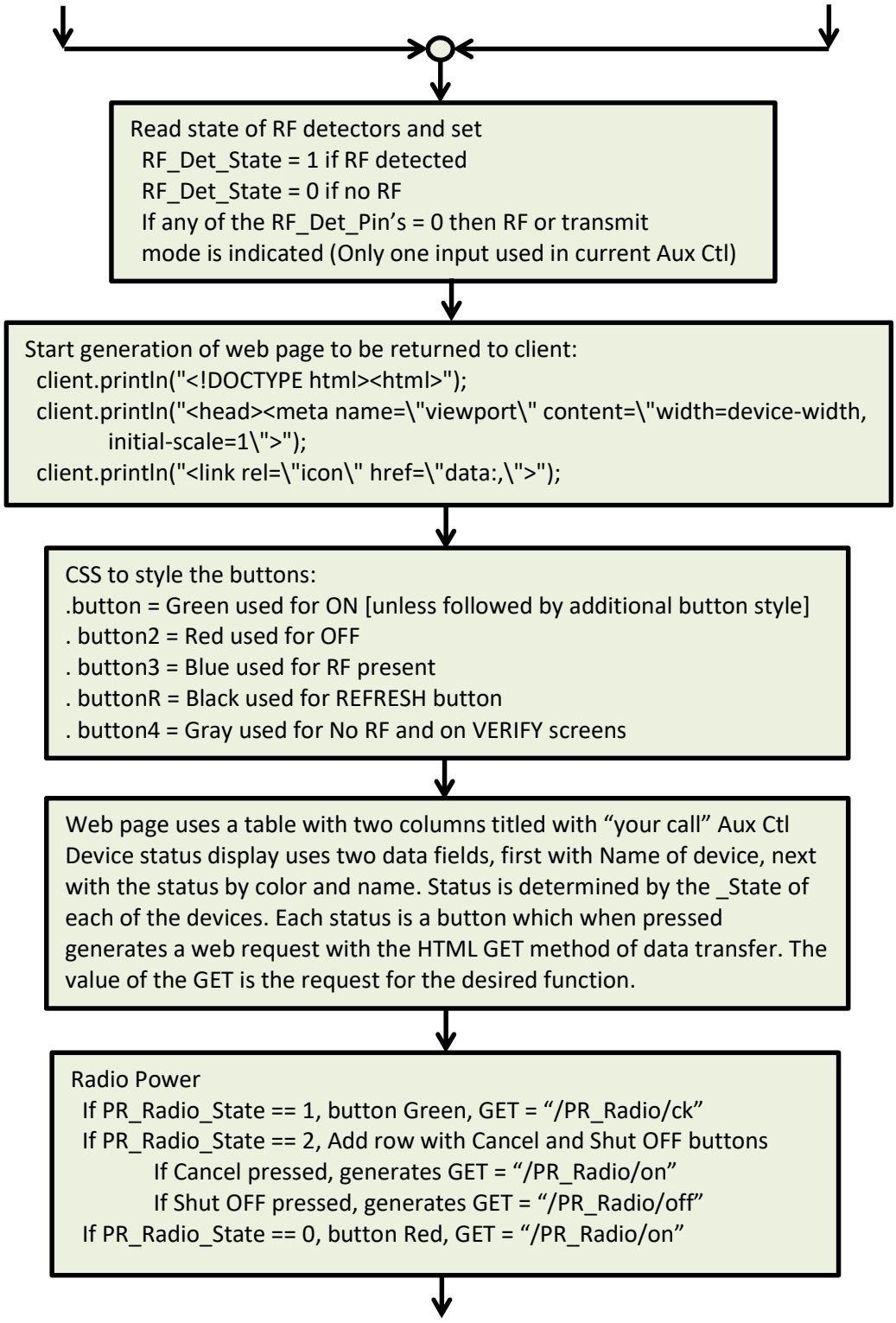

Watchdog Timer by Jack Rickard https://www.youtube.com/watch?v=7kLy2iwIvy8
Initialize Software Watchdog (5000000 = 5sec)
watchdogTimer = timerBegin(0, 80, true); //timer 0, divisor 80
timerAlarmWrite(watchdogTimer, 5000000, false);
// set time in uS must be fed within this time or ESP32 will reboot
timerAttachInterrupt(watchdogTimer, &interruptReboot, true);
timerAlarmEnable(watchdogTimer); // enable interrupt
Connect to Wi-Fi network with SSID and password
Print to serial port [ssid, IP address, MAC address
start web server
server.begin;
void setup() runs once then proceeds to
void loop() as Main program

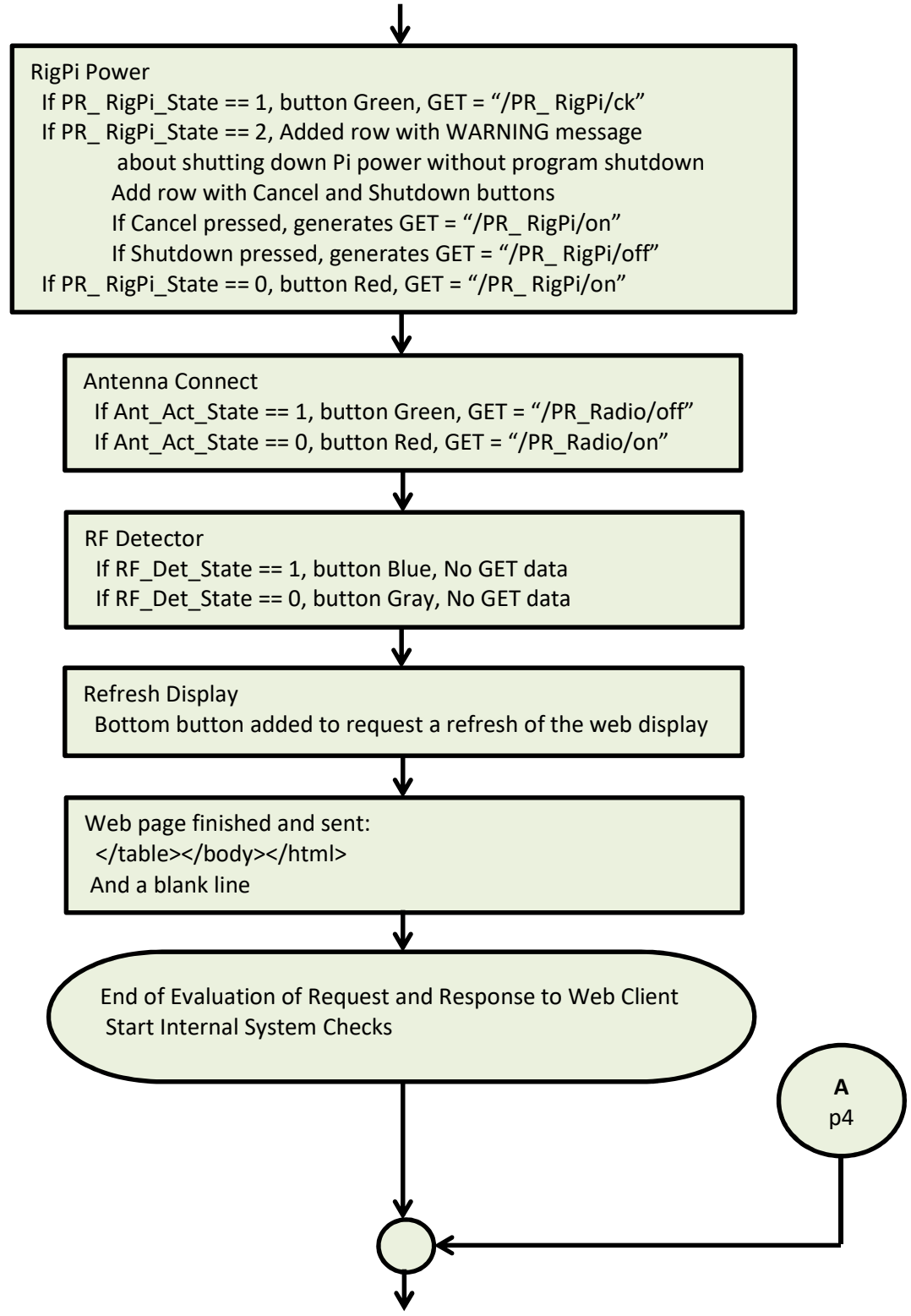

void loop()
Runs main program then loops back here to run again
server.available{};
NO
A
p19
YES
a new client connected
RTN Web Control Routine:
1. Generate string header to be evaluated for HTML "GET/xx" values.
2. Check for validation of user:password
3. Manage connection and terminate when complete
Reading request web page is complete.
Proceed with evaluation
Use the "GET/xx" in "header" to select action for a specified deice.
This procedure will be repeated for each of the three devices to be controlled. The "GET/ xx" is generated when your browser responds to the button being pushed. Also included in this "set" of the variables are some of the conditions being monitored by the Aux Ctl.
PR_Radio = Power Relay for radio [Energize for 12 volts to the radio with output LOW]
PR_RigPi = Power Relay for Pi computer [Energize for 5 volts OFF with output LOW]
Ant_Act = Antenna Active relay [Energize for antenna connected to radio with output LOW]
PR_Radio and PR_RigPi request for OFF must be verified before action initiated.
Evaluate HTML request for "GET/" then
Generate HTML response page

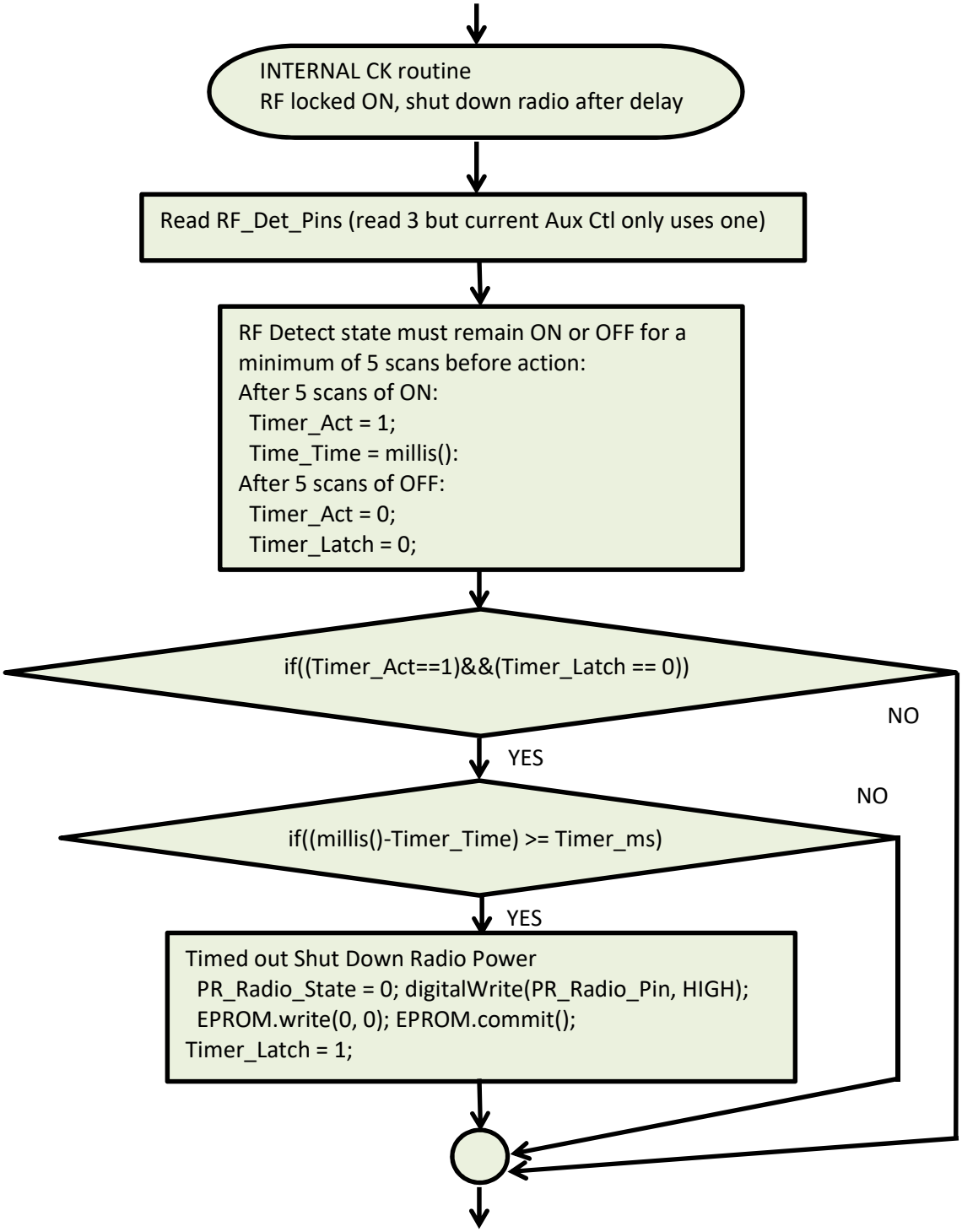

CHECK for Radio Power
State 1 = output LOW
if ((header.indexOf(
"GET /PR_Radio/on") >= 0)
&& (Timer_Latch==0))
NO
YES
Failure condition:
Timer_Latch keeps from turning on radio if shutdown from Transmit Mode timeout still exists after shutdown
Turn PR_Radio ON and record in EEPROM
PR_Radio_State = 1;
digitalWrite(PR_Radio_Pin, LOW);
EEPROM.write(0, 1); EEPROM.commit();
Else if (header.indexOf(
"GET /PR_Radio/off") >= 0)
NO
YES
Turn PR_Radio OFF and record in EEPROM
PR_Radio_State = 0;
digitalWrite(PR_Radio_Pin, HIGH);
EEPROM.write(0, 0); EEPROM.commit();
Else if (header.indexOf(
"GET /PR_Radio/ck") >= 0)
NO
YES
Display Radio OFF / VERIFY screen
PR_Radio_State = 2;

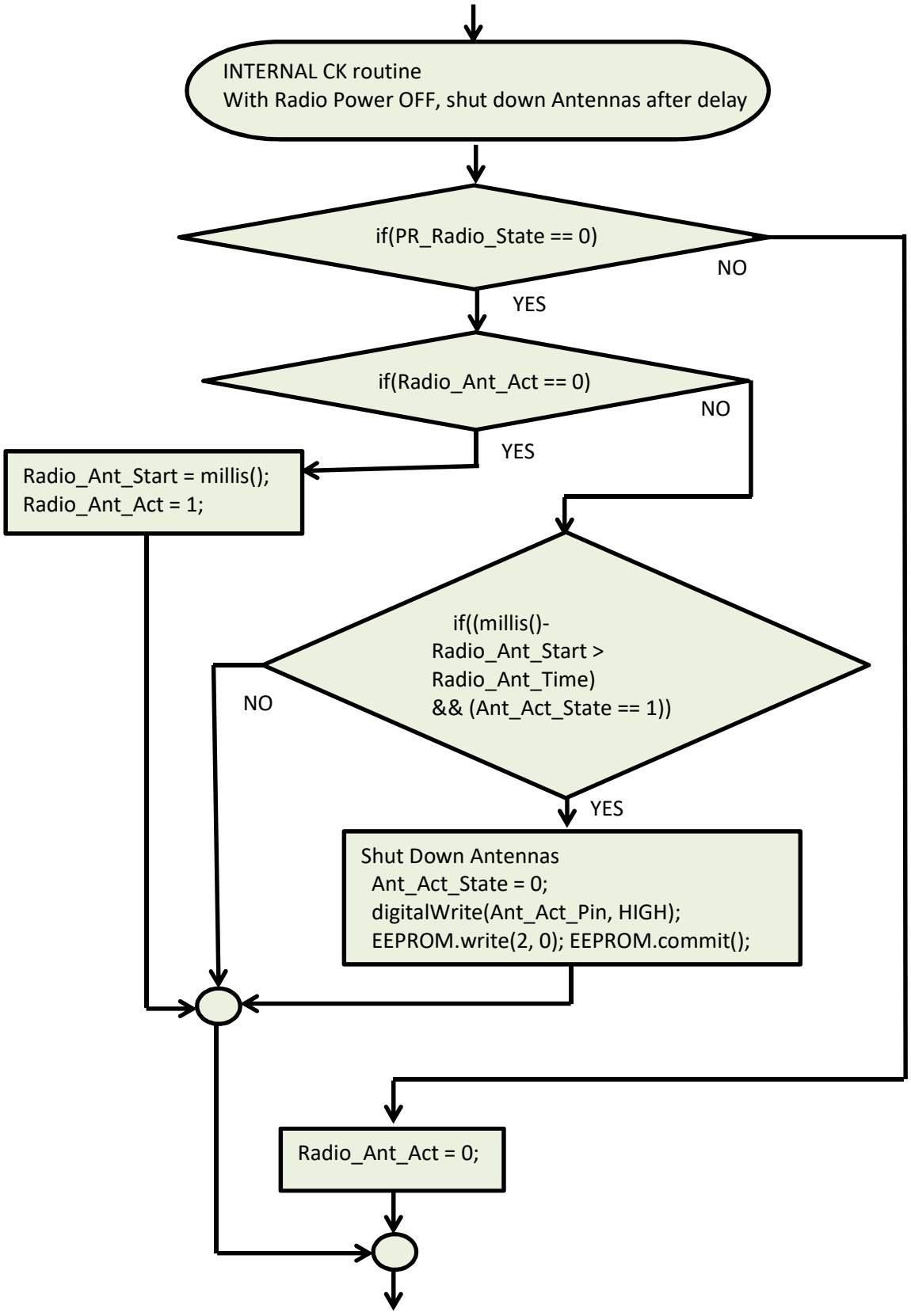

Else if (header.indexOf(
"GET /PR_Radio/can") >= 0)
NO
YES
Cancel OFF request and return to ON state
PR_Radio_State = 1;
CHECK for RigPi power
State 1 = output HIGH
if (header.indexOf(
"GET /PR_RigPi/on") >= 0)
NO
YES
Turn PR_RigPi ON and record in EEPROM
PR_RigPi_State = 1;
digitalWrite(PR_RigPi_Pin, HIGH);
EEPROM.write(1, 1); EEPROM.commit();
Else if ((header.indexOf(
"GET /PR_RigPi/off") >= 0)
NO
YES
Turn PR_RigPi OFF and record in EEPROM
PR_RigPi_State = 0;
digitalWrite(PR_RigPi_Pin, LOW);
EEPROM.write(1, 0); EEPROM.commit();
Else if ((header.indexOf(
"GET /PR_RigPi/ck") >= 0)
NO
YES

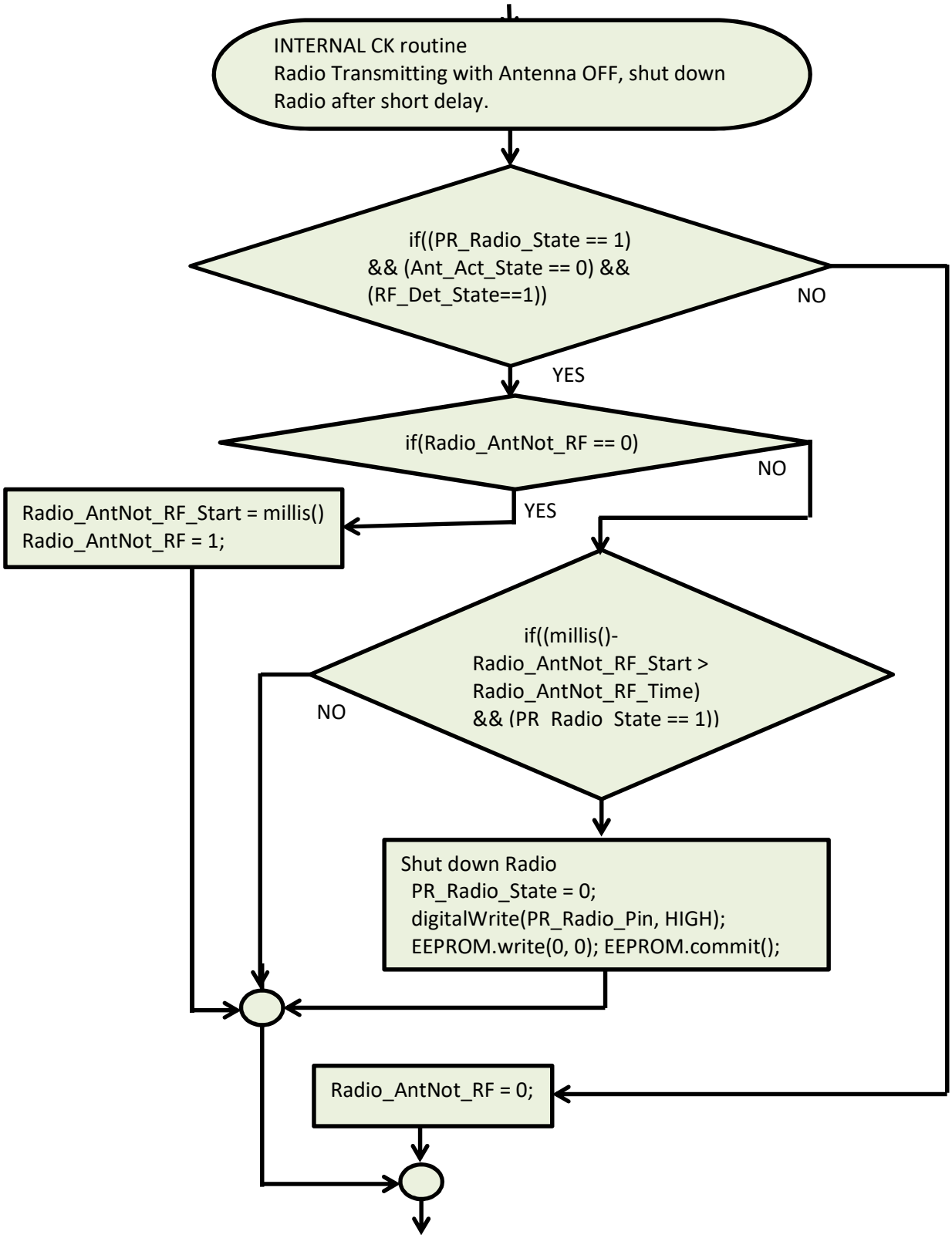

Display RigPi OFF / VERIFY and Warning screen
PR_Radio_State = 2;
Else if (header.indexOf(
"GET /PR_RigPi/can") >= 0)
NO
YES
Cancel OFF request and return to ON state
PR_RigPi_State = 1;
CHECK for Antenna Connect
State 1 = output LOW
if (header.indexOf(
"GET /Ant_Act/on") >= 0)
NO
YES
Turn Ant_Act ON and record in EEPROM
Ant_Act_State = 1;
digitalWrite(Ant_Act _Pin, LOW);
EEPROM.write(2, 1); EEPROM.commit();
Else if ((header.indexOf(
"GET /Ant_Act/off") >= 0)
NO
YES
Turn Ant_Act OFF and record in EEPROM
Ant_Act State = 0;
digitalWrite(Ant_Act _Pin, HIGH);
EEPROM.write(2, 0); EEPROM.commit();
Radio_Ant_Act = 0; //Clear ant off after radio flag

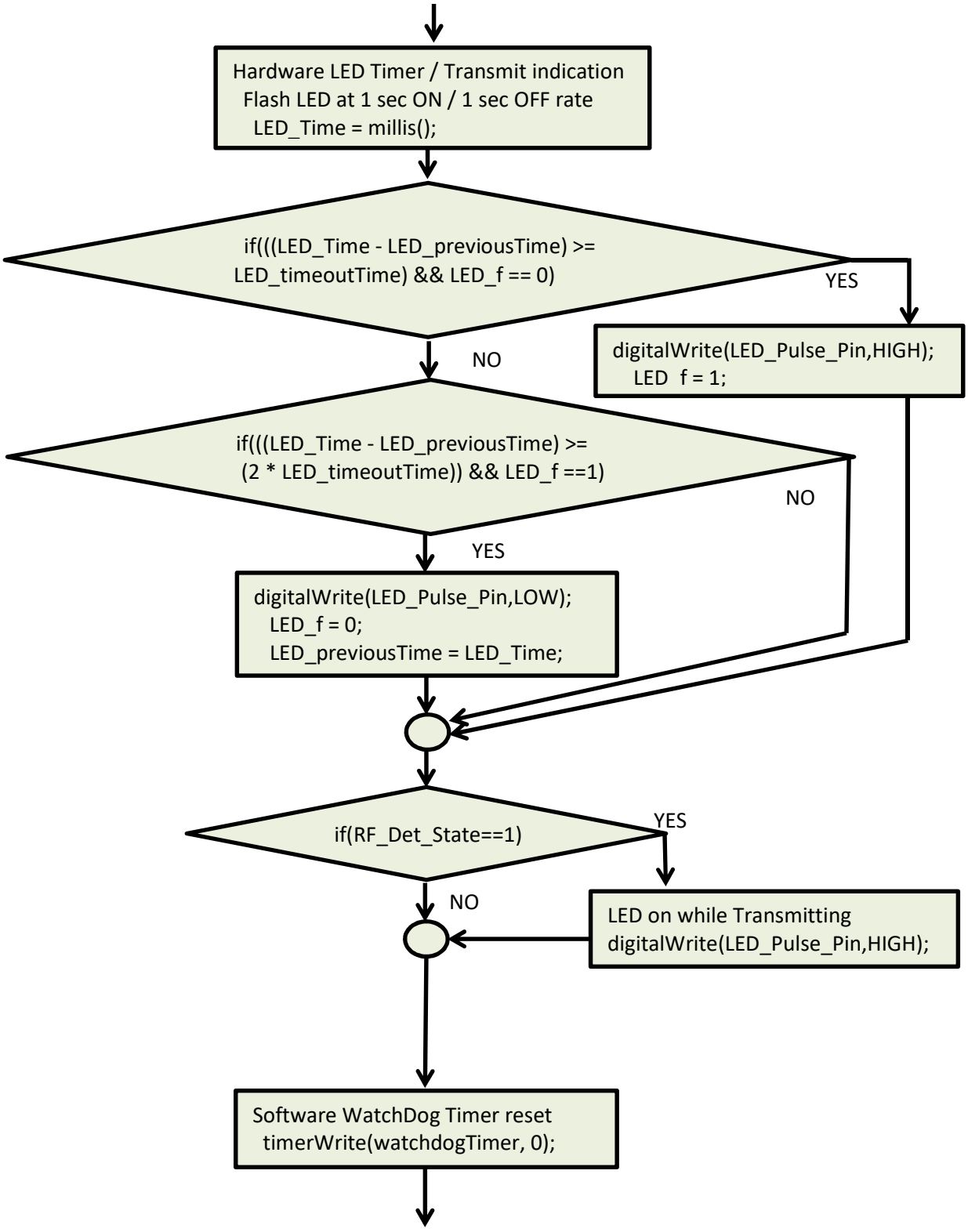

Read state of RF detectors and set
RF_Det_State = 1 if RF detected
RF_Det_State = 0 if no RF
If any of the RF_Det_Pin's = 0 then RF or transmit
mode is indicated (Only one input used in current Aux Ctl)
Start generation of web page to be returned to client:
client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");
client.println("<link rel=\"icon\" href=\"data:,\">");
CSS to style the buttons:
.button = Green used for ON [unless followed by additional button style]
. button2 = Red used for OFF
. button3 = Blue used for RF present
. buttonR = Black used for REFRESH button
. button4 = Gray used for No RF and on VERIFY screens
Web page uses a table with two columns titled with "your call" Aux Ctl
Device status display uses two data fields, first with Name of device, next
with the status by color and name. Status is determined by the _State of
each of the devices. Each status is a button which when pressed
generates a web request with the HTML GET method of data transfer. The
value of the GET is the request for the desired function.
Radio Power
If PR_Radio_State == 1, button Green, GET = "/PR_Radio/ck"
If PR_Radio_State == 2, Add row with Cancel and Shut OFF buttons
If Cancel pressed, generates GET = "/PR_Radio/on"
If Shut OFF pressed, generates GET = "/PR_Radio/off"
If PR_Radio_State == 0, button Red, GET = "/PR_Radio/on"

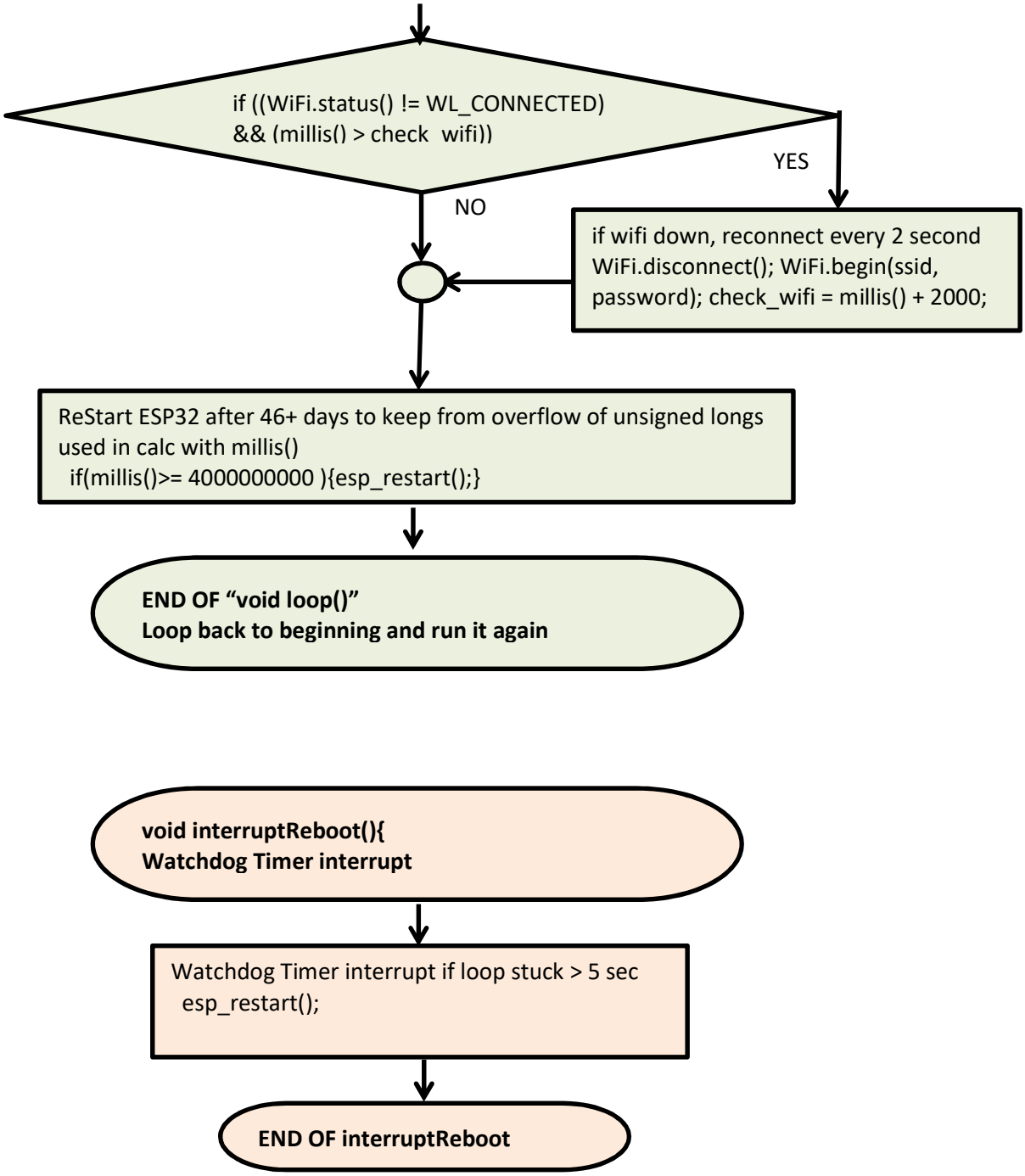

RigPi Power
If PR_ RigPi_State == 1, button Green, GET = "/PR_ RigPi/ck"
If PR_ RigPi_State == 2, Added row with WARNING message
about shutting down Pi power without program shutdown
Add row with Cancel and Shutdown buttons
If Cancel pressed, generates GET = "/PR_ RigPi/on"
If Shutdown pressed, generates GET = "/PR_ RigPi/off"
If PR_ RigPi_State == 0, button Red, GET = "/PR_ RigPi/on"
Antenna Connect
If Ant_Act_State == 1, button Green, GET = "/PR_Radio/off"
If Ant_Act_State == 0, button Red, GET = "/PR_Radio/on"
RF Detector
If RF_Det_State == 1, button Blue, No GET data
If RF_Det_State == 0, button Gray, No GET data
Refresh Display
Bottom button added to request a refresh of the web display
Web page finished and sent:
</table></body></html>
And a blank line
End of Evaluation of Request and Response to Web Client
Start Internal System Checks
A
p4


INTERNAL CK routine
RF locked ON, shut down radio after delay
Read RF_Det_Pins (read 3 but current Aux Ctl only uses one)
RF Detect state must remain ON or OFF for a minimum of 5 scans before action:
After 5 scans of ON:
Timer_Act = 1;
Time_Time = millis():
After 5 scans of OFF:
Timer_Act = 0;
Timer_Latch = 0;
if((Timer_Act==1)&&(Timer_Latch == 0))
NO
YES
NO
if((millis()-Timer_Time) >= Timer_ms)
YES
Timed out Shut Down Radio Power
PR_Radio_State = 0; digitalWrite(PR_Radio_Pin, HIGH);
EPROM.write(0, 0); EPROM.commit();
Timer_Latch = 1;


INTERNAL CK routine
With Radio Power OFF, shut down Antennas after delay
if(PR_Radio_State == 0)
NO
YES
if(Radio_Ant_Act == 0)
NO
YES
Radio_Ant_Start = millis();
Radio_Ant_Act = 1;
if((millis()-
Radio_Ant_Start >
Radio_Ant_Time)
&& (Ant_Act_State == 1))
NO
YES
Shut Down Antennas
Ant_Act_State = 0;
digitalWrite(Ant_Act_Pin, HIGH);
EEPROM.write(2, 0); EEPROM.commit();
Radio_Ant_Act = 0;


INTERNAL CK routine
Radio Transmitting with Antenna OFF, shut down
Radio after short delay.
if((PR_Radio_State == 1)
&& (Ant_Act_State == 0) &&
(RF_Det_State==1))
NO
YES
if(Radio_AntNot_RF == 0)
NO
YES
Radio_AntNot_RF_Start = millis()
Radio_AntNot_RF = 1;
if((millis()-
Radio_AntNot_RF_Start >
Radio_AntNot_RF_Time)
&& (PR Radio State == 1))
NO
Shut down Radio
PR_Radio_State = 0;
digitalWrite(PR_Radio_Pin, HIGH);
EEPROM.write(0, 0); EEPROM.commit();
Radio_AntNot_RF = 0;


Hardware LED Timer / Transmit indication
Flash LED at 1 sec ON / 1 sec OFF rate
LED_Time = millis();
if(((LED_Time - LED_previousTime) >= LED_timeoutTime) && LED_f == 0)
YES
NO
digitalWrite(LED_Pulse_Pin,HIGH);
LED f = 1;
if(((LED_Time - LED_previousTime) >= (2 * LED_timeoutTime)) && LED_f ==1)
NO
YES
digitalWrite(LED_Pulse_Pin,LOW);
LED_f = 0;
LED_previousTime = LED_Time;
if(RF_Det_State==1)
YES
NO
LED on while Transmitting
digitalWrite(LED_Pulse_Pin,HIGH);
Software WatchDog Timer reset
timerWrite(watchdogTimer, 0);


if ((WiFi.status() != WL_CONNECTED)
&& (millis() > check wifi))
YES
NO
if wifi down, reconnect every 2 second
WiFi.disconnect(); WiFi.begin(ssid,
password); check_wifi = millis() + 2000;
ReStart ESP32 after 46+ days to keep from overflow of unsigned longs
used in calc with millis()
if(millis()>= 4000000000){esp_restart();}
END OF "void loop()"
Loop back to beginning and run it again
void interruptReboot(){
Watchdog Timer interrupt
Watchdog Timer interrupt if loop stuck > 5 sec
esp_restart();
END OF interruptReboot

```
/*********
  Aux Control as independent  monitor of RF output from rig controlled by RigPi
   RigPi tm of W6HN
   Aux Control by WA3RTC 5/2020 based on:
     Rui Santo's RNT
     Complete project details at http://randomnerdtutorials.com

   RigPi_Aux05 06/22/2020 shp
*********/

// Load Wi-Fi library
#include <WiFi.h>

// Load flash memory library and define number of bytes of EEPROM used
#include <EEPROM.h>
#define EEPROM_SIZE 3

// set network credentials
const char* ssid     = "YOUR SSID";
const char* password = "YOUR PASWORD";
unsigned long check_wifi = 2000;

// For ESP32-2 (192.168.10.193) Set web server port number 8093 Fwd to 8888
WiFiServer server(8888);

// Variable to store the HTTP request as a String
String header;

/* Next 3 statements part of fix for Chrome not dropping connection*/
unsigned long currentTime = millis();
unsigned long previousTime = 0;
unsigned long timeoutTime = 2000;

// Auxiliay routine constants
const char* call = "YOUR CALL";
#define PR_Spare_Pin 27
#define PR_Radio_Pin 33
#define PR_RigPi_Pin 26
#define Ant_Act_Pin 25
#define RF_Det_Pin1 13
#define RF_Det_Pin2 12
#define RF_Det_Pin3 14
#define LED_Pulse_Pin 32

// Auxiliay routine State variables
int PR_Radio_State;
int PR_RigPi_State;
int Ant_Act_State;
int RF_Det_State;
```

```
// Antenna off if Radio power off
int Radio_Ant_Act = 0;
unsigned long Radio_Ant_Time = 900000; // Set timer for  15 min
unsigned long Radio_Ant_Start;

// RF Fail ON detect parameters
int Timer_Cnt = 0;
int Timer_Act = 0;
int Timer_Latch = 0;
float Timer_min = 2.9; // Power off timer in minutes
unsigned long Timer_ms = long(Timer_min * 60000);
unsigned long Timer_Time = 0;

// RF on with Ant Off detect parameters
int Radio_AntNot_RF = 0;
unsigned long Radio_AntNot_RF_Start = millis();
unsigned long Radio_AntNot_RF_Time = 20000;   //If Antenna off and RF detected for this time (msec)
                                                Shut down Rado

// Hardware LED flash rate
unsigned long LED_Time = millis();
unsigned long LED_previousTime = 0;
unsigned long LED_timeoutTime = 1000;  //Timeout time in milliseconds
int LED_f = 0;

// Software Watchdog parameters
hw_timer_t *watchdogTimer = NULL;
long looptime=0;

//*********************************
void setup() {
 Serial.begin(115200);

 // initialize EEPROM with predefined size
 EEPROM.begin(EEPROM_SIZE);
 //Set Relay states to last states before reset from EPROM memory
   PR_Radio_State = EEPROM.read(0);
   PR_RigPi_State = EEPROM.read(1);
   Ant_Act_State = EEPROM.read(2);
 // if first time ESP32 unit run, set memory
    if((PR_Radio_State > 2) || (PR_Radio_State < 0)){EEPROM.write(0, 0); EEPROM.commit();}
    if((PR_RigPi_State > 2) || (PR_RigPi_State < 0)){EEPROM.write(1, 0); EEPROM.commit();}
    if((Ant_Act_State > 1) || (Ant_Act_State < 0)){EEPROM.write(2, 0); EEPROM.commit();}

 // Initialize PR_Radio I/O to output in State before reset
 pinMode(PR_Radio_Pin, OUTPUT);
 if (PR_Radio_State == 1) {
   digitalWrite(PR_Radio_Pin, LOW);
 }
```

```
else {
  digitalWrite(PR_Radio_Pin, HIGH);
}

// Initialize PR_RigPi I/O to output in State before reset
pinMode(PR_RigPi_Pin, OUTPUT);
if (PR_RigPi_State == 1) {
  digitalWrite(PR_RigPi_Pin, HIGH);
}
else {
  digitalWrite(PR_RigPi_Pin, LOW);
}
// Initialize Ant_Act I/O to output in State before reset
pinMode(Ant_Act_Pin, OUTPUT);
if (Ant_Act_State == 1) {
  digitalWrite(Ant_Act_Pin, LOW);
}
else {
  digitalWrite(Ant_Act_Pin, HIGH);
}
// Initialize PR_Spare I/O as output and de-energized
pinMode(PR_Spare_Pin, OUTPUT);
digitalWrite(PR_Spare_Pin, HIGH);

// Initialize RF_Det as inputs
pinMode(RF_Det_Pin1,INPUT_PULLUP);
pinMode(RF_Det_Pin2,INPUT_PULLUP);
pinMode(RF_Det_Pin3,INPUT_PULLUP);

// Initialize hardware LED I/O
pinMode(LED_Pulse_Pin,OUTPUT);
digitalWrite(LED_Pulse_Pin, LOW);

// ESP32 Tutorial - Watchdog Timer by Jack Rickard  https://www.youtube.com/watch?v=7kLy2iwIvy8
// Initialize Software Watchdog (5000000 = 5sec)
watchdogTimer = timerBegin(0, 80, true);  //timer 0, divisor 80
timerAlarmWrite(watchdogTimer, 5000000, false); // set time in uS must be fed with in this time
                                                   or reboot
timerAttachInterrupt(watchdogTimer, &interruptReboot, true);
timerAlarmEnable(watchdogTimer);  // enable interrupt

// Connect to Wi-Fi network with SSID and password
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
}
// Print local IP and mac addresses
Serial.print("IP address: ");
```

```
  Serial.print(WiFi.localIP());
  Serial.println(":8888");
  Serial.print("ESP Board MAC Address:  ");
  Serial.println(WiFi.macAddress());

  // start web server
  server.begin();
}

//*******************************
void loop() {
  WiFiClient client = server.available();   // Listen for incoming clients

  if (client) {                     // If a new client connects,
  currentTime = millis();
  previousTime = currentTime;
    String currentLine = "";             // make a String to hold incoming data from the client
    while (client.connected() && currentTime - previousTime <= timeoutTime) { // loop while the client's
connected
    currentTime = millis();
     if (client.available()) {           // if there's bytes to read from the client,
       char c = client.read();           // read a byte, then
       header += c;
       if (c == '\n') {                  // if the byte is a newline character
        // if the current line is blank, you got two newline characters in a row.
        // that's the end of the client HTTP request, so send a response:
        if (currentLine.length() == 0) {
          // checking if header is valid
          // •••••••••••••••••• = •••••:••••• base64 encode
          // Finding the right credential string, then loads web page
          if (header.indexOf("••••••••••••••••••") >= 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
            // and a content-type so the client knows what's coming, then a blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

// USE the "GET/xx" HTML header to select action and set _State
        // turns the PR_Radio GPIO on and off (ENERGIZE [ON] = Power ON to Radio)
        // "Timer_Latch" part of function which turns Radio OFF if RF locked ON
            if ((header.indexOf("GET /PR_Radio/on") >= 0)&&(Timer_Latch==0)) {
//("PR_Radio on")
              PR_Radio_State = 1;
              digitalWrite(PR_Radio_Pin, LOW);
                EEPROM.write(0, 1);
                EEPROM.commit();
            } else if (header.indexOf("GET /PR_Radio/off") >= 0) {
//("PR_Radio off")
              PR_Radio_State = 0;
```

```
            digitalWrite(PR_Radio_Pin, HIGH);
              EEPROM.write(0, 0);
              EEPROM.commit();
            } else if (header.indexOf("GET /PR_Radio/ck") >= 0) {
//"PR_Radio ck") can = cancel and return to ON State
              PR_Radio_State = 2;
            } else if (header.indexOf("GET /PR_Radio/can") >= 0) {
//("PR_Radio on")
              PR_Radio_State = 1;
            }

        // turns the PR_RigPi GPIO on and off (ENERGIZE [LOW] = Power OFF to RigPi)
            if (header.indexOf("GET /PR_RigPi/on") >= 0) {
//("PR_RigPi on")
              PR_RigPi_State = 1;
              digitalWrite(PR_RigPi_Pin, HIGH);
              EEPROM.write(1, 1);
              EEPROM.commit();
            } else if (header.indexOf("GET /PR_RigPi/off") >= 0) {
//("PR_RigPi off")
              PR_RigPi_State = 0;
              digitalWrite(PR_RigPi_Pin, LOW);
              EEPROM.write(1, 0);
              EEPROM.commit();
            } else if (header.indexOf("GET /PR_RigPi/ck") >= 0) {
//("PR_RigPi ck") can = cancel and return to de-energized (1) State
              PR_RigPi_State = 2;
            } else if (header.indexOf("GET /PR_RigPi/can") >= 0) {
//("PR_RigPi on")
              PR_RigPi_State = 1;
            }

        // Ant_Act_State control
            if (header.indexOf("GET /Ant_Act/on") >= 0) {
//("Ant_Act on")
              Ant_Act_State = 1;
              digitalWrite(Ant_Act_Pin, LOW);
              EEPROM.write(2, 1);
              EEPROM.commit();
              Radio_Ant_Act = 0; //Clear antenna off after radio flag
            } else if (header.indexOf("GET /Ant_Act/off") >= 0) {
//("Ant_Act off")
              Ant_Act_State = 0;
              digitalWrite(Ant_Act_Pin, HIGH);
              EEPROM.write(2, 0);
              EEPROM.commit();
              Radio_Ant_Act = 0; //Clear antenna off after radio flag
            }

        // Reads state of RF detector: 1 = RF present, 0 = no RF
```

```
if((digitalRead(RF_Det_Pin3))&&(digitalRead(RF_Det_Pin2))&&(digitalRead(RF_Det_Pin1))){RF_Det_Stat
e = 0;}
          else{RF_Det_State=1;}

        // Display the HTML RigPi Aux Ctl web page
          client.println("<!DOCTYPE html><html>");
          client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-
scale=1\">");
          client.println("<link rel=\"icon\" href=\"data:,\">");

        // CSS to style the on/off buttons
          // background-color and font-size attributes set here
          client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-
align: center; font-size: 30px;}");
          client.println(".button { background-color: green; border: none; color: white; padding: 2px;");
          client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");
          client.println(".button2 {background-color: red;}");
          client.println(".button3 {background-color: blue;}");
          client.println(".buttonR {background-color: black;}");
          client.println(".button4 {background-color: gray;}</style></head>");

        // Web Page Heading in table of "<your call> Aux Ctl"
          client.println("<body><table BORDER = 3 Width = 300px><tr><th colspan = 2>");
          client.print("<b>");
          client.print(call);
          client.println(" Aux Ctl</b></th>");
          client.println("<tr><th Width = 200px><b>Device</b><th><b>Status</b></th></tr>");

        // Display current state, and ON/OFF buttons for PR_Radio, Verify for OFF Request
          client.println("<tr><td>Radio Power</td>");
          // If the PR_Radio_State is ON, it displays the GREEN button
          if (PR_Radio_State == 1) {
            client.println("<td><a href=\"/PR_Radio/ck\"><button class=\"button\">ON
</button></a></td></tr>");
            // If the PR_Radio_State is OFF, it displays the RED button2
          } else if (PR_Radio_State == 0) {
            client.println("<td><a href=\"/PR_Radio/on\"><button class=\"button
button2\">OFF</button></a></td></tr>");
          } else if (PR_Radio_State == 2) {
            // If the PR_Radio_State is ck, it displays the GRAY buttons(4) to cancel or verify
            client.println("<td><button class=\"button button3\">VERIFY</button></td></tr>");
            client.println("<tr><td><a href=\"/PR_Radio/can\"><button class=\"button
button4\">Cancel</button></a></td>");
            client.println("<td><a href=\"/PR_Radio/off\"><button class=\"button button4\">Shut
down</button></a></td></tr>");
          }

        // Display current state, and ON/OFF buttons for PR_RigPi, Verify & Warn for OFF Request
          client.println("<tr><td>RigPi Power</td>");
```

```
          // If the PR_RigPi_State is ON, it displays the GREEN button
          if (PR_RigPi_State == 1) {
            client.println("<td><a href=\"/PR_RigPi/ck\"><button class=\"button\">ON
</button></a></td></tr>");
            // If the PR_RigPi_State is OFF, it displays the RED button2
          } else if (PR_RigPi_State == 0) {
            client.println("<td><a href=\"/PR_RigPi/on\"><button class=\"button
button2\">OFF</button></a></td></tr>");
          } else if (PR_RigPi_State == 2) {
            // If the PR_RigPi_State is ck, it displays the GRAY buttons(4) to cancel or verify
            client.println("<td><button class=\"button button3\">VERIFY</button></td></tr>");
            // Warning message to shut down RigPi before power OFF
            client.println("<tr><td colspan = 2><button class=\"button button2\">Power Down RigPi and
wait 30 seconds before Power Shut Down</button></td></tr>");
            client.println("<tr><td><a href=\"/PR_RigPi/can\"><button class=\"button
button4\">Cancel</button></a></td>");
            client.println("<td><a href=\"/PR_RigPi/off\"><button class=\"button button4\">Shut
down</button></a></td></tr>");
          }

        // Display current state, and ON/OFF buttons for Antenna Grounding Relay
          client.println("<tr><td>Antenna Act</td>");
          // If the Ant_Act_State is ON, it displays the GREEN button
          if (Ant_Act_State == 1) {
            client.println("<td><a href=\"/Ant_Act/off\"><button class=\"button\">ON
</button></a></td></tr>");
            // If the Ant_Act_State is OFF, it displays the RED button2
          } else if (Ant_Act_State == 0) {
            client.println("<td><a href=\"/Ant_Act/on\"><button class=\"button
button2\">OFF</button></a></td></tr>");
          }

        // Display current state of RF from detector
          client.println("<tr><td>RF Detector</td>");
          // Displays the BLUE for RF detected, Gray for no RF. Push RF to clear last Request
//(RF_Det_State)
          if (RF_Det_State == 0) {
            client.println("<td><a href=\"/\"><button class=\"button
button4\">RF</button></a></td></tr>");
          } else {
            client.println("<td><a href=\"/\"><button class=\"button
button3\">RF</button></a></td></tr>");
          }

        // Request Refresh of screen in BLACK across bottom of Webpage table
          client.println("<tr><td colspan=2><a href=\"/\"><button class=\"button buttonR\">Refresh
Screen</button></a></td></tr>");


        // The HTTP response ends by closeing table, body, html and another blank line
```

```
            client.println("</table>");
            client.println("</body></html>");
            client.println();

      // Break out of the while loop
            break;
           }
//************
  // Wrong user or password, so HTTP request fails...
           else {
            client.println("HTTP/1.1 401 Unauthorized");
            client.println("WWW-Authenticate: Basic realm=\"Secure\"");
            client.println("Content-Type: text/html");
            client.println();
            client.println("<html>Authentication failed</html>");
           }
         } else { // if you got a newline, then clear currentLine
           currentLine = "";
         }
       } else if (c != '\r') {  // if you got anything else but a carriage return character,
         currentLine += c;      // add it to the end of the currentLine
       }
     }
   }
   // Clear the header variable
   header = "";
   // Close the connection
   client.stop();
//("Client disconnected.")
  }

//************
// INTERNAL CK: RF locked ON, shut down Radio after delay
//Timer for RF output debounce ON/OFF with 5 loop passes

if((digitalRead(RF_Det_Pin3))&&(digitalRead(RF_Det_Pin2))&&(digitalRead(RF_Det_Pin1))){RF_Det_Stat
e = 0;}
  else{RF_Det_State=1;}

  if(RF_Det_State==1){
     Timer_Cnt++;
     if(Timer_Cnt >= 5){
       Timer_Cnt=5;
       if(Timer_Act == 0){
         Timer_Time = millis();
         Timer_Act = 1;}
     }
   }else{
     Timer_Cnt--;
     if(Timer_Cnt <= 0){
```

```
      Timer_Cnt = 0;
      Timer_Act = 0;
      Timer_Latch = 0;}
   }

   if((Timer_Act==1)&&(Timer_Latch == 0)){
    if((millis()-Timer_Time) >= Timer_ms){
      PR_Radio_State = 0;
      digitalWrite(PR_Radio_Pin, HIGH);
       EEPROM.write(0, 0);
       EEPROM.commit();
      Timer_Latch = 1;
//("Timer timeout")
      }
   }
//************
// INTERNAL CK: Radio Power OFF then shut down Antennas after delay
// If Radio Power is down for Radio_Ant_Time mSec then shut down Antenna
 if(PR_Radio_State == 0){
   if(Radio_Ant_Act == 0){
    Radio_Ant_Start = millis();
    Radio_Ant_Act = 1;
   }else{
    if((millis()-Radio_Ant_Start > Radio_Ant_Time)&&(Ant_Act_State == 1)){
      Ant_Act_State = 0;
      digitalWrite(Ant_Act_Pin, HIGH);
       EEPROM.write(2, 0);
       EEPROM.commit();
    }
   }
 }else{
   Radio_Ant_Act = 0;
 }

//************
// INTERNAL CK: Radio ON and Transmitting with Antenna OFF, shut down Radio after delay
// If Radio Power is ON AND Ant_Act OFF AND RF_Det_State ON for Radio_AntNot_RF_Time mSec
   then shut down Radio Power
 if((PR_Radio_State == 1)&&(Ant_Act_State == 0)&&(RF_Det_State==1)){
   if(Radio_AntNot_RF == 0){
    Radio_AntNot_RF_Start = millis();
    Radio_AntNot_RF = 1;
   }else{
    if((millis()-Radio_AntNot_RF_Start > Radio_AntNot_RF_Time)&&(PR_Radio_State == 1)){
      PR_Radio_State = 0;
      digitalWrite(PR_Radio_Pin, HIGH);
       EEPROM.write(0, 0);
       EEPROM.commit();
    }
   }
```

```
 }else{
   Radio_AntNot_RF = 0;
 }

//************
// Flash LED at 1 sec ON / 1 sec OFF rate
// LED delays while responding to Web inputs
// LED ON if PTT active
// Hardware LED Timer / Xmit indication
 LED_Time = millis();
 if(((LED_Time - LED_previousTime) >= LED_timeoutTime) && LED_f == 0){
   //Turn on LED Pulse
   digitalWrite(LED_Pulse_Pin,HIGH);
   LED_f = 1;
 }
 if(((LED_Time - LED_previousTime) >= (2 * LED_timeoutTime)) && LED_f ==1){
   //Turn off LED Pulse and reset previous time value
     digitalWrite(LED_Pulse_Pin,LOW);
      LED_f = 0;
     LED_previousTime = LED_Time;
 }
 // LED on while Transmitting
 if(RF_Det_State==1){
   digitalWrite(LED_Pulse_Pin,HIGH);}

//************
// Software WatchDog Timer reset
 timerWrite(watchdogTimer, 0);

//************
 // if wifi is down, try reconnecting every 2 second
 if ((WiFi.status() != WL_CONNECTED) && (millis() > check_wifi)) {
//(" ... Reconnecting to WiFi...")
   WiFi.disconnect();
   WiFi.begin(ssid, password);
   check_wifi = millis() + 2000;
 }
 //ReStart ESP32 after 46+ days to keep from overflow of unsigned longs
 //used in calc with millis()
 if(millis()>= 4000000000 ){esp_restart();}
}

void interruptReboot(){
 //Watchdog Timer interrupt if loop stuck > 5 sec
 esp_restart();
}
```