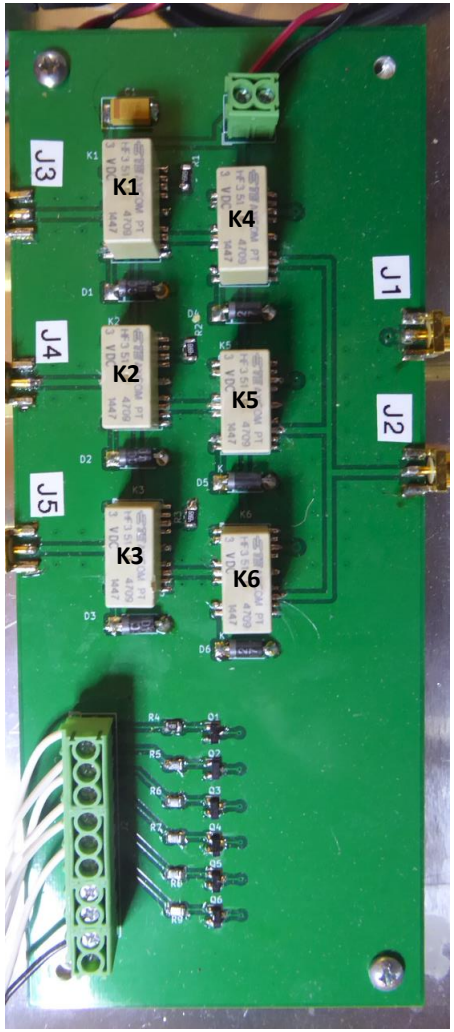
100W 3:2 HF Switch Matrix

Having three antennas and two HF rigs located in different parts of the house presented the opportunity to design a remote controlled switch matrix that would allow connecting of any of the three antennas to the two rigs.

Described here is a 3:2 HF switch matrix controlled via a RasbperryPi 3B+. A rudimentary web server is used to allow control from a web page.

The critical RF design criteria are VSWR, isolation and power handling capability. My two rigs are low power (<=100W) therefore the switch matrix was designed to handle 150W.

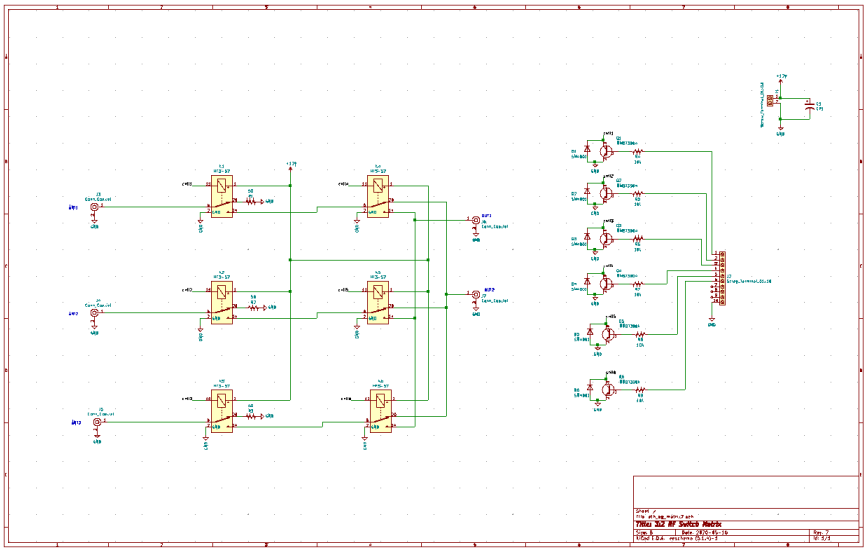
Six relays are mounted on a 4-layer FR4 PCB. PCBWay (https://www.pcbway.com/), a low cost PCB fab located in China was used to manufacture the PCB. To keep the cost low, the default 4-layer stackup was used. This resulted in acceptable VSWR at 30 MHz.



100W 3:2 HF Switch Matrix

Most parts were purchased from DigiKey with the exception of the edge launch SMA connectors. I had a stash of these acquired from a surplus house.
Ebay may be a possibility for lower cost connectors.

Ref Des	Vendor Part Number	DigiKey Part Number	Description	Vendor	Cost Ea.
K1 - K6	1462051-1	PB1102CT-ND	RF Relay	TE Connectivity	$14.25
Q1 - Q6	MMBT2222ALT1G	MMBT2222ALT1GOSCT-ND	NPN Transistor	Onsemi	$0.18
D1 - D6	1N4001-G	641-1310-1-ND	Diode	Comchip Technology	$0.21
R1 - R3	RNCP0805FTD49R9	RNCP0805FTD49R9CT-ND	50Ω Resistor	Stackpole Electronics	$0.10
R4 - R9	RNCP0805FTD10K0	RNCP0805FTD10K0CT-ND	10kΩ Resistor	Stackpole Electronics	$0.10
J1	1984617	277-1721-ND	2 Pos Terminal Block Connector	Phoenix Contact	$0.55
J2	1984691	277-1727-ND	10 Pos Terminal Block Connector	Phoenix Contact	$2.59
J3 - J7	142-0711-821	J629-ND	SMA Edge Launch Connector	Cinch Connectivity Solutions Johnson	$7.20



100W 3:2 HF Switch Matrix

The default 4-layer PCB stackup is shown in Figure 1. The relatively thin first and third dielectrics make the default 50 ohm microstrip trace width quite narrow. To allow for the voltage and current requirements (Figure 2), a 30 mil CPWG (co-planar waveguide) linewidth was chosen (Figure 3). This results in an impedance of 22.5Ω (VSWR = 2.2). Being that the PCB traces are relatively short with respect to a wavelength at 30MHz, this is an acceptable tradeoff.

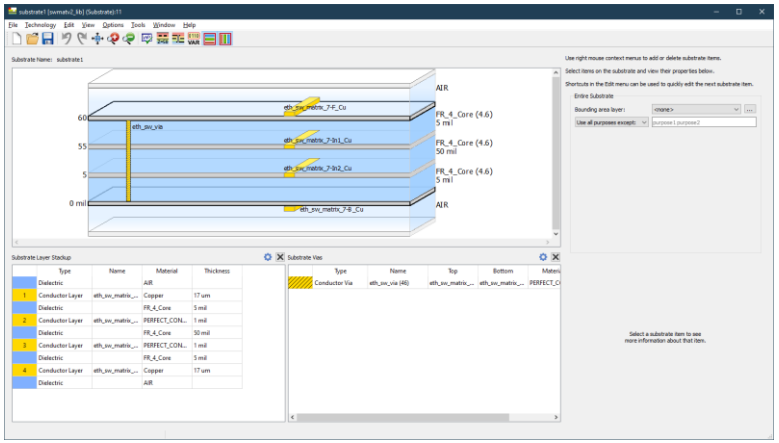


Figure 1 – 4 layer PCB stackup

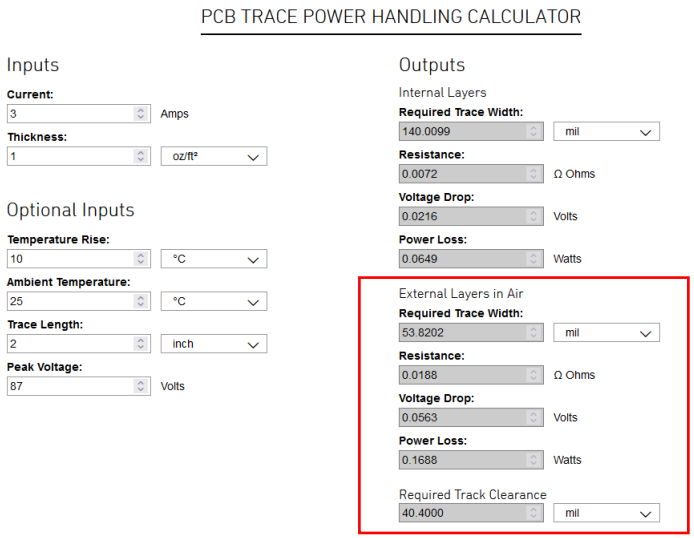


Figure 2 – PCB trace power calculation

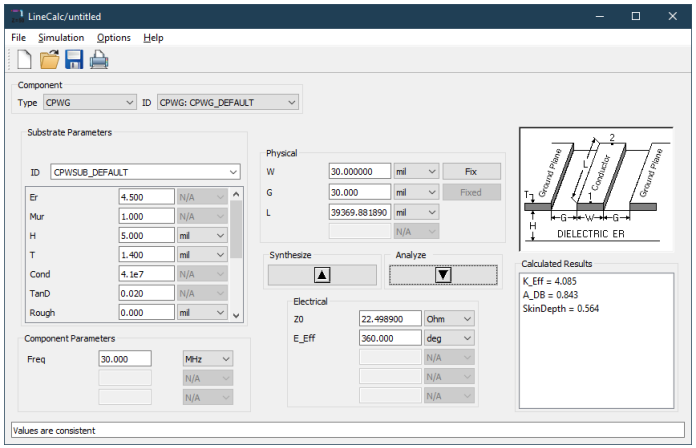


Figure 3 – PCB trace impedance

100W 3:2 HF Switch Matrix

The schematic / layout was first performed in KiCad, then the individual layers were imported into Keysight ADS and an EM (Keysight Momentum) simulation of the layout was performed to get a handle on the expected port return losses and port to port isolation.

Having used Keysight ADS in my prior life (RFIC design), the simulation setup was fairly easy. Keysight is very good at allowing experimenters 30 day trial licenses for projects such as this. The only issue with this sort of simulation is that it is very memory intensive. My computer setup consists of a 16GB i7 machine running Win10 Pro.

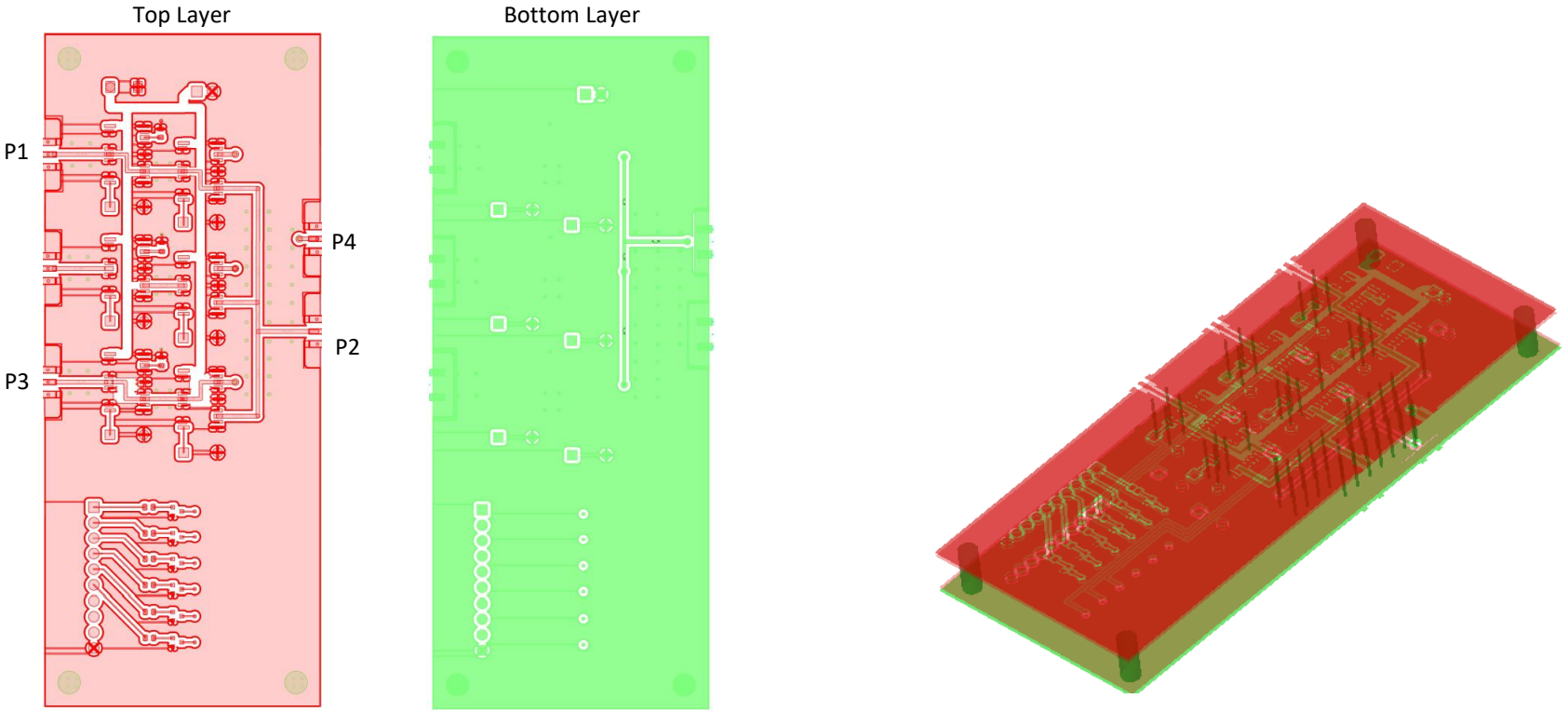


Figure 4 – PCB layout

100W 3:2 HF Switch Matrix

EM simulation results (Figure 5) show port VSWR below 1.5:1 and port to port isolation below 60dB @ 30 MHz.

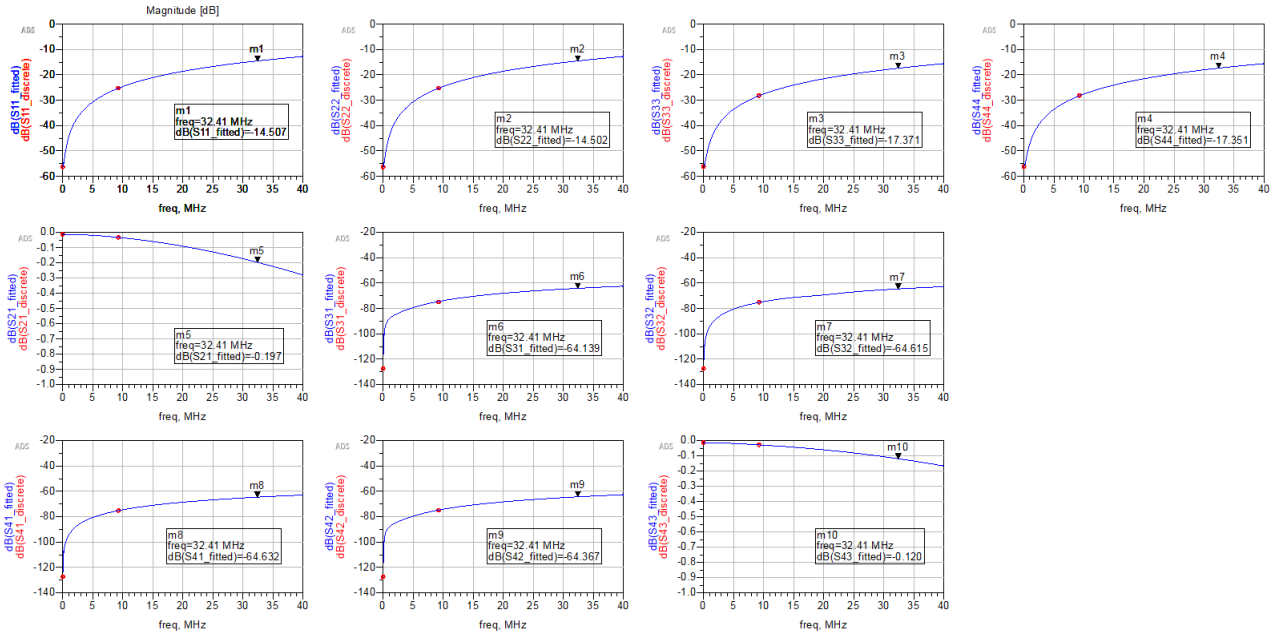


Figure 5 – EM Simulation Results

100W 3:2 HF Switch Matrix

Bench measurements were performed on the finished switch matrix with an SDR Kits VNWA. Simulated vs measured results are shown in Figure 6.

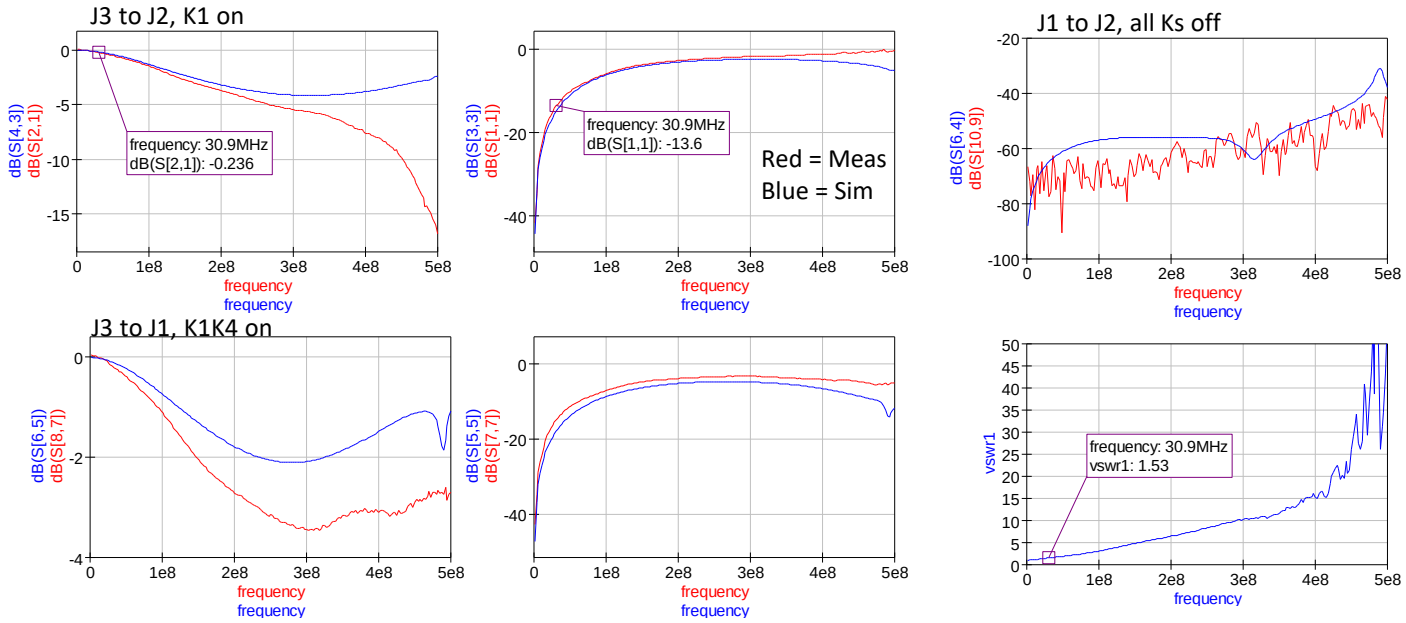


Figure 6 – Measured vs Simulation Results

100W 3:2 HF Switch Matrix

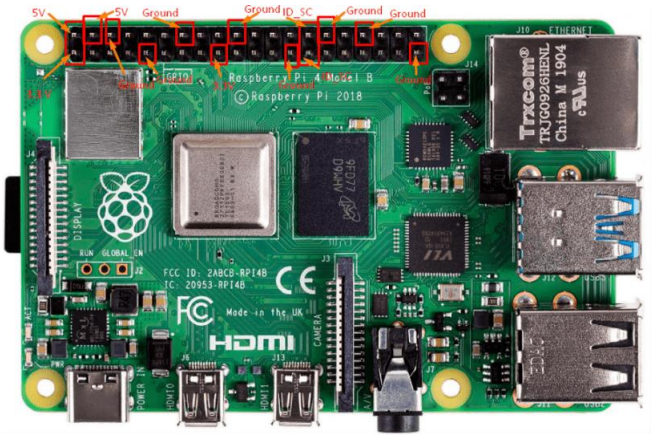
The Raspberry Pi GPIO outputs are 0 / 3.3V. The GPIOs used to control the relays are shown in Figure 9.

The standard Raspberry Pi OS is used to implement the control program. (https://www.raspberrypi.com/software/operating-systems/). Using the version with desktop and recommended software allows for implementing remote desktop and includes python3. Install 'xrdp' to utilize windows like remote desktop.

My setup uses the ethernet connection, the Raspberry Pi 3 also includes a WiFi option. Setup via the raspi-config command line routine.

Flask must also be installed using apt.

```
sudo apt-get install python3-flask
sudo apt-get install xrdp
```



	PIN	NAME		NAME	PIN	
	01	3.3V DC Power		5V DC Power	02	
	03	GPIO02 (SDA1,I²C)		5V DC Power	04	
	05	GPIO03 (SDL1,I²C)		Ground	06	
	07	GPIO04 (GPCLK0)		GPIO14 (TXD0, UART)	08	
	09	Ground		GPIO15 (RXD0, UART)	10	
Relay 1	11	GPIO17		GPIO18(PWM0)	12	Relay 2
Relay 3	13	GPIO27		Ground	14	
Relay 4	15	GPIO22		GPIO23	16	Relay 5
	17	3.3V DC Power		GPIO24	18	Relay 6
	19	GPIO10 (SP10_MOSI)		Ground	20	
	21	GPIO09 (SP10_MISO)		GPIO25	22	
	23	GPIO11 (SP10_CLK)		GPIO08 (SPI0_CE0_N)	24	
	25	Ground		GPIO07 (SPI0_CE1_N)	26	
	27	GPIO00 (SDA0, I²C)		GPIO07 (SCL0, I²C)	28	
	29	GPIO05		Ground	30	
	31	GPIO06		GPIO12 (PWM0)	32	
	33	GPIO13 (PWM1)		Ground	34	
	35	GPIO19		GPIO16	36	
	37	GPIO26		GPIO20	38	
	39	Ground		GPIO21	40	

Figure 9 – Raspberry Pi GPIO pinout

100W 3:2 HF Switch Matrix

I have yet to master website design although have been become somewhat proficient in Python with the Raspberry Pi OS. Python has a simple web server package called Flask (https://pythonbasics.org/what-is-flask-python/).

With this package, It is quite easy to design a simple web server. The web server provides control of each of the relays by pressing the respective button. The webpage is initiated by inserting the IP address in a web browser.

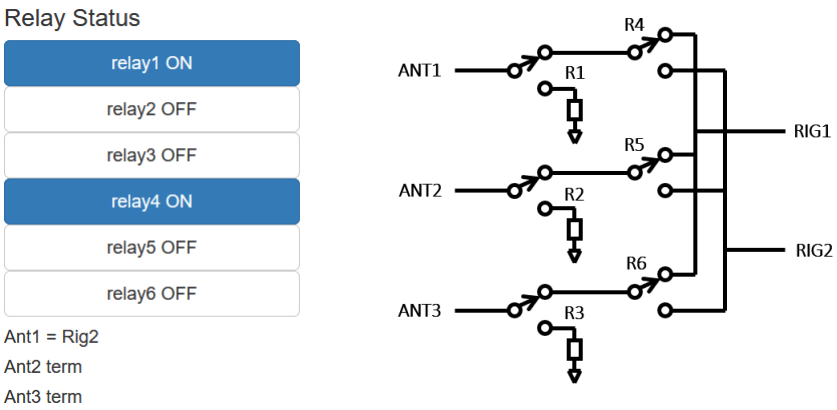


Figure 7 – Flask webpage

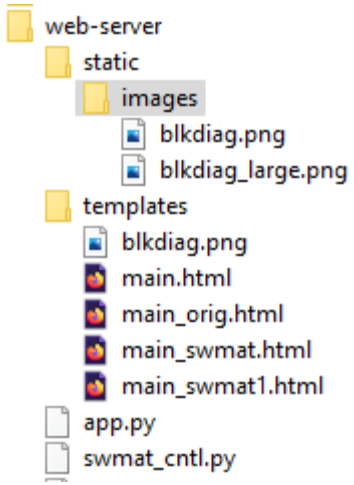


Figure 8 – Raspberry Pi Flask Directory Structure

The Flask directory structure is shown in Figure 8. The server is initiated by running the command 'sudo python3 swmat_cntl.py' from the web-server directory.

All of the web server files and KiCad files are available on my git hub page:

https://github.com/greggdaug

100W 3:2 HF Switch Matrix

swmat_cntl.py

```
import RPi.GPIO as GPIO
from flask import Flask, render_template, request
app = Flask(__name__)

GPIO.setmode(GPIO.BCM)

# Create a dictionary called pins to store the pin number, name, and pin state:
# GPIO 17 = GPIO_GEN0
# GPIO 18 = GPIO_GEN1
# GPIO 27 = GPIO_GEN2
# GPIO 22 = GPIO_GEN3
# GPIO 23 = GPIO_GEN4
# GPIO 24 = GPIO_GEN5

pins = {
   17 : {'name' : 'relay1', 'state' : GPIO.LOW},
   18 : {'name' : 'relay2', 'state' : GPIO.LOW},
   27 : {'name' : 'relay3', 'state' : GPIO.LOW},
   22 : {'name' : 'relay4', 'state' : GPIO.LOW},
   23 : {'name' : 'relay5', 'state' : GPIO.LOW},
   24 : {'name' : 'relay6', 'state' : GPIO.LOW}
   }

# Set each pin as an output and make it low:
for pin in pins:
   GPIO.setup(pin, GPIO.OUT)
   GPIO.output(pin, GPIO.LOW)

@app.route("/")
def main():
   # For each pin, read the pin state and store it in the pins dictionary:
   for pin in pins:
      pins[pin]['state'] = GPIO.input(pin)
   # Put the pin dictionary into the template data dictionary:
   templateData = {
      'pins' : pins
      }
   # Pass the template data into the template main.html and return it to the user
   return render_template('main.html', **templateData)

# The function below is executed when someone requests a URL with the pin number and action in it:
@app.route("/<changePin>/<action>")
def action(changePin, action):
   # Convert the pin from the URL into an integer:
   changePin = int(changePin)
   # Get the device name for the pin being changed:
   deviceName = pins[changePin]['name']
   # If the action part of the URL is "on," execute the code indented below:
   if action == "on":
      # Set the pin high:
      GPIO.output(changePin, GPIO.HIGH)
      # Save the status message to be passed into the template:
      message = "Turned " + deviceName + " on."
   if action == "off":
      GPIO.output(changePin, GPIO.LOW)
      message = "Turned " + deviceName + " off."

   # For each pin, read the pin state and store it in the pins dictionary:
   for pin in pins:
      pins[pin]['state'] = GPIO.input(pin)

   # Along with the pin dictionary, put the message into the template data dictionary:
   templateData = {
      'pins' : pins
   }

   return render_template('main.html', **templateData)

if __name__ == "__main__":
   app.run(host='0.0.0.0', port=80, debug=True)
```

100W 3:2 HF Switch Matrix

main.html

```
<!DOCTYPE html>
<head>
   <title>RF Switch Matrix Control</title>
   <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
   <!-- Latest compiled and minified CSS -->
   <!-- <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" integrity="sha384-
1q8mTJOASx8j1Au+a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7" crossorigin="anonymous"> -->
   <!-- Optional theme -->
   <!-- <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-theme.min.css" integrity="sha384-
fLW2N01lMqjakBkx3l/M9EahuwpSfeNvV63J5ezn3uZzapT0u7EYsXMjQV+0En5r" crossorigin="anonymous"> -->
   <!-- Latest compiled and minified JavaScript -->
   <!-- <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js" integrity="sha384-0mSbJDEHialfmuBBQP6A4Qrprq5OVfW37PRR3j5ELqxss1yVqOtnepnHVP9aJ7xS"
crossorigin="anonymous"></script> -->
</head>

<body>
   <h3>Relay Status</h3>
   {% for pin in pins %}
   {% if pins[pin].state == false %}
      <div class="row"><div class="col-md-2">
      <a href="/{{pin}}/on" class="btn btn-block btn-lg btn-default" role="button">{{ pins[pin].name }} OFF</a></div></div>
   {% else %}
      <div class="row"><div class="col-md-2">
      <a href="/{{pin}}/off" class="btn btn-block btn-lg btn-primary" role="button">{{ pins[pin].name }} ON</a></div></div>
   {% endif %}
   {% endfor %}

   {% if pins[17].state == true %}
       {% if pins[22].state == false %}
           <h4> Ant1 = Rig1 </h4>
       {% elif pins[22].state == true %}
           <h4> Ant1 = Rig2 </h4>
       {% endif %}
    {% else %}
       <h4> Ant1 term </h4>
    {% endif %}

    {% if pins[18].state == true %}
       {% if pins[23].state == false %}
           <h4> Ant2 = Rig1 </h4>
       {% elif pins[23].state == true %}
           <h4> Ant2 = Rig2 </h4>
       {% endif %}
    {% else %}
       <h4> Ant2 term </h4>
    {% endif %}

    {% if pins[27].state == true %}
       {% if pins[24].state == false %}
           <h4> Ant3 = Rig1 </h4>
       {% elif pins[24].state == true %}
           <h4> Ant3 = Rig2 </h4>
       {% endif %}
    {% else %}
       <h4> Ant3 term </h4>
    {% endif %}

   <img class="img-responsive" src="/static/images/blkdiag.png">
</body>
</html>
```

100W 3:2 HF Switch Matrix

Conclusion

The completed switch matrix performs well for power levels at 100W and below. It should be noted that at 100W (+50dBm), 60dB isolation would result in -10dBm at the alternate output. Therefore care must be taken to insure that a receiver connected to the alternate port is not overdriven. It is also recommended that switching should be initiated with the transmitters off.

For additional cost (~2x), it would be possible to design a better 4-layer stackup that would provide better port VSWR and still maintain the PCB trace power requirements. Being retired (and having to get projects approved from the 'boss'), I decided to keep cost as low as possible.

I have not tested the switch matrix with antennas that present very high VSWRs. This may further limit the power output that could be used due to increased peak voltages. To alleviate this, automatic tuner(s) could be attached to the high VSWR input(s).