Introduction

The RF transmitter add-on board can be used with the R-PI to control up to 4 simple Energenie radio controlled sockets independently using a small software program. We have written a simple program in Python to allow us to switch the sockets on and off with a single keyboard press.

The add-on board connects to the row of pins called the GPIO which can be controlled as either input or output lines under your software control.

Installing the board

![R-Pi with piggy back RF-transmitter board](image)

Fig. 1: R-Pi with piggy back RF-transmitter board

Install the board on to the row of pins as show in the picture and connect your Raspberry-Pi as normal to a monitor, mouse, keyboard and USB power supply.
How it is controlled by Software

The pin header connects to the add-on board as follows to allow you to control the GPIO lines as outputs to drive the radio frequency transmitter.

**GPIO Pin Header Rev 2.0**
view from top

<table>
<thead>
<tr>
<th>3V3</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>GPIO3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>GPIO4</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Ground</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Encoder signal D0
- GPIO17: 11
- GPIO22: 15

Encoder signal D3
- GPIO27: 13

Encoder signal D1
- GPIO18: 16

3V3
- GPIO10: 17
- GPIO9: 21
- GPIO11: 23
- Ground: 25

5V
- GPIO14: 1
- GPIO15: 4
- GPIO23: 15
- GPIO24: 18
- GPIO25: 20
- GPIO26: 24
- GPIO27: 22
- GPIO7: 26

Your software will use the outputs on the GPIO connector to drive the add-on board. At the end of this document is some sample code written in Python to switch the Energenie sockets on and off.

Each board transmits a frame of information using On-Off-Keying (OOK) which is a basic form of Amplitude Shift Keying (ASK). This frame includes source address (20 bits) and control data (4 bits). See
Appendix 1. The source address is randomly selected (so unique for that board), pre-programmed and cannot be changed. The socket will accept commands frames that have the source address that it learns during the learning process described later.

Here are the pairs of codes using D0-D3 signals that can be sent to control sockets.

<table>
<thead>
<tr>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>Meaning</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>All on</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>All off</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>socket 1 on</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>socket 1 off</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>socket 2 on</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>socket 2 off</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>socket 3 on</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>socket 3 off</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>socket 4 on</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>socket 4 off</td>
</tr>
</tbody>
</table>

The pcb can therefore control a maximum of 4 Energenie radio sockets independently. However, more than one socket may be controlled by the same control code.

There are two parts to the transmitter. The encoder and the modulator.
1) The encoder will accept 4 input signal levels programmed onto 4 of the GPIO lines (D0-D3) as shown above. It will then serialise them on a single line to the modulator part.
2) The modulator transmits the serialised signal. It needs to be programmed in Amplitude-Shift Keying (ASK) mode for the sockets using a GPIO signal. It also needs to be enabled by a separate GPIO signal.

Firstly, we need to install a Python module, RPi.GPIO, to enable software control of the GPIO pins for the Raspberry Pi. Open an LXterminal from the desktop and type the following lines:

```
sudo apt-get install python-rpi.gpio
```

Now, open a text editor such as Leafpad and type the python code listed at the end called ENER002.py into a new file under home/pi (or, more easily, download it from the energenie4u.co.uk website and copy using a USB flash drive). This program will allow us to send coded commands to the sockets to program them and then switch them on and off when the return key on the keyboard is hit.

**Controlling the sockets**

Launch your program by typing the following command at the prompt in the LXterminal window:

```
sudo python ENER002.py
```

Then insert a radio controlled socket into a mains wall socket which is switched on. The socket must then be programmed to learn a control code from the transmitter. To do this the socket must be in learning mode indicated by the lamp on the front of the socket housing flashing slowly. If it is not doing this, press and hold the green button on the from of the housing (while the lamp is off), for 5 seconds or more and then release it when the lamp starts to flash at 1 second intervals. Then send a signal to it from you program by hitting the return key. Acceptance will be indicated by a brief quick flashing of the lamp on the housing which will then extinguish. Program first one socket then the other in this way, otherwise they will react to the same signal.
You can then toggle the socket on and off by hitting the return key. You can also switch it manually on and off by briefly pressing the button on the front housing.

You can always reset the socket programming by holding down the green button for 5 seconds or more as mentioned before.

To increase the range of the transmitter you may wish to add an extra antenna to the circuit board. You can do this by soldering a piece of ordinary copper wire 13.5cm long into the hole marked ANT1 on the circuit board.

**IMPORTANT:** If using more than one socket, they will need to be inserted into separate mains wall sockets with a physical separation of at least 2 metres to ensure they don't interfere with each other. Do not put into a single extension lead.

Here is the program ENER002.py

```python
# import the required modules
import RPi.GPIO as GPIO
import time

# set the pins numbering mode
GPIO.setmode(GPIO.BOARD)

# Select the GPIO pins used for the encoder K0-K3 data inputs
GPIO.setup(11, GPIO.OUT)
GPIO.setup(15, GPIO.OUT)
GPIO.setup(16, GPIO.OUT)
GPIO.setup(13, GPIO.OUT)

# Select the signal used to select ASK/FSK
GPIO.setup(18, GPIO.OUT)

# Select the signal used to enable/disable the modulator
GPIO.setup(22, GPIO.OUT)

# Disable the modulator by setting CE pin lo
GPIO.output (22, False)

# Set the modulator to ASK for On Off Keying
# by setting MODSEL pin lo
GPIO.output (18, False)

# Initialise K0-K3 inputs of the encoder to 0000
GPIO.output (11, False)
GPIO.output (15, False)
GPIO.output (16, False)
GPIO.output (13, False)

# The On/Off code pairs correspond to the hand controller codes.
# True = '1', False = '0'

print "To clear the socket programming, press the green button"
print "for 5 seconds or more until the red light flashes slowly"
print "The socket is now in its learning mode and listening for"
print "a control code to be sent. It will accept the following"
print "code pairs"
print "0011 and 1011 all ON and OFF"
print "1111 and 0111 socket 1"
```
print "1110 and 0110 socket 2"
print "1101 and 0101 socket 3"
print "1100 and 0100 socket 4"
print "Hit CTL C for a clean exit"

try:
    # We will just loop round switching the unit on and off
    while True:
        raw_input('hit return key to send socket 1 ON code')
        # Set K0-K3
        print "sending code 1111 socket 1 on"
        GPIO.output (11, True)
        GPIO.output (15, True)
        GPIO.output (16, True)
        GPIO.output (13, True)
        # let it settle, encoder requires this
        time.sleep(0.1)
        # Enable the modulator
        GPIO.output (22, True)
        # keep enabled for a period
        time.sleep(0.25)
        # Disable the modulator
        GPIO.output (22, False)

        raw_input('hit return key to send socket 1 OFF code')
        # Set K0-K3
        print "sending code 0111 Socket 1 off"
        GPIO.output (11, True)
        GPIO.output (15, True)
        GPIO.output (16, True)
        GPIO.output (13, False)
        # let it settle, encoder requires this
        time.sleep(0.1)
        # Enable the modulator
        GPIO.output (22, True)
        # keep enabled for a period
        time.sleep(0.25)
        # Disable the modulator
        GPIO.output (22, False)

        raw_input('hit return key to send ALL ON code')
        # Set K0-K3
        print "sending code 1011 ALL on"
        GPIO.output (11, True)
        GPIO.output (15, True)
        GPIO.output (16, False)
        GPIO.output (13, True)
        # let it settle, encoder requires this
        time.sleep(0.1)
        # Enable the modulator
        GPIO.output (22, True)
        # keep enabled for a period
        time.sleep(0.25)
        # Disable the modulator
        GPIO.output (22, False)

        raw_input('hit return key to send ALL OFF code')
        # Set K0-K3
        print "sending code 0011 All off"
        GPIO.output (11, True)
        GPIO.output (15, True)
        GPIO.output (16, False)
        GPIO.output (13, False)
        # let it settle, encoder requires this
```python
time.sleep(0.1)
# Enable the modulator
GPIO.output(22, True)
# keep enabled for a period
time.sleep(0.25)
# Disable the modulator
GPIO.output(22, False)

# Clean up the GPIOs for next time
except KeyboardInterrupt:
    GPIO.cleanup()
```

**Appendix 1 – Tx Frame format**

<table>
<thead>
<tr>
<th>Preamble</th>
<th>A0 - A19 (20 bits - preset)</th>
<th>D0 - D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>preamble</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data (H)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data (L)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ick=256clk