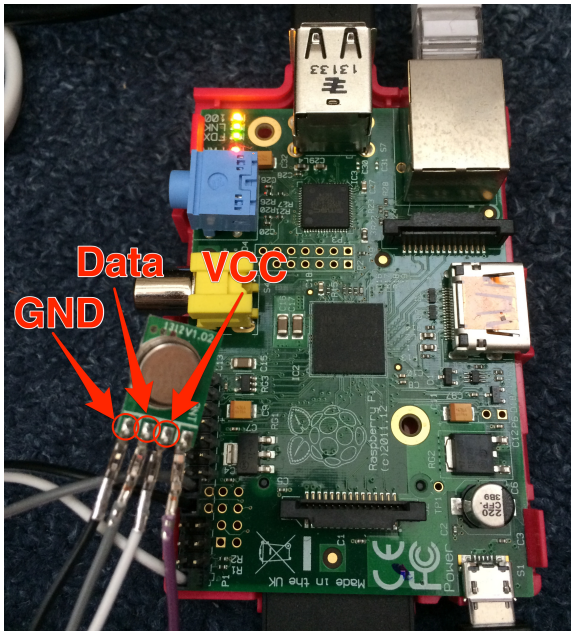
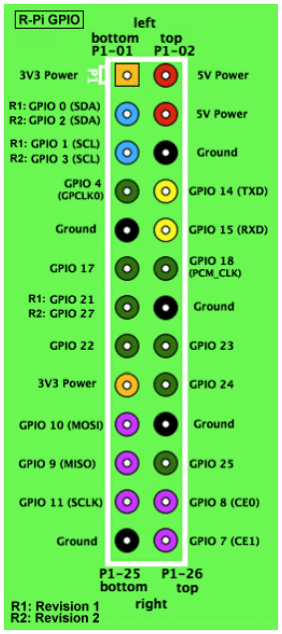
Assignment 4

Signal Modulation, Homeautomation, Android

Deadline: Monday, 9th December 2015

In this assignment you will deal with cheap wireless switchable power plugs. They use 433MHz radio technology. Therefore you will be given a Raspberry Pi which has a 433MHz radio attached to its General Purpose Input and Output (GPIO) pins. The radio will send everything it receives on its data pin. You will first create a program to modulate the signal to switch the plugs and secondly an Android application to access them. Here you see a Raspberry Pi with attached 433MHz module:





The VCC is connected to either 3.3V (here using pin 1) or 5V on the board. Data is connected to a GPIO pin (here using GPIO 17 which is pin 11) and GND (here using pin 9) is connected to a ground pin. The 4th, not annotated pin is a $1/4\lambda$ whip antenna. As we are dealing with 433MHz frequencies we already cut the wire to roughly 17cm, because:

$$\lambda = \frac{c}{f} = \frac{300000000m/s}{433000000Hz} = 0.6928406467m \rightarrow \frac{\lambda}{4} = 0,1732101617m$$

Please be careful: shorting or false wiring will destroy the Raspberry Pi.

Task 4.1: Preparing the Raspberry Pi (0 Points)

The Raspberry comes with no Operating System on it. Before starting you need to setup your Pi. You may choose what ever linux distribution you like, but we would recommend an Arch Linux (`https://wiki.archlinux.org/index.php/Raspberry_Pi`). They have a very well documented wiki which will guide you through the installation process. It is probably best to prepare the image on your PC using the SD card adapter.

If you do not like to connect the Pi to your computer's screen but rather just put it on your LAN and access it via SSH, you may use the MAC address that is printed on the back of the device to discover it in the network. You may simply observe your ARP table, e.g. using `arp -a`. This will show all devices

Teaching assistants:
Jan Rüth: `Jan.Rueth@comsys.rwth-aachen.de`
Jó Ágila Bitsch: `jo.bitsch@cs.rwth-aachen.de`
Martin Serror: `martin.serror@comsys.rwth-aachen.de`

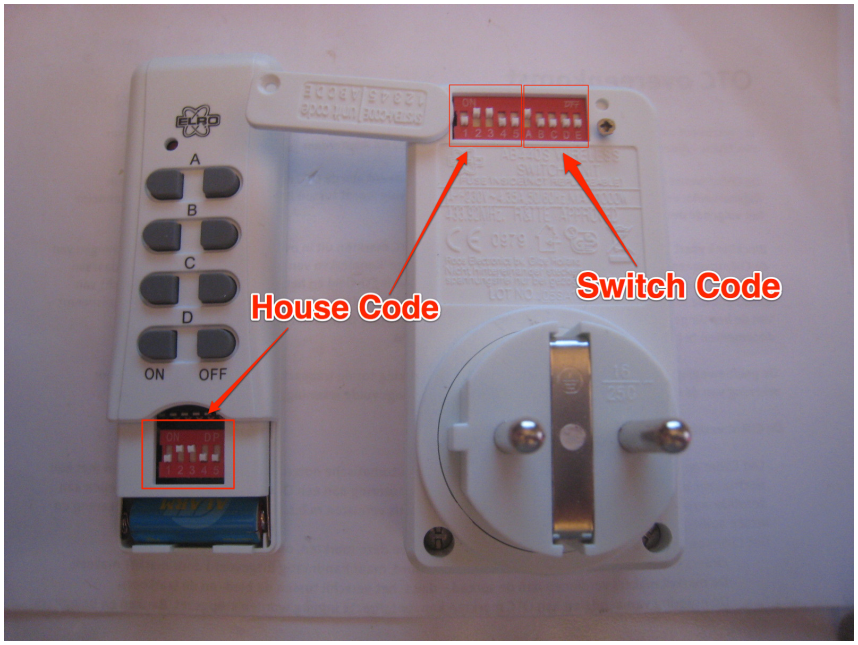
that have an entry in your Address Resolution Protocol table (that is that thing that translates IP to Mac addresses, I hope you remember). So once you have communicated with the Pi it should show up. But how do you communicate, you may either ping your broadcast address (see ifconfig which one it is) and hope that the Pi answers to those requests (it probably won't). Otherwise you can simply ping all devices on your subnet and then checkout your arp table. E.g. if you know that your network is 137.226.12.0/24 and .1 is the router and 2..254 (as it is a /24 subnet) are valid addresses you may just ping everything:

```
for i in {2..254}; do ping -c1 -W1 137.226.12.${i}&  done
```

(c1 = only 1 packet; W1 wait no more that 1 sec for an answer; the & does this command in the background) And if you now look at your arp table, e.g. `arp -a | sort -k4` you will easily find the IP of the Pi.

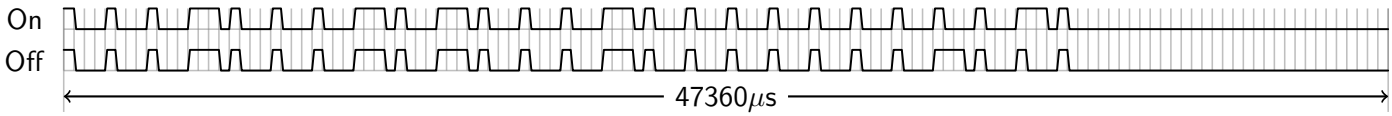
Task 4.2: Modulating the Signal (5 Points)

In this task we want to write a C-Program that modulates a signal for radio transmission. The plugs can be configured to a specific sequence using DIP switches. Usually the first 5 switches declare the house code and the following 5 switches declare the actual switch. The remote control that is included with the plugs can also be configured to the house code and has the buttons A-D (which correspond to the switches A-E in the plug, and yes there are more switches than buttons) to switch 4 plugs. As the switches can either be on or off we will write 1 for on and 0 for off. See the following image for clarification:



Teaching assistants:
Jan Rüth: Jan.Rueth@comsys.rwth-aachen.de
Jó Ágila Bitsch: jo.bitsch@cs.rwth-aachen.de
Martin Serror: martin.serror@comsys.rwth-aachen.de

The plugs listen to the radio and when their specific sequence is received they either switch on or off. The signal is composed of 3 parts: the `plugs code` (house code + switch code), an `on` or `off` code and a `synchronization sequence`. When the plug receives the signal 2 or more times it switches. Use the following image to work out the actual modulation of the signal that is sent to the plug with the house code: 10100 and switch code: 10111 to switch it on/off:



The length of the whole signal is is 47360μs. Please use the provided C-code to get started, it contains code to switch the pins of the Raspberry Pi.

Hints:

1. To work out the modulation, divide the signal into three pieces according to plug code, on/off code, and synchronization.

   You should find the boundaries by comparing the on and off signal.

   Check the length of the plug code and determine how long a 1 or 0 is in the modulated signal.

   Compare the plug code with the modulated signal, sequence by sequence to find out the modulation for a 1 and 0.

   Now you should be able to work out the on/off code.

   As you know the duration of the whole signal you should be able to find out the length of a single high/low interval.

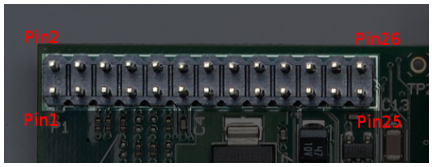
2. When writing your program, it should be capable of getting an input sequence and a state.

   Find an appropriate representation to transform the input sequence to the modulated signal.

   Remember that the plug has to receive the signal multiple times, so send it a couple of times.

   Use `nanosleep` to time your code correctly.

3. Make sure to switch the correct GPIO pin. You must address the pins by their number:

   So the top row is always an even numbered pin and the bottom row always odd numbered one, so pin 3 is the second pin in the bottom row... Attention: if you have a Raspberry Pi B+, you have more GPIO pins, but up to PIN 26, they are compatible with the old layout.

4. Be aware: to switch the GPIO pins you must run your program as root.

Teaching assistants:
Jan Rüth: `Jan.Rueth@comsys.rwth-aachen.de`
Jó Ágila Bitsch: `jo.bitsch@cs.rwth-aachen.de`
Martin Serror: `martin.serror@comsys.rwth-aachen.de`

Task 4.3: Binding to Android (5 Points + 2* Points)

The Android application we are about to write should be capable of switching different plugs. However, the different plugs should be configurable in the server. Extend your program from Task 2 to act as a server, or write a new program in a language of your choice that uses your program from Task 2 (e.g., by giving it a command line argument, or allow interaction of the two programs using local sockets or unix domain sockets, by using pipes, ...). Your server should be configurable via a simple textfile, that contains, in a format/serialization of your choice: an identifier for each switchable plug, a human readable name for the plug, the plugs code. Your server should serve the identifier and human readable name to clients in a format of your choice, thereby allowing them to dynamically create a user interface. Also you are free to use UDP or TCP or X on the transport layer, furthermore there are no restrictions on the protocol mechanism you implement on top of it. Apart from serving the configuration to clients your protocol must offer a way to switch a plug on or off by giving the identifier and the state. Think about additional functionality the server could offer, e.g. add a new plug to the server's configuration or triggers.
Your Android application should be configurable to contact the server on a specific IP and port. It should fetch the list of switchable plugs and create a user interface, showing the human readable description of the plug and the possibility to switch it on or off.

***To gain 2 extra Points**: You program transmits the information in the clear, it is easy to work out your protocol using tools like Wireshark. Furthermore, not only allows this to monitor which plugs are switched, it also allows everybody to access your plugs as your protocol has no kind of access control (assuming the server is reachable from the Internet). Add an access control mechanism that protects against unauthorized access. If you have time left also encrypt the communication. You are free in choice of technology, but we recommend to choose well established and library ready mechanisms (to save your time and to guarantee that it really works).

Teaching assistants:
Jan Rüth: Jan.Rueth@comsys.rwth-aachen.de
Jó Ágila Bitsch: jo.bitsch@cs.rwth-aachen.de
Martin Serror: martin.serror@comsys.rwth-aachen.de