gazerbeam library

Configure microcontroller's WiFi using Android phone's flash light.

18.3.2020/Pekka Lehtikoski

Table of Contents

1. Introduction

200318, updated 20.3.2020/pekka

The “Gazerbeam” library enables configuring micro-controller's Wi-Fi network name (SSID) and password (PSK) using LED flash of Android phone, etc.

- The phone needs APP to blink configuration information, example Android Java application named “blinky”.
- A photo transistor is connected to micro-controller with simple high pass filter/digitalization electronics.
- Micro-controller gets signal from electronics as digital input. It is set up so that changes to the signal trigger interrupt. Gazerbeam library handles this and converts it to message.
- The transferred data is encoded into light signal using short and long intervals between light on/off changes

1.1 Android application - Blinky

200318, updated 20.3.2020/pekka

Android application is simple: To provide user form to fill in wifi network name, password and other optional configuration settings. When user fills these in and clicks “blink” button, the app starts sending this data repeatedly with LED flash until the “blink” button is clicked again.

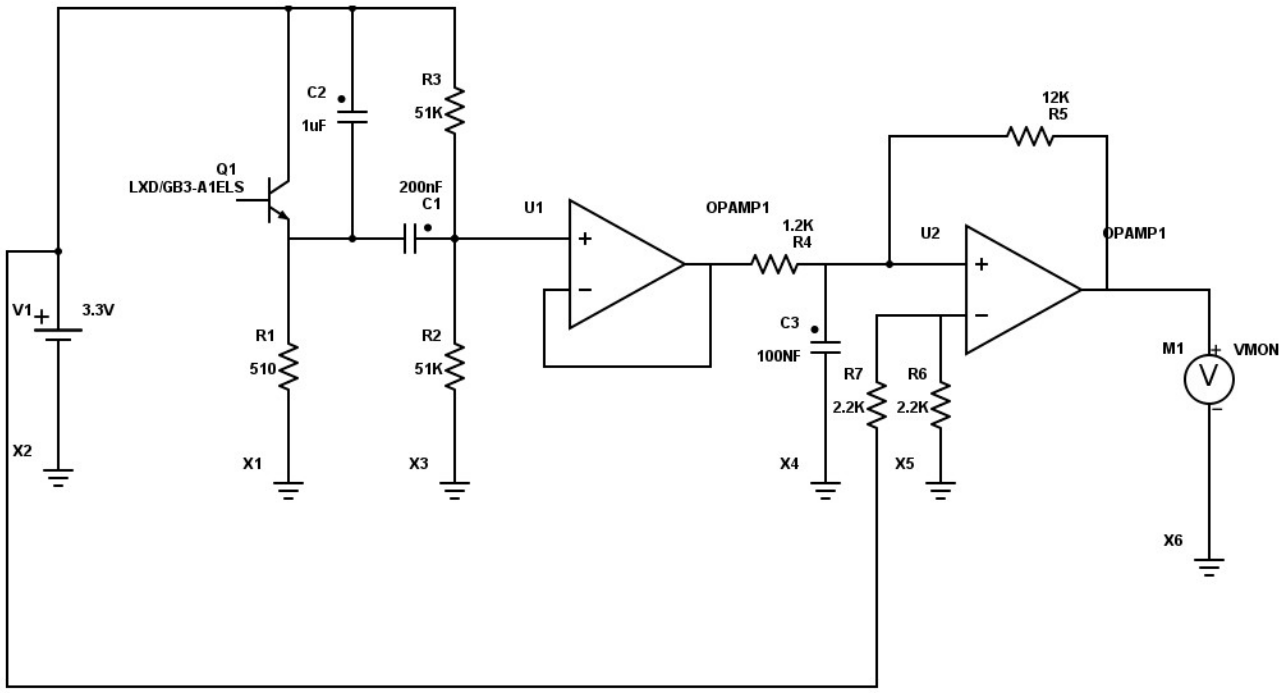
The android application’s Java code is in iocom/extensions/gazerbeam/examples/blinky directory.

1.2 Electronics

200318, updated 20.3.2020/pekka

An LM6144 operational amplifier (U1 and U2) is used to high pass and to digitize the signal of LXD/GB3-A1ELS

photo transistor (Q1).



The first operation amplifier U1 does high pass filtering of the signal. This removes ambient light and centers the signal at 1.65V.

The U2 is comparator, which compares if input voltage is less or greater than 1.65 V changes the voltage to 0V or 3.3V (rougly) Resistor R5 adds a threshold to change output state, to prevent output from filtering. In addition there is mild low pass filtering by RC circuit C3/R4.

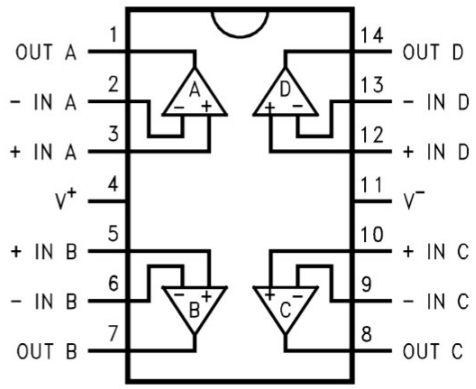
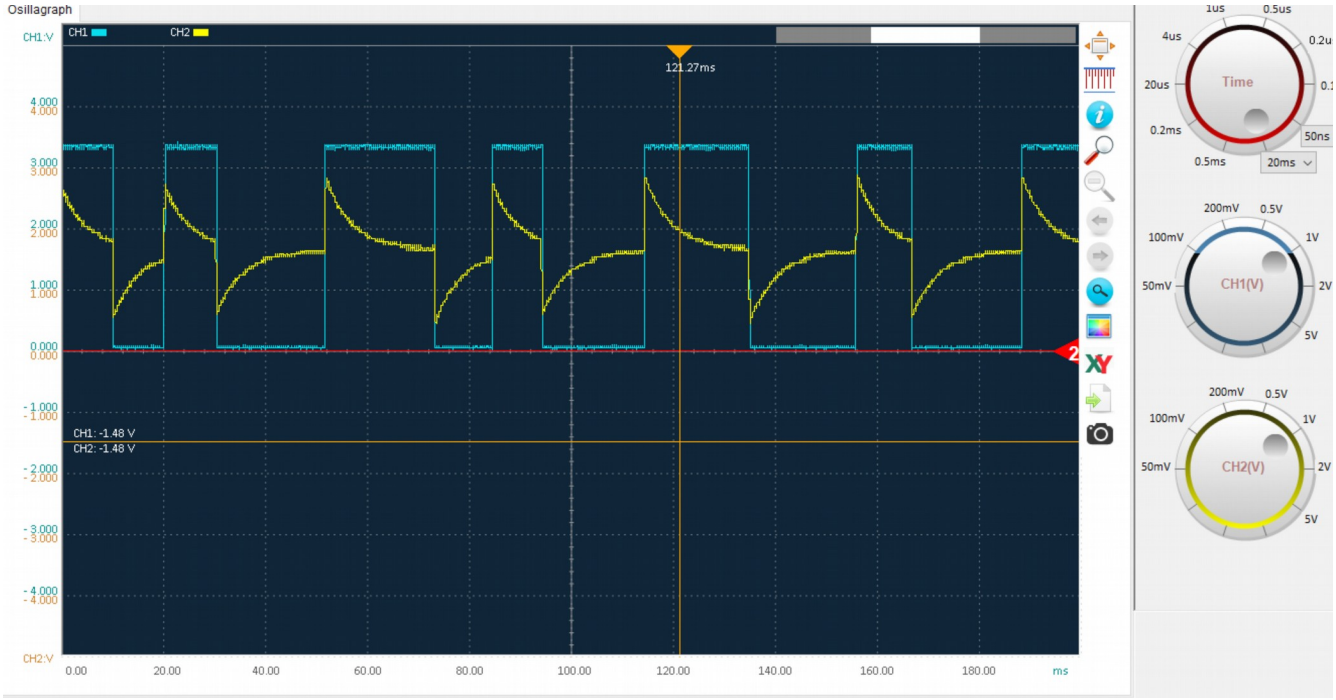


Figure 3. 14-Pin PDIP/SOIC Top View



Yellow is high pass filtered signal after first OP-amp U1. The blue is digitalized signal after second OP-amp U2.

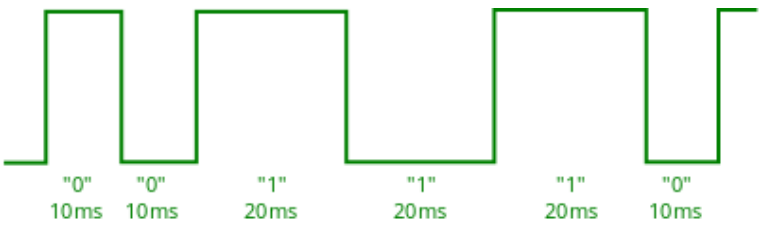
1.3 Signal and data encoding

200318, updated 20.3.2020/pekka

An Android phone can only turn flash on or off, the shortest pulse which can be reliably generated is approximately 10ms. This is enough, amount of data to be transferred is small. Typically WiFi network name and password, and optionally some extra information.

1.3.1 Zero and one bits

When sending Android's LED keeps on turning on and off continuously. What matters is how long LED stays in same state: 10ms for "0" bit and 20 ms for "1" bit. So time between transitions matters, direction of transition is ignored. If doesn't matter if LED switches from on to off or vice versa.



The Gazerbeam receiver is not very time critical, timing can be changed quite much and it should still work. Pulse length is recorded on fly and basic rule is that "1" must be clearly longer than "0". Anyhow if taken very far from tested, the high pass filtering RC may need to be adjusted.

1.3.2 From bits to messages

A message contains all data what user entered and is repeated endlessly: Android application doesn't know when it is received by micro-controller.

- The message starts with 14 zero bits and then one bit, like “000000000000001”.
- Then the 7 data bit for each data byte follow, least significant bit first.
- The data transferred is 7 bit ASCII with legimate values from 1 to 127, it can never form nine consequent zeros and can be separated from beginning of message. Like “0101010”.
- There is no message length. Previous message is processed when a new message begins, at that point the previous message is considered “ready”.
- First three data bytes contain always checksum (MODBUS CRC), less significant byte first. First byte contains 6 least significant bits + 1, next byte 6 bits + 1 and the third byte 4 bits + 1.

1.3.3 Message content - fields

The message starts with three byte checksum. This needs to be generated by Android code, and is checked and stripped away by the Gazerbeam library.

Following the check sum there are 1 or more fields:

- A field starts with field id byte.
- Next byte is field data length, specifies number of data bytes.
- N data bytes, ASCII codes from 1 to 127

The field IDs are (**list here**)

1.4 Using the library

200318, updated 20.3.2020/pekka

1.4.1 Data types and functions

The Gazerbeam receiver structure holds settings and current state of monitoring photo transistor signal.

- Initialize gazerbeam structure and set pin to monitor.
- Poll for new received messages.

1.4.2 The library source code

- The source code is located in /coderoot/iocom/extensions/gazerbeam directory.

1.4.3 Building the library

How to build the library depends on target platform and tools used. Cmake builds are quite straight forward, there is CMakeLists.txt in root folder, which can be opened for example in QT creator.

- Cmake: Either build project separately or include it as subproject within in your project
- ESP32/Arduino/Platform IO. Copy files. Configure platformio.ini.
- **TODO: Add here link to build instructions for different platforms.**

1.4.4 Include headers and link library with your project

The path iocom/extensions/gazerbeam should be added to your project build. This allows including the library headers by “#include “gazerbeam.h”.

1.4.5 Dependencies

The gazerbeam depends on some libraries

- eosal - platform/operating system abstraction
- pins - portable access to IO pins
- iocom - some data types, not really much needed but dependency is there.

1.4.6 HW support

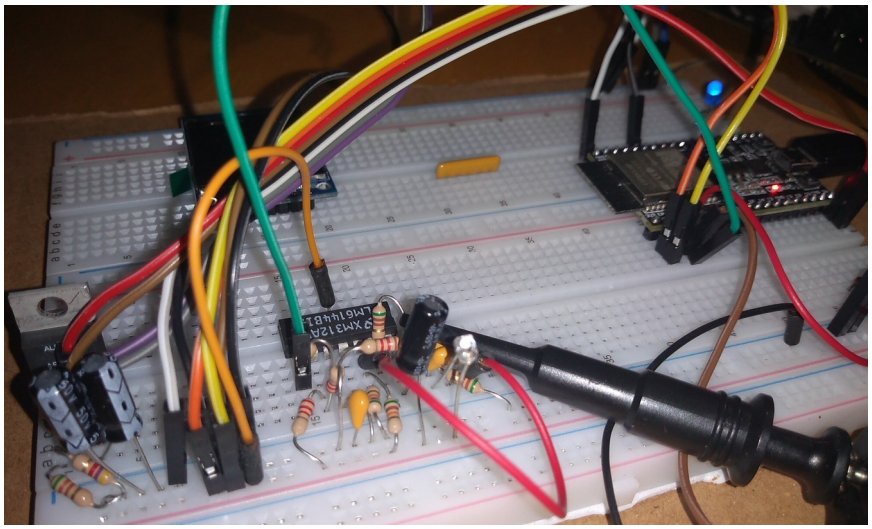
- Tested on ESP32/Arduino/PlatformIO environment.

- To port to other hardwares, the HW support needs to be implemented in eosal and pins libraries.

1.5 Testing and prototyping hints

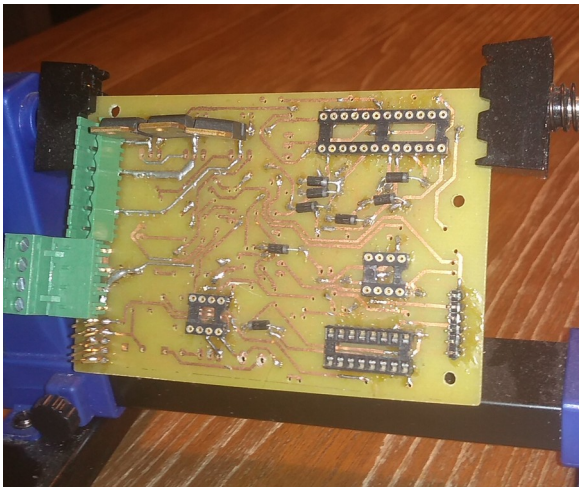
200319, updated 20.3.2020/pekka

I am C coder, only hobbyist in electronics. So I draw any analog electronics design with partsim, (free on line tool) and run simulation it. Next I test the setup on breadboard, as in picture below. Components I like to buy from AliExpress and keep extras at home.



Bread board test. 5 volt regulation on left (ESP32 regulates 5V down to 3.3V, which runs the electronics), ESP-32 WROOM DEV KIT on back right and Gazerbeam photo transistor and signal filtering around LM6144 chip.

Once ready to make a “product”, which often is just hobby project: I start to think about dimensions, connectors, transients and finally open Eagle CAD (AutoDesk, free for home projects) to and draw the design. I still use old laser printer, heat transfer paper, iron and ferro-chloride. With care and enough time it is possible to make quite nice PCB, although sending Eagle CAD file to China would yield better quality and be perhaps more cost effective.



A home made two sided PCB (brush-less motor control). This one was dropped to ferro-chloride straight after transferring laser print with iron. Using “green” iron foil before etching would make copper wires better and liquid tin afterwards to protect the PCB.

1.6 MIT license

190625, updated 27.6.2019/pekka

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.