Tolteca Project Documentation

March 10, 2021

Centro de Investigación y de Estudios Avanzados

del Instituto Politécnico Nacional

Unidad Unidad Guadalajara

Author: Emilio Tonix

Tech Lead: Dr. Fernando Peña Campos

Manager: Dr. Ramón Parra Michel

Contents

Introduction

This document provides information about feature development of ad9361. Starting from NuandRF project to a custom product. The idea is to provide firmware and integration tools for deployment in the transreciever aplications. We provide a code package made of 3 main modules. Host **BladeRF, FPGA_FW** and **GUI**. Note: For bladeRF, you may need legacy APP first in order to use custom project.

Nuand Legacy App, BladeRF2
https://github.com/Nuand/bladeRF

Our custom Project
https://github.com/Tonix22/NuandRFProyect

Nuand / bladeRF

It is a C/C++ legacy package taken from Nuand bladeRF V.2, it is an open source platform based on easy to use software. It provide access to ad9361 radio Frequency Transcreciver directly from CPU applications, this chip offers frequency range from 47 MHz to 6 GHz, and 61.44 MHz sampling rate. Through libbladeRF the bladeRF 2.0 micro is compatible with GNURadio, GQRX, SDR-Radio, SoapySDR, and more on Windows, Linux and macOS. In general we could say bladeRF is non custom software and hardware to access ad9361, but give you such potential of the device, in a project easy work.

Hands on client

The client is known as the console aplication to access and modify registers of Nuand, through the General architecture shown in Figure 1. Client is connect by an usb port, and then comunications flow through the cypress FX3, CYCLON V and finally to the ad9361. You could have both options to work with the Blade RF setup: **Legacy and Custom**. The legacy package comes from Nuand which it is company that design SDR solutions.

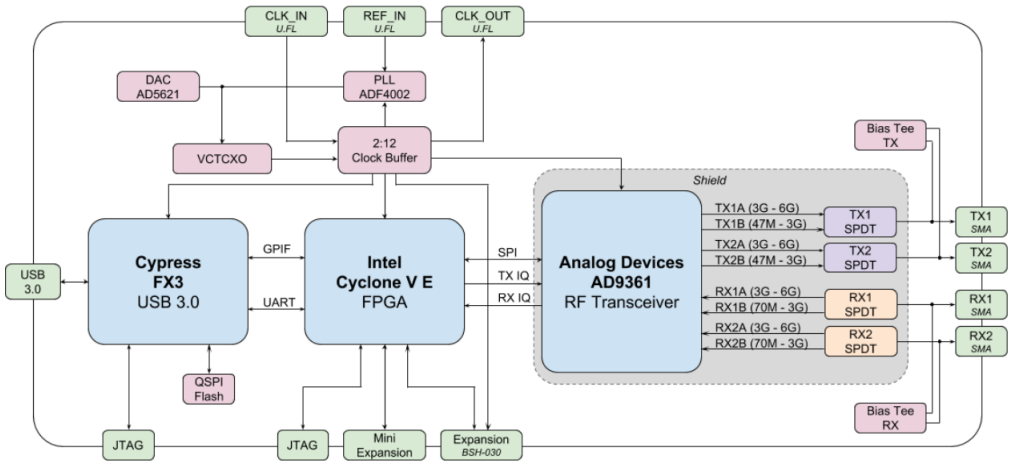

Figure 1: General Architecture of Host BladeRF2

Sections brief

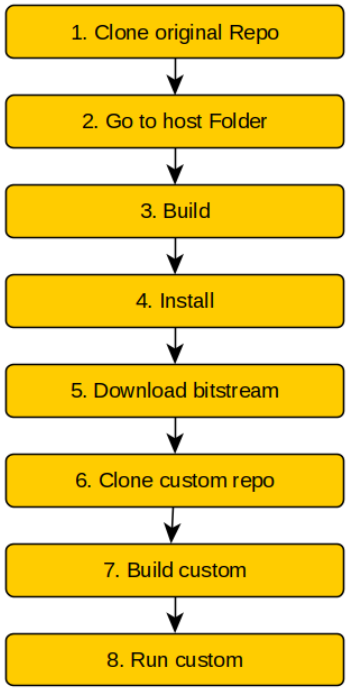

Figure 2: Brief of the following sections

1. Nuand Original Aplication

This one is heavier than custom, but it is more complete. And provides you a comple comand line interface to interact with the Nuand.

Repository link: https://github.com/Nuand/bladeRF

Clone the repository :

```
$ git clone https://github.com/Nuand/bladeRF.git
```

2. Go to host folder

```
$ cd host
```

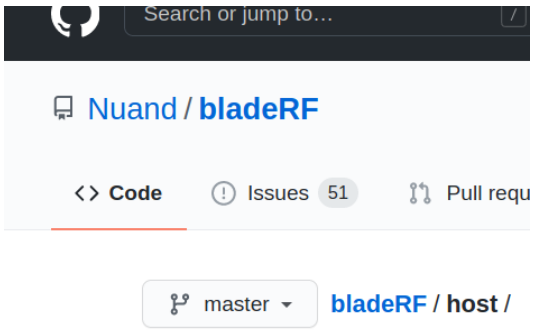

Figure 3: Host folder

3. Build Code

Firstly go to the host path, and then run a cmake building. NOTE: that you must have **cmake** and **libusb**, installed.

```
$ mkdir -p build
$ cd build
$ cmake [options] ../
$ make
```

Install the bladerf framework into the system, this one will help you to download the bitstream into to the Nuand.

```
$ sudo make install
$ sudo ldconfig
```

4. Install

Test that the bladerfcli has been installed correctly

```
$ cd ../
$ bladeRF-cli --version
```

This will show your cli version, if it was installed correctly. Which actually is the repo git commit or branch.

Fore more info, there is a wiki to show how to use the CLI.

https://github.com/Nuand/bladeRF/wiki/bladeRF-CLI-Tips-and-Tricks#receiving-samples

5. Download Bitstream

For Custom and Nuand code, the bistream is the same, they share the same internal logic.

Go to : https://www.nuand.com/fpga_images/

Figure 4: Download A4 in the given link

For this proyect download A4 image

Once downloaded, copy the bitsream path and, run the ./bladeRF-cli command as follows.

```
$ bladeRF-cli -d "libusb: instance=0" -l /path/to/fpga.rbf -i
```

Here is an example of what you can do with nuand client terminal

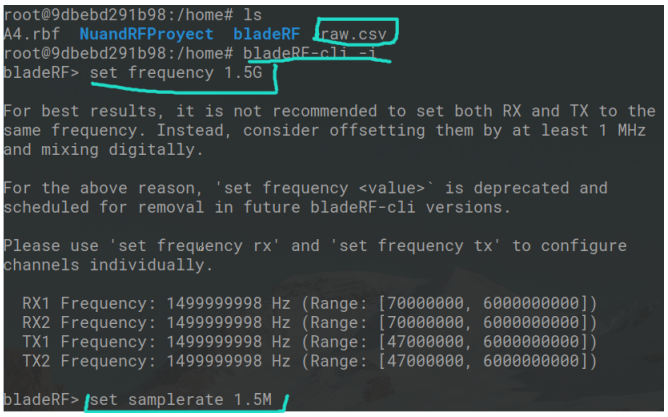

Figure 5: Terminal Example part1

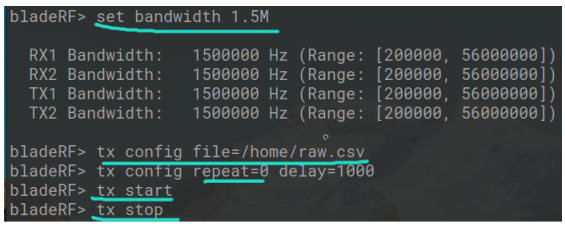

Figure 6: Terminal Example part2

6. Clone custom Repo

Inside the repository there is a customised and reduced console aplication. Folder destination is:

```
$ git clone https://github.com/Tonix22/NuandRFProyect.git
```

7. Build Custom

There are the similar steps as building original, except that there is not need an instalation, and folder path changes.

```
$ cd /NuandRFProyect/Nuand_Blade_Base/host
$ mkdir -p build
$ cd build
$ cmake [options] ../
$ make
```

8. Run custom

```
$ cd host/build/output
$./bladerf-cli
```

NOTE:"./" is important to differ with Nuand blade-cli previously installed, this one is locally builded.

This is a code example to modify, basic parameters for software defined radio in a raw API fashion. So there will be shown almost all functions with ad9361 prefix

Code main sections are in :

NuandRFProyect/Nuand_Blade_Base/host/utilities/bladeRF-cli/src/main.c

1. OPEN DEVICE

Opens usb port to stablish comunication with fx3

```
bladerf_open(&state->dev, rc.device);
```

2. RX AND TX Threads init

```
cli_start_tasks(state);
```

3. Frequency: 70Mhz - 5Ghz

```
bladerf_frequency frequency = FREQ; // 150MHZ
/* Set up band selection */
CHECK_STATUS(board_data->rfic->select_band(state->dev, BLADERF_CHANNEL_TX(0), frequency));

ad9361_set_tx_lo_freq(phy, frequency);
ad9361_set_rx_lo_freq(phy, FREQ);//950MHZ
board_data->rfic->set_gain(state->dev, BLADERF_CHANNEL_TX(0),60);
```

4. Sample Rate 520,834 Khz - 61.440Mhz

To modify the sample rate, there must be verified the interpolation and decimation

```
bladerf_sample_rate current;
bladerf_rfic_rxfir  rxfir;
bladerf_rfic_txfir  txfir;
bladerf_sample_rate rate = SAMPLE_RATE; // 10Mhz
/*Range for interpolation required*/
int max_range = 2083334; //2MHZ
int min_range = 520834; // 520KHz
bool old_rate, new_rate;

old_rate = (current >= min_range) && ( current <= max_range);
new_rate = (rate    >= min_range) && ( rate    <= max_range);

///* Get current filter status */
if(old_rate || new_rate){
  rxfir = board_data->rxfir;
  txfir = board_data->txfir;
}
if(new_rate) // check if rfic needs configuration
{
  if (rxfir!= BLADERF_RFIC_RXFIR_DEC4 ||
    txfir!=BLADERF_RFIC_TXFIR_INT4){
    //fpga_common/src/ad936x_params.c:604
      ad9361_set_rx_fir_config(phy,bladerf2_rfic_rx_fir_config_dec4);
        ad9361_set_rx_fir_en_dis(phy, 1);//1 = enable
        ad9361_set_tx_fir_config(phy,bladerf2_rfic_tx_fir_config_int4);
        ad9361_set_tx_fir_en_dis(phy,1);// 1 = enable
  }
}
ad9361_set_tx_sampling_freq(phy, rate);
ad9361_set_rx_sampling_freq(phy, rate);
```

5. Bandwidth 200Khz -- 56Mhz

```
ad9361_set_tx_rf_bandwidth(phy,BANDWIDTH_RX);
ad9361_set_rx_rf_bandwidth(phy,BANDWIDHT_TX);
```

6. RX AGC MODE OFF

```
enum rf_gain_ctrl_mode gc_mode;
gc_mode = RF_GAIN_MGC; // MANUAL GAIN CONTROL
ad9361_set_rx_gain_control_mode(phy, 0, RF_GAIN_MGC);// RX channel 0
ad9361_set_rx_gain_control_mode(phy, 1, RF_GAIN_MGC); // RX channel 1
```

7. RX GAIN VALUES

```
int val;
int gain = 10;
float offset = -17.0f;// depends on frequency bladerf2_rx_gain_ranges, bladerf2_common.h
struct bladerf_range const *range      = NULL;
gain = gain - offset;
state->dev->board->get_gain_stage_range(state->dev, 0, "full", &range);
val = __scale_int(range, gain);
ad9361_set_rx_rf_gain(phy, 0, val);
ad9361_set_rx_rf_gain(phy, 1, val);
```

8. Read/Write data

First you shall setup path to the .csv file, which it will be used to read data or write data.

```
//tx state
rxtx_set_file_path(state->tx,"/home/tx.csv");
rxtx_set_file_format(state->tx,RXTX_FMT_CSV_SC16Q11);
// rx state
rxtx_set_file_path(state->rx, "/home/rx.csv");
rxtx_set_file_format(state->rx,RXTX_FMT_CSV_SC16Q11);
```

Secondly, select delay and repeat, repeat = 0, means infinit loop.

```
struct tx_params *tx_params = state->tx->params;
tx_params->repeat = 0;//Example: tx_params->repeat_delay = 1000;
```

Finally setup if you want to start RX or TX. NOTE: Select one at each time

```
// start tx
tx_cmd_start(state);
//start rx
rx_cmd_start(state);
```

Sleep the main tread the time you want to sample, and then stop the transmition.

```
usleep(1000*2000); // sample 2 seconds
rxtx_cmd_stop(state,state->rx); // stop transmition
```

FPGA_FW

As we seen before Nuand has its own product to comunicate with ad9361 transreciever, however this platform is not complety flexible, and has some limitations, some of them are that we need a CPU host to exectue certain orders into the radio. What we are looking for is a custom project mounted inside a softcore in the FPGA. Intead of a host relationship, we will use a internal softcore agreement.

The softcore has two internal memories for input and output data respectively, this ones are drive by an extneral middleware.Then read data is parsed and send to an internal ad9361 driver, which process the query or setter, and is sent to the transreciever by SPI.

Middleware is not seen by user, intead user has a GUI where parameters such as frquency, sampling rate and other ones,are changed.

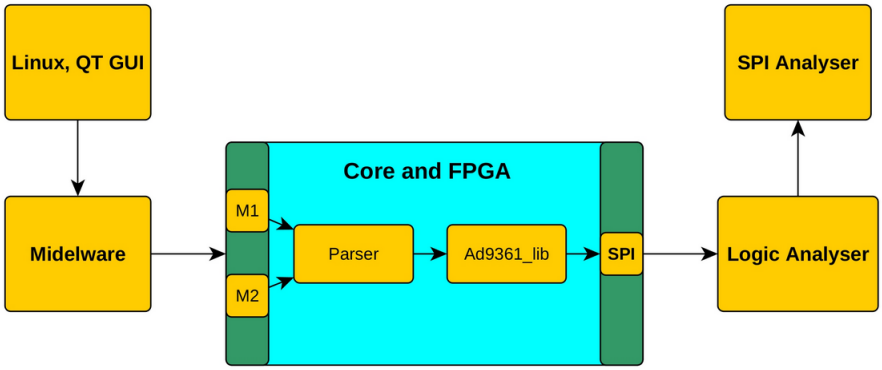

Figure 7: Block diagram of Custom FPGA FW

Parser

Analog Devices follow a convetion to name its APIs. Which actually becomes useful to sort sets of functions.
The pattern followed by the company is shown as next. **<device>_<set/get>_<tx/rx>_<action>**
As consequence we could use those keys words for naming functions and think the as mnemonics.
For example , for ad9361 we could see the following examples:

ad9361_set_tx_lo_freq

ad9361_get_tx_sampling_freq

ad9361_set_tx_rf_bandwidth

Given this sort of combinations and knowing the set of <action>, we could draw a Graph that present all possible paths to generate a valid API call.

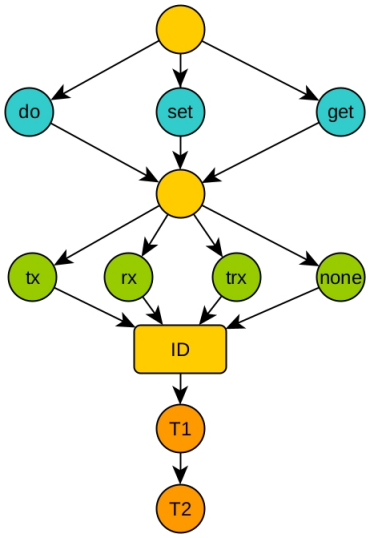

Figure 8: Mnemonics Tree

For each level of the tree there are N posible nodes which represent the possible paths. Over the API we assume that we are using ad9361 device so we ommit the possibility of another device. Given the nodes in each level we fit them in a binary representation, since N elements can be represented in binary set of M bits.

3bit	3bit	6 bit	2bit	2 bits
None	None	ID	other	init
uint8_t	uint8_t		tx	do
uint32_t	uint32_t		rx	set
int32_t	int32_t		trx	get
uint64_t	uint64_t			
struct	struct			
RXFIRConfig	RXFIRConfig			
TXFIRConfig	TXFIRConfig			

Table 1: Mnemonics tree in a 16 bit

The APIs set is given by 64 functions, however some names are repeated and defer only in the setter or getter. Doing analysis in the graph above and in the opcode we can reduce up to 30 possible ID's calls,. For example

```
1 int32_t ad9361_set_rx_rf_bandwidth(struct ad9361_rf_phy *phy,uint32_t bandwidth_hz);
2 int32_t ad9361_get_rx_rf_bandwidth(struct ad9361_rf_phy *phy,uint32_t *bandwidth_hz);
```

Following the given rules, it was made a CPP code to take all possible functions and parse them automatically. Then this functions are saved in an independent csv file, where we can visualize them and verify data. The ad9361 functions list is taken from :

Original Header

Ad9361 functions list

Parser Generator

Framework Installation

General Overview

Required packages and their goals

- Quartus Prime Programmer Version 20.1.1(linux)
  - Download bistream
- RISCV GCC
  - Compiler of softcore
- openocd 0.10.0+dev-g1e85cf0-dirty
  - Debugger interconnect

Quartus installation

You must be registered in the web page to download software.

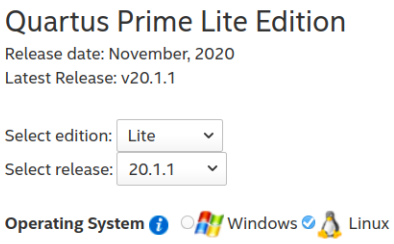

Figure 9: Quartus download

Once you download the package, decrompress into a folder, and run the **./setup.sh**
There are not requeried other devices different from Cyclone V, and neither multisim. Just install quartus app to download bitstreams with the usb blaster.

RISCV GCC

RISC-V is an open standard instruction set architecture (ISA) based on established reduced instruction set computer (RISC) principles. It is provided under open source licenses that do not require fees to use. A number of companies are offering or have announced RISC-V hardware, open-source operating systems are available and the instruction set is supported in several popular software toolchains. Internal FPGA firmware is build in this architecture, that means softcore works with this open architecture.

To install firmware compiler, download and run the following comands.

```
$ sudo apt-get install wget
$ wget https://static.dev.sifive.com/dev-tools/riscv64-unknown-elf-gcc-20171231-x86_64-linux-centos6.tar.gz
$ tar -xzvf riscv64-unknown-elf-gcc-20171231-x86_64-linux-centos6.tar.gz
$ sudo mv riscv64-unknown-elf-gcc-20171231-x86_64-linux-centos6 /opt/riscv64-unknown-elf-gcc-20171231-x86_64-
    linux-centos6
$ sudo mv /opt/riscv64-unknown-elf-gcc-20171231-x86_64-linux-centos6 /opt/riscv
$ echo 'export PATH=/opt/riscv/bin:$PATH' >> ~/.bashrc
```

OpenOCD

OpenOCD is a tool for softcore debugging.

1. Install dependencies

   (a) `$ sudo apt-get install libtool automake libusb-1.0.0-dev texinfo libusb-dev libyaml-dev pkg-config`

2. Build provide the interface to debug the softcore firmware. Folder in repository has the required version.

   (a)
   ```
   $ cd NuandRFProyect/FPGA_FW/openOCD
   $ ./bootstrap
   $ ./configure --enable-ftdi --enable-dummy
   $ make
   $ sudo make install
   ```

   (b) Build could fail on earlier versions of ubuntu. Higher than 18.04.

      i. If build fails intall a gcc version lower than 8.

      ii. To manage multiple gcc versions go to this link

      iii. Open makefile and search -Werror flag and delete it.

Vscode Extensions

1. You may have installed Visual Studio Code for this step, and also it could be optional to run the code. If you don't use Vscode you could use other IDE or simply use gdb. How to use the gdb is not described in this document.
2. Open the extensions section in vscode, and search in market place "Native Debug" . This will provide your gdb connection.

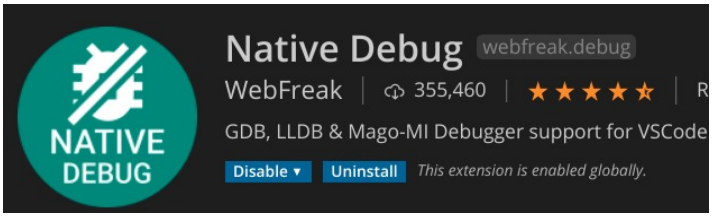

Figure 10: Native debug

Hands on using FPGA firmware

Clone the repo

Our custom Project
Check that you have already installed the repository. It is the same used on host section.

https://github.com/Tonix22/NuandRFProyect

```
$cd NuandRFProyect/FPGA_FW
```

framework.sh

There was written an script in order to deal with multiple comands and setup. Read the file first and check that the variable paths are setup correctly. If not so, then search for the current file paths and update the script.

Download bitstream

First ensure that board is power by an exteneral source of 5V, and JTAG port is connected correctly.

```
$./framework.sh -bitstream
```

Known Issues, USB blaster, Linux

Permisions problems on JTAG port https://rocketboards.org/foswiki/Documentation/UsingUSBBlasterUnderLinux

Compile Code

You could go to dupinSoC/fw/<project_folder>, and then use simple **make** comand to compile and **make clean** to clear compiled cache. In the other hand you could use the framework script .sh. In some cases script will be more comfortable. Note: Folder's names are in dupinSoC/fw/

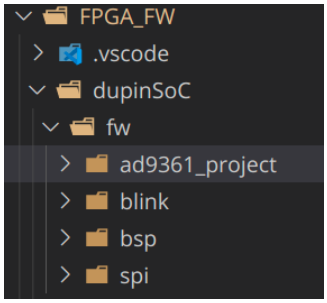

Figure 11: Projects example in dupinSoC/fw/

Compile

```
$./framework.sh -all <project_folder_name>
For example
$./framework.sh -all ad9361_project
```

Clean

```
$./framework.sh -clean <project_folder_name>
For example
$./framework.sh -clean ad9361_project
```

Open ocd

Ocd is an interface to debug the board using gdb, check that the debug port is connected, and then open debug sesion.

```
$./framework.sh -ocd
```

Debug

Once ocd is open , go to the debug tab in vscode, ensure you are on the FPGA_FW folder. Click on run -> add configuration.

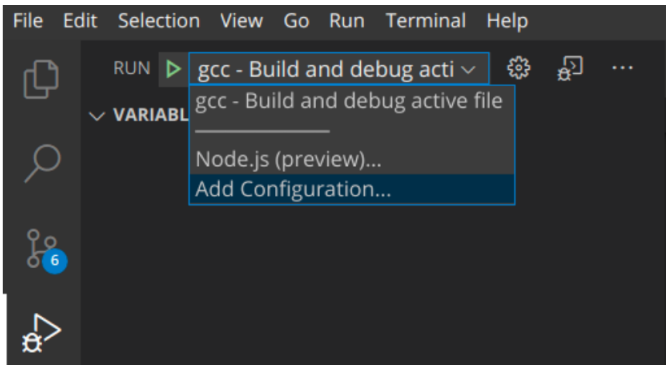

Figure 12: Add config

Vscode will open a .vscode/task.json file which will be configure as the image below. Check that the elf path and riscv-gdb match with yours.

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/
    "version": "0.2.0",
    "configurations": [
        {
            "name": "GDB",
            "type": "gdb",
            "request": "launch",
            "cwd": "${workspaceRoot}",
            "target": "${workspaceRoot}/blink/debug/blink.elf",
            "gdbpath" : "/opt/riscv/bin/riscv64-unknown-elf-gdb",
            "autorun": [
                "target remote localhost:3333",
                "set remotetimeout 60",
                "set arch riscv:rv32",
                "monitor reset halt",
                "load"
            ]
        }
    ]
}
```

Figure 13: .vscode/task.json

Go to the debug menu, and click the green triangle. Debug may start in a .S file, after some step over, it will go to the main.c file. A shortcut you can use, is to setup a breakpoint in the main, and just run the code until reach the breakpoint.

```
10  void delay()
11  {
12      for(uint32_t idx = 0;idx < 500000;idx++) asm volat
13  }
14
15  void main() {
16      // Set GPIO LSB as outputs
17      GPIO->output_enable = 0x0000FFFF;       Edgardo Ser
18      // Set initial value of GPIO
19          GPIO->output        = 0x00000000;
20
21      while(1){
22                  Digitalwrite(0xFF00);
23                  delay();
24          Digitalwrite(0x00FF);
25          delay();
26
27          for(uint32_t j = 0; j <=7; j++)
28          {
29              GPIO->output <<= 1;
30              delay();
31          }
32
33          for(uint32_t j = 0; j < 7; j++)
```

Figure 14: Debug working

Firmware structure