Part 1. Hardware description

My hardware set-up is the following:

STM32F4 Discovery board

USB-Serial converter (such as FTDI or PL2303)

5v<->3v logic level converter

Can Transceiver TJA1042 but can be any equivalent

Arduino female-female cables

Here is a sketch of the connections:

CAN\_L

CAN\_H

Tx

Rx

Tx

Rx

3v-5v converter

USB-USB programmer

USB-serial

CAN transceiver

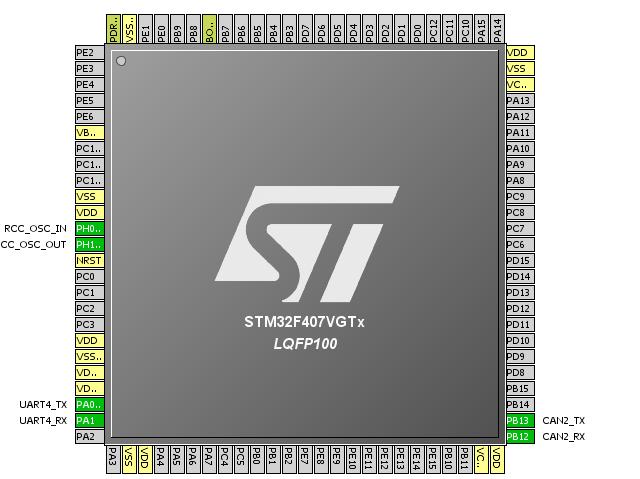
I used a 3v-5v logic converter because my transceiver works with 5volt level and the Discovery microcontroller works with 3.3volts. If you have a 3v transceiver, then you don’t need the converter.

Please note that the USB-serial converter must work with 3.3v level, otherwise you have to route the wires through the 3v-5v converter.

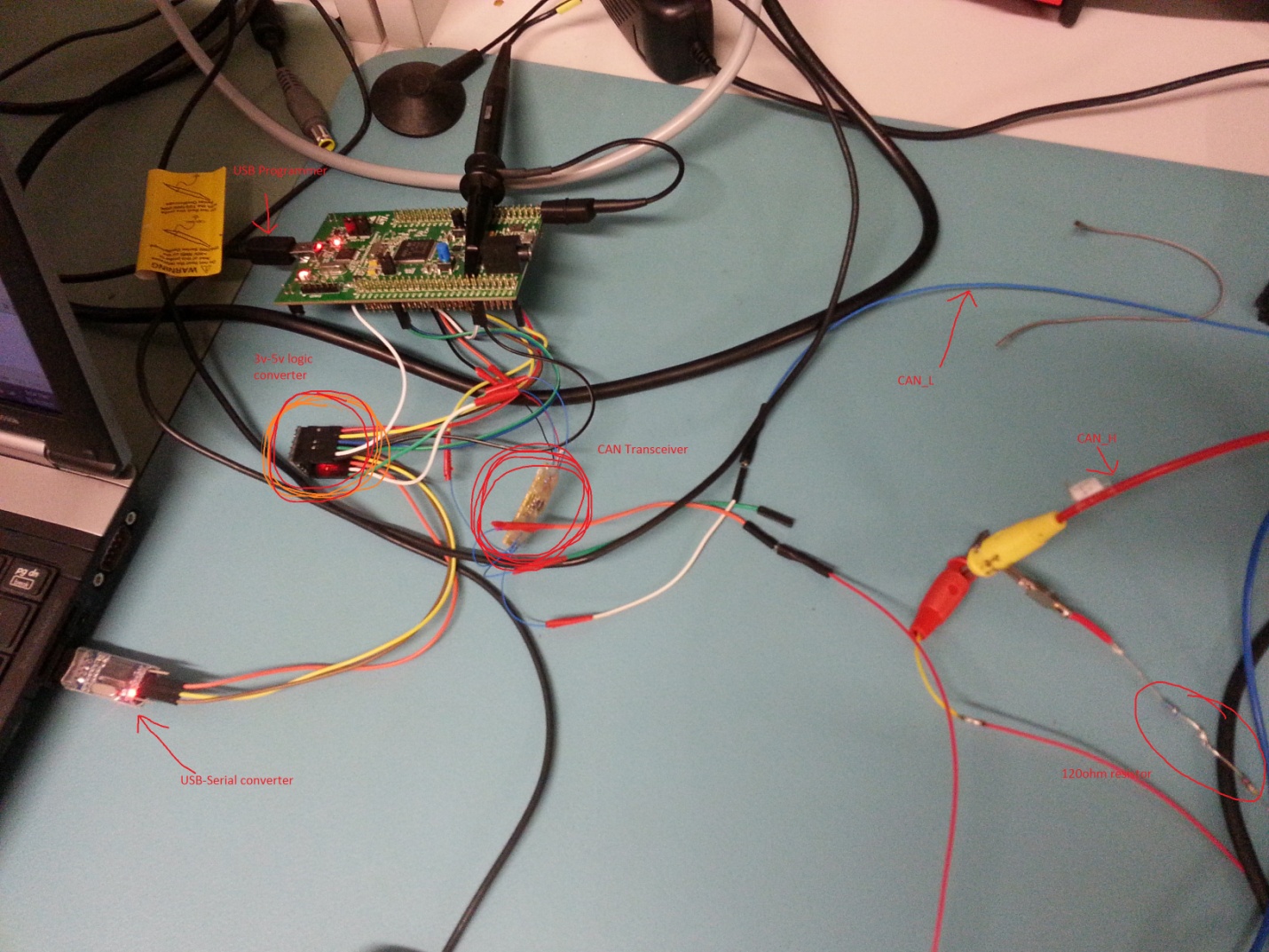
Sometimes the CAN bus requires a 120ohm resistor between CAN\_L and CAN\_H lines. In my case I had to put this resistance for the communication to work. This is called a line termination and ECU’s in a car have this usually.

My CAN transceiver also has a STBN pin which needs to be pulled down for the transceiver to work properly.

Here is a picture of the microcontroller pins to be used:



Here’s my setup explained:



Part 2 Software configuration

As a prerequisite, you need Keil uVision v4 or v5(recommended) installed. Keil uVision can be downloaded from their site freely, it is only limited to 32kb of executable size (more than enough for us).

The project can be started from accessing the folder CAN\_project\_v1\_0\Projects\MDK-ARM\ and double click on CAN\_project.uvproj

You can Compile and flash the software on the Discovery board by clicking 2 buttons (Compile All followed by Download).

After this, you need to install an application to handle the serial communication between the Discovery and the PC. I used Arduino IDE -> Serial Monitor. Please select the correct COM port and 115200 baudrate and “No line ending”.

From the serial monitor application, you can issue commands now to the Discovery board.

Here are the commands implemented:

Commands implemented:

-> “1” = send single frame

-> “2” = send repetitive frame each 1 ms

-> “3” = send repetitive frame each 5 ms

-> “4” = send repetitive frame each 10 ms

-> “5” = stop repetitive frames

-> “6” = baudrate 125kbps

-> “7” = baudrate 250 kbps

-> “8” = baudrate 500kbps

-> “9” = baudrate 1Mbps

-> “A” = read bus for 5 seconds

If you need to input a custom CAN frame, please change the following defines (file state\_handler.c):

#define canMASK 0x0 🡨 Mask for the input frames (which frame is accepted)

#define canID 0x0 🡨 Which ID is accepted

#define txID 0x5 🡨 ID of the TX message

#define txDLC 2 🡨 How many bytes you want to send in the TX message (maximum 8 bytes)

#define txPayload 0xAA 🡨 This is the payload for the TX frame (stuffing byte)

Please note that for any command that you will issue, the Software will respond with a status so that you know what it is doing. Look for the response on the serial communication window.

About the incoming frame filtering: the software is configured so that it will accept any frame on the CAN bus. This is not the best choice because it can overload the application. If you need to see only a frame, you need to change the canMASK and canID macros.

canMASK -> any bit “1” means that that bit is a “care” bit, while “0” means “don’t care”

canID -> which ID will be accepted

The easyest way to configure is:

#define canMASK 0xFF 🡨 Mask for the input frames (which frame is accepted)

#define canID **{put here the ID you want to receive}** 🡨 Which ID is accepted

I hope this information is helpful for you.

If you need more help, please contact me.