Seeed-Studio / **CAN_BUS_Shield**

CAN Bus Shield - MCP2515&MCP2551

MIT License

468 stars **313** forks

Star Watch

Code Issues 31 Pull requests Actions Projects Wiki Security Insights

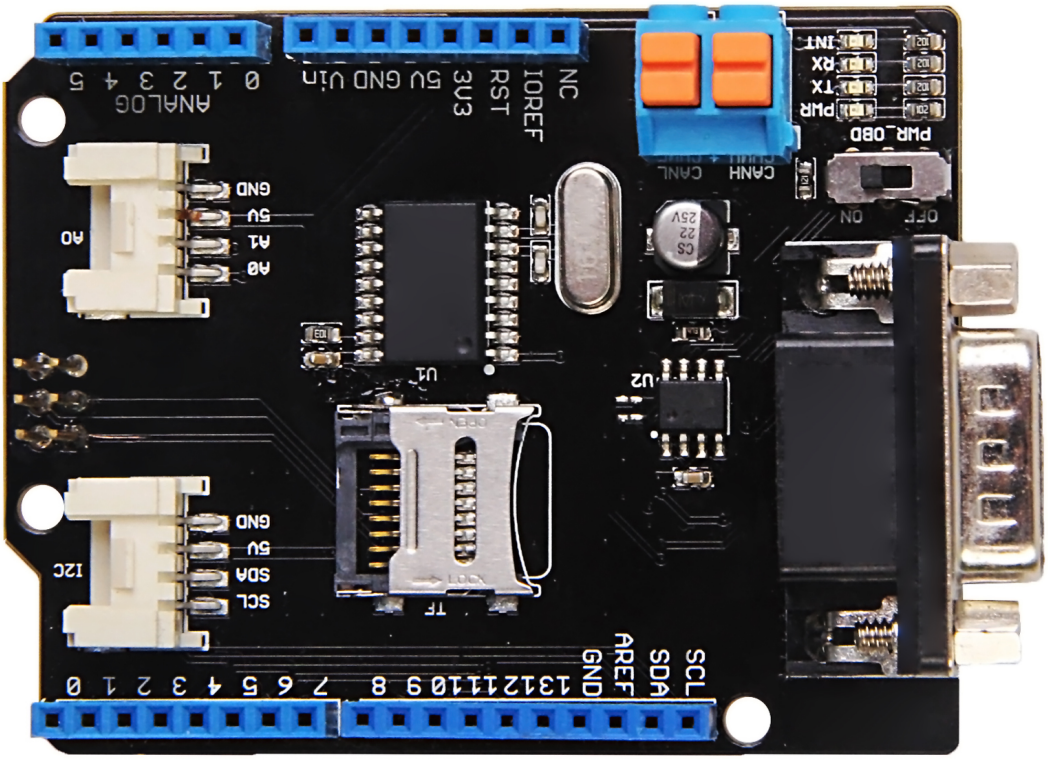
master

baorepo authored and **Pillar1989** committed on Mar 30 ···

View code

README.md

CAN BUS Shield build failing



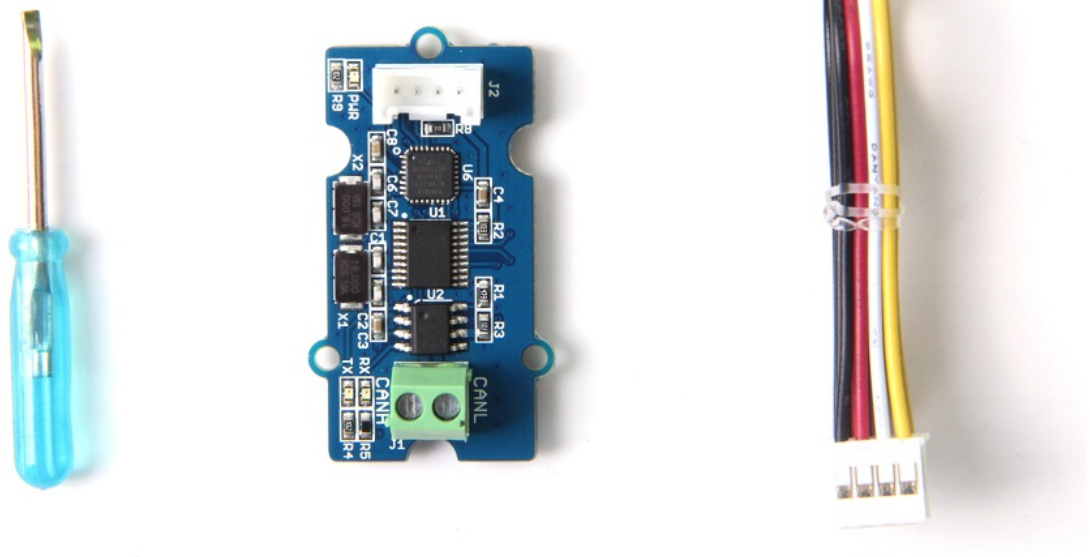
CAN-BUS Shield V2

CAN-BUS is a common industrial bus because of its long travel distance, medium communication speed and high reliability. It is commonly found on modern machine tools and as an automotive diagnostic bus. This CAN-BUS Shield adopts MCP2515 CAN Bus controller with SPI interface and MCP2551 CAN transceiver to give your Arduino/Seeeduino CAN-BUS capability. With an OBD-II converter cable added on and the OBD-II library imported, you are ready to build an onboard diagnostic device or data logger.

- Implements CAN V2.0B at up to 1 Mb/s
- SPI Interface up to 10 MHz
- Standard (11 bit) and extended (29 bit) data and remote frames
- Two receive buffers with prioritized message storage
- Industrial standard 9 pin sub-D connector
- Two LED indicators

Alternative Choice

If your project is space limited, here is a Serial CAN-BUS module which has the full features of CAN Bus. The Serial CAN-BUS provides your Arduino or others MCU with the capability to communication to CAN Bus, such as hacking your vehicle. This Grove CAN-BUS module is controled by UART, that means if your MCU has a UART interface, this serial CAN BUS is available. Good news is that this Serial CAN BUS module needs only $14.9



Installation:

```
git clone https://github.com/Seeed-Studio/CAN_BUS_Shield.git
```

or download the zip.

Usage:

Simply copy the CAN_BUS_Shield folder to your Arduino library collection. For example, arduino-1.6.12/libraries. Next time you run the Arduino IDE, you'll have a new option in Sketch -> Include Library -> CAN_BUS_Shield. Review the included examples in CAN_BUS_Shield/examples.

1. Set the BaudRate

This function is used to initialize the baudrate of the CAN Bus system.

The available baudrates are listed as follows:

```
#define CAN_5KBPS    1
#define CAN_10KBPS   2
#define CAN_20KBPS   3
#define CAN_25KBPS   4
#define CAN_31K25BPS 5
#define CAN_33KBPS   6
#define CAN_40KBPS   7
#define CAN_50KBPS   8
#define CAN_80KBPS   9
#define CAN_83K3BPS  10
#define CAN_95KBPS   11
```

```
#define CAN_100KBPS  12
#define CAN_125KBPS  13
#define CAN_200KBPS  14
#define CAN_250KBPS  15
#define CAN_500KBPS  16
#define CAN_666kbps  17
#define CAN_1000KBPS 18
```

2. Set Receive Mask and Filter

There are 2 receive mask registers and 5 filter registers on the controller chip that guarantee you get data from the target device. They are useful especially in a large network consisting of numerous nodes.

We provide two functions for you to utilize these mask and filter registers. They are:

```
init_Mask(unsigned char num, unsigned char ext, unsigned char ulData);
init_Filt(unsigned char num, unsigned char ext, unsigned char ulData);
```

num represents which register to use. You can fill 0 or 1 for mask and 0 to 5 for filter.

ext represents the status of the frame. 0 means it's a mask or filter for a standard frame. 1 means it's for a extended frame.

ulData represents the content of the mask of filter.

3. Check Receive

The MCP2515 can operate in either a polled mode, where the software checks for a received frame, or using additional pins to signal that a frame has been received or transmit completed. Use the following function to poll for received frames.

```
INT8U MCP_CAN::checkReceive(void);
```

The function will return 1 if a frame arrives, and 0 if nothing arrives.

4. Get CAN ID

When some data arrives, you can use the following function to get the CAN ID of the "send" node.

```
INT32U MCP_CAN::getCanId(void);
```

5. Send Data

```
CAN.sendMsgBuf(INT32U id, INT8U ext, INT8U len, INT8U *buf);
```

This is a function to send data onto the bus. In which:

id represents where the data come from.

ext represents the status of the frame. '0' means standard frame. '1' means extended frame.

len represents the length of this frame.

buf is the content of this message.

For example, In the 'send' example, we have:

```
unsigned char stmp[8] = {0, 1, 2, 3, 4, 5, 6, 7};

CAN.sendMsgBuf(0x00, 0, 8, stmp); //send out the message 'stmp' to the bus
and tell other devices this is a standard frame from 0x00.
```

6. Receive Data

The following function is used to receive data on the 'receive' node:

```
CAN.readMsgBuf(INT8U *len, INT8U *buf);
```

Under the condition that masks and filters have been set, this function will only get frames that meet the requirements of those masks and filters.

len represents the data length.

buf is where you store the data.

7. Check additional flags

When frame is received you may check whether it was remote request and whether it was an extended (29bit) frame.

```
CAN.isRemoteRequest();
CAN.isExtendedFrame();
```

return value is '0' for a negative response and '1' for a positive

8. Sleep Mode

By setting the MCU, the CAN controller (MCP2515) and the transceiver (MCP2551) into sleep mode, you can reduce the power consumption of the whole setup from around 50mA down to 240uA (Arduino directly connected to 5V, regulator and power LED removed). The node will wake up when a new message arrives, process the message and go back to sleep afterwards.

Look at the examples "receive_sleep" and "send_sleep" for more info.

In order to set the MCP2551 CAN transceiver on the shield into sleep/standby mode, a small hardware modification is necessary. The Rs pin of the transceiver must either be connected to pin RX0BF of the MCP2515 or to a free output of the Arduino, both via the resistor R1 (17k). Cut the connection to ground after R1 and solder in a wire to one of the pins. Note however that from now on, you have to pull this pin low in software before using the transceiver. Pulling the pin high will set the transceiver into standby mode.

Without this modification, the transceiver will stay awake and the power consumption in sleep mode will be around 8mA - still a significant improvement!

For more information, please refer to wiki page.

Contributing to this software is warmly welcomed. You can do this basically by forking, committing modifications and then pulling requests (follow the links above for operating guide). Adding change log and your contact into file header is encouraged. Thanks for your contribution.

Seeed Studio is an open hardware facilitation company based in Shenzhen, China. Benefiting from local manufacture power and convenient global logistic system, we integrate resources to serve new era of innovation. Seeed also works with global distributors and partners to push open hardware movement.

Latest release

v1.3.0
on Apr 3, 2018

+ 1 release

Contributors 34

+ 23 contributors

Languages

- **C++** 83.7%
- **C** 16.3%