CAN-BUS Shield

Contents

Introduction

CAN-BUS is a common industrial bus because of its long travel distance, medium communication speed and high reliability. It is widely used as the automotive diagnostic bus, and also commonly used on modern machine tools. This CAN-Bus shield gives the Arduino CAN-Bus capability, It uses the Microchip MCP2515 CAN controller with MCP2551 CAN transceiver, the CAN connection is via a standard 9-way sub-D for use with OBD-II cable, ideal for automotive CAN application. The shield also has a Micro SD card holder, which helps you store the diagnostic information in the SD card directly, making this shield ideal for data logging application.

Model: AS54887CAN (http://www.elecrow.com/canbus-shield-p-1133.html)

Features

- Arduino Mega and Leonardo compatible
- 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
- SD card holder for information storage
- Dimensions(mm):74.3(L)x53.6(W)x23.5(H)

Usage

Hardware Installation

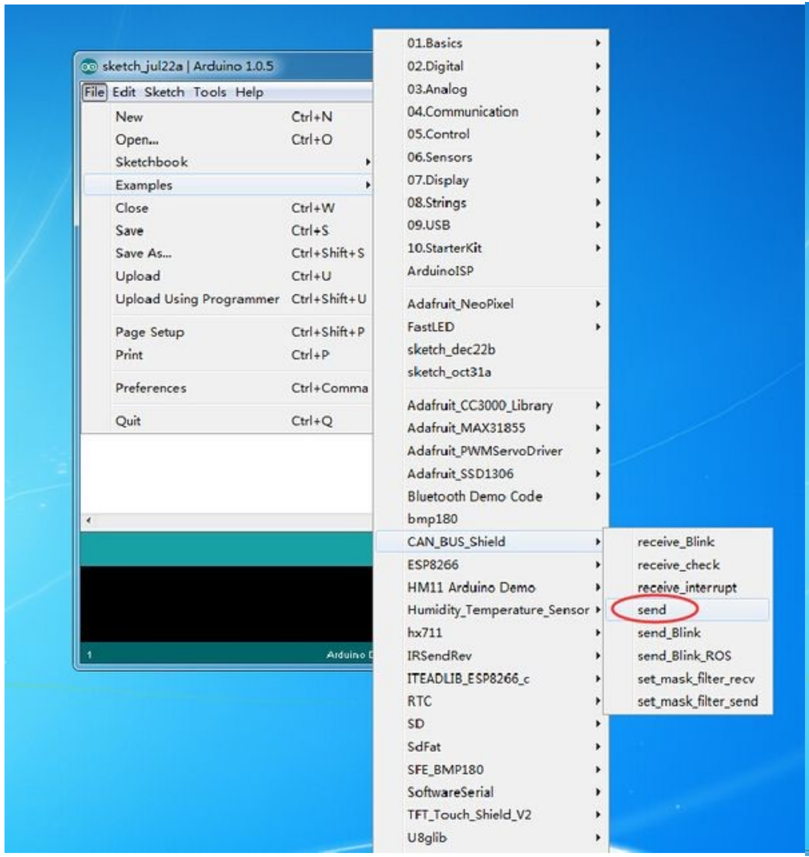
Plug the CAN-BUS Shield onto the Arduino. And then connect the Crowduino to PC with USB cable.

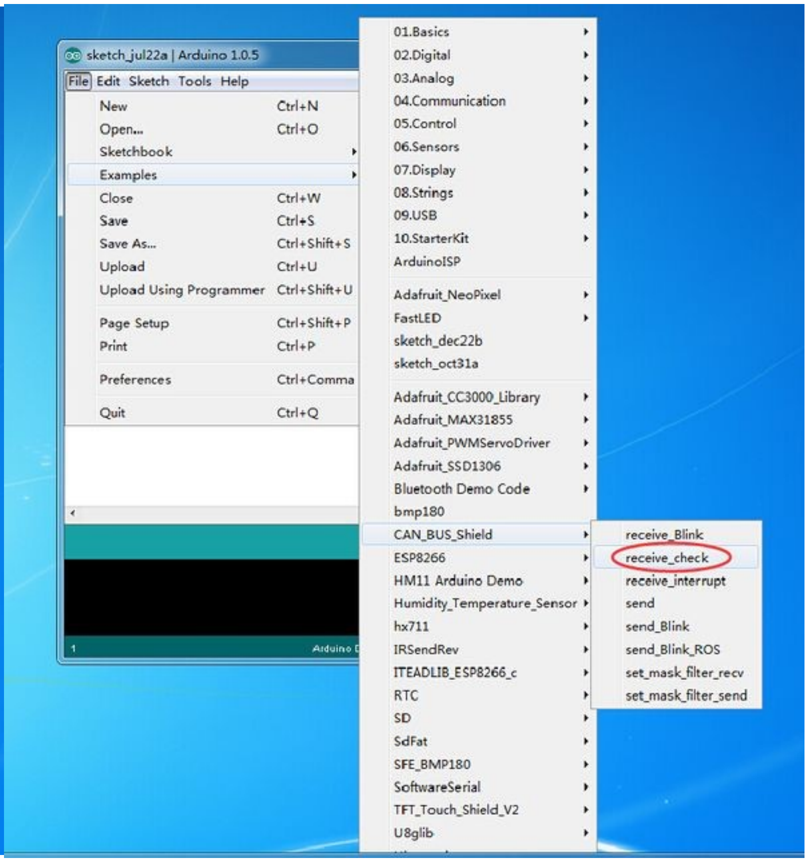
Upload the program

1. Download the CAN-BUS Source code file for Arduino 1.0 (http://www.elecrow.com/wiki/index.php?title=File:CAN_BUS_Shield_master.zip) and release it in the libraries file in the Arduino-1.0 program.: ..\arduino-1.0\libraries.

If the folder name include "-master", just remove it.

2. Open the Arduino-1.0, and you will find 8 examples: receive_check ,send and receive_interrupt and so on. Here we'll use send and receive_check, open it then you should get two programming windows now.





2.1 Send data:

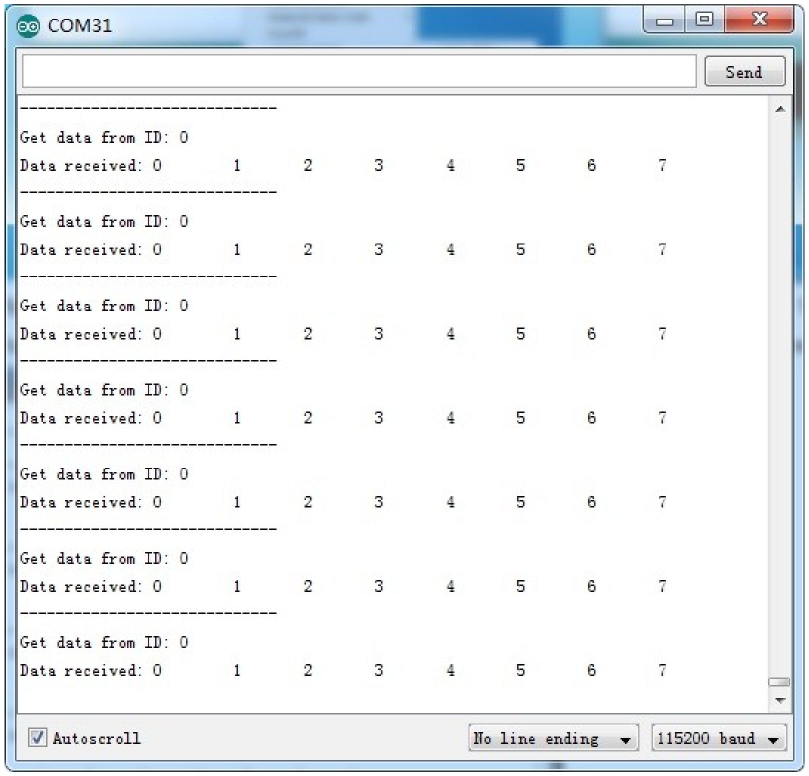
```
// demo: CAN-BUS Shield, send data
#include <mcp_can.h>
#include <SPI.h>

// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 9;

MCP_CAN CAN(SPI_CS_PIN);                                    // Set CS pin

void setup()
{
    Serial.begin(115200);

START_INIT:

    if(CAN_OK == CAN.begin(CAN_500KBPS))                   // init can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init ok!");
    }
    else
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println("Init CAN BUS Shield again");
        delay(100);
        goto START_INIT;
    }
}

unsigned char stmp[8] = {0, 1, 2, 3, 4, 5, 6, 7};
void loop()
{
    // send data:  id = 0x00, standrad frame, data len = 8, stmp: data buf
    CAN.sendMsgBuf(0x00, 0, 8, stmp);
    delay(100);                       // send data per 100ms
}
```

2.2 Receive data

```
// demo: CAN-BUS Shield, receive data with check mode
// send data coming to fast, such as less than 10ms, you can use this way


#include <SPI.h>
#include "mcp_can.h"


// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 9;

MCP_CAN CAN(SPI_CS_PIN);                                    // Set CS pin

void setup()
{
    Serial.begin(115200);

START_INIT:

    if(CAN_OK == CAN.begin(CAN_500KBPS))                   // init can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init ok!");
    }
    else
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println("Init CAN BUS Shield again");
        delay(100);
        goto START_INIT;
    }
}


void loop()
{
    unsigned char len = 0;
    unsigned char buf[8];
    if(CAN_MSGAVAIL == CAN.checkReceive())            // check if data coming
    {
        CAN.readMsgBuf(&len, buf);    // read data,  len: data length, buf: data buf

        unsigned char canId = CAN.getCanId();

        Serial.println("-----------------------------");
        Serial.println("get data from ID: ");
        Serial.println(canId);

        for(int i = 0; i<len; i++)    // print the data
        {
            Serial.print(buf[i]);
            Serial.print("\t");
        }
        Serial.println();
    }
}
```

3. Upload two examples to two boards separately. Choose the board via the path: Tools -->Serial Port-->COMX. Note down which board is assigned as a "send" node and which board is assigned as a "receive" node.

4. Open the "Serial Monitor" on the "receive" COM, you will get message sent from the "send" node. Here we have the preset message "0 1 2 3 4 5 6 7" showing in the following picture.



```
-----------------------------
Get data from ID: 0
Data received: 0	1	2	3	4	5	6	7
-----------------------------
Get data from ID: 0
Data received: 0	1	2	3	4	5	6	7
-----------------------------
Get data from ID: 0
Data received: 0	1	2	3	4	5	6	7
-----------------------------
Get data from ID: 0
Data received: 0	1	2	3	4	5	6	7
-----------------------------
Get data from ID: 0
Data received: 0	1	2	3	4	5	6	7
-----------------------------
Get data from ID: 0
Data received: 0	1	2	3	4	5	6	7
-----------------------------
Get data from ID: 0
Data received: 0	1	2	3	4	5	6	7
```

Reference

1. Set the BaudRate

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

The available baudrates are listed as follws:

CAN_5KBPS, CAN_10KBPS, CAN_20KBPS, CAN_40KBPS, CAN_50KBPS, CAN_80KBPS, CAN_100KBPS, CAN_125KBPS, CAN_200KBPS, CAN_250KBPS, CAN_500KBPS and CAN_1000KBPS

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 arrive, 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(INT8U id, INT8U ext, INT8U len, data_buf);

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.

"data_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(unsigned char len, unsigned char buf);

In conditions that masks and filters have been set. This function can only get frames that meet the requirements of masks and filters.

"len" represents the data length.

"buf" is where you store the data.

Resources

- CAN-BUS Source code file for Arduino 1.0 (http://www.elecrow.com/wiki/index.php?title=File:CAN_BUS_Shield_master.zip)
- MCP2551 datasheet (http://www.elecrow.com/wiki/index.php?title=File:Mcp2551en.pdf)
- MCP2515 datasheet (http://www.elecrow.com/wiki/index.php?title=File:MCP2515.pdf)

Retrieved from "https://www.elecrow.com/wiki/index.php?title=CAN-BUS_Shield&oldid=63"

This page was last edited on 22 August 2019, at 04:08.