

# Controlling over I2C port

Applies to EVB.1.2.0 firmware where I2C is implemented

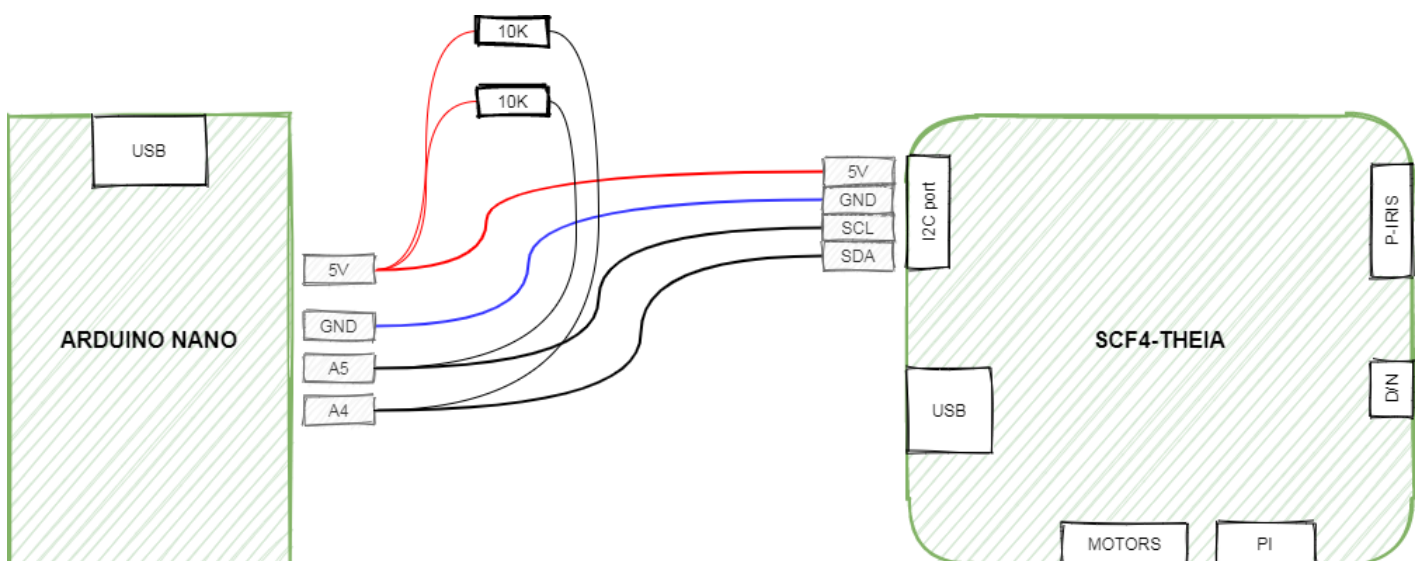
Slave address: 0x33

## About

In order to simplify I2C communication, firmware utilizes one one direction read/write operations. All commands expect 5 bytes, where first byte is function address, remaining bytes is payload.

USB-CDC works in parallel with I2C functionality and independently, but it is recommended to use single communication channel once controller is initialized.

## Wiring



In order to avoid power loops, do not connect both (SCF4 and Arduino) USB ports at the same time.

## Write data

All commands are fixed length consisting of 5 bytes.

W: I2C ADDR	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
-------------	----------	-------	-------	-------	-------

## Read data

First issue write command (register values are ignored). This step performs necessary calculations and fills memory with registers ready for reading in next step.

W: I2C ADDR	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
-------------	----------	-------	-------	-------	-------

Read command also consists of 5 bytes. FUNCTION repeats last write command value.

R: I2C ADDR	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
-------------	----------	-------	-------	-------	-------

## Commands

ADDR	REGISTER NAME	R/W	FUNCTION
0x02	RESET_CPU	W	Reset STM32 CPU
0x03	INIT_DRV	W	Reset and initialize motor driver
0x05	AUX_OUT	W	Control AUX output on/off
0x06	MODE	W	Normal / forced move
0x07	STOP	W	Compulsory stop
0x08	PI_LEDS	W	Switch on ON or OFF PI LEDs
0x09	MOTOR_SLEEP_PWR	W	Set motor sleep power
0x0A	MOTOR_DRV_PWR	W	Set motor working power
0x0B	MOTOR_SPEED	W	Set motor speed
0x0C	PI_THRESHOLD	W	Set PI detector thresholds
0x0D	READ_STATUS	R	Read controller status
0x0E	SET_MOTOR_POS	W	Set current motor position
0x0F	SET_MICROSTEPPING	W	Set motor micro-stepping mode
0x10	DN_SWITCH	W	IR fitler

0x20	<b>MOVE</b>	W	Move motor
------	-------------	---	------------

## Command explanation

### **RESET\_CPU** - Reset CPU

Resets CPU. Other values are ignored.

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Reset CPU	0x02	ignored	ignored	ignored	<b>0x32</b>

### **INIT\_DRV** - Reset motor driver

Resets and initializes motor controller. Other values are ignored.

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Initialize motor driver	0x03	ignored	ignored	ignored	<b>0x32</b>

### **AUX\_OUT** - Control AUX output

Controls GPIO output

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Set AUX to Low	0x05	ignored	ignored	ignored	<b>0x00</b>
Set AUX to High	0x05	ignored	ignored	ignored	<b>0x01</b>

### **MODE** - Normal / forced move

Selects between normal and normal+forced move mode

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Normal move mode	0x06	ignored	ignored	<b>Channel</b>	<b>0x00</b>
Normal+Forced move mode	0x06	ignored	ignored	<b>Channel</b>	<b>0x01</b>

Channel encoding:

Channel	BYTE3 value
A	0x01

B	0x02
C	0x03

## **STOP** - Compulsory stop

Stop movement of all motors

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Reset CPU	0x07	ignored	ignored	ignored	<b>0x32</b>

## **PI\_LEDS** - Switch on ON or OFF PI LEDs

Control LEDs used in homing procedure.

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
OFF	0x08	ignored	ignored	ignored	<b>0x00</b>
ON	0x08	ignored	ignored	ignored	<b>0x01</b>

## **MOTOR\_SLEEP\_PWR** - Set motor sleep power

Set motor sleep current

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Set current	0x09	ignored	ignored	<b>Channel</b>	<b>Power</b>

Channel encoding:

Channel	BYTE3 value
A	0x01
B	0x02
C	0x03

## **MOTOR\_DRV\_PWR** - Set motor working power

Set motor operating current

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Set current	0x0A	ignored	ignored	<b>Channel</b>	<b>Power</b>

Channel encoding:

Channel	BYTE3 value
A	0x01
B	0x02
C	0x03
D	0x04

**MOTOR\_SPEED** - Set motor speed

Set motor speed

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Set speed	0x0B	ignored	Channel	Speed [15:8]	Speed [7:0]

Channel encoding:

Channel	BYTE2 value
A	0x01
B	0x02
C	0x03

**PI\_THRESHOLD** - Set PI detector thresholds

Set limit switch optocoupler detector thresholds

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Set threshold	0x0C	ignored	Channel	Signal [15:8]	Signal [7:0]

Channel encoding:

Channel	BYTE2 value
A LOW	0x01
B LOW	0x02

C LOW	0x03
A HIGH	0x04
B HIGH	0x05
C HIGH	0x06

READ\_STATUS

- Read controller status

Read motor status, PI status and motor positions

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Set current	0x0D	ignored	ignored	ignored	Channel

Channel encoding:

Channel	BYTE2 value
A position	0x01
B position	0x02
C position	0x03
PI_A status	0x04
PI_B status	0x05
PI_C status	0x06
A moving	0x07
B moving	0x08
C moving	0x09

Returns:

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Status return value	0x0D	Value [31:24]	Value [23:16]	Value [15:8]	Value [7:0]

## SET\_MOTOR\_POS - Set position

Redefine current position

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Set position	0x0E	Channel	Speed [23:16]	Speed [15:8]	Speed [7:0]

Channel encoding:

Channel	BYTE1 value
A	0x01
B	0x02
C	0x03

## SET\_MICROSTEPPING - Set microstepping mode

Set microstepping mode for defined channel

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
Set microstepping	0x0F	ignored	ignored	Channel	Mode

Channel encoding:

Channel	BYTE1 value
A	0x01
B	0x02
C	0x03

## MOVE - Move motor

Move motor defined step count.

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
-------------	----------	-------	-------	-------	-------

Steps	0x20	Channel	Direction	Steps [15:8]	Steps [7:0]
-------	------	---------	-----------	--------------	-------------

Channel encoding:

Channel	BYTE2 value
A	0x01
B	0x02
C	0x03

MAX steps is: 0xFFFF-1 = **0xFFFE**

Theia lens exceeds 0xFFFF step count, thus absolute positioning has to be implemented on client side code

**DN\_SWITCH** - IR filter swith

Switch filter to day or night position

Description	FUNCTION	BYTE1	BYTE2	BYTE3	BYTE4
POS1	0x10	ignored	ignored	ignored	<b>0x00</b>
POS2	0x10	ignored	ignored	ignored	<b>0x01</b>

## Arduino sketch example

Demo output should look like

```
SCF4-M I2C tester
```

```
Init driver
```

```
Set microstepping
```

```
Move mode
```

```
PI LEDS
```

```
Set drive pwr
```

```
Set sleep pwr
```

```
Set motor speeds
```

```
Set PI thresholds
```

```
Read status: posA
```



```
D 0 0 0 0
Read status: PI A
D 0 0 0 1
Homing A
Homing B
Move +A
Move -A
Move +B
Move -B
Move +C
Move -C
Set position
DN 0
DN 1
DN 0
DN 1
Last return from I2C port: ok
Loop...
```

tester.ino

```
#include <Wire.h>

#define CH_A 0x01
#define CH_B 0x02
#define CH_C 0x03
#define CH_D 0x04

#define CCW 0
#define CW 1

void err_print(byte err)
{
    if (err == 0)
    {
        Serial.println("ok");
    }
    else if (err==4)
    {
```

```
        Serial.println("failed");
    }
}

void setup()
{
    Wire.begin();
    Serial.begin(115200);
    while (!Serial);
    Serial.println("\nSCF4- M I2C tester\n");
}

void loop()
{
    byte error;

    Serial.println("Init driver");
    SCF4_INIT_DRIVER();

    Serial.println("Set microstepping");
    SCF4_MICROSTEPPING(2, CH_A);
    SCF4_MICROSTEPPING(2, CH_B);
    SCF4_MICROSTEPPING(6, CH_C);

    Serial.println("Move mode");
    SCF4_MODE(0x00, CH_A);
    SCF4_MODE(0x00, CH_B);
    SCF4_MODE(0x00, CH_C);

    Serial.println("PI LEDS");
    SCF4_PI_LEDS(0x01);

    Serial.println("Set drive pwr");
    SCF4_DRV_PWR(180, CH_A);
    SCF4_DRV_PWR(180, CH_B);
    SCF4_DRV_PWR(180, CH_C);
    SCF4_DRV_PWR(90, CH_D);

    Serial.println("Set sleep pwr");
    SCF4_SLEEP_PWR(50, CH_A);
```

```
SCF4_SLEEP_PWR( 50, CH_B);
```

```
SCF4_SLEEP_PWR( 50, CH_C);
```

```
Serial.println("Set motor speeds");
```

```
SCF4_MOTOR_SPEED( 5000, CH_A);
```

```
SCF4_MOTOR_SPEED( 5000, CH_B);
```

```
SCF4_MOTOR_SPEED( 5000, CH_C);
```

```
Serial.println("Set PI thresholds");
```

```
SCF4_PI_THRESHOLD( 2000, 0x01);
```

```
SCF4_PI_THRESHOLD( 2000, 0x02);
```

```
SCF4_PI_THRESHOLD( 2000, 0x03);
```

```
SCF4_PI_THRESHOLD( 3000, 0x04);
```

```
SCF4_PI_THRESHOLD( 3000, 0x05);
```

```
SCF4_PI_THRESHOLD( 3000, 0x06);
```

```
Serial.println("Read status: posA");
```

```
SCF4_READ_STATUS( 0x01);
```

```
Serial.println("Read status: PI A");
```

```
SCF4_READ_STATUS( 0x04);
```

```
Serial.println("Homing A");
```

```
MOVE( 30000, CW, CH_A);
```

```
delay( 2000);
```

```
SCF4_MODE( 0x01, CH_A);
```

```
MOVE( 0x100, CCW, CH_A);
```

```
delay(15000); // status reading is not implemented, for testing 15s timeout is used
```

```
SCF4_MODE( 0x00, CH_A);
```

```
SET_MOTOR_POS(100, CH_A);
```

```
Serial.println("Homing B");
```

```
MOVE( 30000, CW, CH_B);
```

```
delay( 2000);
```

```
SCF4_MODE( 0x01, CH_B);
```

```
MOVE( 0x100, CCW, CH_B);
```

```
delay(15000); // status reading is not implemented, for testing 15s timeout is used
```

```
SCF4_MODE( 0x00, CH_B);
```

```
SET_MOTOR_POS(100, CH_B);

// normal operation starts here


Serial.println("Move +A");
MOVE(0xFFFE, CW, CH_A);
delay(5000);
Serial.println("Move -A");
MOVE(0xFFFE, CCW, CH_A);
delay(5000);


Serial.println("Move +B");
MOVE(0xFFFE, CW, CH_B);
delay(5000);
Serial.println("Move -B");
MOVE(0xFFFE, CCW, CH_B);
delay(5000);


Serial.println("Move +C");
MOVE(1000, CW, CH_C);
delay(2000);
Serial.println("Move -C");
MOVE(1000, CCW, CH_C);
delay(2000);


Serial.println("Set position");
//SET_MOTOR_POS(100, 0x01);
//SET_MOTOR_POS(200, 0x01);
//SET_MOTOR_POS(300, 0x01);


//STOP();


Serial.println("DN 0");
DN_SWITCH(0);
delay(1000);
Serial.println("DN 1");
```

```

    DN_SWITCH(1);
    delay(1000);
    Serial.println("DN 0");
    DN_SWITCH(0);
    delay(1000);
    Serial.println("DN 1");
    DN_SWITCH(1);
    delay(1000);

    Serial.print("Last return from I2C port: ");
    err_print(error);

    Serial.println("Loop...");
    while(1)
    {
    }
}

```

`scf4_i2c.ino`

```

// SCL - A5
// SDA - A4

#include <Wire.h>
#define SCF4_ADDR 0x33

// SCF4 is not signaling busy status, thus fixed delay is added
// If next I2C command is sent too soon it might be ignored
#define I2C_SLEEP 200

byte SCF4_AUX(byte status)
{
    byte function = 0x05;
    byte error;

    //byte w1 = (counter&0xFF);
    //byte w2 = ((counter>>8) &0xFF);
    //byte w3 = ((counter>>16) &0xFF);
    //byte w4 = ((counter>>24) &0xFF);
}

```

```
Wire.beginTransmission( SCF4_ADDR);  
Wire.write( function);  
Wire.write( 0);  
Wire.write( 0);  
Wire.write( 0);  
Wire.write(status);  
error = Wire.endTransmission();  
delay( I2C_SLEEP);
```

```
return error;  
}
```

```
byte SCF4_RESET_CPU( void)
```

```
{  
    byte function = 0x02;  
    byte error;  
  
    Wire.beginTransmission( SCF4_ADDR);  
    Wire.write( function);  
    Wire.write( 0);  
    Wire.write( 0);  
    Wire.write( 0);  
    Wire.write( 0x32);  
    error = Wire.endTransmission();  
    delay( I2C_SLEEP);  
  
    return error;  
}
```

```
byte SCF4_INIT_DRIVER( void)
```

```
{  
    byte function = 0x03;  
    byte error;  
  
    Wire.beginTransmission( SCF4_ADDR);  
    Wire.write( function);  
    Wire.write( 0);  
    Wire.write( 0);  
    Wire.write( 0);  
    Wire.write( 0x32);
```

```

    error = Wire.endTransmission();
    delay(I2C_SLEEP);

    return error;
}

byte SCF4_MODE(byte mode, byte ch)
{
    byte function = 0x06;
    byte error;

    Wire.beginTransmission( SCF4_ADDR);
    Wire.write(function);
    Wire.write(0);
    Wire.write(0);
    Wire.write(ch);
    Wire.write(mode);
    error = Wire.endTransmission();
    delay(I2C_SLEEP);

    return error;
}

byte SCF4_PI_LEDS(byte mode)
{
    byte function = 0x08;
    byte error;

    Wire.beginTransmission( SCF4_ADDR);
    Wire.write(function);
    Wire.write(0);
    Wire.write(0);
    Wire.write(0);
    Wire.write(mode);
    error = Wire.endTransmission();
    delay(I2C_SLEEP);

    return error;
}

byte SCF4_SLEEP_PWR(byte pwr, byte ch)

```

```

{
    byte function = 0x09;
    byte error;

    Wire.beginTransmission( SCF4_ADDR);
    Wire.write( function);
    Wire.write( 0);
    Wire.write( 0);
    Wire.write( ch);
    Wire.write( pwr);
    error = Wire.endTransmission();
    delay( I2C_SLEEP);

    return error;
}

byte SCF4_DRV_PWR( byte pwr, byte ch)
{
    byte function = 0x0A;
    byte error;

    Wire.beginTransmission( SCF4_ADDR);
    Wire.write( function);
    Wire.write( 0);
    Wire.write( 0);
    Wire.write( ch);
    Wire.write( pwr);
    error = Wire.endTransmission();
    delay( I2C_SLEEP);

    return error;
}

byte SCF4_MOTOR_SPEED( int speed, byte ch)
{
    byte function = 0x0B;
    byte error;

    Wire.beginTransmission( SCF4_ADDR);
    Wire.write( function);
    Wire.write( 0);

```



```

Wire.write(ch);
Wire.write(highByte(speed));
Wire.write(lowByte(speed));
error = Wire.endTransmission();
delay(I2C_SLEEP);

return error;
}

byte SCF4_PI_THRESHOLD(int level, byte ch)
{
    byte function = 0x0C;
    byte error;

    Wire.beginTransmission(SCF4_ADDR);
    Wire.write(function);
    Wire.write(0);
    Wire.write(ch);
    Wire.write(highByte(level));
    Wire.write(lowByte(level));
    error = Wire.endTransmission();
    delay(I2C_SLEEP);

    return error;
}

byte SCF4_READ_STATUS(byte ch)
{
    byte function = 0x0D;
    byte error;

    /*
    * channel:
    * 0x00 - dummy, reads 0x87, 0x65, 0x43, 0x21
    * 0x01 - chA.position
    * 0x02 - chB.position
    * 0x03 - chC.position
    * 0x04 - piA.status
    * 0x05 - piB.status
    * 0x06 - piC.status
    * 0x07 - chA.moving

```

```
* 0x08 - chB.moving
* 0x09 - chC.moving
*/

Wire.beginTransmission( SCF4_ADDR);
Wire.write(function);
Wire.write(0);
Wire.write(0);
Wire.write(0);
Wire.write(ch);
error = Wire.endTransmission();

delay( I2C_SLEEP);

byte w1 = 0xff;
byte w2 = 0xff;
byte w3 = 0xff;
byte w4 = 0xff;
byte w5 = 0xff;

Wire.requestFrom( SCF4_ADDR, 5);
w1 = Wire.read();
w2 = Wire.read();
w3 = Wire.read();
w4 = Wire.read();
w5 = Wire.read();

Serial.print(w1, HEX);
Serial.print(" ");
Serial.print(w2, HEX);
Serial.print(" ");
Serial.print(w3, HEX);
Serial.print(" ");
Serial.print(w4, HEX);
Serial.print(" ");
Serial.print(w5, HEX);
Serial.println();

delay( I2C_SLEEP);

// TODO: return read values
```

```

    return error;
}

byte SET_MOTOR_POS(int pos, byte ch)
{
    byte function = 0x0E;
    byte error;

    Wire.beginTransmission( SCF4_ADDR);
    Wire.write( function);
    Wire.write( ch);
    Wire.write( 0);
    Wire.write( highByte( pos));
    Wire.write( lowByte( pos));
    error = Wire.endTransmission();
    delay( I2C_SLEEP);

    return error;
}

byte MOVE(unsigned int steps, byte dir, byte ch)
{
    byte function = 0x20;
    byte error;

    Wire.beginTransmission( SCF4_ADDR);
    Wire.write( function);
    Wire.write( ch);
    Wire.write( dir);
    Wire.write( highByte( steps));
    Wire.write( lowByte( steps));
    error = Wire.endTransmission();
    delay( I2C_SLEEP);

    return error;
}

byte DN_SWITCH(byte mode)
{
    byte function = 0x10;

```

```

byte error;

Wire.beginTransmission( SCF4_ADDR);
Wire.write( function);
Wire.write( 0);
Wire.write( 0);
Wire.write( 0);
Wire.write( mode);
error = Wire.endTransmission();
delay( I2C_SLEEP);

return error;
}

byte SCF4_MICROSTEPPING(byte stepping, byte ch)
{
    byte function = 0x0F;
    byte error;

    Wire.beginTransmission( SCF4_ADDR);
    Wire.write( function);
    Wire.write( 0);
    Wire.write( 0);
    Wire.write( ch);
    Wire.write( stepping);
    error = Wire.endTransmission();
    delay( I2C_SLEEP);

    return error;
}

byte STOP( void)
{
    byte function = 0x07;
    byte error;

    Wire.beginTransmission( SCF4_ADDR);
    Wire.write( function);
    Wire.write( 0);
    Wire.write( 0);
    Wire.write( 0);

```

```
Wire.write(0x32);  
error = Wire.endTransmission();  
delay(I2C_SLEEP);  
  
return error;  
}
```

---

Revision #76

Created 9 August 2020 13:14:02 by Saulius

Updated 6 February 2021 12:09:15 by Saulius