SBGC32_I2C_Drv expansion module reference manual

ver. 0.2 – 28.03.2015 – first edition
ver. 0.3 – 11.04.2015 – Add “Q&A”, “Flashing MCU” sections.
ver. 0.4 – 27.12.2017 – Add "API and examples" section

Overview

This module is intended to work as a part of the SimpleBGC camera stabilizer system, performing a motor driver function. Unlike regular scheme, where all motors and encoders are connected to the main board and driven by the single MCU, in the modular scheme each motor is driven by its own MCU, that lets to optimize cabling: encoder is integrated into the PCB for each module, and motor's cable goes to it by the shortest way. See Appendix A for an example of connection. Schematics, firmware and other resources you can find on the product's page: http://www.basecamelectronics.com/sbgc32_i2c_drv/

This expansion module is supported only in the encoder-enabled version of SimpleBGC firmware: http://www.basecamelectronics.com/encoders/

Features:

- **Flexibility -** This module can replace one or two motor+encoder pairs. Using it for all three motors is possible, but not recommended because I2C transfer on 400kHz rate will not fit into 800us cycle time. There is an option to increase I2C speed to 800kHz for small systems.
- **Low cost -** Each module consists of entry-level STM32F051 series MCU, cheap magnetic or analog encoder (6+ models are suported by firmware), FET- or IC-based motor driver, minimal number of other components.
- **Better cabling -** only 5 wires go between modules
- **Compact size -** MCU is available in UFQFPN32 package, allowing ultra-compact design and installation into small motors.
- **Reliable, low power consumption -** rotary encoders allow apply advanced FOC-algorithm for driving motors
- **Software compatibility -** fully compatible with SimpleBGC software stack (including GUI, mobile applications, Serial API, etc.)

Motor driver output circuit

Module provides 6 PWM outputs to drive 3 half-bridges at ultra-sonic frequency. 300-400ns dead-time is inserted. It lets to use simple FET-based output circuit, or use integrated circuits like DRV8839, DRV8313, and any other IC that encloses 3 half-bridges and all protection circuits. Several output driver circuits are provided in the schematics, as example.

Encoder

One of the purposes of this module is to let to integrate encoder and motor driver into single PCB to minimize overall size of stabilizer system. For this target, magnetic encoder will be a best choice.

List of supported encoders with its props and cons:

Model	*Interface & Setup*	*Props*	*Cons*
AS5048A	SPI, internal	small size, high resolution, high update rate, perfect integration	expensive
AS5048B	I2C	small size, high resolution, high update rate, perfect integration	Expensive; requires 48pin or 64pin MCU
AMT203	SPI, external	shaft with pass-through hole is possible	Big size; no integration; medium update rate; expensive
MA3 (10bit, 12bit)	PWM, external		no integration; slow update rate; expensive
Analog	Internal or external	Low cost, good integration, high update rate	360 degree is not supported
AS5600	I2C, internal	Low cost, perfect integration, high update rate	Requires 48pin or 64pin MCU
AS5050A	SPI, internal	Low cost, perfect integration	Medium update rate; low resolution
AS5055A	SPI, internal	Low cost, perfect integration	Medium update rate
AS5045	PWM, internal	Perfect integration	Slow update rate

Notes:

1. Update rate is not crucial parameter for general application of the SimpleBGC system, so all types of supported encoders suit.
2. With magnetic or analog encoders, special effort is required when designing system that should support infinite rotation of axis, because they do not allow to have pass-through hole.

More information about using and configuring encoders in the SimpleBGC system you can find in this document:
http://www.basecamelectronics.com/files/SimpleBGC_32bit_Encoders.pdf

Notes on schematics

We provide a reference schematics, which contains a circuits for all types of supported encoders, several examples of output motor driver circuits and several packages of MCU (you can chose from LQFP32, UFQFPN32, LQFP48, UFQFPN48, UFBGA64 case, 32k or more FLASH).

There are 2 options for flashing MCU: via SWD port and ST-Link utility, or via UART port and integrated bootloader. For second option, additional components are required, as shown on the schematics.

For I2C encoders, 32-pin MCU case is not applicable. Use 48- or 64-pin case.

Flashing MCU

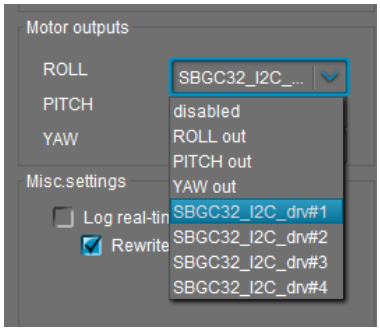
To upload firmware, you can use UART port and **Flash Loader Demonstrator** tool from ST company, or any other flashing tool that can communicate with standard STM32 bootloader, including our GUI "Upgrade" tab in the manual mode. Second option is SWD port and **ST-Link** tool and utility (can be bought separately or found as a part of some "STM32Discovery" boards). Both interfaces are shown in the schematics, choose one that you like.

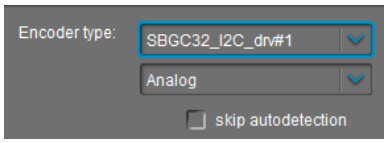
Configuring SimpleBGC 32bit controller to work with the expansion modules

1. Open SimpleBGC GUI , go to "Advanced" tab, "Motor outputs" group. Chose "SBGC32_I2C_drv#1..4" module in the drop-down list for any axis you want. Number 1..4 is configured by setting ADDR0, ADDR1 jumpers on the module: solder jumper to connect address pin to VDD for high level, or leave it floating to set low level (pin is pulled-down internally).

Role	*ADDR0*	*ADDR1*
SBGC32_I2C_drv#1	0 (low)	0 (low)
SBGC32_I2C_drv#2	1 (high)	0 (low)
SBGC32_I2C_drv#3	0 (low)	1 (high)
SBGC32_I2C_drv#4	1 (high)	1 (high)

2. In the "Encoders" tab, chose the same module for corresponding axis. Select a type of encoder, installed on the module, in the drop-down list below.





3. Its better to increase I2C speed by selecting "I2C high speed" in the "Advanced" tab, to lower delays caused by I2C data transfer. But it may be required to decrease overall resistance of pullups on SDA, SCL lines to 1..2k, otherwise I2C errors may come. Please pay attention to this fact when designing your own system.

4. Second (frame) IMU is not required in this setup and may be omitted.

All other settings for motor output and encoder are remain the same as in regular system. You

can find tuning instructions in the user manual and encoder-enabled firmware reference, available to download from our web-site.

Licensing

Binary firmware and schematics for SBGC32_I2C_Drv module are provided free of charge, limited to use only as a part of the SimpleBGC 32bit controller-based system. Source code is a property of Basecamelectronics and is not licensed.

Q&A

Q: How does it identify the motor pitch and the motor roll?

A: There are 2 ADDR pins to chose I2C slave device address from four options. Set any unique address for each module, that defines them as “module 1..4”, than choose corresponding module in the GUI. More information can be found in the section "Configuring SimpleBGC 32bit controller to work with the expansion modules".

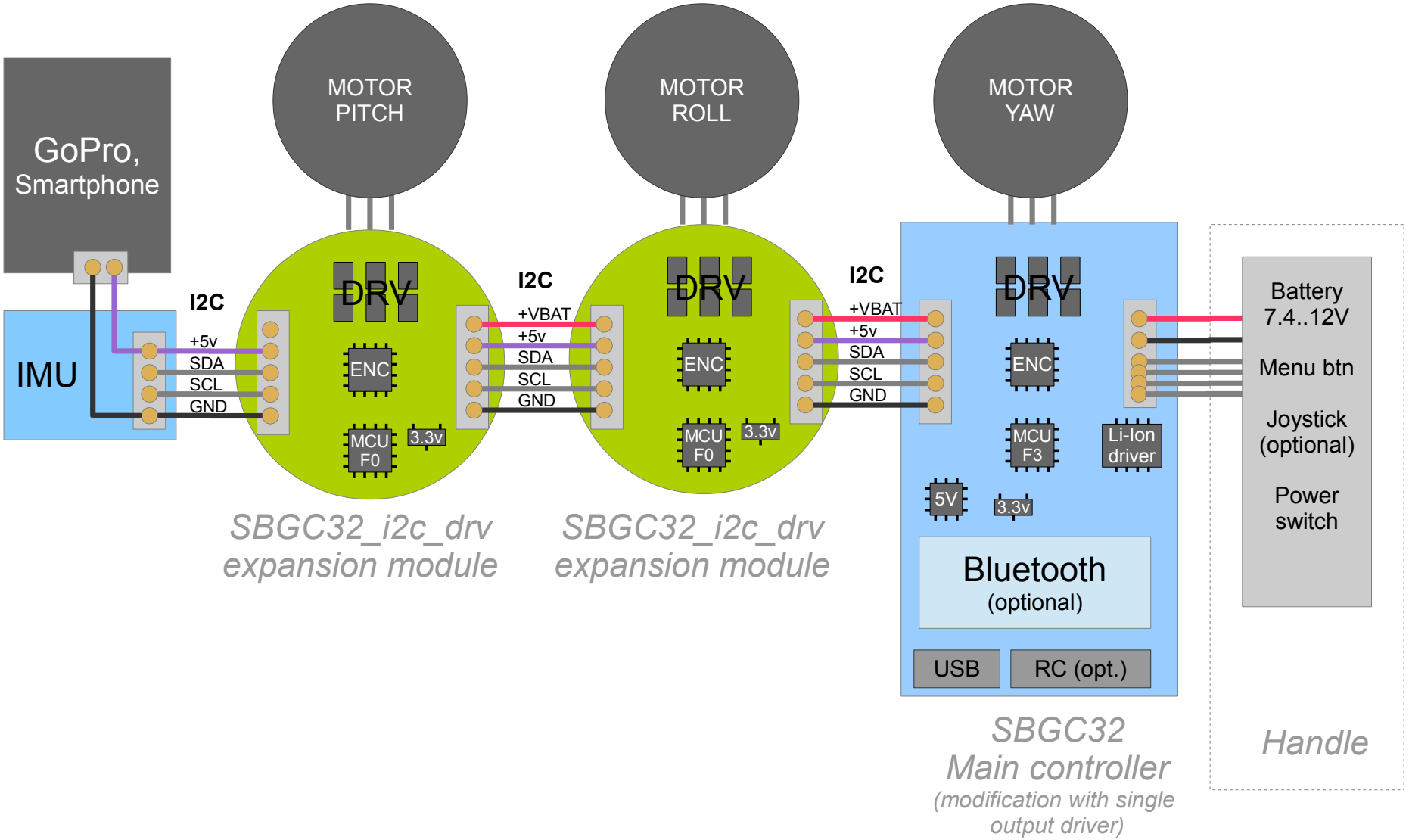
Q: I cannot find the circuit diagram of the main controller. What can I refer to in order to find it?

A: Main controller is the regular 32bit board with removed 1, 2 or 3 drivers. All other service electronics like LiPo charger, bluetooth module, are shown in the connection diagram just for example and does not relate to stabilization function. All functions and interfaces of the main controller remains available as before, and you can still use main controller to drive any motor, assigning direct motor outputs in the GUI.

Q: Is it possible to apply DRV8313 instead of the original motor driver, shown in the schematics? If it is not possible, how can we get increased motor power?

A: Yes, you can apply DRV8313 or any other motor driver that supports 3 PWM + ENABLE inputs.

Appendix A: SBGC32_I2C_Drv connection diagram



Appendix B: I2C_Drv API and examples

The API consists of the definition of I2C registers. The master controller reads and writes them to operate the module.

I2C_Drv.h

```
/*
 This is a part of SimpleBGC project source code
 Copyright (c) 2015 Aleksei Moskalenko

 SBGC32_i2c_drv -  expansion board with brushless driver, encoder and I2C interface
*/
#ifndef I2C_DRV_H_
#define I2C_DRV_H_


// Device address (7bit)
#define I2C_DRV_START_ADDR  0x19

// Device identifier. Check it when discovering
#define I2C_DRV_DEVICE_ID 0x14



/************** Register map ************************/
/* Note that though registers are 16bit,  lower byte can be read/written by 8bit
transactions */
// Encoder
#define I2C_DRV_REG_ENC_RAW_ANGLE       40
#define I2C_DRV_REG_ENC_ANGLE           41
#define I2C_DRV_REG_ENC_INFO            6
#define I2C_DRV_REG_ENC_ERR_CNTR        42

// Motor driver
#define I2C_DRV_REG_SET_POWER_ANGLE     0
#define I2C_DRV_REG_SET_POWER           0
#define I2C_DRV_REG_SET_ANGLE           1
#define I2C_DRV_REG_SET_FORCE_POWER     2
#define I2C_DRV_REG_SET_ENABLE          3

// Configuration
#define I2C_DRV_REG_ENC_TYPE            4
#define I2C_DRV_REG_ENC_CONF            5
#define I2C_DRV_REG_ENC_FLD_OFFSET      7

// Device info
#define I2C_DRV_REG_DEVICE_ID           39
#define I2C_DRV_REG_FIRMWARE_VER        32
#define I2C_DRV_REG_MCU_ID              33
#define I2C_DRV_MCU_ID_SIZE             6
#define I2C_DRV_REG_I2CS_ERR_CNTR       43

// Misc. functions
#define I2C_DRV_REG_RESET_MODE          8
```

```
/*****************************************************/


// Types of supported encoders
#define I2C_DRV_ENC_TYPE_AS5048A 1
#define I2C_DRV_ENC_TYPE_AS5048B 2
#define I2C_DRV_ENC_TYPE_AMT203 3
#define I2C_DRV_ENC_TYPE_MA3_10BIT 4
#define I2C_DRV_ENC_TYPE_MA3_12BIT 5
#define I2C_DRV_ENC_TYPE_ANALOG 6
#define I2C_DRV_ENC_TYPE_AS5600 7
#define I2C_DRV_ENC_TYPE_AS5050A 8
#define I2C_DRV_ENC_TYPE_AS5055A 9


#endif /* I2C_DRV_H_ */
```

Example of using I2C_Drv

```
uint8_t OutputI2CDrv::init(uint8_t _out_port, uint8_t _axis)
{
    addr = (I2C_DRV_START_ADDR - 3) + _out_port;
    cur_power = 0;

    // Check if I2C board is connected (wait 2 sec)
    i2c::select_line(I2C_LINE_A);
    for(uint8_t i=0; i<100; i++) {
        if(i2c::read_reg_byte(addr, I2C_DRV_REG_DEVICE_ID) == I2C_DRV_DEVICE_ID) { //
check device ID
            return 1;
        }
    }

    return 0;
}

void OutputI2CDrv::write_reg_byte(uint8_t reg, uint8_t val) {
    i2c::select_line(I2C_LINE_A);
    i2c::write_reg_byte(addr, reg, val);
}

void OutputI2CDrv::powerOn() {
    write_reg_byte(I2C_DRV_REG_SET_ENABLE, 1);
}

void OutputI2CDrv::powerOff() {
    write_reg_byte(I2C_DRV_REG_SET_ENABLE, 0);
}


void OutputI2CDrv::output16(uint16_t el_angle) {
    i2c::select_line(I2C_LINE_A);
    uint16_t buf[2] = { cur_power, el_angle };
    i2c::write_reg_buf(addr, I2C_DRV_REG_SET_POWER_ANGLE, (void*)buf, sizeof(buf));

    // TODO: We can write force power, if encoder fld.offset is calibrated
    // (not yet implemented in the I2C_DRV firmware)
```

```
}


// Search for the I2C_Drv-connected encoder
void Encoder::_init_i2c_drv(uint8_t type) {
    i2c.addr = (I2C_DRV_START_ADDR - ENC_TYPE_I2C_DRV1) + type;

    i2c::select_line(I2C_LINE_A);


    if(i2c::read_reg_byte(i2c.addr, I2C_DRV_REG_DEVICE_ID)  == I2C_DRV_DEVICE_ID) {
        // Get firmware version, if required
        //uint16_t frw_ver;
        //i2c::read_reg_buf(addr, I2C_DRV_REG_FIRMWARE_VER, &frw_ver, 2);

        // Configure the type of internal encoder
        if(i2c::write_reg_byte(i2c.addr, I2C_DRV_REG_ENC_TYPE, cfg)) {
            // Read it back to confirm activation. Wait max. 50ms
            for(uint8_t i=0; i<50 && !encoder_type; i++) {
                Time::delay_us(1000);
                if(i2c::read_reg_byte(i2c.addr, I2C_DRV_REG_ENC_TYPE) == cfg) {
                    encoder_type = type;
                    // configure field calibration
                    //i2c::write_reg_buf(addr, I2C_DRV_REG_ENC_FLD_OFFSET,
&(params.encoder_fld_offset[axis]), 2);

                    // Configure encoder
                    //      bits 0..3: LPF factor
                    //      bit 4: I2C fast mode
                    uint16_t conf = 5;
                    i2c::write_reg_buf(i2c.addr, I2C_DRV_REG_ENC_CONF, &conf,
sizeof(conf));
                }
            }
        }

    }
    i2c::errors_count = 0;
}

uint16_t Encoder::_read_i2c_drv() {
    uint16_t angle;

    i2c::select_line(I2C_LINE_A);
    if (i2c::read_reg_buf(i2c.addr, I2C_DRV_REG_ENC_ANGLE, &angle, 2)) {
        read_error = 0;

        return angle;
    }

    return 0;
}



void Encoder::_request_info_i2c_drv(uint8_t info[4]) {
    // info[0] - encoder error counter
    // info[1] - I2C slave error counter
    // Sub-type AS5048A, AS5048B:
```

```
    //   info[2] - magnitude
    //   info[3] - diagnostic register

    i2c::select_line(I2C_LINE_A);
    info[0] = i2c::read_reg_byte(i2c.addr, I2C_DRV_REG_ENC_ERR_CNTR);
    info[1] = i2c::read_reg_byte(i2c.addr, I2C_DRV_REG_I2CS_ERR_CNTR);

    // Write any data to ENC_INFO reg to queue information from encoder
    if(i2c::write_reg_byte(i2c.addr, I2C_DRV_REG_ENC_INFO, 0)) {
        Time::delay_ms(50); // wait a bit to let device to read information
        i2c::read_reg_buf(i2c.addr, I2C_DRV_REG_ENC_INFO, &info[2], 2);
    }
}
```