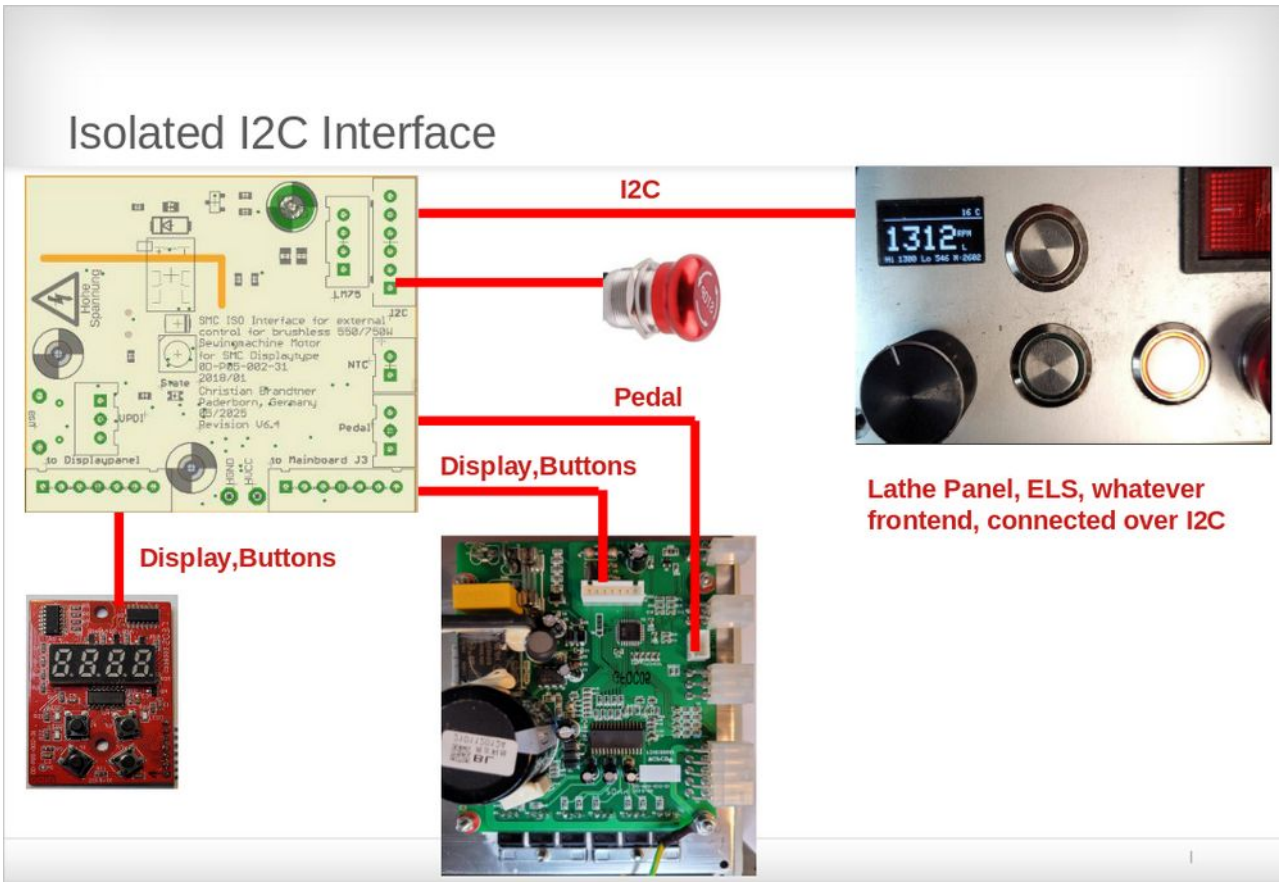
Brushless BLDC Motor isolated I2C interface

Version 1.0 May 2025

Features:

- electrically isolated from the mains voltage on SMC controllerboard
- no changes on Controllerboard or Panelboard needed
- can be installed in the housing on existing attachments
- I2C speedcontrol
- I2C start/stop control
- I2C rotation direction control
- I2C temperature control
- complete configuration of the SMC controller via I2C
- securityswitch input for emergency stop button
- opt. NTC temperature control

Using for this brushless servo motor 550/750 Watt
Displayboard 0D-P05-002-31 2018/01
Controllerboard 0D-P05-010-01 2020/11



GENERAL DESCRIPTION

The ISO interface is used to control the brushless servo motor controller (SMC) via an I2C interface to change the motor speed and direction.
To do this, the ISO interface simulates a display panel with four buttons, a four-segment LED, and three LEDs towards the SMC.
The display data and LED states are decoded from the seven-segment information, and the four buttons are simulated. The existing SMC panel can continue to be used with all its functions.
The decoded data is passed to an external controller via the I2C interface, and conversely, the external controller can simulate the four buttons to control the controller.

THEORY OF OPERATION

When the device is connected to the I2C bus line, the device is working as a slave device. The Master (MCU) can write/read the ISO Interface input register using the I2C interface command.
The ISO Interface device address contains 7 bits I2C Adress 0x36 = x011 0110

The following sections describe the communication protocol to send or read the data code using the I2C interface.

An important feature is the native keypad control of the SMC control panel. This allows all operating functions to be implemented natively with the connected controller via I2C and is based on the manual operation of the SMC control panel.

Write Commands

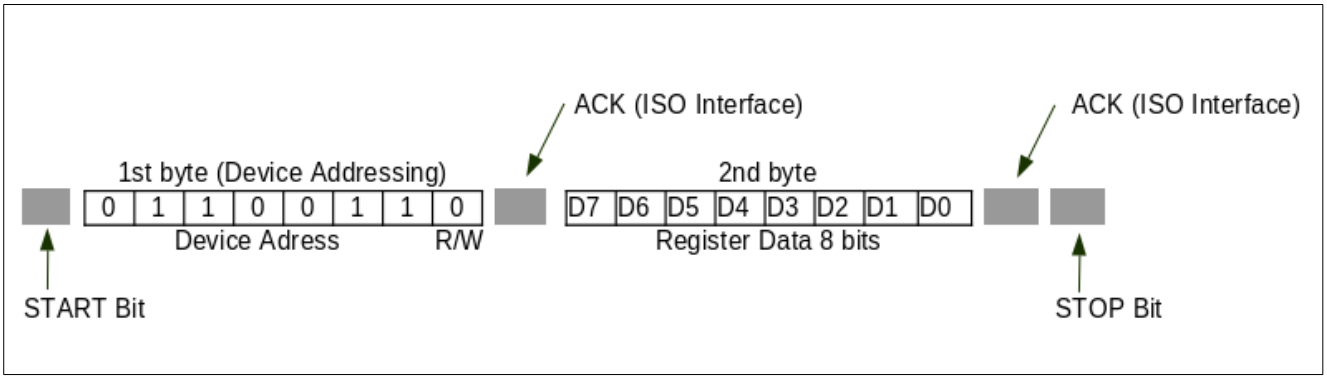
The write commands are used to:
1. simultate the Panelbuttons to the SMC
2. set the desired DAC value for speedcontrol
3. release the DAC to start the motor
4. set smooth motor stop
5. request the displaydata (raw data)
6. reset the ISO Interface
7. send Alive command

Table 1-1 shows the write command types and their functions.

Table 1-1 Write command type

Command Types	Command name	Function
0	Reserved	Reserved for future use
1	Alive	Alive check
2	Panel Reset	clear Panel Data
3	Reserved	Reserved for future use
4	Button P	press Button P
5	Button S	press Button S
6	Button up	press Button up
7	Button Down	press Button down
8	Button P & Button S	press Button P & S
9	Button CDR	longpress Button S to change rotation direction
10	GetDisplay	SMC Displaydata request
11	DAC value	set rotational speed e.g.DAC Value
12	SPI Resync	Restart of SMC Display detection
13	DAC Stop	Motor stop smooth
14	DAC Release	set DAC in workmode, ready to accept DAC value
15	Reserved	Reserved for future use

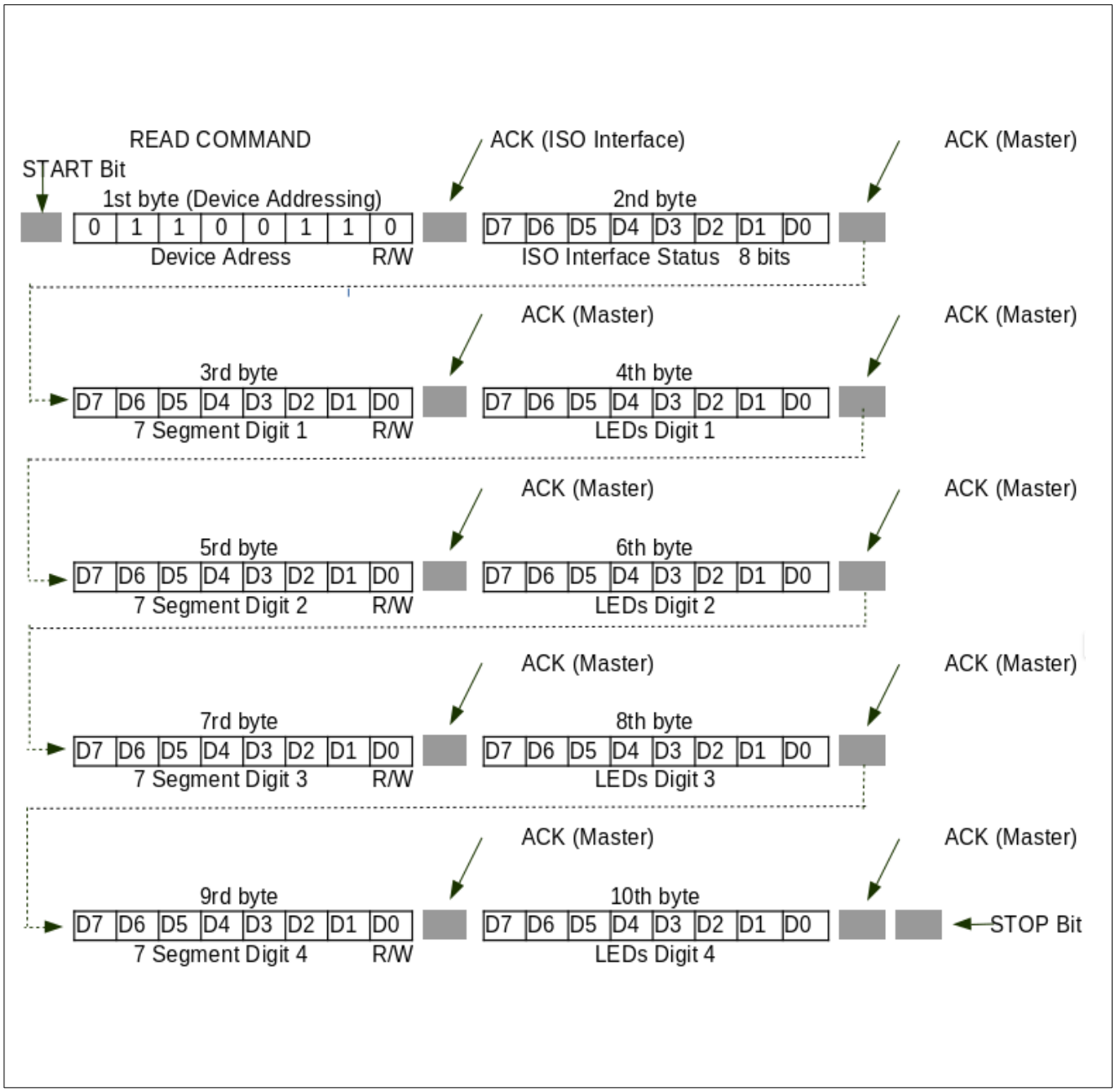
Write ISO Interface Register



Read ISO Interface Register

I2C Register

I2C_REG[0] = ISO Interface Status
I2C_REG[1] = 7 Segment Digit 1
I2C_REG[2] = 3 LEDs Digit 1
I2C_REG[3] = 7 Segment Digit 2
I2C_REG[4] = 3 LEDs Digit 2 , depreciated
I2C_REG[5] = 7 Segment Digit 3
I2C_REG[6] = 3 LEDs Digit 3 , depreciated
I2C_REG[7] = 7 Segment Digit 4



I2C_REG[8] = 3 LEDs Digit 4 , depreciated

Register Definitions

ISO Interface Status **uint8_t**

```
#define PANEL_STATUS_MOTOR_RUN          0
#define PANEL_STATUS_ENABLED_RELAIS     1
#define PANEL_STATUS_SMC_SYNC_ERROR     2
#define PANEL_STATUS_SMC_BUTTON_ERROR   3
#define PANEL_STATUS_I2C_ERROR          4
#define PANEL_STATUS_SPI_ERROR          5
#define PANEL_STATUS_WATCHDOG_ERROR     6
#define PANEL_STATUS_DAC_STOP           7
```

---

7 Segment Digit **uint8_t**

```
          gafbcPde        dez          aktiv Low
#define b10000100     /* zero  132     */
#define b11100111     /* one   231     */
#define b00101100     /* Two   44      */
#define b00100101     /* Three 37      */
#define b01000111     /* Four  71      */
#define b00010101     /* Five  21      */
#define b00010100     /* Six   20      */
#define b10100111     /* Seven 167     */
#define b00000100     /* Eight 4       */
#define b00000101     /* Nine  5       */
#define b11111011     /* DP    251     */

//      gafbcPde, animated 0 anmiert

#define b10000101     /*        133, 85        */
#define b10000110     /*        134, 86        */
#define b10001100     /*        140, 8C        */
#define b10010100     /*        184, 94        */
#define b11000100     /*        196, C4        */
#define b10100100     /*        164, A4        */
```

3 Leds and Digits **uint8_t**

```
// Segments and LED's

#define Bit_LED_NeedleUp            0       // Led1    aktiv High
#define Bit_LED_NeedleDown          1       // Led2    aktiv High
#define Bit_LED_Lamp                2       // Led3    aktiv High

#define Bit_Disp_Segment1           3       // 8       aktiv Low
#define Bit_Disp_Segment4           4       // 16      aktiv Low
#define Bit_Disp_Segment3           5       // 32      aktiv Low
#define Bit_Disp_Segment2           6       // 64      aktiv Low
```

Structures

```
// 7 Segment Digit, 4 Digit select, 3 Leds

typedef struct {
        uint8_t Digit;                  // 7 Segment
        uint8_t Segment_Led;            // Selected Display 1-4, 3 Leds
} PANEL_DISPLAY_MSGQUEUE_OBJ_t;

typedef struct{
        PANEL_DISPLAY_MSGQUEUE_OBJ_t SMCDigit1;
        PANEL_DISPLAY_MSGQUEUE_OBJ_t SMCDigit2;
        PANEL_DISPLAY_MSGQUEUE_OBJ_t SMCDigit3;
        PANEL_DISPLAY_MSGQUEUE_OBJ_t SMCDigit4;
}DISPLAY_DATA_OBJ_t;

typedef struct{
        uint8_t PanelStatusFlags;
}PANEL_STUFF_OBJ_t;

typedef struct{
        DISPLAY_DATA_OBJ_t      DisplayData;
        PANEL_STUFF_OBJ_t       StuffData;
}I2C_MESSAGE_OBJ_t;
```