Product Document

Published by
ams OSRAM Group

austriamicrosystems AG

is now

ams AG

The technical content of this austriamicrosystems application note is still valid.

Contact information:

Headquarters:
ams AG
Tobelbaderstrasse 30
8141 Unterpremstaetten, Austria
Tel: +43 (0) 3136 500 0
e-Mail: ams_sales@ams.com

Please visit our website at **www.ams.com**

AN5115-40 EASYZAPP Programming MAGNETIC ROTARY ENCODER OTP Programming Guide

APPLICATION NOTE

This application note describes the available options for EASYZAPP programming of AS5115 and AS5215 (dual die version).

Starting with the general permanent programming of the OTP memory register, it also describes the "soft writing" for single or multiple non-permanent writing of the OTP and the features for verification after programming.

For an overall description of the device, please refer to the relevant datasheet.

1 Hardware Connections for OTP Memory Programming

For OTP memory access, 3 signals are required: DIO, CS and CLK; for programming in addition PROG. The related programming voltage Vprog (switch position 3) is always between 8V....8.5V. It has to be buffered by a fast 100nF and a 10uF capacitor, which should be placed as close as possible to the PROG pin.

For the AS5215, below shown signal lines must be carried out twice (3 lines per device).

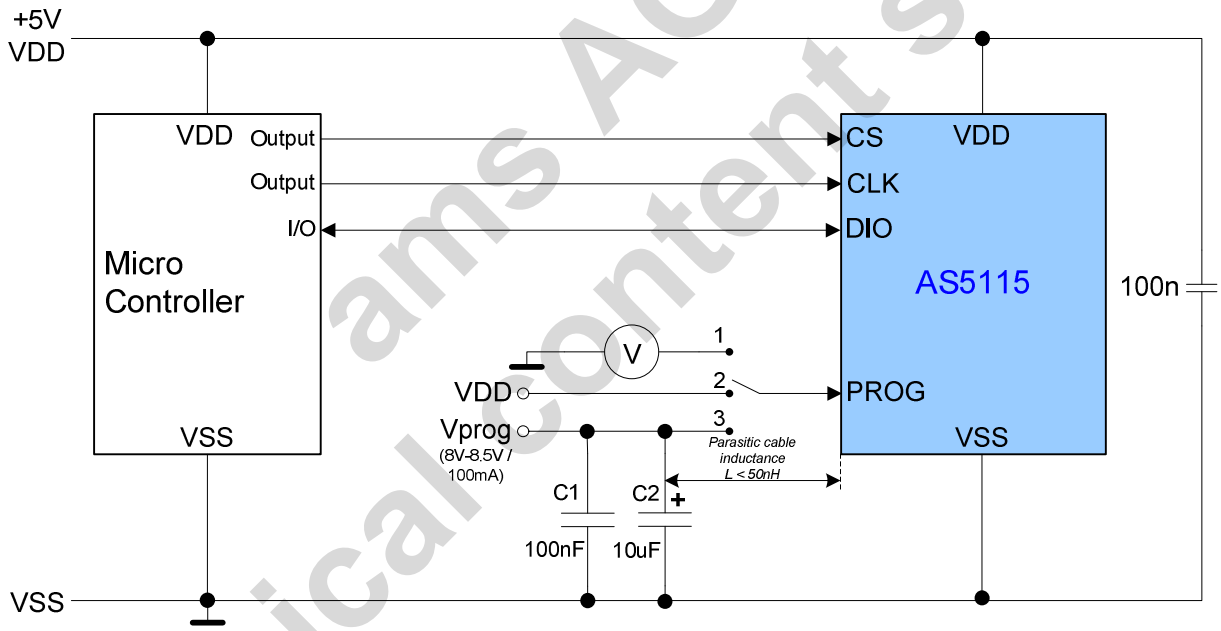

Figure 1: Programming circuit of AS5xxx

Note: During normal operation the PROG pin should be supplied with 5V (**Figure 1, switch position 2**)!

2 One Time Programmable Register (OTP)

The OTP block should add flexibility to the user. It allows defining user parameter like a new zero position, sensitivity and several other output modes (depending on the product). It is not recommended to change the factory settings of the device.

By default the periphery pins CS, CLK, PDIO are used for the SSI interface. The device starts in *Normal Operating Mode* – one command-/data-bit needs one clock cycle – as shown in Figure 2.

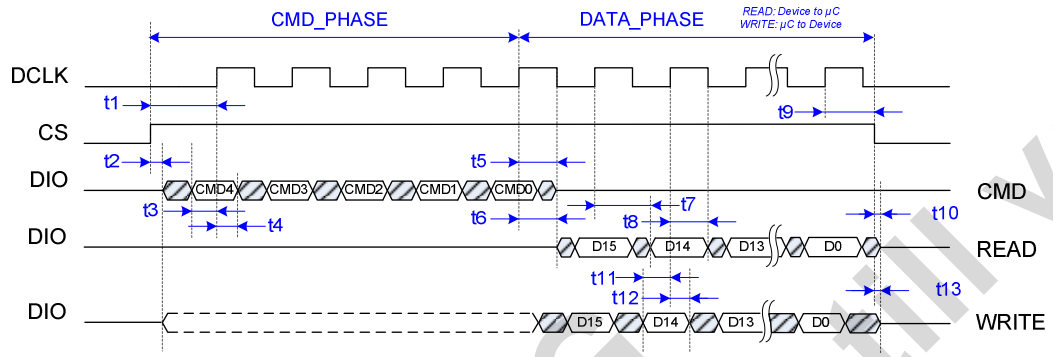

Figure 2: Normal Operating Mode

Access to the OTP block is granted by sending a special command (EN_PROG) over the SSI Interface to the OTP block. This command sets the device in *Extended Operating Mode*, where the device is able to handle signals for special requirements. In this mode, the digital interface needs four clock cycles for one data-bit (see Figure 3).

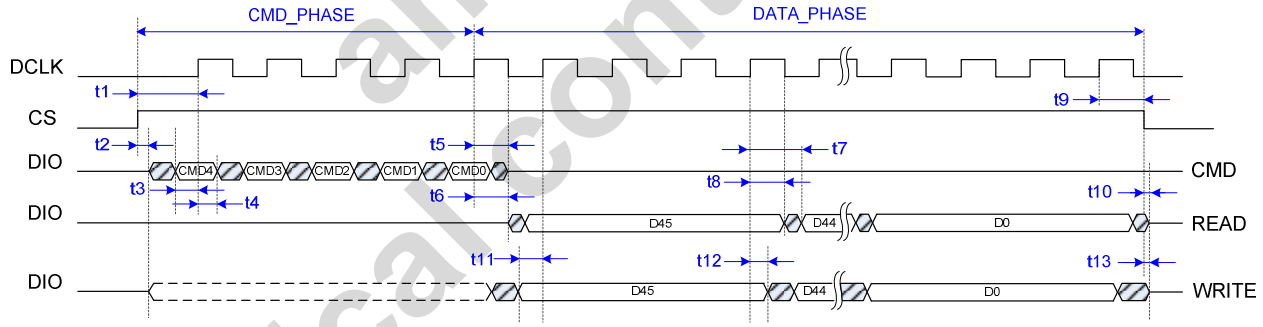

Figure 3: Extended Operating Mode

Each command (READ, WRITE, EN_PROG, etc.) consists of 5 bits, whereas the amount of data bits is depending on the operating mode. The OTP buffer structure of both devices is shown in chapter 4.

Note: Please be aware that there exist two sets of commands, one for *Normal Operating Mode* and one for *Extended Operating Mode*.

2.1 Commands

Below table shows an overview of all commands of AS5115 and AS5215.

Mode	Command	bin	Protocol
Normal Mode	WRITE_CONFIG	10111	5 command bits + 16 data bits 1 CLK per data bit
	EN_PROG	10000	5 command bits + data bits: 1000110010101110 1 CLK per data bit
Extended Mode	WRITE_OTP	11111	5 command bits + 46 data bits 4 CLK per data bit
	READ_OTP	01111	
	PROG_OTP	11001	
	READ_OTP_ANA	01001	

Table 1: SSI Commands

For detailed information concerning buffer structure and data bit sequence, please refer to chapter 4.

2.2 Timings

The following table contains all timings related to Figure 2 and Figure 3.

Symbol	Parameter	Min	Max	Unit
t1	Chip select to positive edge of CLK	30	-	ns
t2	Chip select to drive bus externally	0	-	ns
t3	Setup time command bit Data valid to positive edge of CLK	30	-	ns
t4	Hold time command bit Data valid after positive edge of CLK	15	-	ns
t5	Float time Positive edge of CLK for last command bit to bus float	-	CLK/2+0	ns
t6	Bus driving time Positive edge of CLK for last command bit to bus drive	CLK/2 +0	-	ns
t7	Setup time data bit Data valid to positive edge of CLK	CLK/2 +0	CLK/2 +30	ns
t8	Hold time data bit Data valid after positive edge of CLK	CLK/2 +0	-	ns
t9	Hold time chip select Positive edge CLK to negative edge of chip select	CLK/2 +0	-	ns
t10	Bus floating time Negative edge of chip select to float bus	-	30	ns
t11	Hold time data bit @ write access Data valid to positive edge of CLK	30	-	ns
t12	Hold time data bit @ write access Data valid after positive edge of CLK	15	-	ns
t13	Bus floating time Negative edge of chip select to float bus	-	30	ns

Table 2: Timings

2.3 Programming Sequence

This chapter shows a complete programming sequence, including a verification of all settings with analog-readback.

After power up, the device is by default in Normal Operating Mode.

For programming the device, please proceed accordingly:

- Set device into Extended Mode (EN_PROG - "10000") → now take commands for *Extended Operating Mode*
- "Soft-Write" settings into OTP (WRITE_OTP – "10111") for checking impact of new parameters
- Read back new settings (READ_OTP – "01111") → verifies the "soft-write" values
- Ensure that Vprog (8V-8.5V) is applied to PROG pin (Figure 1, switch position 3)
- Perform permanent writing (PROG_OTP – "11001")
- Verify new settings by analog reading the fuse values
- Redirect PROG pin signal to voltmeter (Figure 1, switch position 1)
- Send READ_ANA – "01001" command → PROG pin becomes an output
- Four clock cycles represent the readback (analog voltage) of one fuse (pattern looks like Figure 3), starting with the MSB.
  A voltage of < 500mV indicates a correctly programmed bit ("1") while a voltage level from 2.2V to 3.5V indicates an unprogrammed bit ("0"). Any voltage level in between refer to incorrect programming!

3 Operating Conditions

Following table shows the most important programming parameters and its conditions.

PROG Operation

PARAMETER	SYMBOL	MIN	MAX	UNIT	NOTE
Positive Supply Voltage	VDD	4.5V	5.5V	V	
Negative Supply Voltage	VSS	0V	0V	V	Ground =0V
Programming Voltage	Vprog	8.0	8.5	V	
Programming Current into PROG	IPROG,prog		100	mA	required current to program a single element
Programming ambient temperature	Tamb,prog	0	85	°C	During programming
Programming time	tprog	2	4	us	Timing is internally generated
Analog readback voltage	Vp,prog		0.5	V	During analog readback at PROG pin
	Vp, unprog	2.2	3.5	V	
The PolyFuse cell can be programmed only once					

Table 3: Operating Conditions

4 Example source code

The following source code is taken from the AS5000-Programmer firmware. The microcontroller is a SiLabs C8051F342. It contains the communication over the 3-wire interface and the PROG pin.

Five commands are needed for programming:

- twiRead
- twiWrite
- twiReadExtended
- twiWriteExtended
- twiReadExtendedAnalog

Buffer structure for AS5115/AS5215 (normal mode):

Number of bits	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Bit position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit	go2sleep	gen_rst					analog_sig	OB_bypassed								

Buffer structure for AS5115/AS5215 (extended mode):

Number of bits	2	18	3	3	2	4	1	1	1	1	1	2	1	6
Bit position	45..44	43..26	25..23	22..20	19..18	17..14	13	12	11	10	9	8..7	6	5..0
Bit	fs	ID	fs	fs	fs	fs	fs	fs	Invert_chanel	cm_sin	cm_cos	gain	dc.offset	Hall_bias

Note: All bits highlighted in blue are factory settings (fs) and must not be overwritten!

```
#define CLEAR_CS()      do { TWI_CS = 0; } while(0)
#define SET_CS()        do { TWI_CS = 1; } while(0)
#define CLEAR_CLK()     do { TWI_CLK = 0; } while(0)
#define SET_CLK()       do { TWI_CLK = 1; } while(0)

#define DIO_HIGH_IMP()  do { TWI_DIO = 1; } while(0)
#define SET_DIO()       do { TWI_DIO = 1; } while(0)
#define CLEAR_DIO()     do { TWI_DIO = 0; } while(0)
#define VAL_DIO         TWI_DIO


/**
 * These macros are for dealing witch the PROG line.
 */

#define PROG_HIGH_IMPED() VZAP_SWITCH_STATE = 0
#define PROG_LOW_IMPED()  VZAP_SWITCH_STATE = 1

// Timings in microseconds.
// T = 4 us -> 250 kHz. That frequency is usable for all operations //
(RD/WRITE/PROG) except ANALOG READ
#define T_TWI_CLK_2     2
#define T_TWI_CLK_4     1

// These are the timings for the analog readback in milliseconds.
// 25 kHz is chosen here. As long we operate in three wire mode, the
frequency has no lower limit.
#define T_TWI_ARB_2     20
#define T_TWI_ARB_4     10

/**
 * Sets certain pins to certain states.
 * Set VZAP_SWITCH_STATE to 1 after a call to this function in order to
complete the TWI initialization.
 */
void initTWI()
{
  //TODO: #define this
```

```
  TWI_C2 = 0;     // 3-wire mode

  TWI_DX = 1;     // High Impedance
  DIO_HIGH_IMP();
  CLEAR_CS();
  CLEAR_CLK();

static void clkPulses(unsigned char num_pulses)
{
  do
  {
    delay_us(T_TWI_CLK_2);  //twiDelay(400);
    SET_CLK();
    delay_us(T_TWI_CLK_2);  //twiDelay(400);
    CLEAR_CLK();
  } while (--num_pulses);
}

static void clkPulsesAnalog(unsigned char num_pulses)
{
  do
  {
    delay_us(T_TWI_ARB_2);
    SET_CLK();
    delay_us(T_TWI_ARB_2);
    CLEAR_CLK();
  } while (--num_pulses);
}

/**
 * command phase.
 */
static void write(unsigned char* buffer, unsigned char num_bits)
{
  xdata unsigned char current_bit;

  for (current_bit = num_bits; current_bit; current_bit--)
  {
```

```
    unsigned char this_bit = ((buffer[(current_bit-1) >> 3]) >>
((current_bit-1) & 0x07)) & 0x01;

    if (this_bit)
          SET_DIO();
    else
      CLEAR_DIO();

    if (current_bit == 1)
    {
      // Last Bit has been written:
      delay_us(T_TWI_CLK_2); //twiDelay(400);
      SET_CLK();
      delay_us(T_TWI_CLK_4); //twiDelay(400);
      DIO_HIGH_IMP();
      delay_us(T_TWI_CLK_4); //twiDelay(400);
      CLEAR_CLK();
    }
    else
    {
      clkPulses(1);
    }
  }
}

static void writeExtended(unsigned char* buffer, unsigned char num_bits)
{
  xdata unsigned char current_bit;

  //-- write Data --
  for (current_bit = num_bits; current_bit; current_bit--)
  {
    unsigned char this_bit = ((buffer[(current_bit-1) >> 3]) >>
((current_bit-1)&0x07)) & 0x01;

    if(this_bit)
          SET_DIO();
    else
          CLEAR_DIO();
```

```
    clkPulses(4);
  }
}

/**
 * WORKS.
 */
static void read(unsigned char* buffer, unsigned char num_bits)
{
  xdata unsigned char current_bit;
  xdata unsigned char temp;

  //-- read SSI Data --
  temp = 0;
  for (current_bit = num_bits; current_bit; current_bit--)
  {
    temp <<= 1;

    delay_us(T_TWI_CLK_2); // t7     //twiDelay(400);

      temp += (VAL_DIO) ? 1 : 0;

    SET_CLK();
    delay_us(T_TWI_CLK_2);
    CLEAR_CLK();

    // Save the bits to the buffer
    if (((current_bit - 1) & 0x07) == 0)
    {
      buffer[current_bit >> 3] = temp;    // Normally it should be
[(current_bit-1) >> 3], but it has the same results.
      temp = 0;
    }
  }
}

static void readExtended(unsigned char* buffer, unsigned char num_bits)
```

```
{
  xdata unsigned char current_bit;
  xdata unsigned char temp;

  if(!num_bits) return;


  //-- read SSI Data --
  temp = 0;
  for(current_bit = num_bits; current_bit; current_bit--)
  {
    temp <<= 1;

       clkPulses(2);

    temp += (VAL_DIO) ? 1 : 0;

    delay_us(T_TWI_CLK_2); //twiDelay(400);
    SET_CLK();
    delay_us(T_TWI_CLK_2); //twiDelay(400);
    CLEAR_CLK();

    // This is the 4th clock pulse:
       clkPulses(1);

    // Save the bits read so far.
    if(((current_bit - 1) & 0x07) == 0)
    {
      buffer[(current_bit-1) >> 3] = temp;   // Normally it should be
[(current_bit-1) >> 3], but current_bit >> 3 has the same results.
      temp = 0;
    }
  }
}

static void readAnalog(unsigned char* buffer, unsigned char num_bits)
{
  xdata unsigned char current_bit = 0;
```

```
  if (!num_bits) return;

  //twiDelay(400);

  //-- read SSI Data --
  for (current_bit = num_bits; current_bit; current_bit--)
  {
    clkPulsesAnalog(3);
    delay_us(T_TWI_ARB_2);  //twiDelay(400);

       buffer[current_bit-1] = adcSample(0) >> 2; // Read analog voltage
from PROG

    SET_CLK();
    delay_us(T_TWI_ARB_2);  //twiDelay(1500);
    CLEAR_CLK();
  }
}

/*
 * Interface Functions are following.
 */

// cmd_byte[(num_cmd_bits - 1)/8].((num_cmd_bits-1)%8) is written first
// bit ordering: buffer[0].0 equ bit  0 (read last)
//               buffer[2].3 equ bit 19 (read earlier)
void  twiRead(unsigned  char  *cmd_buffer,  unsigned  char  num_cmd_bits,
unsigned char *data_buffer, unsigned char num_data_bits)
{
  SET_CS();
  delay_us(T_TWI_CLK_2);  //twiDelay(400);
  write(cmd_buffer, num_cmd_bits);

  DIO_HIGH_IMP(); // Already done be write, but done again here in case of
num_cmd_bits = 0
  read(data_buffer, num_data_bits);

  delay_us(T_TWI_CLK_2);  //twiDelay(400);
```

```
  CLEAR_CS();
}


// cmd_byte[(num_cmd_bits - 1)/8].((num_cmd_bits-1)%8) is written first
// bit ordering: buffer[0].0 equ bit  0 (read last)
//               buffer[2].3 equ bit 19 (read earlier)
void    twiReadExtended(unsigned    char    *cmd_buffer,    unsigned    char
num_cmd_bits, unsigned char *data_buffer, unsigned char num_data_bits)
{
  SET_CS();
  delay_us(T_TWI_CLK_2);  //twiDelay(400);
  write(cmd_buffer, num_cmd_bits);

  DIO_HIGH_IMP();
  readExtended(data_buffer, num_data_bits);

  delay_us(T_TWI_CLK_2);  //twiDelay(400);
  CLEAR_CS();
}

// cmd_byte[(num_cmd_bits - 1)/8].((num_cmd_bits-1)%8) is written first
// bit ordering: buffer[0].0 equ bit  0 (written last)
//               buffer[2].3 equ bit 19 (written earlier)
void  twiWrite(unsigned  char  *cmd_buffer,  unsigned  char  num_cmd_bits,
unsigned char *data_buffer, unsigned char num_data_bits)
{
  SET_CS();
  delay_us(T_TWI_CLK_2);  //twiDelay(400);
  write(cmd_buffer, num_cmd_bits);

  write(data_buffer, num_data_bits);

  delay_us(T_TWI_CLK_2);  //twiDelay(400);
  CLEAR_CS();
}


// cmd_byte[(num_cmd_bits - 1)/8].((num_cmd_bits-1)%8) is written first
```

```
// bit ordering: buffer[0].0 equ bit  0 (written last)
//               buffer[2].3 equ bit 19 (written earlier)
void   twiWriteExtended(unsigned   char   *cmd_buffer,   unsigned   char
num_cmd_bits, unsigned char *data_buffer, unsigned char num_data_bits)
{
  SET_CS();
  delay_us(T_TWI_CLK_2);  //twiDelay(400);
  write(cmd_buffer, num_cmd_bits);

  writeExtended(data_buffer, num_data_bits);

  delay_us(T_TWI_CLK_2);  //twiDelay(400);
  CLEAR_CS();
}

// cmd_byte[(num_cmd_bits - 1)/8].((num_cmd_bits-1)%8) is written first
// data_buffer will hold the 8 MSB of the ADC result
void  twiReadExtendedAnalog(unsigned  char  *cmd_buffer,  unsigned  char
num_cmd_bits, unsigned char *data_buffer, unsigned char num_data_bits)
{
  SET_CS();
  delay_us(T_TWI_CLK_2);  //twiDelay(400);
  write(cmd_buffer, num_cmd_bits);

  DIO_HIGH_IMP();
  readAnalog(data_buffer, num_data_bits);

  delay_us(T_TWI_CLK_2);  //twiDelay(400);
  CLEAR_CS();
}
```

Revision History

Revision	Date	Description
1.0	January, 2010	initial revision

Copyrights

Disclaimer

Devices sold by austriamicrosystems AG are covered by the warranty and patent indemnification provisions appearing in its Term of Sale. austriamicrosystems AG makes no warranty, express, statutory, implied, or by description regarding the information set forth herein or regarding the freedom of the described devices from patent infringement. austriamicrosystems AG reserves the right to change specifications and prices at any time and without notice. Therefore, prior to designing this product into a system, it is necessary to check with austriamicrosystems AG for current information. This product is intended for use in normal commercial applications. Applications requiring extended temperature range, unusual environmental requirements, or high reliability applications, such as military, medical life-support or lifesustaining equipment are specifically not recommended without additional processing by austriamicrosystems AG for each application.

The information furnished here by austriamicrosystems AG is believed to be correct and accurate. However, austriamicrosystems AG shall not be liable to recipient or any third party for any damages, including but not limited to personal injury, property damage, loss of profits, loss of use, interruption of business or indirect, special, incidental or consequential damages, of any kind, in connection with or arising out of the furnishing, performance or use of the technical data herein. No obligation or liability to recipient or any third party shall arise or flow out of austriamicrosystems AG rendering of technical or other services.

Contact Information

Headquarters

austriamicrosystems AG

A-8141 Schloss Premstaetten, Austria

Tel: +43 (0) 3136 500 0

Fax: +43 (0) 3136 525 01

For Sales Offices, Distributors and Representatives, please visit:

http://www.austriamicrosystems.com