Technical Notes for SMCI33-2

Content

1 - Serial setup .....
2 - Running without Java script .....
3 - Running with Java script .....
4 - List of commands .....
5 - Examples .....
5.1 - With Java script running .....
5.2 - Without Java script .....
5.3 - Universal commands .....
6 - Java Program .....

Serial setup

Command	Response	Description
#<motor>:baud	<motor>:baud	Sets the baud rate of the controller 12=115200 (default) see 1.5.40

After each command the controller estimates an carriage return ! Otherwise the command is not beeing executed.

Running without Java script

You have to configure the controller manually like limit switch behavior and function of the digital inputs.

Command	Response	Description
#<motor>l[2565 to 17442]	<motor>l[number]	Sets the limit switch behavior by a bitmask: **Internal limit switch during a reference run:** Bit0: Free travel forwards Bit1: Free travel backwards (default value) **Internal limit switch during a normal run:** Bit2: Free travel forwards Bit3: Free travel backwards Bit4: Stop Bit5: Ignore (default value) **External limit switch during a reference run:** Bit9: Free forwards Bit10: Free backwards (default value) **External limit switch during a normal run:** Bit11: Free travel forwards Bit12: Free travel backwards Bit13: Stop Bit14: Ignore (default value) see 1.5.9
#<motor>i[0 to 100]	<motor>i[0 to 100]	Sets the phase current in percent. For ST2818M1006-A in bipolar serial mode the right value is 33percent (0,66A)
#<motor>r[0 to 100]	<motor>r[0 to 100]	Sets the phase current at standstill.

#<motor>:port_in_a to :port_in_h	<motor>:port_in_<symbol><number>	Sets the function of the digital inputs. User defined = 0 External reference switch = 7 see 1.5.25 for more settings
#<motor>h[0 to 4294967295]	<motor>h[0 to 4294967295]	Sets a bit mask with which the user can reverse the polarity of the inputs and outputs. If the bit of the corresponding I/O is set to `'1'`, there is no polarity reversal. If it is set to `'0'`, the polarity of the I/O is inverted. The bit assignment is shown below: Bit0: Input 1 Bit1: Input 2 Bit2: Input 3 Bit3: Input 4 Bit4: Input 5 Bit5: Input 6 Bit16: Output 1 Bit17: Output 2 Bit18: Output 3 see 1.5.28

Example:

#2l9250	Sets Bit 1,5,10,13 to 1
#2i33	Sets current to 33% of 2A
#2r0	Sets hold current to 0% of 2A
#2:port_in_a0	Sets Input1 to user defined
#2h458815	Sets polarity of input 1 to 6 to not inverted (closer)

Running with Java script

Configuration

On the SMCI33-2 runs a JAVA program in a loop to control the end switch behaviour and takes care that the driver operates how it should be. Furthermore the motor configuration is set like standby current (0%) and operating current (33%) of the motor. Without this program the end switch behaviour is unassigned and the motor could be damaged.

First of all take care that the JAVA program stays up all the time. After each command the firmware is returning the command, without the '#', if it was executed successfully. If the command was wrong you will get it back with an '?' at the end. The following sequence is to make the driver ready to operate:

Check if the autorun of the JAVA program is enabled on motor 1 with '#1(JB'. If the firmware returns '1(JB+1' everything is fine and autorun is enabled. The flashing red LED shows that the program is running. Now it is beneficial to make a reference drive to find and set automatically the zero position which is always next to the negative end switch. Run the commands to do this on motor 1:
'#1p4' (set reference mode on motor 1)
'#1A' (start the motor with given values)
If the slide is always inside the negative end switch it will automatically do a reference drive after the command '#1p4'. After that the operation mode is changed to relative positioning automatically. Now the actuator is ready to drive.

If you want to drive 2000 steps towards the positive switch relative, use the commands:
'#1s2000' (set 2000 steps)
'#1d1' (set direction towards positive switch)
'#1A' (start the motor with given values)

In relative mode only positive step values are allowed, negative values are automatically switched to positive ones.

If you want to drive 2000 steps towards the positive switch absolute the direction command is ignored. You set the direction with positive or negative step values like
'#1p2' (set absolute mode)
'#1s2000' (set target to 2000 steps from zero)
'#1A' (start the motor)
In absolute mode the 's' command sets the target position.

All the time you can check the position with '#1C'. After each drive the firmware sends a command through the comm interface like '001j161'. '001' is the motor number, 'j' is the automatic status of the control and '161' is a 8bit bitmask.
Bitmask:
Bit 0: 1: control ready
Bit 1: 1: zero positioning reached
Bit 2: 1: positioning error

List of commands

command	description	example
#<Motor-ID>A	Start the motor	#1A
#<Motor-ID>S	Stop the motor	#1S
#<Motor-ID>(JA	Start the JAVA Program on the SMCI-33	#1(JA
#<Motor-ID>(JS	Stop the JAVA Program on the SMCI-33	#1(JS
#<Motor-ID>(JB	Show the status of the autorun of the JAVA program	#1(JB
#<Motor-ID>(JB1	Enable the autorun of the JAVA program after boot up the SMCI-33	#1(JB1
#<Motor-ID>(JB0	Disable the autorun of the JAVA program after boot up the SMCI-33	#1(JB0
#<Motor-ID>Zp	show operating mode (4=reference,1=relative,2=absolute)	#1Zp
#<Motor-ID>p<number>	set operation mode (4=reference drive,1=relative,2=absolute)	#1p4
#<Motor-ID>Zs	show way to drive in micro steps	#1Zs
#<Motor-ID>s<number>	set way to drive in micro steps	#1s1000
#<Motor-ID>Zd	show drive direction (0=to the right,1=to the left)	#1Zd
#<Motor-ID>d<number>	set drive direction	#1d1
#<Motor-ID>C	show position	#1C
#<Motor-ID>Zg	show step mode (default=2 Halfstep)	#1Zg
#<Motor-ID>g<number>	set step mode (valid values are 1,2,4,5,8,10,16,32,64) (1=fullstep,2=halfstep...)	#1g16
#<Motor-ID>ZK	Get switch debounce time (default=40ms)	#1ZK
#<Motor-ID>K<number>	Set debounce time for all switches	#1K80
#<Motor-ID>Zh	Get inputs polarity as a bitmask (458812=input 1+2 as opener(inverted), 458815=input 1+2 as closer)	#1Zh
#<Motor-ID>h<number>	Set inputs polarity	#1h458812

Examples

With Java script running

do a reference drive on motor 1:
#1p4 (set reference mode)
#1A (start the motor)

drive 2000 steps to the positive direction in relative mode on motor 2 in half steps:
#2p1 (set mode to relative)
#2d1 (set positive direction)
#2g2 (set halfstep mode) (default value after boot up)
#2s2000 (set steps to drive)
#2A (start the motor)

drive to zero position (only if motor is referenced after reference drive and only in absolute mode) on motor 1:
#1p2 (set mode to absolute)
#1s0 (set drive way to zero)
#1A (start the motor)

Without Java script

Reference run

Command	Response	Description
#<motor>p4	<motor>p4	Set positioning mode external reference run
#<motor>d0	<motor>d0	Sets the direction of rotation: 0: Left
#<motor>A	<motor>A After completing movement: status of the firmware as a bit mask	Starting motor
#<motor>:is_referenced	<motor>:is_referenced+1	If the motor has already been referenced, '1' is returned, otherwise '0'.
#<motor>C	<motor>C[max +/-100000000]	Returns the current position

Example:

#2p4	2p4
#2d0	2d0
#2A	2A 002j163
#2:is_referenced	2:is_referenced
#2C	2C+0

Universal commands

Relative positioning

Command	Response	Description
#<motor>p1	<motor>p1	Set positioning mode relative positioning
#<motor>d[0/1]	<motor>d[0/1]	Sets the direction of rotation: 0: Left 1:right
#<motor>s[0 to +100,000,000]	<motor>s[0 to +100,000,000]	This command specifies the travel distance in (micro-)steps. Only positive values are allowed for the relative positioning. The direc tion is set with command ‘d’.
#<motor>A	<motor>A After completing movement: status of the firmware as a bit mask	Starting motor
#<motor>C	<motor>C[max +/- 100000000]	Returns the current position

Absolute positioning

Command	Response	Description
#<motor>p2	<motor>p2	the specified position is moved to as an absolute position
#<motor>s[0 to +100,000,000]	<motor>s[0 to +100,000,000]	This command specifies the travel distance in (micro-)steps. Only positive values are allowed for the relative positioning. The direc tion is set with command ‘d’.
#<motor>A	<motor>A After completing movement: status of the firmware as a bit mask	Starting motor
#<motor>C	<motor>C[max +/- 100000000]	Returns the current position

Recovering after occurrence of a position error
#<motor>$ reports a Position error by rising bit 3.

Command	Response	Description
#<motor>D	<motor>D	Resetting the position error to default (0). See 1.5.17

It is strongly recommended to start a reference run afterwards

Else

Command	Response	Description
#<motor>$	status of the firmware as a bit mask The bit mask has 8 bits.	Bit 0: 1: Controller ready Bit 1: 1: Zero position reached Bit 2: 1: Position error Bit 3: 1: Input 1 is set while the controller is ready again. This occurs when the controller is started via input 1 and the controller is ready before the input has been reset. Bits 4 and 6 are always set to 0, bits 5 and 7 are always set to 1. See 1.5.22.
#<motor>v		Reading out the firmware version (communication check). See 1.5.23.

Java Program

```
import nanotec.*;

class schalter
{
	// sets config for reference drive
	public static void reference_config()
	{
		drive.StopDrive(0);	// home found
		drive.SetMode(1);
		drive.SetTargetPos(2000);
		drive.SetDirection(1);
		drive.SetMaxSpeed(500);
		drive.StartDrive();	// home distance
		while(!util.TestBit(drive.GetStatus(),0))
		{			// wait for motor to be ready
		}

		drive.SetTargetPos(4000);
		drive.SetDirection(0);
		drive.StartDrive();	// back home
		while(!util.TestBit(io.GetDigitalInput(),1))
		{			// drive back to end switch
		}
		drive.StopDrive(0); // then stop

		drive.SetTargetPos(6000);
		drive.SetDirection(1);
		while(util.TestBit(io.GetDigitalInput(),1))
		{
			drive.StartDrive();	// out of home down from the end switch to zero position
		}
		drive.StopDrive(1);
		util.Sleep(100);
		drive.SetPosition(0);
	}
```

```
public static void main()
{

	//engine configuration
	drive.SetMode(1); // relative positioning
	drive.SetMinSpeed(400);
	drive.SetAcceleration(50); // ramp
	drive.SetCurrent(33); // set current of the motor in percent (0.66A)
	drive.SetCurrentReduction(0); // set standby current
	drive.SetMaxSpeed(32000); // set max speed in steps per second
	drive.SetMaxSpeed2(32000); // set top speed
	util.SetStepMode(64);   // set step mode to 1/64
	config.SetSendStatusWhenCompleted(1);

	// switch configuration
	config.SetLimitSwitchBehavior(17442);
	io.SetInput1Selection(0); // set input1 to user define
	io.SetInput2Selection(0); // set input2 to user define

	// variable
	int input = 0;
	int target = 0;
	int direction = 0;
	int speed = 0;
	int steps = 0;
	boolean stop = true;
	int target_original = drive.GetTargetPos();
	int count = 0;

	// main loop
	while(true)
	{
		input = io.GetDigitalInput(); // read i/o
		target = drive.GetTargetPos(); // get traverse path
		direction = drive.GetDirection(); // get direction 0=left, 1=right
		speed = drive.GetMaxSpeed(); // get max speed
		steps = util.GetStepMode(); // get step mode
		stop = true;  // failsafe stop

		count++;
		if (count > 10)  // execute only every 10 loops to avoid slowdown of the script
		{
			// calculates the target for full steps
			if (target/steps != target_original)
			{
				target_original = target;
				drive.SetTargetPos(target*steps);
			}
			io.SetLED(1);
			count = 0;
		}
		else
		{
			io.SetLED(0);
		}

		// relative positioning (1)
		if (1 == drive.GetMode())
		{
			if (target < 0)
			{
				drive.SetTargetPos(-target); // only positive values allowed
			}
```

```
		if( util.TestBit(input,0) ) // at left end switch
		{
			if (0 == direction) // if there and wants to drive down
			{
				stop = false;
			}
		}
		else if( util.TestBit(input,1) ) // at right end switch
		{
			if (1 == direction) // if there and wants to drive down
			{
				stop = false;
			}
		}
}

// absolute positioning (2)
if (2 == drive.GetMode())
{
		if( util.TestBit(input,0) ) // at left end switch
		{
			if (target < 0) // if there and wants to drive down
			{
				stop = false;
			}
		}
		else if( util.TestBit(input,1) ) // at right end switch
		{
			if (target > 0) // if there and wants to drive down
			{
				stop = false;
			}
		}
}

// reference drive
if (4 == drive.GetMode())
{
		if (target < 0)
		{
			drive.SetTargetPos(-target); // only positive values allowed
		}
		stop = false;
		util.SetStepMode(2);
		drive.SetMaxSpeed(1000);
		drive.SetDirection(0);
		while( !util.TestBit(io.GetDigitalInput(),1) && 4 == drive.GetMode() )
		{		// wait for drive hit the negative end switch
		}
		if (4 == drive.GetMode())
		{
			reference_config();
		}
		else
		{
			util.SetStepMode(64);
			drive.SetMaxSpeed(32000);
		}
		if( util.TestBit(input,0) ) // at left end switch (should never happen)
		{
			stop = true;
			if (0 == direction) // if there and wants to drive down
			{
```

```
                                        stop = false;
                                }
                        }
                }

                // Stop at end switch
                if( (util.TestBit(input,0) || util.TestBit(input,1)) && stop)
                {
                        drive.StopDrive(0);
                        // send just every 9th round
                        if (count > 9)
                        {
                                // send position across comm
                                if(util.TestBit(input,0))
                                {
                                        comm.SendInt(0);
                                }
                                else
                                {
                                        comm.SendInt(1);
                                }
                                count = 0; // reset counter
                        }
                }

        }// while end
    }// main()
}// class
```