

SCpar100.dll Instruction Manual

This document explains the functions and exports found in the SCpar100.dll file, downloadable from <http://www.salemcontrols.com/>

The SCpar100.dll is a dynamically linked library that can be used in VB or VC++ programs to control stepper motors through the parallel of a PC using controller boards from Salem Controls Inc. The .dll provides a number of functions that will take care of the timing and parallel port communications needed to send the appropriate signals to the controller board to cause the motors to rotate.

A .zip file containing the following files can be downloaded for free from <http://www.salemcontrols.com/software/parallel.html>:

exports.def
SCpar100.dll
SCpar100.exp
SCpar100.H
SCpar100.lib

Below is a description of each of these files.

exports.def -- This file contains a listing of the functions found in the .dll file, SCPar100.dll

SCPar100.dll -- This is the .dll file that can be used in various software programs to rotate stepper motors using the parallel port and stepper motor controllers from Salem Controls Inc.

SCPar100.exp -- Exports file for the .dll containing export definitions.

SCpar100.H -- C header file for the .dll. Contains a list of the functions with their types and the types of variables they need to be passed.

SCpar100.lib -- Library file for the .dll.

Some of these files will be of use to you in writing your own software and others will not. A listing of the functions contained in the .dll file along with their descriptions can be found below.

SCpar100.dll Function Descriptions

void **OutStepDLL**(*short* PortAddress, *short* Data)

This function writes the data contained in Data to the physical port address/memory location contained in PortAddress.

short **InStepDLL**(*short* PortAddress)

Returns the value found at the port address/memory location PortAddress.

int **Set_Port** (*int* parallel_port)

Pass either a 1, 2, or 3 to set the port to be used to LPT1, LPT2, or LPT3. The values for the Data, Status, and Control Registers for the three port values are shown below:

parallel_port	DataRegister	StatusRegister	ControlRegister
1	888	889	890
2	632	633	634

3	956	957	958
---	-----	-----	-----

```
void set_m1_dir (int motor1_dir)
void set_m2_dir (int motor2_dir)
void set_m3_dir (int motor3_dir)
void set_m4_dir (int motor4_dir)
```

Pass either a 0 or 1 to each of these functions to set the direction pin for that function's respective motor. For example, the function call "set_m1_dir(0)" will set pin 3 of the parallel port to low. The pins of the parallel port affected by the above are pins 3, 5, 7, and 9 respectively.

```
void set_outputs (short outval)
```

This function accepts a short integer value from 0 to 15 that is passed to the Control Register of the parallel port. The effect of this is that the 4 pins of the Control Register can be turned on or off. Control Register bits 0, 1, 2, and 3 correspond to pins 1, 14, 16, and 17 of the parallel port, respectively. When a call is made such as "set_outputs(15)" the binary value 1111 is passed to the Control Register, turning the 4 output pins on. Similarly, "set_outputs(5)" would send 1010 to the Control Register, turning pins 1 and 16 on, and 14 and 17 off.

These pins can be used as outputs for driving relays and external devices, or they can enable and disable the windings of the motors when connected to certain Salem Controls Inc. controller boards.

```
int get_inputs ()
```

This function returns the value of the Status Register. Status Register bits 3, 4, 5, 6, and 7 correspond to pins 15, 13, 12, 10, and 11, respectively, of the parallel port. Status Register bit 7, pin 11 of the parallel port, is inverted.

For example, if pin 15 of the parallel port (Status Register bit 3) is at +5V, a value of 4 will be added to the Status Register. If pin 13 of the parallel port (Status Register bit 4) is at +5V, a value of 8 will be added to the Status Register. One note is that pin 11, or Status Register bit 7, is inverted. The table below shows values of the status register (values that will be returned by a call to "get_inputs()") for various voltage levels on the parallel port pins.

Parallel Port Pin (Status Register Bit)					Value Returned by "get_inputs()"
15 (3)	13 (4)	12 (5)	10 (6)	11 (7)	
GND	GND	GND	GND	GND	128
+5V	GND	GND	GND	GND	136
GND	+5V	GND	GND	GND	142
+5V	+5V	GND	GND	GND	150
.
.
GND	GND	+5V	+5V	+5V	96
+5V	GND	+5V	+5V	+5V	104
GND	+5V	+5V	+5V	+5V	112
+5V	+5V	+5V	+5V	+5V	120

```
void set_m1_del(int m1_delay_time)
void set_m2_del(int m2_delay_time)
void set_m3_del(int m3_delay_time)
void set_m4_del(int m4_delay_time)
```

These functions set the delay times of the 4 motors. The integer value "mX_delay_time" that is passed to the function represents the delay, in milliseconds, between the execution of steps for that particular motor. The value must be an integer and 1 or greater. For example, a call of "set_m1_del(5)" would set the speed of rotation for motor 1 to 200 steps per second, or 5 milliseconds between individual steps.

```
int get_m1_del()  
int get_m2_del()  
int get_m3_del()  
int get_m4_del()
```

These functions return the integer values for each axis representing the number of milliseconds between individual steps. See also **set_mX_del**(*int* mX_delay_time) above.

```
void rotate_motors(long m1_StepsVal, long m2_StepsVal, long m3_StepsVal, long  
m4_StepsVal)
```

This function causes the motors to begin rotation at the step rates defined by the **set_mX_del**(*int* mX_delay_time) functions. The function requires 4 arguments, which represent the number of steps to be executed by each of the 4 motors. For example, to rotate motor 1 200 steps, motor 4 58 steps, and motors 2 and 3 no steps, the call would look like "rotate_motors(200, 0, 0, 58)"

```
void rotate_motors_thread(void *param)
```

This function wraps the rotation of the motors in a thread so that the PC can perform other functions while the motors are rotating. This function should NOT be called directly.

```
void stop_m1()  
void stop_m2()  
void stop_m3()  
void stop_m4()
```

Calling these function sets the number of steps remaining for each motor to execute to 0, effectively stopping the rotation of the motor.

```
void stop_all()
```

This function sets the number of steps left to execute for all 4 motors to 0, effectively stopping the rotation of the motors.

```
int CheckThreadRunning()
```

This function returns a 1 if the motor rotation thread is still running, or a 0 if it is not. This allows the software to check to see if the motors are still rotating.

```
const double DLLVersion()
```

Returns the version of the .dll as a *double*.