tbd 2016 V 0.4.0 beta. Doug Johnson (finson@whidbey.com)

DeviceFeature and Device Drivers - Appendix B: Device Registers and Parameter Block Layout

The document device-driver.md describes the overall device feature proposal and most parts of the detailed design. Appendix A lists the standard virtual registers that all devices can implement.

This Appendix B defines the registers that are actually implemented by several known device drivers and the detailed format of the parameter block bodies they expect.

Common Device Registers (CDR_)

`CDR_DriverVersion`

`CDR_LibraryVersion`

Get version number and name of the DeviceDriver or supporting library.

Version numbers are in the Semantic Versioning format x.y.z-a.b.c See semver.org for more info.

The first byte of the return data buffer is the size in bytes of the version identifier packet that follows. Name strings use UTF-8 encoding and are null-terminated.

In the initial implementation, the size of the version identifier packet is 6 bytes. The name string immediately follows the version identifier packet and is limited to 128 bytes maximum, including the null terminator.

The size of the receiving buffer should be large enough to hold the 1-byte packet size, a version identifier packet, and a name string (including the null terminator). If the buffer size is not large enough, an error will be returned (`EMSGSIZE`).

Method signature

`int status(int handle, CDR_DriverVersion, int bufSize, byte *buf)`
`int status(int handle, CDR_LibraryVersion, int bufSize, byte *buf)`

Return data buffer

```
0  version descriptor packet size (6, in this example)
1  major version (x)
2  minor version (y)
3  patch version (z)
4  pre-release (a)
5  pre-release (b)
6  pre-release (c)
7..n  name string (UTF-8, null terminated)



0  version descriptor packet size (3, in this example)
1  major version (x)
2  minor version (y)
3  patch version (z)
4..n  name string (UTF-8, null terminated)
```

StepperDriverBasic

`CDR_Configure`

Get/set stepper motor interface configuration

Configure the stepper motor attached to the given handle and its associated software object.

Method signature

`int status(int handle, CDR_Configure, int count, byte *buf)` `int control(int handle, CDR_Configure, int count, byte *buf)`

Parameter block buffer

2-wire interface (EasyDriver: step and direction)

```
 0  interface type (0)
 1  steps per revolution (LSB)
 2  steps per revolution (MSB)
 3  direction pin (LSB)
 4  direction pin (MSB)
 5  step pin (LSB)
 6  step pin (MSB)
```

2-wire interface (unipolar motor)

```
 0  interface type (1)
 1  steps per revolution (LSB)
 2  steps per revolution (MSB)
 3  pin1 (LSB)
 4  pin1 (MSB)
 5  pin2 (LSB)
 6  pin2 (MSB)
```

4-wire (bipolar motor)

```
 0  interface type (2)
 1  steps per revolution (LSB)
 2  steps per revolution (MSB)
 3  pin1 (LSB)
 4  pin1 (MSB)
 5  pin2 (LSB)
 6  pin2 (MSB)
 7  pin3 (LSB)
 8  pin3 (MSB)
 9  pin4 (LSB)
10  pin4 (MSB)
```

`STP_MoveR`

Move to new relative position

Method signature

`int control(int handle, STP_MoveR, 5, byte *buf);`

Parameter block buffer

```
 0  delta position (steps) (LSB)
 1  delta position (steps)
 2  delta position (steps)
 3  delta position (steps) (MSB)
 4  block? (0 or 1)
```

`STP_RPMSpeed`

Set stepping speed using Revolutions Per Minute

Method signature

`int control(int handle, STP_RPMSpeed, 4, byte *buf);`

Parameter block buffer

```
0  RPM (LSB)
1  RPM
2  RPM
3  RPM (MSB)
```

MCP9808 Temperature Sensor

AMBIENT_TEMP = 5, MANUF_ID = 6, DEVICE_ID = 7, RESOLUTION = 8

`CDR_Configure`

Get/set MCP9808 interface configuration

Get/set the configuration of the temperature sensor attached to the given handle and its associated software object.

Method signatures

`int status(int handle, CDR_Configure, int count, byte *buf)` `int control(int handle, CDR_Configure, int count, byte *buf)`

Parameter block buffer

```
0  interface type (0)
1  steps per revolution (LSB)
2  steps per revolution (MSB)
3  direction pin (LSB)
4  direction pin (MSB)
5  step pin (LSB)
6  step pin (MSB)
```

`UPPER_TEMP`

`LOWER_TEMP`

`CRIT_TEMP`

Get/set upper/lower/critical temperature limit registers.

Method signatures

`int status(int handle, int reg, 2, byte *buf);` `int control(int handle, int reg, 2, byte *buf);`

Parameter block buffer

```
0  temperature °C (LSB)
1  temperature °C (MSB)
```

`AMBIENT_TEMP`

Get ambient temperature in degrees C.

Method signature

`int status(int handle, AMBIENT_TEMP, 2, byte *buf);`

Parameter block buffer

```
0  temperature °C (LSB)
1  temperature °C (MSB)
```

`MANUF_ID`

`DEVICE_ID`

Get Manufacturer ID or Device ID

Method signature

`int status(int handle, MANUF_ID, 2, byte *buf);` `int status(int handle, DEVICE_ID, 2, byte *buf);`

Parameter block buffer

```
0  0x00 (manufacturer ID - MSB)
1  0x54 (manufacturer ID - LSB)
```

or

```
0  0x04 (device ID - MSB)
1  0x00 (device ID - LSB)
```

`RESOLUTION`

Get/set sensor resolution

Method signature

`int status(int handle, RESOLUTION, 1, byte *buf);` `int control(int handle, RESOLUTION, 1, byte *buf);`

Parameter block buffer

```
0  0,1,2, or 3 (resolution control bits)
```