SuperServo I2C Protocol Specification

Colin Mackenzie
mailto: colin@colinmackenzie.net
http://www.colinmackenzie.net

http://OpenServo.com

Created
Aug 28th, 2003

Updated
Jan 10th, 2006

The I2C implementation on the SuperServo is based on sending commands to the servo to read or write register locations inside the PIC microcontroller that change the operation of the servo. Care must be taken on the user's part not to write to registers that may cause undesired operation of the servo.

Many operations perform the same semantic operation on the servo and their registers. These semantically equivalent operations exist to decrease the amount of bus traffic between the host and the servo that is required to service your application of the servo. For example, registers that reside next to each other in the microcontroller's memory can be read sequentially in one bus transaction.

A transaction on the I2C bus is defined in this document to be the bytes sent between each I2C start and stop condition. A command message is 1, 2, or more associated transactions that make up a single servo command operation.

Commands sent to the servo are either single operation (SOP) command messages or dual/multi operation (DOP) messages. Single operation command messages contain the complete servo command in a single I2C bus transaction. Multi operation command messages require at least 2 bus transactions. However, for DOP transactions you may repeat the second transaction any number of times to receive updated contents of the given registers. Also, single operation command messages do not interfere with the setup of multi-operation command messages and can be interplexed with them.

How to interpret command descriptions in this document:

[DA]	The device (servo) address. This is configured via the solderable address jumper on the servo or by the value stored in eeprom.
rn	where n is 0,1,2,...n. The index location of the specified register.
[x]	The contents of the specified register 'x'.
(x)	The byte x is not required to be sent in this I2C message.
{	The I2C start signal
}	The I2C stop signal
0xff	or any hex value means the literal byte.

Author Notes

This document is still considered to be preliminary specification. The operation of the i2c bus implementation may yet change as openservo.com standards are developed.

I have to confirm the operation of the ‘0x00 Null Command’. I can’t recall exactly when the 0x00 is required for the second transaction of DOP messages. I will soon resolve this and correct any errata in this document.

All SOP operations are confirmed fully operational. Except currently no bootloader is implemented for the Firmware Programming command 0x1f.

DOP operations are partially tested but not yet widely used in operation.

I2C Command Index

Dual/Multi Operation (DOP) Commands

0x00	Reserved / Null Command
0x01/R	Read device registers
0x01/W	Write device registers
0x02/R	Read device register region
0x02/W	Write device register region
0x0a/R	Read device info
0x0d/R	Echo DOP Command and Register Set

Single Operation (SOP) Commands

0x11/W	Write device registers
0x12/W	Write device registers region
0x14/W	Set servo desired position
0x15/W	Set servo desired speed
0x16	Set servo output enable
0x17	Set servo output disable
0x18	Restore Servo Settings
0x19	Load Servo Settings
0x1a	Save Servo Settings
0x1e	Reset servo
0x1f	Enter device firmware programming mode

DUAL OPERATION (DOP) COMMANDS

Dual operation commands require at minimum two i2c bus transactions to complete. The advantage of dual operation commands is that the first (setup) transaction can be sent once, and the second and later transactions can be used to repeat the operation. Depending on your application you may only need to send one setup transaction during power-up stating only those registers you wish to read/write throughout the operation of the servo.

0x00 Reserved / Null Command

This is used to repeat the second part of a multi operation command message. You may take advantage of this if you are continually reading or writing to the same register locations.

0x01/R Read device registers

```
Send      { [DA] 0x01 r0 r1 r2 ... rn }
Receive   { [DA] 0x00 [r0] [r1] [r2] ... [rn] }
```

A command message is first sent to specify the register locations whose contents are to be read. After which, a read operation can be performed to read the contents of the specified registers in the given sequence.

After a single read command message is sent. Repeated read operations can be performed to continually read the contents of the given register sequence.

The content of the next register is read and transferred into the i2c shift register on each ACK bit that occurs on the i2c bus. Thus, the continuous reading operations will read the up-to-date contents of the registers.

It is not necessary to read all the contents of the registers in the read operation. If a stop condition is received before all registers are transferred the next read operation will read the contents from the BEGINNING of the sequence. In this way, you can prioritize the most often read registers at the beginning and read the later registers only when required, while still not having to send a new read command.

If you read beyond the end of the register sequence the sequence will wrap to the beginning and continue. This allows for continuous scanning of a register sequence without starting a new message.

0x01/W Write device registers

```
Send { [DA] 0x01 r0 r1 r2 ... rn }
Send { [DA] 0x00 [r0] [r1] [r2] ... [rn] }
```

A command message is first sent to specify the register locations whose contents are to be written to. A second message is then sent to specify the contents of the registers. Notice the 0x00 after the device address is sent. This is to signal the servo that this write message is associated with the previous multi-operation command message. If this byte was other than 0x00 the servo would interpret the write message as a new command.

Many of the same principles of the read device registers apply here. Repeated write operations can be performed by sending multiple messages or wrapping beyond the register sequence. It is also not necessary to write all registers in each message. The register is written at the end of each byte transferred.

0x02/R Read device register region

```
Send { [DA] 0x02 region }
Receive { [DA] r0 r1 r2 r3 ... rn }
```

This command reads from a region of registers starting at the specified region offset. The first byte is read from the register at [region], the next byte from [region+1], the next byte from [region+2], etc.

Repeated read operations can be performed by sending multiple messages. The offset is reset to 'region' at the beginning of each message.

The register is read at the start of each byte transmission.

0x02/W Write device register region

```
Send { [DA] 0x02 region }
Send { [DA] 0x00 r0 r1 r2 r3 ... rn }
```

Writes to a region of registers. The first byte is written to the register at [region], the next byte to [region+1], the next byte to [region+2], etc. Repeated write operations can be performed by sending multiple messages. The offset is reset to 'region' at the beginning of each message. The register is written at the end of each byte transferred.

0x0a/R Read device info

```
Send { [DA] 0x0a }
Receive { [DA] 'S' 'u' 'p' 'e' 'r' 'S' 'e' 'r' 'v' 'o' tab ... 0x00 }
```

Reads the servo device information which can contain the SuperServo string, the version and copyright string. Fields are separated by the tab character and the line is terminated by the null character. You should continue reading from the device until a 0x00 (null) is encountered.

0x0d/R Echo DOP Command and Register Set

```
Send { [DA] 0x0d .. .. .. .. }
```

This command echoes the last dual-operation command back to the master controller. It is a convenient way to test the buffer capabilities of the device or ensure the device communications is working properly.

SINGLE OPERATION (SOP) COMMANDS

Single Operation commands can be performed in a single i2c transaction. These commands also do not interrupt the setup of a Dual Operation (DOP) command thus they can be performed in between DOP commands. If you rely on DOP commands through typical servo operation, you can use these commands to perform occasional reads and writes without disturbing your normal operation.

0x11/W Write device registers

```
Send { [DA] 0x11 r0 [r0] r1 [r1] r2 [r2] ... rn [rn] }
```

This command message is performed in a single operation and does not affect setup of dual operation commands.

The contents to be written to the registers are interplexed with the register locations themselves. Thus pairs of [register, content] are sent in a single command message.

As this mode of command does not affect DOP command messages, read command messages can be occurring before and after a SOP write device registers without having to resend the read command operation.

0x12/W Write device registers region

```
Send { [DA] 0x12 region [r0] [r1] [r2] ... [rn] }
```

This command message is performed in a single operation and does not
affect setup of dual operation commands.

This command writes registers in sequence starting at the offset specified by the region byte.

As this mode of command does not affect DOP command messages, read command messages can be occurring before and after a SOP write device registers without having to resend the read command operation.

0x14/W Set servo desired position

```
Send { [DA] 0x14 position (lsb) }
```

This command message is performed in a single operation and does not affect setup of dual operation commands.

Position is the desired position in the range of 0 – 255, or 0 – 1023 if 10bit position mode is defined during code compilation. Typically, for servo control loop performance, the ADC and the PID control loop uses 10bit resolution internally even though the target position may only be given as 8 bits which are then shifted 2 positions to get a 10bit target position. In most applications 10bit target resolution is probably not required.

The ADC that reads the servos current position has a resolution of 10 bits. Further resolution of the desired position can then also be achieved by sending the least significant bits of the position in the lsb byte. Only the least significant 2 bits of the lsb byte are used. If this byte is omitted, the two lsb's of the desired position will be set to 0, thus making the servo target position resolution equivalent to 8bits.

0x15/W Set servo desired speed

```
Send { [DA] 0x15 speed }
```

[not yet implemented]

This command message is performed in a single operation and does not
affect setup of dual operation commands.

Speed is the maximum speed of swing in the range of 0 - 255. This value is used to restrict the duty cycle of the PWM output.

0x16 Set servo output enable

```
Send { [DA] 0x16 }
```

Enables the output of the PWM to the servo. By default, the output of the PWM is disabled so the servos don’t jolt on power up.

0x17 Set servo output disable

```
Send { [DA] 0x17 }
```

Disables the output of the PWM to the servo effectively turning off the motion of the servo. The servo horn is free to move from external forces. The position of the servo can still be read while the output is disabled. This does not affect the state of the servo's dynamic braking if it is enabled.

0x18 Restore Servo Settings

```
Send { [DA] 0x18 }
```

Restores the settings for the servo to factory default values. The restored settings include the servo desired position, speed, status and the PID coefficients.

0x19 Load Servo Settings

```
Send { [DA] 0x19 }
```

Load saved servo settings from the EEPROM on the PIC (if available). The loaded settings include the servo state, desired position, speed, status and the PID coefficients.

0x1a Save Servo Settings

```
Send { [DA] 0x1a }
```

Saves servo settings from the EEPROM on the PIC (if available). The saved settings include the servo desired position, speed, status and the PID coefficients.

0x1e Reset servo

```
Send { [DA] 0x1e }
```

Resets the servo. This simply resets the PC (program counter) register to 0000.

0x1f Enter device firmware programming mode

```
Send { [DA] 0x1f }
```

NOTE: This command is only valid for device with cpu's supporting self-programming and containing the firmware bootloader! This function jumps to PIC program address 0x0001, just after the reset vector. This location should contain the jump vector to high memory where the bootloader resides.

This command places the servo in firmware programming mode enabling the servo firmware can be updated. This is an advanced function and should be used with care as improper operation of the servo can be result.

Once the servo is placed in programming mode, all normal operations of the servo halt including the standard IIC implementation protocols. See the "Firmware Programming" reference for more info.

[BEGIN Preliminary] When the servo is placed in programming mode the reset vector is redirected to the programming firmware. Thus upon reset the device will reenter programming mode and wait for instructions. This is to prevent faulty code or a faulty firmware write from rendering the servo inoperable as upon reset the servo will reenter programming mode.

The redirection of the reset vector is confirmed by resetting the device at the completion of the "Enter device programming mode" command. *[END Preliminary]*