DT0055
Design tip

Maintaining message synchronization with L647x, L648x, and PowerStep01 services

By Dennis Nolan

Main components	
L6470, L6472	Fully integrated microstepping motor driver with motion engine and SPI
L6474	Fully integrated microstepping motor driver
L6480, L6482	Fully integrated microstepping motor controller with motion engine and SPI
powerSTEP01	System-in-package integrating microstepping controller and 10 A power MOSFETs

Purpose and benefits

The family of devices receive commands over a simple SPI interface. However, since different commands have a different number of data bytes associated with them, care must be taken to keep the communications synchronized. In noisy environments or when other errors occur, the communication could lose synchronization and cause improper operation of the system. This document describes a way to restore the synchronization in the communication.

Description

The L6470 and L6480 families of stepper motor drivers and the PowerStep01 share a similar communication and command structure that receives commands and data over an SPI interface and returns requested status and data via the same interface. The command protocol is simple, with each message consisting of a command byte (op code) followed by 0, 1, 2, or 3 data bytes.

While the data sheets for the respective parts are clear as to the number of data bytes which are to be sent with any given command, a mistaken interpretation is certainly possible. Consider the case where the controller sends an absolute GOTO command but, mistakenly, sends only two post bytes of data instead of three. This portends disaster on a grand scale! The next byte that is sent will be intended as the “op code” of a new command but it will be interpreted (and rightly so) by the device as the third data byte of the previous command. Now things get really bad. The next byte is intended as the first data byte of the new command but will be interpreted as a new command. The system will now continue to interpret data as commands, commands as data, and all other possible combinations. This is not a good situation! Random luck may allow the system to re-establish a proper "message sync", but there are no guarantees.

A well-designed system will never make this mistake. But, any system has to tolerate electrical noise and other disturbances. It is not difficult to imagine that, in the life of a typical system, such disturbances could cause the system to go into this same “out of message sync” situation.

However, if the system has a little idle time where there is no command or data being sent to the device, the controller can periodically send a sequence of three NOP commands (a single command byte of 0x00 with no data bytes) during these idle times. This will, if required, reset the command synchronization.

The longest device command string includes one command byte and three data bytes. Therefore, the possible states of the device command interpreter are:

a. Waiting for a command
b. Waiting for the first data byte
c. Waiting for the second data byte
d. Waiting for the third data byte

If the device was waiting for a command (message sync is intact), then it will just receive three consecutive NOPs and everything will continue to operate normally. If it was waiting for one, two, or three data bytes, it will interpret the NOPs as one, two or three bytes with values of zero. If it needs less than three data bytes, the leftover bytes will just be interpreted as NOP commands, which do nothing.

In any case, the device will emerge from receiving the string of three 0x00 bytes always in “message sync”. This can be considered as a system re-sync and, if your system allows you to do it, some fairly cheap insurance. It is certainly a good idea to send this sequence whenever the device is being re-initialized and, if the sequence can be sent periodically during normal operation, so much the better.

Support material

Documentation
Datasheet: L6470, Fully integrated microstepping motor driver with motion engine and SPI
Datasheet: L6472, Fully integrated microstepping motor driver with motion engine and SPI
Datasheet: L6474, Fully integrated microstepping motor driver
Datasheet: L6480, Fully integrated microstepping motor controller with motion engine and SPI
Datasheet: L6482, Fully integrated microstepping motor controller with motion engine and SPI
Datasheet: powerSTEP01, System-in-package integrating microstepping controller and 10 A power MOSFETs

Revision history

Date	Version	Changes
15-Feb-2016	1	Initial release