*Application Report*

# *Daisy Chain Implementation for Serial Peripheral Interface*

*Ishtiaque Amin*
*James Lockridge*

*Brushed DC and Stepper Motors*

**ABSTRACT**

Embedded and electronic systems use serial interfaces to transfer data between connected devices. Serial peripheral interface (SPI) is one type of serial communication interface that provides synchronous transfer of data between a microcontroller (MCU), and one or more peripheral devices. In the SPI protocol, the MCU generates a clock signal (SCLK), a select signal (nSCS), and a serial data out (SDO) signal (e.g. data transferred to the peripheral devices). The peripheral devices receive the data signal on the serial data input (SDI) pin. The data from signal from the MCU SDO to the peripheral SDI synchronizes with the clock signal while the select signal is active. The peripheral devices may also send data back to the MCU on their own SDO outputs to the SDI of the MCU. This output from the peripheral devices also synchronizes with the clock signal. SPI is a common form of interface in automotive applications for better flexibility, configurability, and fault reporting by the electronic components. This report describes the method for synchronous serial communication to multiple peripheral devices (motor driver devices for example) using SPI with daisy chain functionality.

## 1 Trademarks

All trademarks are the property of their respective owners.

## Table of Contents

## List of Figures

## 2 Functional Overview

Figure 2-1 shows daisy chain configuration to keep GPIO ports available when multiple devices are communicating to the same MCU by sharing the SPI bus. The MCU must be configured to generate a chip select signal, a header signal, and multiple address fields depending on the number of devices connected in series. In addition, the MCU generates a clock signal that allows the data to be synchronously transmitted through the chain. From Figure 2-1 the MCU sends out a signal via SDI1 which gets decoded by each device in the chain, and the appropriate commands are executed. Details about the decoding method is described herein.

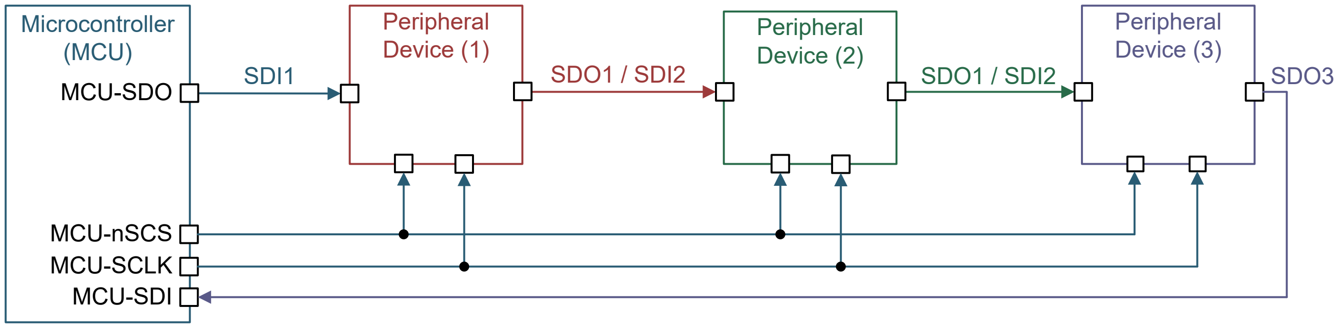

**Figure 2-1. Three Motor Driver Devices Connected in Daisy Chain**

The chip select signal (MCU-nSCS) defines a frame interval for provision to the motor driver devices. The clock signal is to control synchronous transfer of serial data between the MCU and the motor driver devices. The MCU-SDO is the data being sent from the MCU to the first motor driver device in the chain, and M-SDI is the data being received from the last motor driver device in the chain. The MCU-SDO consists of the header bytes, address bytes, and data bytes. The MCU-SDI, generated by the last motor driver device in the chain, consists of status bytes, header bytes (same as MCU-SDO), and report bytes.

From MCU to the motor driver devices, the header field is the first field to be transmitted in the frame interval, and specifies a number of peripheral devices communicatively coupled to the MCU. The header field is followed by address fields. The multiple address fields are to be transmitted in the frame interval. Each of the address fields corresponds to a different motor driver device. A first of the address fields transmitted by the MCU in the frame interval corresponds to the last motor driver device to receive the header field, and vice versa. The address fields are followed by data fields which carry the information to be executed in a given motor driver device.

## 3 Implementation Details

The traditional daisy chain configuration allows for a reduced number of terminals on the microcontroller, but limits communication bandwidth. For example, in some implementations, two transactions (two communication frames) are required to read from a peripheral device. In systems that allow single frame reads, transfer speed of information through the serially connected peripheral devices is reduced as the number of peripheral devices increases.

The synchronous serial communication system described here employs the daisy chain configuration to reduce the number of terminals required on the microcontroller, allows reads in a single frame, and provides a transfer bit rate that is independent of the number of peripheral devices in the chain. Some implementations provide operation with a single peripheral device without introducing additional protocol overhead.

Figure 2-1 shows a block diagram of an example of a synchronous serial communication system in accordance with the daisy chain implementation described in this section. The SPI system includes a MCU and one or more motor driver devices in series to communicate with the MCU.

For example, the MCU outputs data signal and the first motor driver device in the chain receives the data signal (SDI1). After performing any relevant operation, it passes along the outputs signal SDO1/SDI2, which includes a portion of the data provided via SDI1. The same sequencing is repeated throughout the entire chain until the final motor driver device is reached.

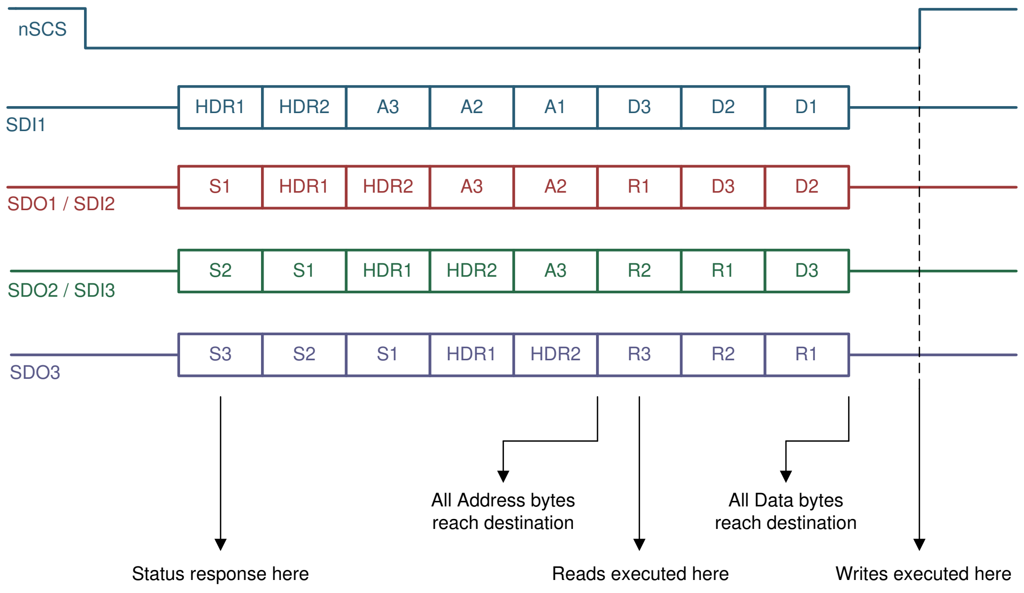
nSCS
SDI1
HDR1 HDR2 A3 A2 A1 D3 D2 D1
SDO1 / SDI2
S1 HDR1 HDR2 A3 A2 R1 D3 D2
SDO2 / SDI3
S2 S1 HDR1 HDR2 A3 R2 R1 D3
SDO3
S3 S2 S1 HDR1 HDR2 R3 R2 R1
All Address bytes reach destination
All Data bytes reach destination
Status response here
Reads executed here
Writes executed here

**Figure 3-1. SPI Frame With Three Motor Driver (Peripheral) Devices**

Figure 3-1 shows the data transmit and receive structure between the MCU and three motor driver devices. The first device in the chain receives data from the MCU in the following format for 3-device configuration: 2 bytes of header (HDRx) followed by 3 bytes of address (Ax) followed by 3 bytes of data (Dx). After the data has been transmitted through the chain, the MCU receives the data string in the following format for 3-device configuration: 3 bytes of status (Sx) followed by 2 bytes of header followed by 3 bytes of report (Rx).

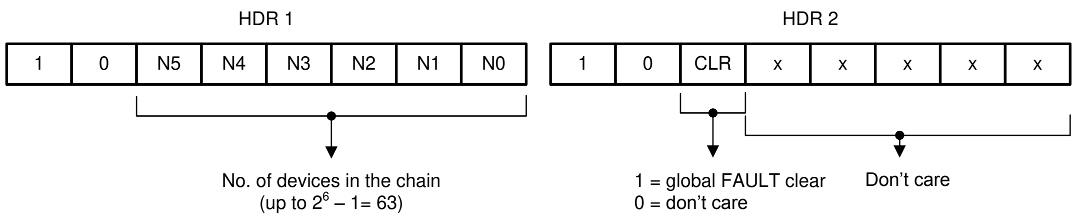

**Figure 3-2. Header Bytes**

The header bytes (HDRx), shown in Figure 3-2 contain information of the number of devices connected in the chain, and a global clear fault command that will clear the fault registers of all the devices on the rising edge of the chip select (nSCS) signal. Header bytes must start with 1 and 0 for the two MSBs. The header identification value is disposed at the start of the header field and is therefore the first value transmitted by the MCU that is received by each of the motor driver devices in a sequence determined by its position in the chain. Header values N5 through N0 are 6 bits dedicated to show the number of devices in the chain. Up to 63 devices can be connected in series for each daisy chain connection. The 5 LSBs of the HDR2 register are don't care bits that can be used by the MCU to determine integrity of the daisy chain connection.

The address field (Ax) is an implementation of the address field from the MCU that needs to be accessed for a particular motor driver device. The address field includes an identification value of 0 that identifies that byte as an address field. It also includes a read/write control value (R/W bit) and an address value in the byte. The identification value specifies whether a location of the motor driver device corresponding to the address value is to be read or written. For example, if the R/W bit is set to logic 1, then the address corresponding address value is to be read. If the R/W bit is a logic 0, then the address corresponding to the address value will be written with the data in the data field.

The data field (Dx) specifies a value to be written at the address value of the motor driver device. For example, a motor driver device writes the value contained in the data field to the address specified in the address field corresponding to the motor driver device at the termination of the frame in which the data field is received.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Header Bytes (HDRx) | 1 | 0 | N5 | N4 | N3 | N2 | N1 | N0 |
| | 1 | 0 | CLR | X | X | X | X | X |
| Status Byte (Sx) | 1 | 1 | OTW | UVLO | CPUV | OCP | TSD | OLD |
| Address Byte (Ax) | 0 | R/W | A4 | A3 | A2 | A1 | A0 | X |
| Data Byte (Dx) | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

**Figure 3-3. Contents of Header, Status, Address, and Data Bytes**

When the chip-select signal is active, the motor driver devices begin to receive clock signal, and at each clock pulse, the motor driver devices transmit a status value. For example, while the motor driver device is receiving the header field, it is transmitting a status bits that provide information about the fault status register for each device in the daisy chain so that the MCU does not have to re-initiate another read command to read the fault status from a given motor driver device. This reduces the number of read commands from the MCU and makes the system more efficient to determining fault conditions flagged by a device.

The status field includes an identification value and a status value. The identification value must start with 1 and 1 for the two MSBs. The status field contains global fault bit identification for the motor driver device from which the status byte was generated. In Figure 3-3 the global fault bits shown refer to the DRV8873-Q1 device as an example. These six global fault bits, following the two identification bits, can vary depending on the motor driver device.

When data passes through a device, it determines the position of itself in the chain by counting the number of total bytes it receives followed by the first header byte. For example, in this 3-device configuration, device 2 in the chain receives two status bytes before receiving the HDR1 byte which is then followed by the HDR2 byte. From the two status bytes, the motor driver device can determine that its position is second in the chain. From the HDR2 byte, each device can determine how many devices are connected in the chain. In this way, the data only loads the relevant address and data byte in its buffer and bypasses the other bits. This protocol allows for faster communication without adding latency to the system for up to 63 devices in the chain. Figure 3-3 shows the encoding of a status byte.

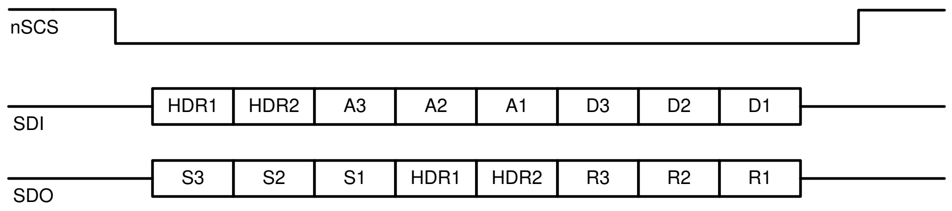

**Figure 3-4. SPI Data Sequence between MCU and Three Motor Driver Devices**

Figure 3-4 shows how the data sequence should look like between the MCU and motor driver devices for three peripheral units connected in the chain. The number of header bytes will always remain 2, but the other bytes (status, address, data, and report) will scale according to the number of devices in series. Such a daisy chain configuration also allows the MCU to control two or more motor driver devices without having to change the SDI data structure. If the MCU controls only one device in the chain, then the header bytes and the daisy chain configuration is not needed. The specific device datasheet describes the application case for communicating with a single peripheral device.

## 4 Application Examples

The following section contains additional application examples for using the daisy chain configuration. The list below overviews some key points to consider during the system design:

- The entire chain of devices must be written or read for each SPI transaction.
- The HDRx bytes must always contain the total number of devices in the chain. Each device counts the number of clock cycles until it receives the header byte. This is how each device finds the number of devices in the chain and its relative location.
- Only one register per device may be read or written per SPI transaction.

### Writing Devices Mid-Chain

In this example, there are 20 devices in the chain, but the microcontroller only needs to write to the 10th device as shown in Figure 4-1.

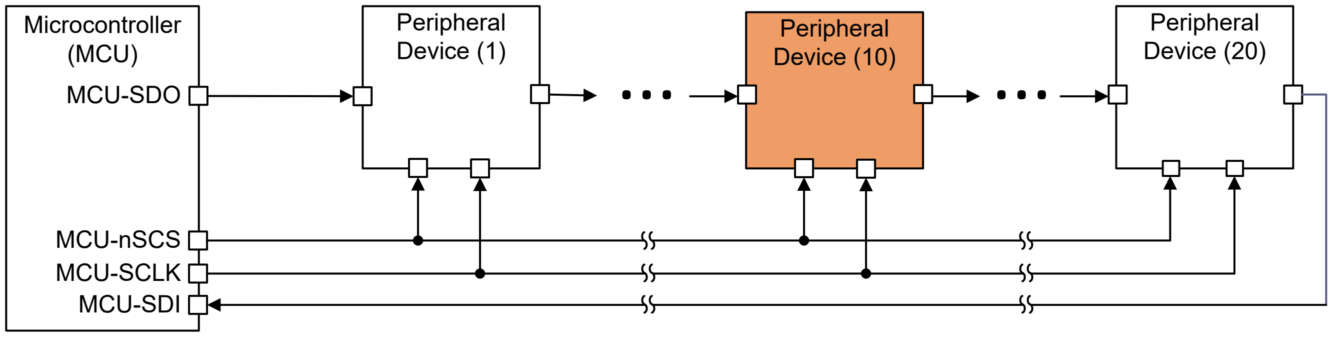

**Figure 4-1. SPI Daisy Chain Implementation Example for 20 Devices**

If no other devices need to update, then the R/W bit in the address bytes (A1-A9 and A11-A20) can be set to "read." The R/W bit of A10 will be set to "write." Each transaction will only write one register in device 10 as selected by the A10 byte. The data written to that address location is sent in the D10 byte. If the microcontroller needs to update multiple addresses in device 10, it will need to send the entire SPI transaction for each additional address update. Figure 4-2 shows an example of the SPI transaction for each address write to device 10.

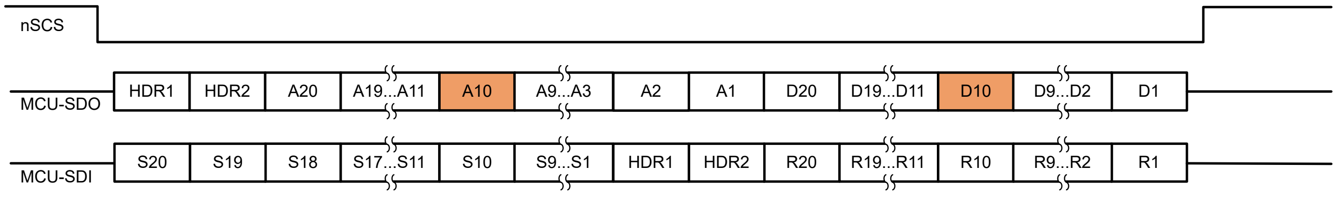

**Figure 4-2. SPI Daisy Chain Data Frame for 20 Devices**

The microcontroller can read or write one register in each device per SPI transaction. If multiple devices need updating, then the microcontroller can set the R/W bits in those addresses to "write."

### Length of Time for Each SPI Transaction

Sometimes SPI transaction time can be a critical constraint in an application with a long daisy chain. This section calculates the amount of time needed for each transaction. Figure 4-3 shows a block diagram for this application example. Table 4-1 shows example application constraints. Figure 4-4 shows the SPI transaction data frame for this example.

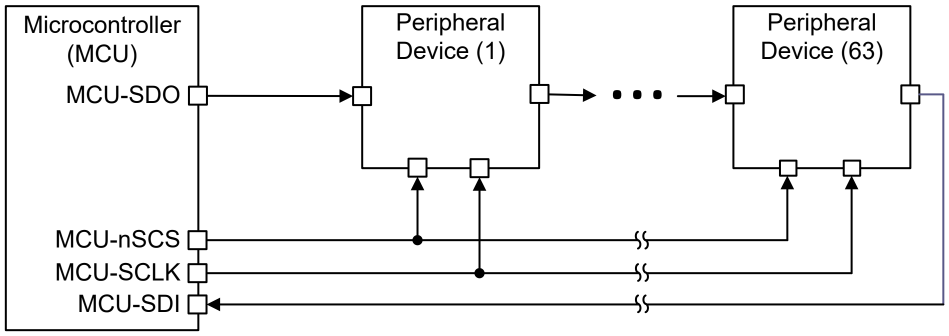

**Figure 4-3. SPI Daisy Chain Block Diagram for 63 Devices**

**Table 4-1. Parameters for SPI Timing Calculation**

| Parameter | Description | Example value |
|---|---|---|
| $N_{devices}$ | Number of devices | 63 |
| $f_{CLK}$ | SPI clock frequency | 5 MHz (bits/second) |

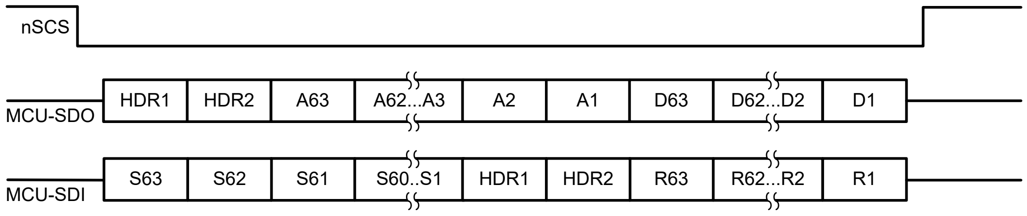

**Figure 4-4. SPI Daisy Chain Data Frame for 63 Devices**

Transmitting the bytes in the SPI frames takes the most time in the SPI transaction. Equation 1 calculates the total number of bits in each SPI transaction. Equation 2 calculates the total amount of time needed for all of the data in the chain. Equation 2 should be sufficient for most SPI timing calculations.

$$N_{bits} = N_{HDR1} + N_{HDR2} + N_{ADDRESSES} + N_{DATA} = 8 + 8 + 8*N_{devices} + 8*N_{devices} = 16 + 16*63 = 1{,}024 \text{ bits} \quad (1)$$

$$t_{bits} = N_{bits} / f_{CLK} = N_{bits}*(t_{(CLKH)} + t_{(CLKL)}) = 1{,}024 \text{ bits} / 5 \text{ MHz} = 0.2048 \text{ ms} \quad (2)$$

Equation 3 calculates the full timing required for each SPI frame, including the initial nSCS setup time ($t_{su(nSCS)}$) and final nSCS hold time ($t_{h(nSCS)}$). Figure 4-5 shows additional SPI timing parameters required by the SPI logic. The specific numbers in the equation are example values. Specific values depend on the particular peripheral device and can be found in the device datasheet.

$$t_{SPI\_FRAME} = t_{bits} + t_{su(nSCS)} + t_{h(nSCS)} = 0.2048 \text{ ms} + 100 \text{ ns} + 100 \text{ ns} = 0.2050 \text{ ms} \quad (3)$$

If multiple SPI frames need to be written in quick succession, $t_{(HI\_nSCS)}$ and $t_{dis(nSCS)}$ must be observed between each frame. Equation 4 calculates the full timing for each SPI transaction.

$$t_{SPI\_TRANSACTION} = t_{SPI\_FRAME} + t_{(HI\_nSCS)} + t_{dis(nSCS)} = 0.2050 \text{ ms} + 600 \text{ ns} + 30 \text{ ns} = 0.20563 \text{ ms} \quad (4)$$

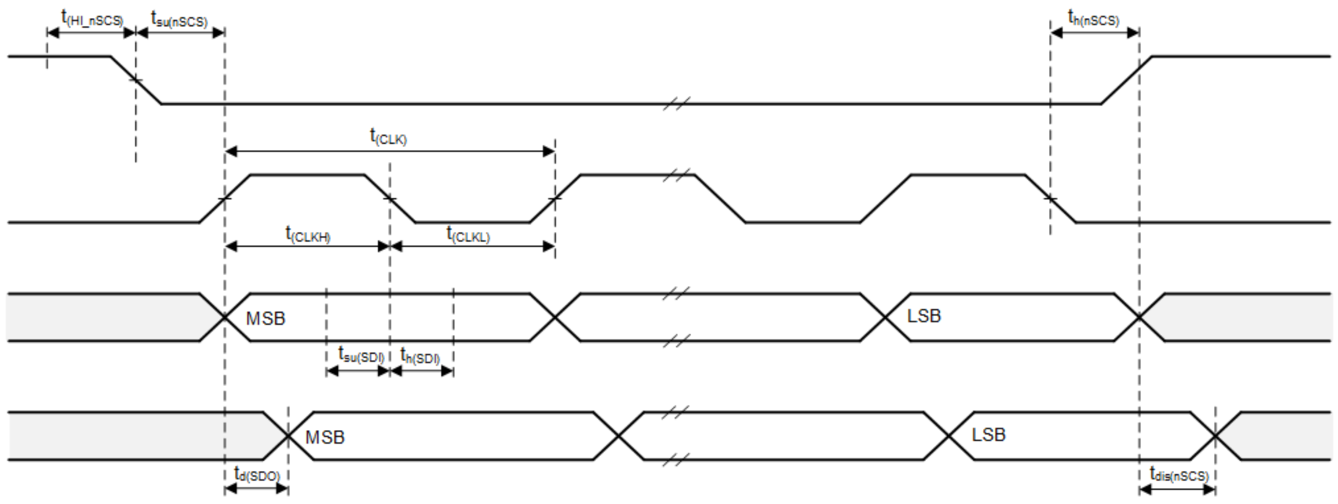

**Figure 4-5. SPI Frame Timing Diagram from DRV8889-Q1 Datasheet**

## Controlling More Than 63 Devices

To control more than 63 devices, the microcontroller will need separate GPIO for multiple nSCS lines. This allows the controller to support multiple daisy chains while minimizing overall GPIO pins needed for SPI communication. Figure 4 shows this. To communicate with a specific chain, that particular nSCS signal must be pulled low while the others remain high. Figure 4-6 shows an example of this.

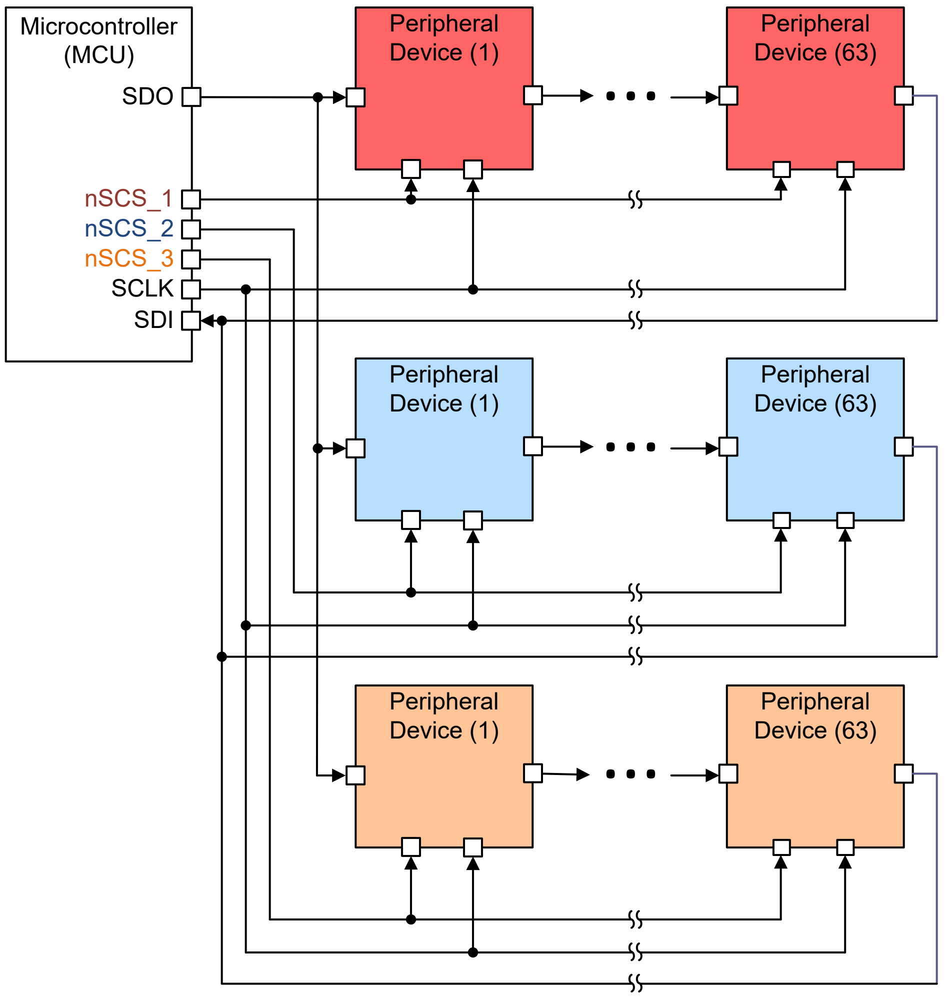

**Figure 4-6. SPI Block Diagram for Controlling More Than 63 Devices**

## 5 Summary

There are certain advantages of implementing daisy chain as described in this report. They include:

- Speed: Large number of SPI peripheral devices (up to 63) can be connected on a single SPI bus without having to reduce frequency of the SPI transaction.
- Same-frame response: For both read and write commands, each motor drive device responds on its SDO line (MCU-SDI line) with the current data at the given register address. This allows each device to update the MCU without needing additional read transactions.
- Robustness: Header bytes sent by the microcontroller return back to the microcontroller after going through the entire chain. This allows the microcontroller to continuously check for integrity of the chain connection.

Refer to the following datasheets for more information.

- *DRV8873-Q1 Automotive H-Bridge Motor Driver* data sheet
- *DRV89xx-Q1 Automotive Multi-Channel Half-Bridge Drivers with Advanced Diagnostics* data sheet
- *DRV8889-Q1, DRV8889A-Q1 Automotive Stepper Driver with Integrated Current Sense, 1/256 Micro-Stepping, and Stall Detection* data sheet
- *DRV8899-Q1 Automotive Stepper Driver with Integrated Current Sense and 1/256 Micro-Stepping* data sheet
- *DRV8434S Stepper Driver With Integrated Current Sense, 1/256 Microstepping, SPI Interface, Smart Tune Technology and Stall Detection* data sheet

# 6 Glossary

## 6.1 Nomenclature Used in this Document

The following acronyms and initialisms are used in ths document:

| | |
|---|---|
| **MCU** | Microcontroller unit |
| **SPI** | Serial peripheral Interface |
| **MSB** | Most significant bit |
| **LSB** | Least significant bit |
| **SDO** | Serial data out |
| **SDI** | Serial data in |
| **GPIO** | General purpose input-output |

For a more comprehensive list of terms, acronyms, and definitions, refer to the *TI Glossary*.