Implementing a Step-Direction Interface-based Stepper Motor Controller using SmartFusion2 Devices

Purpose

This application note describes the Stepper Motor interface that controls a two-phase bipolar stepper motor using a SmartFusion®2 system-on-chip (SoC) field programmable gate array (FPGA) and companion stepper motor driver chip, TMC-262. The control blocks are distributed between the SmartFusion2 microcontroller subsystem (MSS) and the FPGA fabric.

This application note facilitates the designer in quickly designing a stepper motor controller interface for an external driver using the *StepperWrapper* software library and IP blocks or in customizing them for a specific application.

Introduction

The stepper motors are electromechanical actuators that provide precise positioning of the rotor shaft. They are capable of moving to a specified position and holding that position irrespective of the load torque. This capability makes the stepper motors to be used in optics, medical instruments, factory automation, and industrial equipment.

The stepper drive consists of a controller, driver, and stepper motor. The controller provides a direction signal and step pulses, while the driver converts these signals into actual electrical power and supplies them to the motor. The stepper motor moves in steps, each step covering one step angle, which can be described as the rotor displacement corresponding to one step pulse.

The stepper motor can be run in the following modes:

- Full-step
- Half-step
- Microstep

In Full-step mode, the motor phase windings are excited in a way that the rotor moves by one step angle. The angle increment corresponding to the step angle in Full-step mode depends on the motor construction. It is available in the TMC262 Datasheet as step number, which is defined as the number of steps required by a stepper motor to complete one revolution.

In Half-step mode, the phase windings are excited in a way that the rotor moves by an angle equal to half the step angle. This allows for positioning the rotor more precise compared to the Full-step mode. The number of steps required to complete one revolution is twice as that in Full-step mode.

In Microstep mode, the phase currents are modulated in a way that the step angle is divided into smaller steps and allows for even finer positioning of the rotor.

The number of steps required to complete one full rotor revolution is calculated as:

$$Number\ of\ Steps\ per\ Revolution = Step\ Number * Microstep\ Resolution$$

EQ1

To control the speed at which the motor runs to reach the required position, the delay between successive step pulses must be modulated as described in Direction Interface Stepper Drivers section.

The SmartFusion2 stepper motor control hardware platform can control up to six stepper motors in two control modes:

- Continuous (rotation)
- Finite-step

You can also configure:

- Step resolution
- Current drain
- Number of steps
- Motor direction
- Motor speed.

In Continuous-step mode, the motor runs steadily at a set speed until it is interrupted manually. In Finite-step mode, the motor moves through a specified number of steps at the set speed, after which it continues to hold this position and resists motion in either direction until it is interrupted manually.

Figure 1 shows running multiple motors by connecting multiple driver cards to the hardware platform where a single controller controls upto six motors.

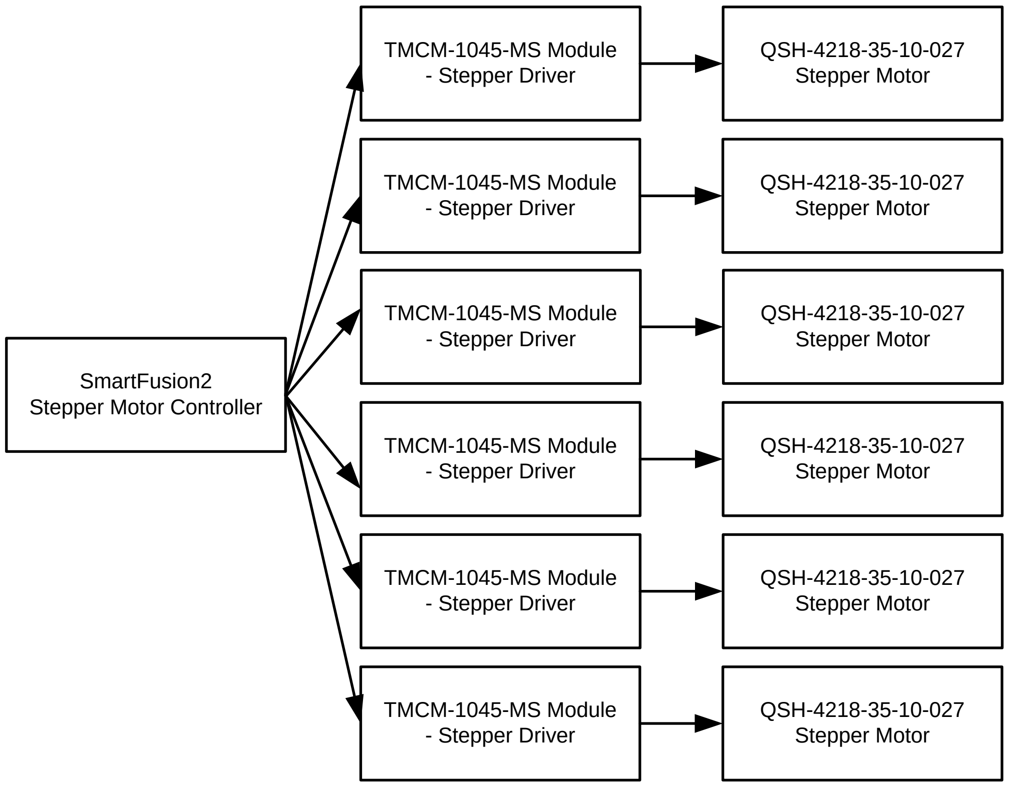
SmartFusion2
Stepper Motor Controller
TMCM-1045-MS Module - Stepper Driver
TMCM-1045-MS Module - Stepper Driver
TMCM-1045-MS Module - Stepper Driver
TMCM-1045-MS Module - Stepper Driver
TMCM-1045-MS Module - Stepper Driver
TMCM-1045-MS Module - Stepper Driver
QSH-4218-35-10-027 Stepper Motor
QSH-4218-35-10-027 Stepper Motor
QSH-4218-35-10-027 Stepper Motor
QSH-4218-35-10-027 Stepper Motor
QSH-4218-35-10-027 Stepper Motor
QSH-4218-35-10-027 Stepper Motor

Figure 1 · SmartFusion2 Stepper Motor Driver for Multiple Stepper Motors

Step Direction Interface

Stepper motor drivers, such as the TMC-262 driver, have a step - direction interface. This interface requires only two signals from the controller to run the motor. A controller provides these signals to the driver which in turn switches the power MOSFETs to make the motor turn by one step angle. The step signal is usually a pulse, which makes the motor respond by turning the stepper by one step. The direction signal determines the direction of this motion. To make the motor move by a fixed number of steps at a particular speed, the number of pulses generated must be equal to the number of steps required, and the speed is achieved by modulating time between the successive step pulses.

Calculating Number of Wait Cycles Between Successive Step Pulses

The controller uses a counter to generate the time delays between pulses. The following steps describe the algorithm to calculate the number of wait cycles that should be counted by the controller for a specified speed.

1. The speed of the motor in number of revolutions per second is N_{RPS} or $\frac{N_{RPM}}{60}$; where N_{RPM} is the motor speed in revolutions per minute; the time taken for each revolution is $\frac{60}{N_{RPM}}$.
2. The Step number of the motor indicates the number of pulses required to run the motor through one complete revolution in Full-step mode. Therefore, the time between two successive pulses is:

$$\frac{\frac{60}{N_{RPM}}}{Step\ Number} = \frac{60}{(N_{RPM}) * (Step\ Number)}$$

EQ2

3. If the clock frequency of the controller is f_{clk} (to the SmartFusion2 device and the TMC-262 driver), the time period of this clock is $\frac{1}{f_{clk}}$.
4. The number of clock cycles to count before sending the successive strobe signals is:

$$\frac{\frac{60}{(N_{RPM})*(Step\ Number)}}{\frac{1}{f_{clk}}} \text{ or } \frac{f_{clk}*60}{(N_{RPM})*(Step\ Number)}$$

EQ3

5. For Microstep mode, each step (θ) is divided into smaller steps (Δ). Each step pulse will now correspond to Δ. For a given number of wait cycles between successive step pulses, the motor will run at N_{RPM} in Full-step mode, but at $\frac{N_{RPM}}{2}$ in Half-step mode, and $(\frac{N_{RPM}}{\text{Number of microsteps}})$ in the Microstep mode. To keep the motor speed consistent, the number of wait cycles is divided by the Number of microsteps. The modified formula for N_{RPS} is:

$$\frac{f_{clk} * 60}{(N_{RPM}) * (Step\ Number) * (Number\ of\ Microsteps)}$$

EQ4

   If it is not divided, the speed of the motor becomes slower than N_{RPS}.

6. The TMC-262 driver requires the number of microsteps to be programmed in using a variable, MRES. Table 3 shows the Microstep resolution and their respective MRES value.
7. Hence, the number of wait cycles between successive strobe pulses is:

$$\frac{f_{clk} * 60 * 2^{(MRES-8)}}{(N_{RPM}) * (Step\ Number)}$$

EQ5

Block Diagram

Figure 2 shows the signal connections between various blocks used in the stepper motor control design. This design uses an advanced peripheral bus (APB) interface to communicate between the MSS and the FPGA fabric.

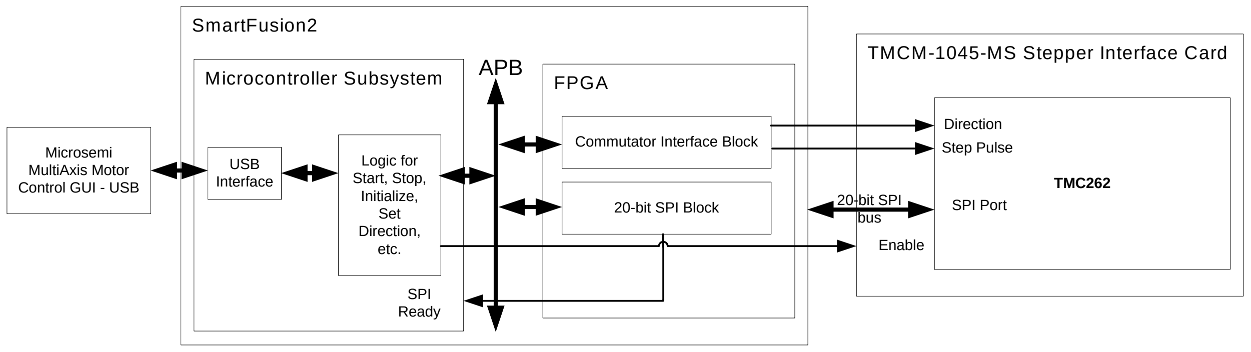
SmartFusion2
Microcontroller Subsystem
APB
FPGA
TMCM-1045-MS Stepper Interface Card
Microsemi MultiAxis Motor Control GUI - USB
USB Interface
Logic for Start, Stop, Initialize, Set Direction, etc.
Commutator Interface Block
20-bit SPI Block
Direction
Step Pulse
TMC262
SPI Port
20-bit SPI bus
Enable
SPI Ready

Figure 2 · System-Level Block Diagram of SmartFusion2 Stepper Motor Controller

Functional Description

This section lists the inputs that are required for the blocks and descriptions of the hardware and software blocks used in controlling the stepper motor.

System Inputs

The following data is required to operate a stepper motor using the Step pulse interface:

- **Current scale** – The quantity by which the motor current is amplified. This is specified in steps 0 through 15, with 0 referring to the lowest current and in turn, lowest torque. 15 referring to the highest current and in turn, highest torque. For more information, refer to the TMC262 Datasheet.
- **Speed** – The number of revolutions of the motor in a given period of time.
- **MRES** – The microstep resolution, refer to Table 3.
- **Direction** – The direction of the motor - clockwise or anticlockwise.
- **Step Number of the motor**: The number of steps required for the motor to complete one full revolution.
- **Number of steps** – The number of steps through which the motor runs, when the controller is running in Finite-step mode. When in Continuous mode, this does not apply as the motor runs till it is expressly stopped manually.

Hardware Blocks (FPGA Blocks)

The hardware blocks use the parameters listed in System Inputs section as inputs to generate the step pulses and the direction signal. Although the step-direction interface only requires two signals to run the motor, several other parameters should be configured on the TMC262 driver. This is configured using the SPI communication block. This section describes:

- Commutator Block
- SPI Block

Commutator Block

Using the inputs listed in System Inputs section, the commutator block generates the direction signal and step pulses and sends them as input to the stepper driver. The time duration between the successive step pulses decides the motor speed.

This time duration (number of clock cycles to wait before sending successive strobe signals) is calculated by using EQ5.

For Continuous (stepping) mode, the required speed is given as input to the system and the step pulses are generated until the commutator block is reset.

For Finite-step mode, the number of steps through which the motor must run is also specified to the block (EQ1). This operation makes the motor to run through a finite number of steps by generating an equivalent number of pulses and stop. After the motor stops, it remains energized and can resist motion in both the directions. Figure 3 shows a logical representation of the commutator block algorithm.

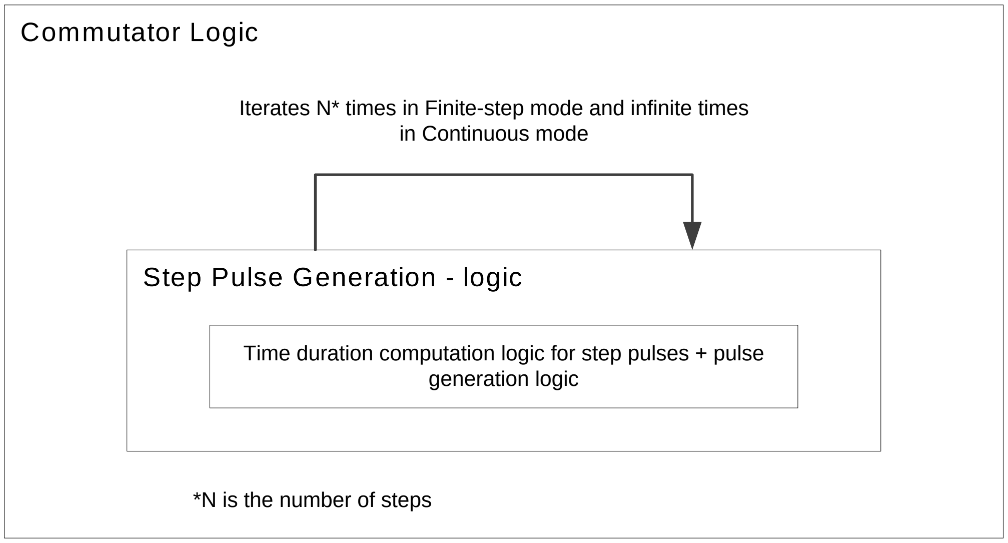
Commutator Logic
Iterates N* times in Finite-step mode and infinite times in Continuous mode
Step Pulse Generation - logic
Time duration computation logic for step pulses + pulse generation logic
*N is the number of steps

Figure 3 · Commutator Block

Figure 4 shows the pseudo code representation of the commutator block.

```
Input: mode, FiniteStepCount, speed, Resolution
Output: StepPulse

PROCESS: Step Pulse Generation
IF ((mode IS Continuous) OR (FiniteStepCount > 0))
{
IF(delay IS 0) THEN
{
TOGGLE StepPulse;
delay <- COMPUTE delay;
}
ELSE
{
DECREMENT delay;
}
}
```

Figure 4 · Pseudo Code Representation of the Commutator Block

SPI Block

The TMC-262 driver has several registers that must be configured through a serial peripheral interface (SPI) communication channel. During the transfer, the TMC-262 driver sends back a frame of data with parameters such as motor stall, over temperature that can be used to ensure safe operation. A 20-bit SPI communication block is used to transfer data between the SmartFusion2 device and the TMC-262 driver. This block also provides a ready signal to indicate that the block is ready to transmit the data. This signal is useful to a controlling block (for example, MSS) when continuous data transmissions are required.

Figure 2 shows the commutator block and SPI block signals that are directly connected to the external stepper driver module (TMCM-1045-MS). The MSS sends inputs to these blocks through an APB interface. On the MSS, the hardware abstraction layer provides application program interface (API)s to communicate with the fabric through the APB interface.

MSS Blocks (Software Library)

The MSS is responsible for logic to:

- Initialize, start, and stop the motor
- Set the motor parameters

The MSS also communicates with the GUI on a Host PC and other blocks on the FPGA fabric. The MSS communicates with the GUI through a human interface device (HID) class, high speed interrupt based USB communication. The interface uses direct memory access (DMA) and requires a minimum of 125 µs between consecutive transfers. The MSS communicates with the fabric through an APB interface.

StepperWrapper Library

The *StepperWrapper* library provides APIs for the stepper motor drive to

- Initialize, start, stop and
- Change motor configuration parameters

The *USBMsgHandler* library calls these functions. *USBMsgHandler* library is used to communicate with the GUI through the USB port. The StepperWrapper library can also be called independently in the firmware.

Table 1 shows a list of APIs provided by the *StepperWrapper* library.

Table 1 · APIs in the StepperWrapper Library

API Name	API (Prototype)	Function

`StepperInit`	`void StepperInit(`
	`mss_gpio_id_t Stepper_slot,`
	`mss_gpio_id_t FAB_SPI_SLOT,`
	`addr_t AXIS_BASE_ADDR,`
	`uint32_t FAB_SPI_RDY_MASK,`
	`Stepper_type *Stepper_ptr,`
	`uint32_t steps,`
	`uint8_t speed,`
	`uint8_t mres,`
	`uint8_t dirn,`
	`uint8_t Current_fil,`
	`uint8_t Mode`
	`);`
		Initializes a stepper motor structure with:
		• The GPIO pins to:
		- Enable the stepper module
		- Receive the SPi ready signals
		• The APB base address
		• Other stepper parameters like resolution, direction, speed and so on

`StepperStart`	`void StepperStart(`
	`Stepper_type *Stepper_ptr`
	`);`
		Loads the number of steps in the Finite-step mode, or the continuous step mask in the Continuous mode and runs the motor.

`StepperStop`	`void StepperStop(`
	`Stepper_type *Stepper_ptr`
	`);`
		Disables the stepper module and puts this axis in Finite-step mode with zero steps.

API Name	API (Prototype)	Function
`StepperSetParam`	`void StepperSetParam( Stepper_type *Stepper_ptr, uint8_t Speed, uint8_t Mres, uint32_t Steps, uint8_t current);`	Sets parameters such as speed, MRES, number of steps, and motor current.
`StepperSetDirn`	`void StepperSetDirn( Stepper_type *Stepper_ptr, uint8_t direction);`	Sets the direction of the stepper. 0: Clockwise 1: Anticlockwise
`SPI_tx_262`	`static inline void SPI_tx_262( addr_t reg_addr, uint32_t value );`	Sends data from the MSS to the stepper driver chip through the fabric SPI block. The MSS sends the data to be transmitted with a start transmission bit. After a fixed period of time, this bit is cleared so that the data is not sent repeatedly.

Table 2 shows the default values that the `StepperInit()` writes to the TMC262 registers automatically.

Table 2 · Default Values for TMC262 Driver Registers

Register	Default Value	Function
DRVCTRL	0x00000 + MRES	Drive Control register (last 4 bits represent MRES, refer to Table 3)
CHOPCONF	0x8B032	Chopper Configuration register
SMARTEN	0xA8160	CoolStep Configuration
SGSCONF	0xD4F00 + Current_scaling	StallGuard Configuration
DRVCONF	0xEF050	Driver Configuration

Table 3 shows the values of MRES corresponding to the various microsteps.

Table 3 · MRES Values for Microstep Resolutions

Resolution (Microsteps)	MRES Value
256	0
128	1
64	2
32	3
16	4
8	5
4	6
Half Step (2)	7
Full Step (1)	8

Note: The speed and direction of the motor and the MRES value can be configured while the motor is running. Current scaling can only be modified when the motor stops.

Setting up the Design

Table 4 lists the hardware and software requirements of the reference design.

Table 4 · Reference Design Requirements

Reference Design Requirements and Details	**Description**
Hardware Requirements	
• SmartFusion2 Development Kit – 12 V adapter (provided along with the kit) – FlashPro4 programmer (provided along with the kit) – USB A to Micro-B cable (provided along with the kit) • TMCM-6930-MS Interposer Card (provided with the motor control kit) • TMCM-1045-MS Stepper Daughter Card (provided with the motor control kit) • Power Supply (provided with the motor control kit)	Rev C or later
Host PC or Laptop	Any Windows 7 Operating System
Software Requirements	
Host PC Drivers (provided along with the design files)	USB Drivers (Provided with the GUI)
Microsemi application	Microsemi Motor Control GUI 2.4 or later

For information on setting up the hardware, connecting it to the GUI and running the design, refer to the SmartFusion2 Six-Axis Motor Control Development Kit Demo Guide.

Figure 5 shows the stepper Daughter card connected to a stepper motor.

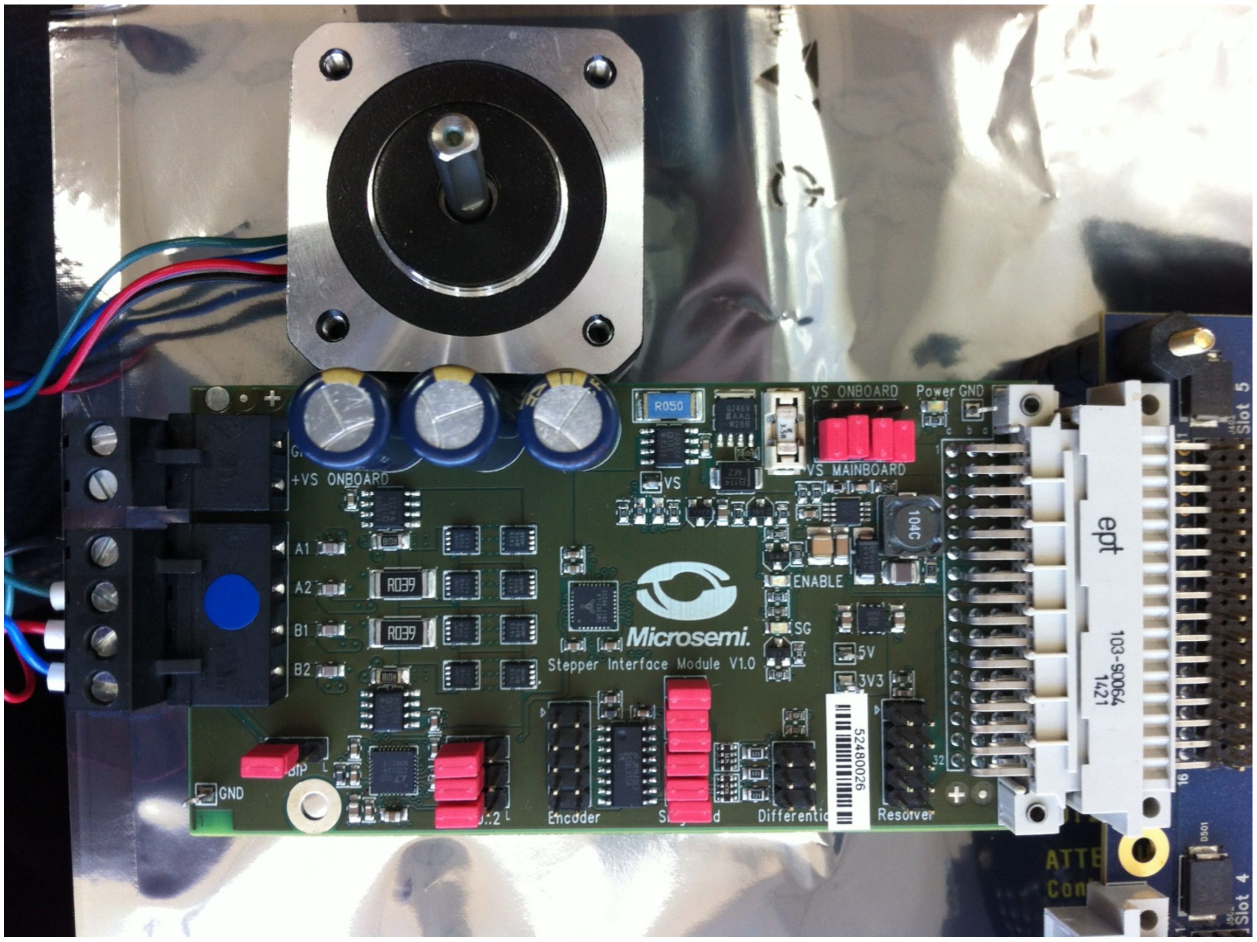

Figure 5 · Stepper Motor Driver Daughter Card connected to a Stepper Motor

Summary

This application note describes the basic step-direction interface control method of a stepper motor drive using SmartFusion2 and the TMCM-1045-MS stepper driver card. The functional operation of the stepper motor drive is described, and the default settings are discussed.

References

1. TMC262 Datasheet
   http://www.trinamic.com/tmctechlibcd/integrated_circuits/TMC262/TMC262_datasheet.pdf
2. Stepping Motors and their Microprocessor controls, Takashi Kenjo, 1994
3. T.C. Chin, D.P. Mital and M.A. Jabbar, A Stepper Motor Controller, 1988.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1(949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996