AN2820
Application note

Driving bipolar stepper motors using a medium-density STM32F103xx microcontroller

Introduction

This application note describes how to achieve compact size, high speed and low cost with less resources when driving bipolar stepper motors using the medium-density STM32F103xx family of cortex-M3-based microcontrollers. It presents a simple method to implement the full-step and half-step operating modes to control stepper motors.

A stepper motor is an electromechanical device that converts electric pulses into discrete mechanical step motions. The shaft of a stepper motor rotates in discrete steps when electric command pulses are applied to it in the proper sequence. Stepper motors are a good choice whenever controlled movement is required. They are particularly useful in applications where rotation angle, speed, position and synchronism control is needed.

The major advantages of stepper motors are that they need no feedback devices, they are inexpensive relative to other motion control systems, they show an excellent low-speed torque and they are stable. Many stepper motor applications could benefit from the power, features and flexibility of the STM32F10xxx devices. They include robotics controllers, turning machine tools, video cameras and other precise shaft-positioning-control environments.

Moreover, the high performance of the STM32F10xxx microcontrollers offers designers the possibility of driving stepper motors reliably with low computing requirements from the controller.

This application note gives a simple method to control stepper motors following a typical run profile. The user can choose the operating mode (full-step or half-step), the rotation sense of the motor (clockwise or counter clockwise) and the control current mode (fast or slow decay).This method uses the medium-density STM32F103xx and the L6208 fully integrated two-phase stepper motor driver. It is the cheapest and simplest way of obtaining minimum CPU load.

Contents

List of tables

List of figures

1 Stepper motor basics

1.1 Stepper motor types

There are three basic stepper motor types:

- Variable reluctance
- Permanent magnet
- Hybrid

The variable-reluctance (VR) motor type has been around for a long time. It is probably the easiest to understand from a structural point of view. This type of motor consists of a soft-Iron multitoothed rotor and a wound stator. When the stator windings are fed with DC current, the poles become magnetized. Rotation occurs when the rotor teeth are attracted to the energized stator poles.

The permanent-magnet (PM) motor type has permanent magnets added to the motor structure. The rotor does not have teeth like in VR motors. Instead, it is magnetized with alternating North and South poles situated in a straight line parallel to the rotor shaft. These magnetized rotor poles provide increased magnetic flux intensity that gives the PM motor improved torque characteristics compared to those of the VR type.

The hybrid (HB) motor type exhibits a better performance in terms of step resolution, torque and speed. This type of motor combines the best features of both the PM and VR stepper motor types. The rotor is multitoothed like in VR motors and contains an axially magnetized concentric magnet around its shaft. The teeth on the rotor provide an even better path which helps guide the magnetic flux to preferred locations in the air gap. This feature increases the detent, holding and dynamic torque characteristics of the motor compared to both the VR and PM motor types.

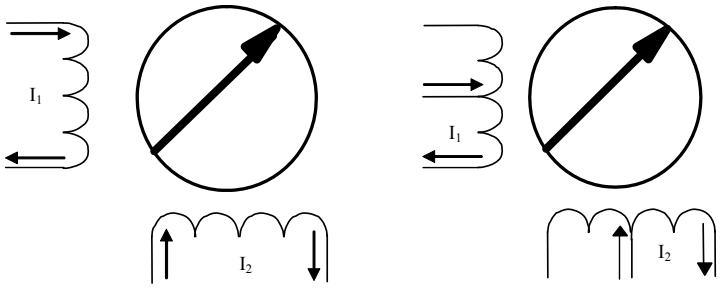
Stepper motors are available in either bipolar or unipolar windings.

Unipolar stepper motors have two identical coils that are not connected electrically and both have center tap. The flux is reversed by powering either end of the bifilar coil pair with the center taps made common. The advantage of unipolar stepper motors is that they need only one changeover switch. However, they require a double bifilar winding, which means that for a given bulk factor the wire is thinner and the resistance much higher.

Bipolar stepper motors are the same as unipolar motors except that the coils do not have center taps. For a bipolar motor, a H-bridge can be used to reverse the polarity of the windings and thus the flux. The advantage of bipolar stepper motors is that they use only one winding with a good bulk factor (low winding resistance).

Unipolar motors are still popular because their drive circuit appears to be simpler when implemented with discrete devices. With the integrated circuits available today, however, bipolar motors can be driven with no more components than unipolar motors.

Figure 1. Stepper motor types



1.2 Drive signals

A direct current motor runs by itself when supplied with voltage, whereas a stepper motor needs commutation signals.

Different modes can be used to drive stepper motors, including the full-step and half-step modes. The full-step mode is normally used by full-step motor drives. In this mode, both phases are always supplied and the motor has a full rated torque. This control mode requires only four rectangular signals that could also be generated by PWM (fixed duty cycle within a one-step duration). Depending on the leading phase, the motor axis rotates clockwise or counter clockwise.
The half-step mode is a bit more complicated. If half-step driving is used, the motor advances half a step after each clock pulse, thus obtaining a higher position resolution and reducing instability.
In both modes, however, the signals are all related to each other in a definite way so that they can be generated using standard logic. A good logic implementation may however be quite expensive and it would be better to use an application-specific integrated circuit. In general, specific integrated circuits contain an internal translator circuit controlled by step-and-direction inputs. The IC (Integrated circuit) motor controller allows operation in three modes only: full-step two phases on, half-step and wave drive. This type of IC needs four signals to the controller which are provided by a microcomputer or another dedicated controller chip. In addition, this solution requires a power stage and a microprocessor to generate the different control signals to each motor.

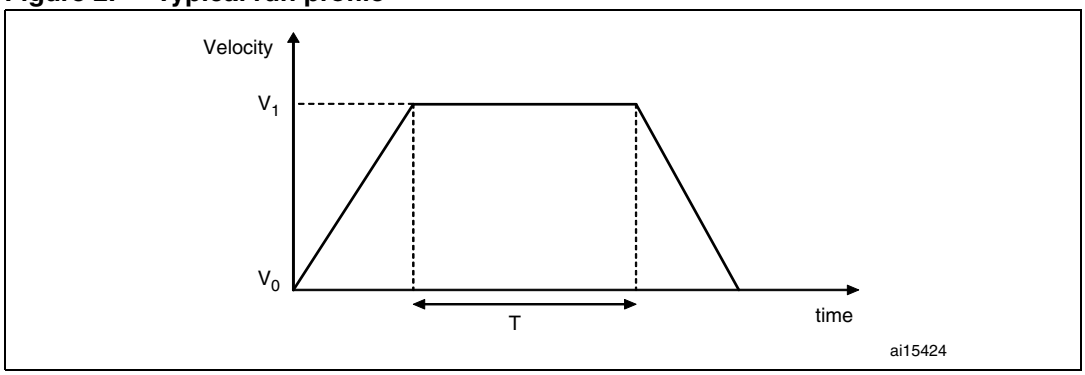
Some known applications need to be able to vary the stepper motor speed in order to generate a determined velocity profile (for example: a typical run profile). In this case, the best way to drive the stepper motor, in each of the three possible modes, is the software solution using a microcontroller circuit that could replace any other controller. In a microcontroller-based application, it is possible to use software and PWM timers, which removes the need for an external controller.

Using this approach, it is possible to realize a high-speed application that uses minimum hardware and creates very low microcontroller CPU load.

2 Driving a bipolar stepper motor using a medium-density STM32F103xx

This section describes how to drive a stepper motor in full- and half-step modes with the medium-density STM32F103xx microcontroller, according to the typical run profile illustrated in *Figure 2*.

Figure 2. Typical run profile



According to *Figure 2*, the stepper motor accelerates from the velocity V_0 up to velocity V_1 that has to be maintained for the period of time T. After this period, the motor decelerates down to the initial velocity V_0. The obtained velocity profile shows the same slope during the acceleration and deceleration.

2.1 Hardware development

As said in the previous section, it is necessary to correctly choose the microcontroller and the IC to implement the best solution to drive the stepper motor. In this application, a medium-density STM32F103xx device and the L6208 DMOS driver have been selected.

2.1.1 STM32F10xxx features used to drive a bipolar stepper motor

The medium-density STM32F103xx has a set of peripherals ideally suited to driving stepper motors. These peripherals include three standard (general-purpose) timers (TIM2, TIM3 and TIM4) with an internal clock frequency of up to 72 MHz and four 16-bit independent channels for high-resolution capture. Moreover, the STM32F103xx peripherals include an advanced-control timer (TIM1) with an internal clock frequency of 72 MHz and four 16-bit high-resolution capture channels. This timer has three channels (CH1, CH2 and CH3) able to generate three complementary signals, and one independent channel (CH4). These timer channels are able to generate signals in PWM or in output compare modes. The PWM and the output compare features are required to generate a regular step clock input for the L6208 to control the stepper motor. This application aims at controlling the stepper motor speed by using TIM2 in output compare toggle mode with a constant duty cycle of 50% and a variable frequency. The latter is used to vary the motor speed in accordance with the typical velocity profile. When changing the clock frequency, a minimum CPU load is required to prevent the driven stepper motor from stalling.

The DMA controller is used to transfer the timer periods, leading to a sped-up CPU operation because, via the DMA controller, the device directly transfers periods from

memory to timers without any CPU intervention. The DMA controller of the STM32F10xxx has seven independently configurable channels with three event flags for each channel: one for DMA Half Transfer, another for DMA Transfer complete and the last one for DMA Transfer Error. In this application, the update event DMA request and the DMA Transfer complete interrupt are used to control the transfer of the periods. For all the timers, the update event request is present in different DMA channels. Thus, the advantage of this choice is that all the timers with their DMA requests can work at the same time to drive different stepper motors.

The SysTick is used to maintain the maximum stepper motor speed during the period T. This flexible system timer allows the generation of an interrupt each time the programmed time base is reached.

The medium-density STM32F103xx has eighty GPIOs that can be configured as alternate function push-pull. These I/Os are able to control the rotation sense of the stepper motor (clockwise or counter clockwise), the step mode (full- or half-step), the decay mode (slow or fast) and the L6208 Chip Enable signal. (Refer to the STM32F10xxx reference manual for more details about the medium-density STM32F103xx features).

In summary, with its peripherals, the medium-density STM32F103xx can drive nineteen stepper motors at the same time in the full- and half-step modes with minimum CPU load. The hardware requirements of this solution are an IC that integrates a basic H-bridge circuit for each winding in the power stage, and a centralized logic mainly used for phase generation.

2.1.2 L6208 DMOS driver for bipolar stepper motor

The drive circuits used for bipolar stepper motors are more complex because bipolar motors do not have a center tap on their windings. To reverse the direction of the field produced by a motor winding, it is therefore necessary to reverse the current through the winding by using a H-bridge circuit. There are a lot of integrated H-bridge drivers on the market. The driver selected for this application is the L6208, a DMOS fully integrated stepper motor driver with non-dissipative overcurrent protection used to drive two-phase bipolar stepper motors. It includes a dual DMOS full bridge, two fixed off-time PWM current controllers (one for each of the bridges) that perform the chopping regulation and a phase sequence generator to generate the stepping sequence.

Figure 3 presents the block diagram of the L6208 IC. With reference to this figure, the main L6208 inputs are:

- **EN logic input:** it is the Chip Enable input. A low logic level switches off all power MOSFETs of both Bridge A and Bridge B. This pin is also connected to the collector of the overcurrent and thermal protection circuits to implement overcurrent protection.
- **CLOCK logic input:** it is the step clock input. The state machine makes one step on each rising edge.
- **CW/$\overline{\text{CCW}}$ logic input:** it selects the sense of rotation. A high logic level sets the clockwise sense, whereas a low logic level sets the counter clockwise sense.
- **HALF/$\overline{\text{FULL}}$ logic input:** it is the step mode selector. A high logic level sets the half-step mode, whereas a low logic level sets the full-step mode.
- **CONTROL logic input:** it is the decay mode selector. A high logic level sets the slow decay mode, whereas a low logic level sets the fast decay mode.
- **VREF_A and VREF_B analog inputs:** they are the bridge A and bridge B source pins, respectively.

Figure 3. L6208 block diagram

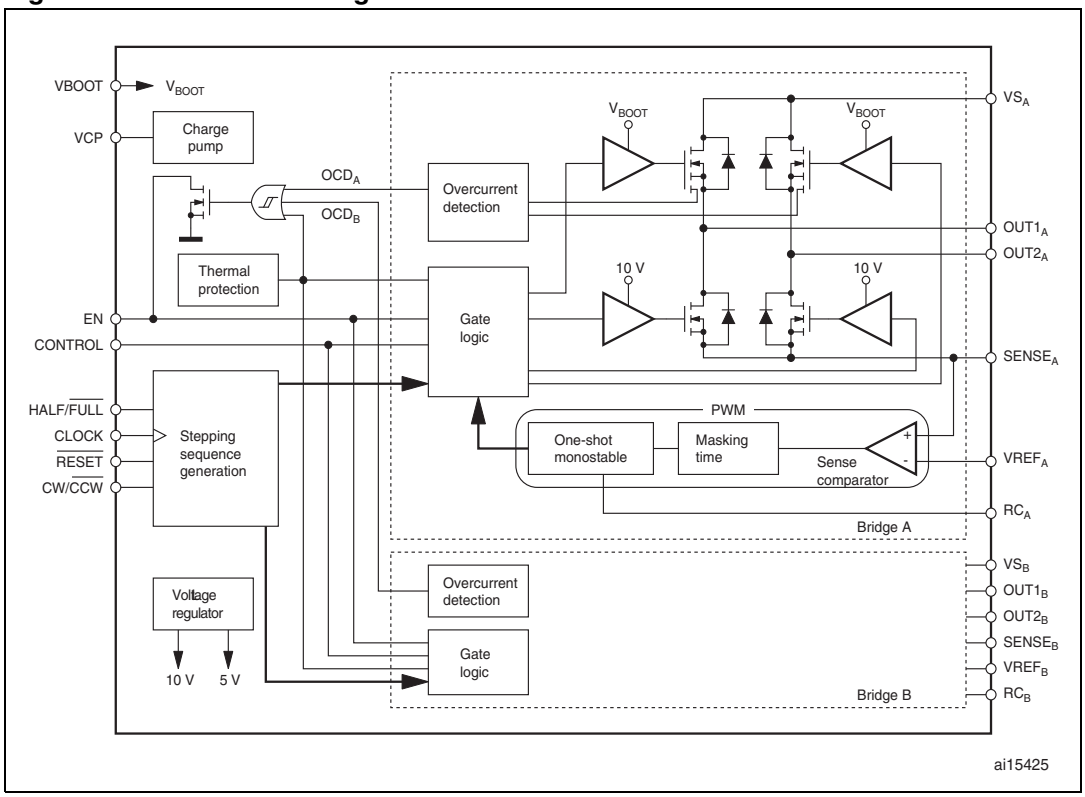


Figure 4 shows a typical bipolar stepper motor driver application using the L6208 driver. It also shows the different pins of the used package, which is the PowerDIP24/SO24. A high-quality ceramic capacitor in the range of 100 to 200 nF should be inserted between the power pins (V_{SA} and V_{SB}) and ground close to the L6208 to improve high-frequency filtering on the power supply and reduce the high-frequency transients generated by switching. The capacitor connected between the EN input and ground sets the shutdown time when an overcurrent is detected. The two current-sensing inputs ($SENSE_A$ and $SENSE_B$) should be connected as close as possible to the sense resistors in the layout. The sense resistors should be non-inductive resistors to minimize the di/dt transients across them. To improve noise immunity, it is preferable to connect unused logic pins (except for EN) to 5 V (high logic level) or to GND (low logic level). It is recommended to keep power ground and signal ground separated on the PCB.

Figure 4. L6208 typical application

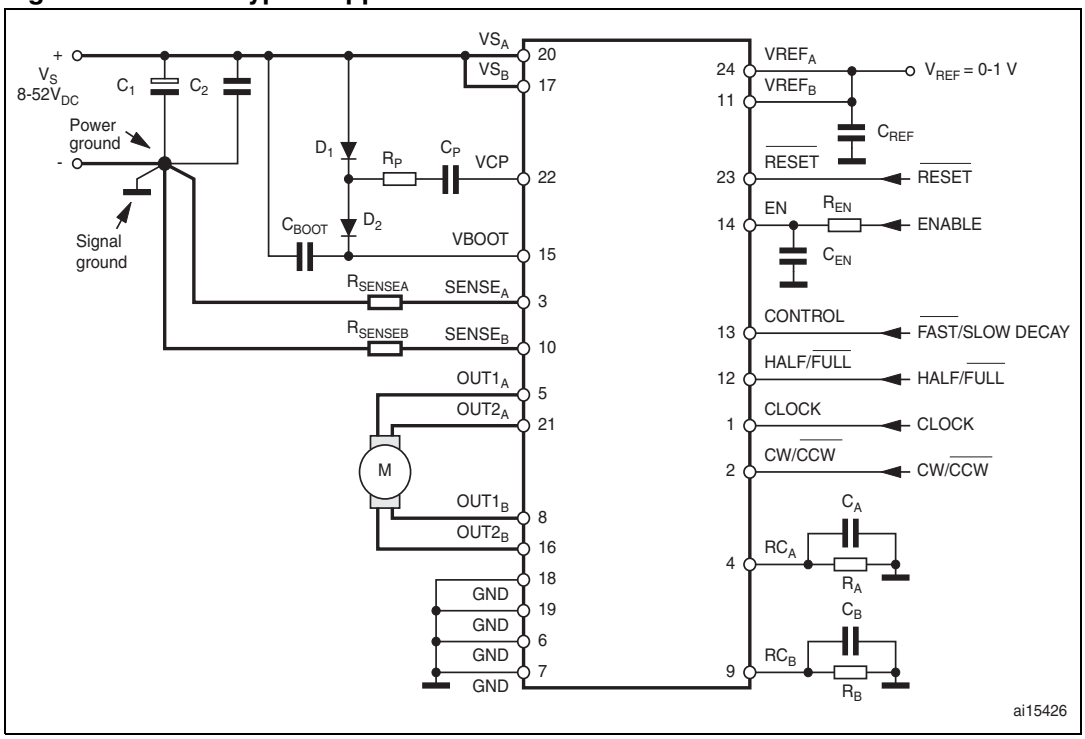


Table 1 gives the electrical characteristics of the L6208 driver.

Table 1. Electrical characteristics of L6208

Parameters	MIN	MAX	Unity
Supply voltage (V_s)	8	52	V
Output current (I_{OUT})		2.8	A
Switching frequency (f_{SW})		100	kHz
Operating Junction temperature (T_j)	–25	+125	°C

The L6208 includes a fixed off-time PWM current controller for each of the two bridges. The current control circuit senses the bridge current by sensing the voltage drop across an external sense resistor connected between the source of the two lower power MOS transistors and ground.

Current from each motor winding is flowing through the corresponding sense resistor, causing a voltage drop that is used, by the logic, to control the peak value of the load current. Two issues must be taken into account when choosing the value of the sense resistors, R_{SENSE}:

- Sense resistors dissipate power and give rise to potentially dangerous negative voltages on the SENSE pin when the current starts flowing again. For this reason the resistance of this component should be kept low.
- The voltage drop across R_{SENSE} is compared with the reference voltage (on V_{ref} pin) by the internal comparator. The lower the R_{SENSE} value, the higher the peak current error. This is due to noise on the V_{ref} pin and to the input offset of the current sense comparator: too small R_{SENSE} values have to be avoided.

A good tradeoff consists in calculating the sense resistor value so that the voltage drop corresponding to the peak current through the load (I_{peak}) is of about 0.5 V:

$R_{SENSE} = 0.5\ V / I_{peak}$.

The sense resistor must mandatorily be of the non-inductive type in order to avoid dangerous negative spikes on the SENSE pins.

Current control modes: fast and slow decay modes

The CONTROL input is used to select the behavior of the bridge during the off time. When the CONTROL pin is low, the fast decay mode is selected and both transistors in the bridge are switched off during the off time. When the CONTROL pin is high, the slow decay mode is selected and only the low-side transistor of the bridge is switched off during the off time.

Stepping sequence generation: half- and full-step modes

The phase sequence generator is a state machine that provides the phase and enables inputs for the two bridges to drive a stepper motor in either full-step or half-step mode. Two full-step modes are possible: the normal drive mode where both phases are on at each step and the wave drive mode where only one phase is on at a time. The drive mode is selected by the HALF/$\overline{FULL}$ input. A rising edge of the CLOCK input causes the state machine to move forward to the next state. The sense of rotation is set by the CW/$\overline{CCW}$ input. The $\overline{RESET}$ input resets the state machine to Home state (State 1).

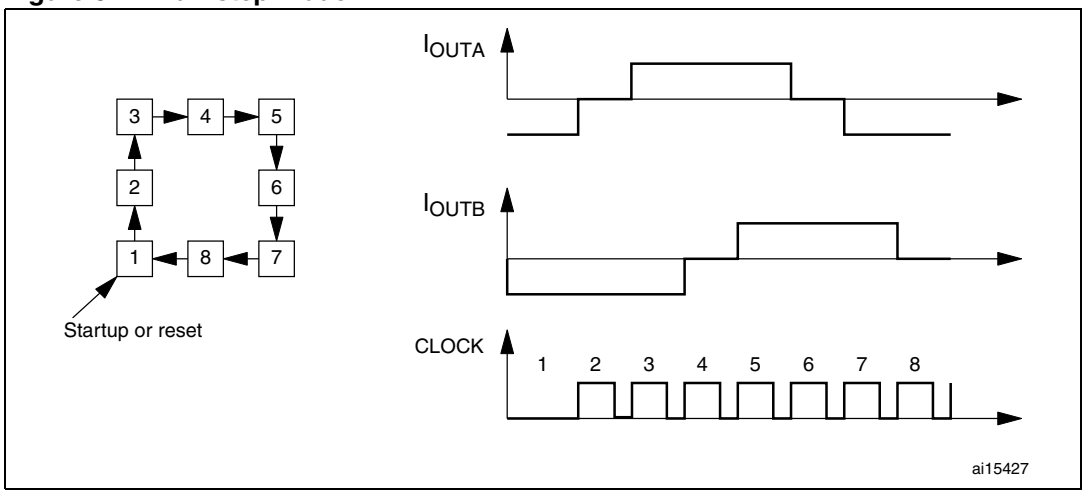
- Half-step mode

  A high logic level on the HALF/$\overline{FULL}$ input selects the half-step mode. *Figure 5* shows the motor current waveforms and the state diagram of the phase sequence generator. At startup or after reset the phase sequence generator is in state 1 (Home state). After each clock pulse the state changes according to the sequence:

  – 1,2,3,4,5,6,7,8,… if CW/$\overline{CCW}$ is high (clockwise sense)

  – 1,8,7,6,5,4,3,2,… if CW/$\overline{CCW}$ is low (counterclockwise sense).

Figure 5. Half-step mode



- Normal drive mode (full-step, two phases on)

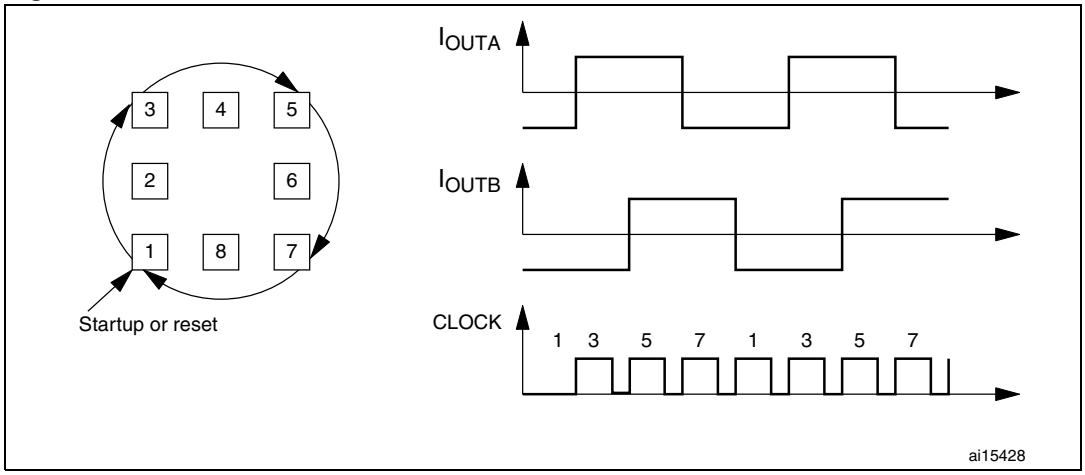
  A low level on the HALF/$\overline{FULL}$ input selects the full-step mode. If the low level is applied when the state machine is at an odd numbered state, the normal drive mode is selected. *Figure 6* shows the motor current waveform and the state diagram of the phase sequence generator. The normal drive mode can easily be selected by holding

the HALF/$\overline{\text{FULL}}$ input low and applying a reset. At startup or after reset the state machine is in state 1. While the HALF/$\overline{\text{FULL}}$ input is kept low, the state changes following the sequence:

– 1,3,5,7,… if CW/$\overline{\text{CCW}}$ is high (clockwise sense)

– 1,7,5,3,… if CW/$\overline{\text{CCW}}$ is low (counterclockwise sense)

Figure 6. Normal drive mode



- Wave drive mode (full-step, one phase on)

  A low level on the HALF/$\overline{\text{FULL}}$ input pin selects the full-step mode. If a low level is applied when the state machine is at an even numbered state, the wave drive mode is selected. *Figure 7* shows the motor current waveform and the state diagram of the phase sequence generator. To enter the wave drive mode, the state machine must be in an even numbered state. A more direct method of entering the wave drive mode consists in first applying a reset, then applying one pulse to the clock input while keeping the HALF/$\overline{\text{FULL}}$ input high, then taking the HALF/$\overline{\text{FULL}}$ input low. This sequence first forces the state machine to sate 1. The clock pulse, with the HALF/$\overline{\text{FULL}}$ input high causes the state machine to switch from state 1 to either state 2 or 8 depending on the CW/$\overline{\text{CCW}}$ input. After each clock pulse (rising edge), the state machine changes states following the sequence:

  – 2,4,6,8,… if CW/$\overline{\text{CCW}}$ is high (clockwise sense)

  – 8,6,4,2,… if CW/$\overline{\text{CCW}}$ is low (counter clockwise sense)

Refer to the L6208 datasheet for more details.

Figure 7. Wave drive mode

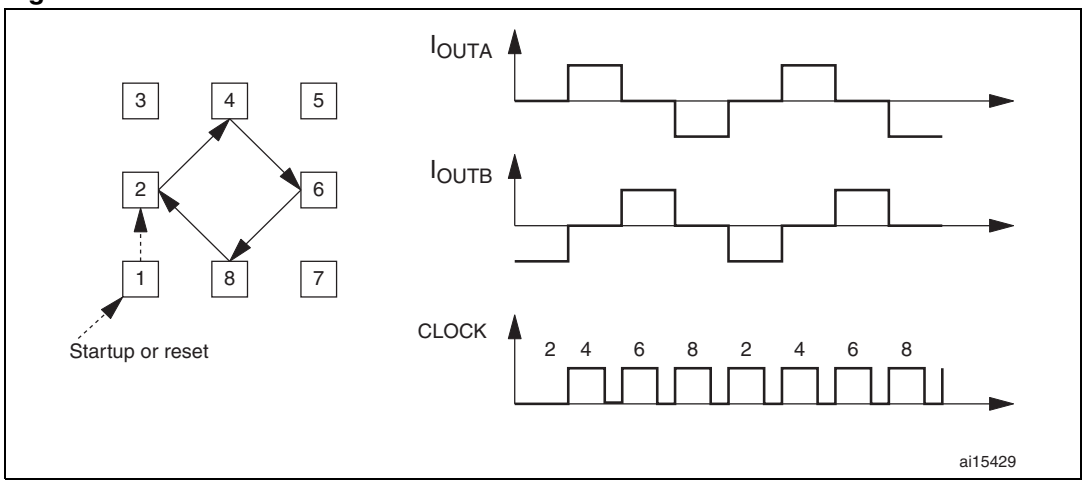
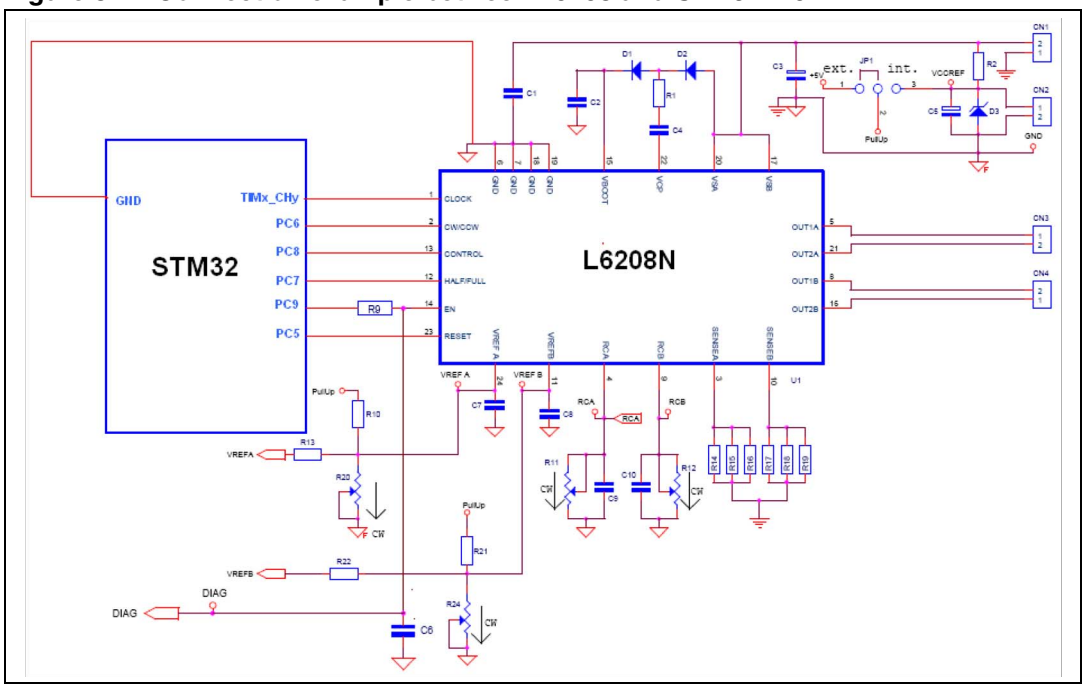


Figure 8. Connection example between L6208 and STM32F10xxx



2.2 Firmware development

In order to guarantee a minimum CPU load when controlling the stepper motor, an optimized program is provided that uses the resources and performance of the STM32F10xxx device.

2.2.1 Firmware description

In this application, TIM2 is used in output compare toggle mode to change the signal clock frequency. The signal frequency is increased from 200 Hz up to 800 Hz during acceleration, and from 800 Hz to 200 Hz during deceleration with a duty cycle of 50%. When defining the frequencies and the pulses, the electrical characteristics of the L6208 must be taken into

account. The maximum clock frequency of the L6208 is 100 kHz and the minimum clock low and high times are of 1 µs. The output compare toggle mode offers the possibility of changing only the frequency and keeping the duty cycle constant to obtain a regular clock signal with minimum CPU load. The transfer of the periods from memory to the ARR timer register is ensured by the DMA controller.

Two buffers are used to change the TIM2 periods. They are stored in SRAM. SRC_Buffer_INC is the buffer for the stepper motor acceleration and SRC_Buffer_DEC is the buffer for the motor deceleration. In this application, the acceleration and deceleration have the same slope. Each buffer contains ten frequencies for the input clock signal.

In the acceleration phase, on completion of the transfer of all the periods in SRC_Buffer_INC, a DMA transfer complete interrupt is generated. This interrupt stops the DMA transfer and enables the SysTick that starts counting a time T of 8 ms, during which the stepper motor continues running at maximum speed. After T, a SysTick interrupt is generated and DMA transfer is enabled for the transfer of the periods in SRC_Buffer_DEC. On completion of the transfer of the last period, the DMA transfer complete interrupt occurs. The whole procedure is repeated every 38 ms.

Figure 9. Clock signal for the stepper motor

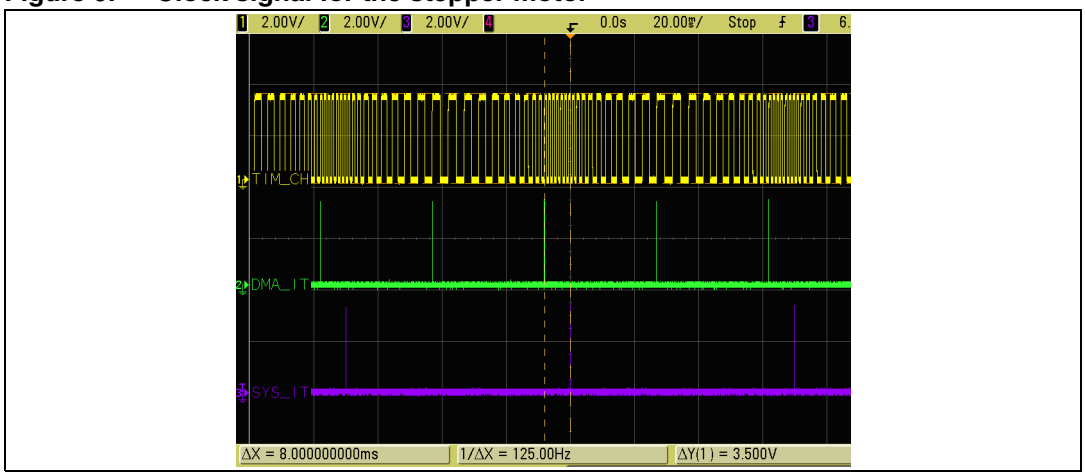


Figure 9 shows the clock signal generated for the stepper motor in yellow. The DMA interrupt and the SysTick interrupt are also represented. Period T is measured when the maximum frequency (800 Hz) is maintained.

2.2.2 Source files

The program provided with this application note includes the source files listed in *Table 2* below.

Table 2. List of sources files

Files	Description
StepperMotor.c	Routines used to control the stepper motors in different configurations and modes.
main.c	Example program.
STM32f10x_it.c	Interrupt service routines.

2.2.3 Main program and routine flowcharts

To test an example of stepper motor control, some `#define` statements should be uncommented in the main source file:

1. Uncomment one of the two possibilities for the step mode:

`#define Half_Step`

The selected mode is the half-step mode.

`#define Full_Step`

The selected mode is the full-step mode.

2. Uncomment one of the two possibilities for the motor rotation sense:

`#define RotationDirection_CW`

The motor rotates in the clockwise sense.

`#define RotationDirection_CCW`

The motor rotates in the counter clockwise sense.

3. Uncomment one of the two possibilities for the current decay mode:

`#define ControlSlow_Current`

The slow decay mode is selected.

`#define ControlFast_Current`

The fast decay mode is selected.

Moreover, the user should select the STM3210B-EVAL board in the tool options.

Figure 9, *Figure 10* and *Figure 11* provide the flowcharts of the main program, the DMA routine and the SysTick routine, respectively.

Figure 10. Main program flowchart

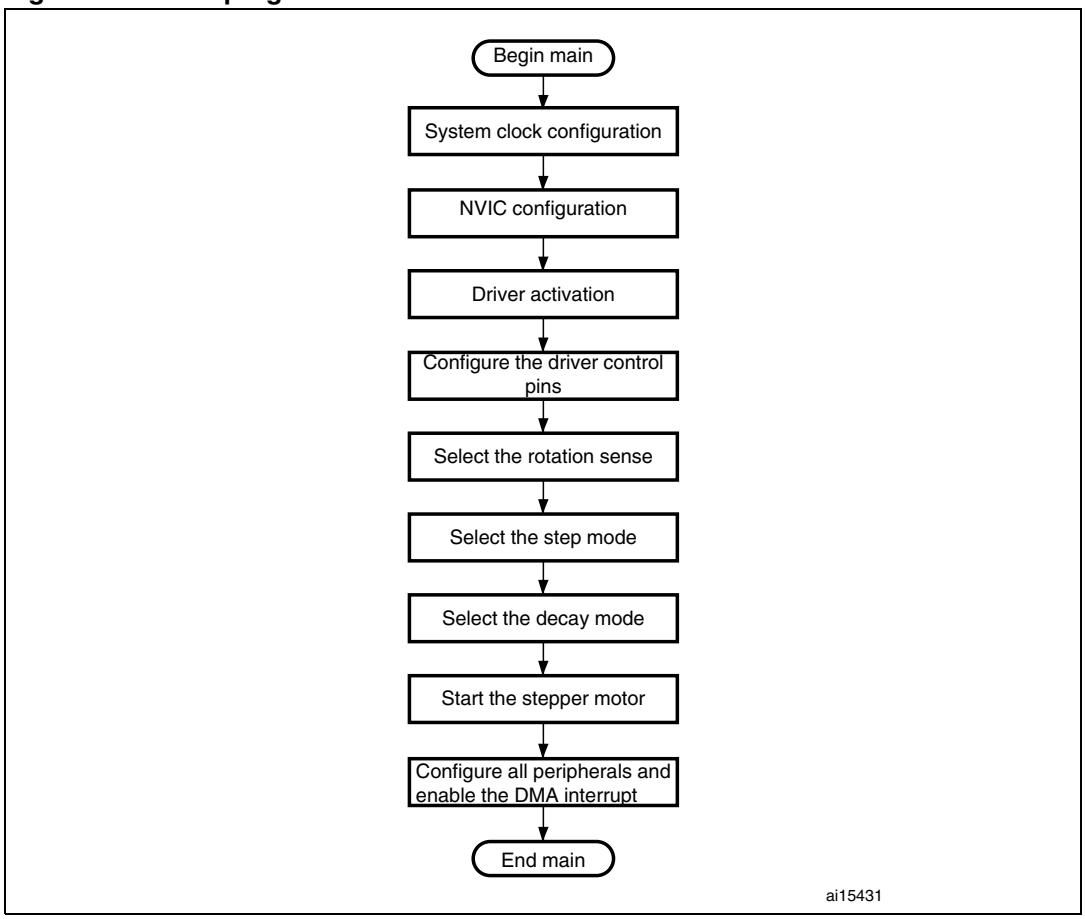


Figure 11. DMA routine flowchart

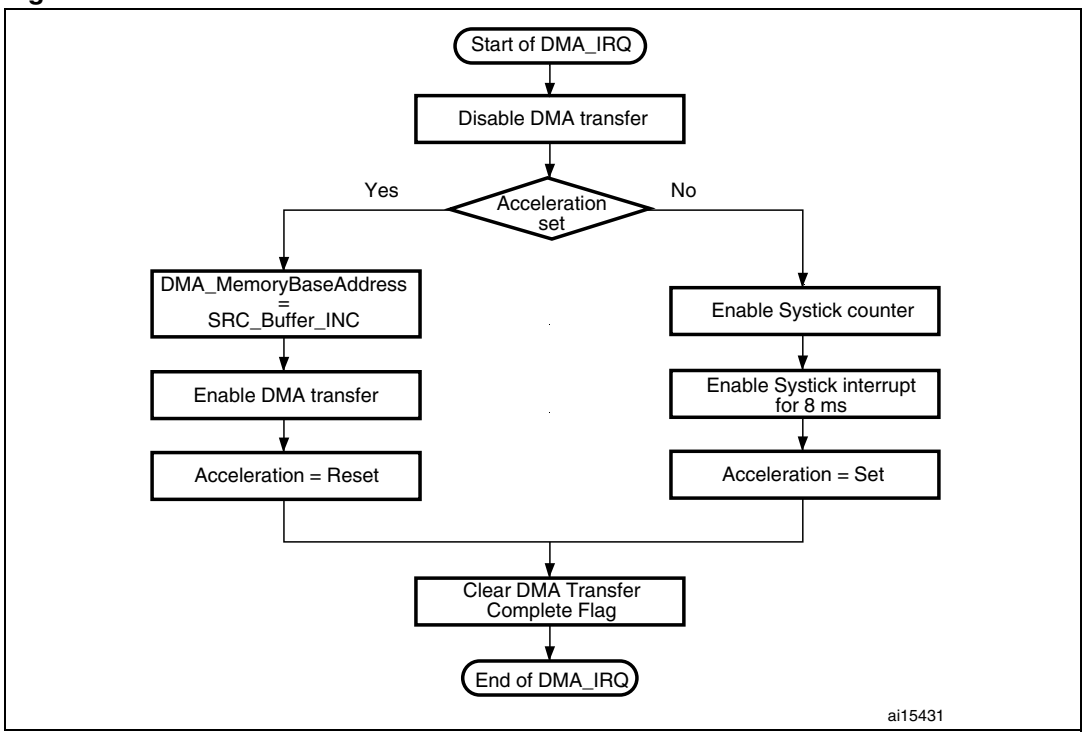
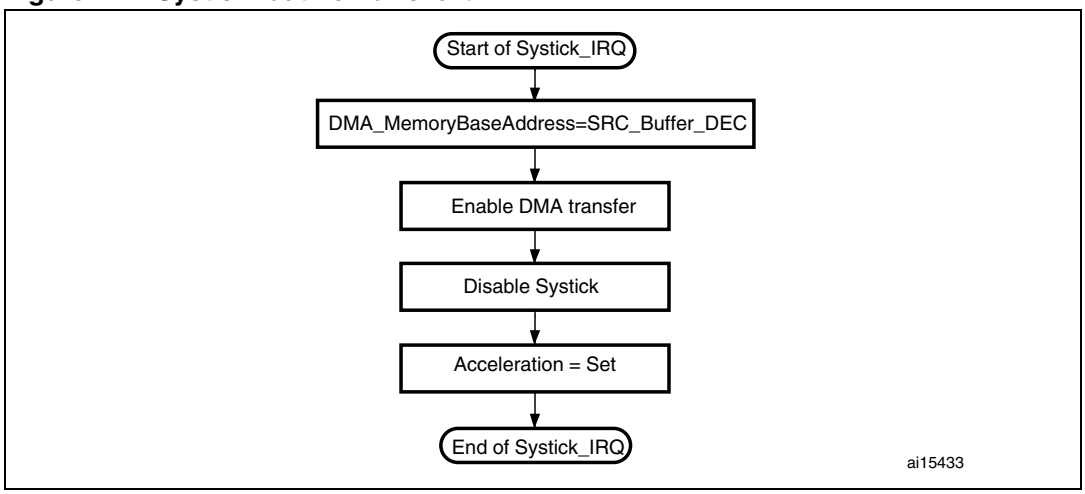


Figure 12. Systick routine flowchart



2.2.4 Software library description of the stepper motor driver

This section describes the software library of the stepper motor driver, gives the details of the related functions and provides an example of use of these functions and an example of a typical run profile.

The stepper motor driver can be used to manage the stepper motor functionality in both full- and half-step modes, in both clockwise and counter clockwise rotation senses and in both slow and fast decay modes.

Table 3 lists the different functions of the stepper motor library.

Table 3. Stepper motor library functions

Function name	Description
`Stepper_ResetDisable`	Disables the initialization of the driver to its default reset value.
`Stepper_Start`	Starts or stops the stepper motor.
`Stepper_Cmd`	Activates or deactivates the driver by enabling or disabling the peripheral clocks.
`Stepper_PinControlConfig`	Configures the driver control pins.
`Stepper_SelectMode`	Selects the step mode: full- or half-step.
`Stepper_SetRotationDirection`	Selects the rotation sense: CW or CCW.
`Stepper_SetControlMode`	Specifies the decay mode: slow or fast.
`Stepper_Init`	Configures all peripherals to control the stepper motor and enables the DMA interrupt.

The `Stepper_ResetDisable` function is described in *Table 4*.

Table 4. `Stepper_ResetDisable` function description

Function name	Stepper_ResetDisable
Function prototype	void Stepper_ResetDisable(void)
Behavior description	Disables the initialization of the driver to its default reset value.
Input	None
Output	None

The `Stepper_Start` function is described in *Table 5*.

Table 5. `Stepper_Start` function description

Function name	Stepper_Start
Function prototype	void Stepper_Start(FunctionalState NewState)
Behavior description	Starts or stops the stepper motor.
Input	NewState: ENABLE or DISABLE
Output	None

The `Stepper_Cmd` function is described in *Table 6*.

Table 6. `Stepper_Cmd` function description

Function name	Stepper_Cmd
Function prototype	void Stepper_Cmd(FunctionalState NewState)
Behavior description	Activates or deactivates the driver by enabling or disabling the peripheral clock.
Input	NewState: ENABLE or DISABLE
Output	None

The `Stepper_PinControlConfig` function is described in *Table 7*.

Table 7. `Stepper_PinControlConfig` function description

Function name	Stepper_PinControlConfig
Function prototype	void Stepper_PinControlConfig(void)
Behavior description	Configures the driver control pins
Input	None
Output	None

The `Stepper_SelectMode` function is described in *Table 8*.

Table 8. `Stepper_SelectMode` function description

Function name	Stepper_SelectMode
Function prototype	void Stepper_SelectMode(uint16_t Stepper_Mode)
Behavior description	Selects the step mode: full or half mode
Input	Stepper_Mode: Stepper_Full or Stepper_Half
Output	None

The `Stepper_SetRotationDirection` function is described in *Table 9*.

Table 9. `Stepper_SetRotationDirection` function description

Function name	Stepper_SetRotationDirection
Function prototype	Stepper_SetRotationDirection(uint16_t Stepper_RotationDirection)
Behavior description	Selects the rotation sense: CW or CCW
Input	Stepper_RotationDirection: Stepper_RotationDirection_CW or Stepper_RotationDirection_CCW
Output	None

The `Stepper_SetControlMode` function is described in *Table 10*.

Table 10. `Stepper_SetControlMode` function description

Function name	Stepper_SetControlMode
Function prototype	void Stepper_SetControlMode(uint16_t Stepper_ControlMode)
Behavior description	Specifies the decay mode.
Input	Stepper_ControlMode: Stepper_ControlFast or Stepper_ControlSlow
Output	None

The `Stepper_Init` function is described in *Table 11*.

Table 11. `Stepper_Init` function description

Function name	Stepper_Init
Function prototype	void Stepper_Init(void)
Behavior description	Configures all peripherals to control the stepper motor and enables the DMA interrupt.
Input	None
Output	None

2.2.5 Firmware performance study

The peripherals used in this application to control a stepper motor following the typical run profile are presented in *Table 12*. *Table 13* indicates the resource requirements.

Table 12. Peripheral usage

Peripheral	Description	Interrupt enable
5 I/O pins	Stepper motor output pins	
1 I/O pin	Used as TIM2 Channel1 in output	
TIM2	Configured in output compare toggle mode	
DMA	Transfers the periods from memory to timer by using the update event request	Transfer complete interrupt
Systick	Maintains the maximum motor speed during time interval T	Generates an interrupt each time interval T is reached

To control a stepper motor following the typical run profile, the software uses three interrupts, a DMA interrupt during acceleration, a SysTick interrupt and another DMA interrupt during deceleration. These interrupts are generated every 38 ms. In this implementation, each interrupt routine takes 12 cycles to enter the interrupt and 12 cycles to exit it. For the three interrupts, this makes a total of 72 cycles. With a clock speed of 72 MHz, it takes less than 1 µs to enter and exit the three interrupts.

The first DMA interrupt takes 3.6 µs, the sysTick interrupt takes 2 µs and the second DMA interrupt takes 2 µs. Stepper motor control therefore takes less than 8.6 µs. If the interrupts are required every 38 ms, stepper motor handling uses only 2.10^{-4}% of the processing power of the CPU.

Table 13. Resource requirements

RAM	Flash memory
1.5 Kbit	7.5 Kbit

3 Conclusion

In this application, the stepper motor is controlled using the medium-density STM32F103xx. The user is able to choose the step mode, the rotation sense and the current control mode. Although on today's market it is relatively easy to find devices capable of assuming a similar function, the solution discussed here offers more flexibility than a dedicated IC. The STM32F10xxx device has a set of peripherals particularly well suited to driving stepper motors with the best efficiency according to a typical run profile. Moreover, the application is implemented in a cost saving way, both in terms of external components and CPU usage as the application uses minimum hardware and very simple and flexible firmware routines.

4 Revision history

Table 14. Document revision history

Date	Revision	Changes
06-Mar-2009	1	Initial release.
30-Apr-2009	2	Occurrences of `u16` updated to `uint16_t` in accordance with the new STM32F10xxx standard peripheral library (StdPeriph_Lib) V3.0.0.