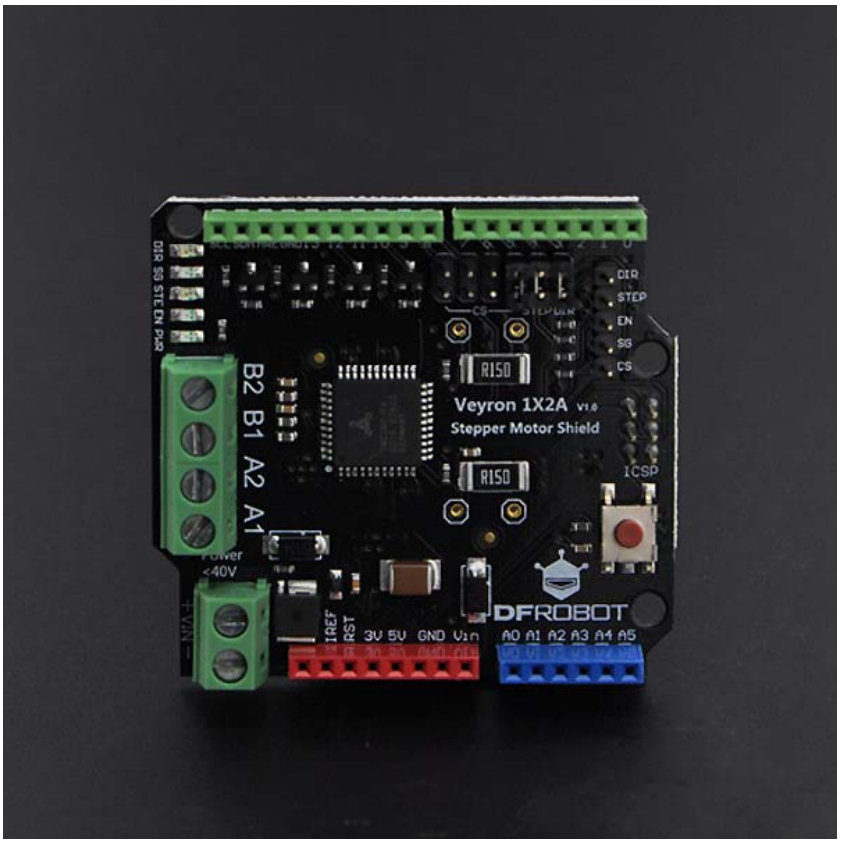
TMC260 Stepper Motor Driver Shield SKU: DRI0035



Contents

- 1 Introduction
- 2 Specification
- 3 Board Overview
  - 3.1 TMC260 Control Mode Selection
- 4 Tutorial
  - 4.1 Requirements
  - 4.2 Sample Code
    - 4.2.1 SPI Sample Code
    - 4.2.2 STEP/DIR Sample Code
- 5 Protocol/Library Explantion

Downloaded from Arrow.com.

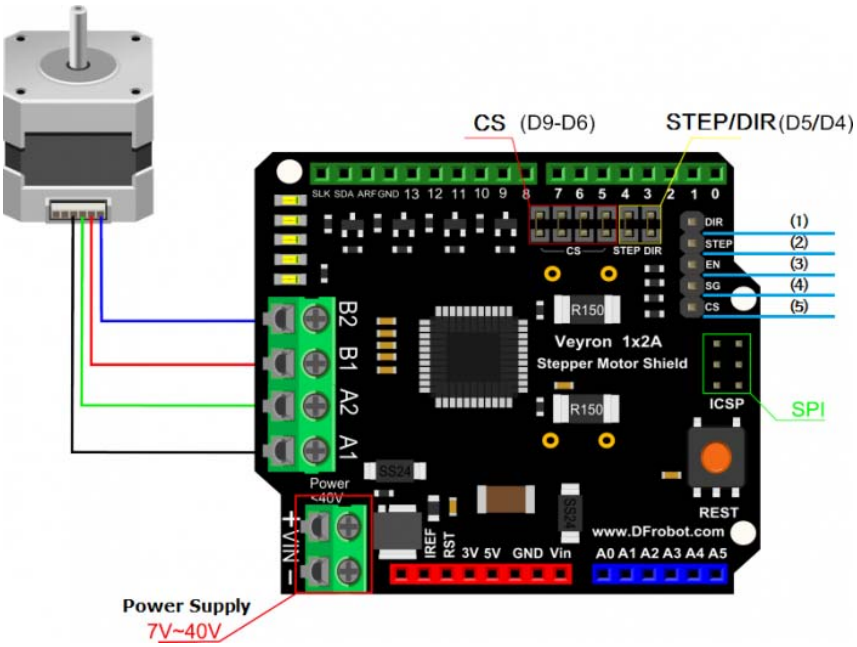
Introduction

Do you want to do some projects with stepper motors? A pair of automatic curtains? An XY Plotter? A 3D printer? Generally, it's not been easy to find a powerful stepper motor driver for Arduino, but now this has changed! DFRobot presents the TMC260 Stepper Motor Driver Shield. This shield allows your Arduino to easily drive stepper motors (up to 2A per motor coil, 40V max).

Specification

Basic Features	Key Features
• Compatible with two-phase stepper motors • Compatible with Arduino • Up to 2A motor current drive capability • Up to 40V DC • Simplified communication with standard SPI™ and STEP/DIR interfaces • Up to 256 microsteps per full step	• StallGuard2™: high precision sensorless motor load detection • CoolStep™ load dependent current control for energy saving up to 75% • microPlyer™ : Microstep interpolator for increased smoothness of microstepping over a STEP/DIR interface • SpreadCycle™ High-precision chopper algorithm available as an alternative to the traditional constant off-time algorithm • Protection & Diagnostics: overcurrent, short to GND,overtemperature & undervoltage • Low Power Dissipation, low RDSON and synchronous rectification

Board Overview



Downloaded from Arrow.com.

NOTE: We have already lead out some necessary pins on the chip and use mini jumpers to avoid a jumble of cables.

Label	Name	Description
DIR(D4)/1	DIR	Connect with D4 pin directly
STEP(D5)/2	STEP	Connect D5 pin directly with a jumper
3	EN	Enable
4	SG	Status value output of motor load detection
CS(D9-D6)/5	CS	Chip Select. You can choose D6, D7, D8 or D9 to be the CS pin in SPI mode
SPI	ICSP	Serial peripheral interface
Power Supply	VIN	Voltage in. Voltage range 7~40V DC, 2A max

TMC260 Control Mode Selection

- **SPI mode:** SDOFF bit is set, the STEP/DIR interface is disabled, and DRVCTRL is the interface for specifying the currents through each coil.
- **STEP/DIR mode:**SDOFF bit is clear, the STEP/DIR interface is enabled, and DRVCTRL is a configuration register for the STEP/DIR interface.

Downloaded from Arrow.com.

Tutorial

Requirements

- **Hardware**
  - Arduino UNO (or similar) x1
  - TMC260 Stepper Motor Driver Shield x1
  - Hybrid Stepper Motor for 3D Printer (3.5kg) x1

- **Software**

Arduino IDE V1.0.6 (link)

http://arduino.cc/download.php?f=/arduino-1.0.6-windows.zip

Sample Code

Please Download Library First: Download Link
https://raw.githubusercontent.com/CainZ/TMC260-Stepper-Motor-Driver-Shield/master/TMC26XStepper%20Libraries.zip

SPI Sample Code

You need to give a "sine" a value in SPI mode using function (tmc26XStepper.SPI_setCoilCurrent(200))

```
#include <SPI.h>
#include <TMC26XStepper.h>

//we have a stepper motor with 200 steps per rotation,CS pin 6, dir pin 4,
step pin 5 and a current of 800mA
TMC26XStepper tmc26XStepper = TMC26XStepper(200,6,4,5,800);
void setup() {
    Serial.begin(9600);
    Serial.println("============================");
    Serial.println("TMC26X Stepper Driver Demo App");
    Serial.println("============================");
    //set this according to you stepper
    Serial.println("Configuring stepper driver");
    //char constant_off_time, char blank_time, char hysteresis_start, char
hysteresis_end, char hysteresis_decrement
    tmc26XStepper.setSpreadCycleChopper(2,24,8,6,0);
    tmc26XStepper.setRandomOffTime(0);
    tmc26XStepper.SPI_setCoilCurrent(100);
    tmc26XStepper.setMicrosteps(128);
    tmc26XStepper.setStallGuardThreshold(4,0);
```

Downloaded from Arrow.com.

```
    Serial.println("config finished, starting");
    Serial.println("started");

      tmc26XStepper.SPI_setSpeed(80);    //Set speed at 80 RPM
      tmc26XStepper.SPI_step(-200);       //set step at -200 steps,  that is
to say stepper will turn a circle reverse
      tmc26XStepper.spi_start() ;         //start stepper
      delay(2000);                         //delay 2s


      tmc26XStepper.SPI_step(200);       // set step at 200 steps,  stepper
will turn a circle forward
      tmc26XStepper.spi_start() ;
      delay(2000);

      tmc26XStepper.SPI_setSpeed(100);   // Set speed at 100 RPM
   tmc26XStepper.SPI_step(-300);       // stepper will turn 1.5 circles
reverse
      tmc26XStepper.spi_start() ;
      delay(2000);

     tmc26XStepper.SPI_setSpeed(120);   // Set speed at 120 RPM
     tmc26XStepper.SPI_step(400);       // stepper will turn 2 circles
forward
     tmc26XStepper.spi_start() ;
     delay(3000);



}
void loop() {


          //you can put stepper control code in loop{} to make stepper works
circularly

}
```

STEP/DIR Sample Code

```
#include <SPI.h>
#include <TMC26XStepper.h>


//we have a stepper motor with 200 steps per rotation, CS pin 2, dir pin 6, s
tep pin 7 and a current of 800mA
TMC26XStepper tmc26XStepper = TMC26XStepper(200,6,4,5,800);
int curr_step;
int speed =  0;
int speedDirection = 100;
```

Downloaded from Arrow.com.

```
int maxSpeed = 1000;


void setup() {
  Serial.begin(9600);
  Serial.println("============================");
  Serial.println("TMC26X Stepper Driver Demo App");
  Serial.println("============================");
  //set this according to you stepper
  Serial.println("Configuring stepper driver");
  //char constant_off_time, char blank_time, char hysteresis_start, char hyst
eresis_end, char hysteresis_decrement
  tmc26XStepper.setSpreadCycleChopper(2,24,8,6,0);
  tmc26XStepper.setRandomOffTime(0);


  tmc26XStepper.setMicrosteps(128);
  tmc26XStepper.setStallGuardThreshold(4,0);
  Serial.println("config finished, starting");
  Serial.println("started");
}


void loop() {
  if (!tmc26XStepper.isMoving()) {
    speed+=speedDirection;
    if (speed>maxSpeed) {
      speed = maxSpeed;
      speedDirection = -speedDirection;
    } else if (speed<0) {
      speedDirection = -speedDirection;
      speed=speedDirection;
    }
    //setting the speed
    Serial.print("setting speed to ");
    Serial.println(speed);
    tmc26XStepper.setSpeed(speed);
```

Downloaded from Arrow.com.

```
    //we want some kind of constant running time - so the length is just a pr
oduct of speed
    Serial.print("Going ");
    Serial.print(10*speed);
    Serial.println(" steps");
    tmc26XStepper.step(10*speed);
  } else {
    //we put out the status every 100 steps
    if (tmc26XStepper.getStepsLeft()%100==0) {
      Serial.print("Stall Guard: ");
      Serial.println(tmc26XStepper.getCurrentStallGuardReading());
    }
  }
  tmc26XStepper.move();
}
```

Protocol/Library Explantion

```
*TMC26XStepper(int number_of_steps, int cs_pin, int dir_pin, int step_pin, un
signed int current, unsigned int resistor)
```

number_of_steps : Number of steps per rotation
cs_pin : CS: enable pin
dir_pin : Dir: direction pin
step_pin : Step: step pin
resistor : sense resistor current scaling default value is 15 ohms(or 150 mohms). It can be ignored when you don't need to use that.

```
*TMC26XSet_SPICS(int cs_pin)
```

cs_pin : CS pin in SPI mode

Downloaded from Arrow.com.

```
*setConstantOffTimeChopper(char constant_off_time, char blank_time, char fast
_decay_time_setting, char sine_wave_offset, unsigned char use_current_compara
tor)
```

constant_off_time： Off time. This setting controls the duration of the slow decay time and limits the maximum chopper frequency. For most applications an off time within the range of 5μs to 20μs will fit.
blank_time： Blanking time.Blanking is the time when the input to the comparator is masked to block these spikes. Set from 0~54。
hysteresis_start： hysteresis start setting that this value is an offset to the hysteresis end value HEND.The range is 1~8
hysteresis_end： hysteresis end setting. Set the hysteresis end value from -3~12
hysteresis_decrement： hysteresis decrementer setting, this setting determines the slope of the hysteresis during on time and during fast decay time

```
*setCurrent(unsigned int current)
```

current： set Current。

```
*SPI_setCoilCurrent(int Current)
```

Current： Coil Current in SPI mode set from 0 to 248

```
*setMicrosteps(int number_of_steps)
```

number_of_steps： Microsteps each step set from 0 to 248

```
*setStallGuardThreshold(char stall_guard_threshold, char stall_guard_filter_e
nabled)
```

stall_guard_threshold： Set threshold of StallGuard to get logical value of StallGuard
stall_guard_filter_enabled： stall guard filter enable（1），stall guard filter disenable（0）。

```
*step(int steps_to_move)
```

Downloaded from Arrow.com.

steps_to_move： Set the steps to move and use positive and negative to determine the direction in STEP/DIR mode

```
*setSpeed(unsigned int whatSpeed)
```

whatSpeed： Set RPM (revolutions per minute) in STEP/DIR mode

```
*SPI_step(int spi_steps_to_move)
```

spi_steps_to_move： Set the steps to move and use positive and negative to determine the direction in SPI mode

```
*SPI_setSpeed(unsigned int whatSpeed)
```

whatSpeed： Set RPM (revolutions per minute) in SPI mode

```
*tmc26XStepper.move()
```

Stepper starts to run n STEP/DIR mode

```
*tmc26XStepper.spi_start()
```

Starts to run in SPI mode

Troubleshooting

Stepper Motor

A stepper motor is a special electromagnetic motor which can convert pulse signals into corresponding angular displacement (or linear displacement). Ordinary motors continuously rotate, but in stepper motors there are two basic states: positioning and revolving. Once there is a pulse input, stepper motor will rotate a certain angle, which we call a "step".

Control Mode

The TMC260 driver chip has two different control modes: SPI and STEP/DIR interface.

Downloaded from Arrow.com.

The TMC260 requires configuration parameters and mode bits to be set through the SPI before the motor can be driven. The SPI also allows reading back status values and bits. This interface must be used to initialize parameters and modes necessary to enable driving the motor. This interface may also be used for directly setting the currents flowing through the motor coils, as an alternative to stepping the motor using the STEP and DIR signals, so the motor can be controlled through the SPI interface alone.

The STEP/DIR interface is a traditional motor control interface available for adapting existing designs to use TRINAMIC motor drivers. Using only the SPI interface requires slightly more CPU overhead to look up the sine tables and send out new current values for the coils.

NOTE: Our developers didn't find the specific formula in the datasheet to calculate this sine. We welcome you to join in development and help us.

1.SPI

The SPI (Serial Peripheral Interface) is a bit-serial interface synchronous to a bus clock. For every bit sent from the bus master to the bus slave, another bit is sent simultaneously from the slave to the master. Communication between an SPI master and the TMC260 or TMC261 slave always consists of sending one 20-bit command word and receiving one 20-bit status word.

The SPI command rate typically corresponds to the microstep rate at low velocities. At high velocities, the rate may be limited by CPU bandwidth to 10-100 thousand commands per second, so the application may need to change to fullstep resolution.

2.STEP/DIR

The STEP/DIR interface is enabled by default. On each active edge, the state sampled from the DIR input determines whether to step forward or back. Each step can be a fullstep or a microstep, in which there are 2, 4, 8, 16, 32, 64, 128, or 256 microsteps per fullstep. During microstepping, a step impulse with a low state on DIR increases the microstep counter and a high decreases the counter by an amount controlled by the microstep resolution. An internal table translates the counter value into the sine and cosine values which control the motor current for microstepping.

If you have any questions or cool ideas to share, please visit **DFRobot Forum**

Powered By DFRobot © 2008-2017

Downloaded from Arrow.com.