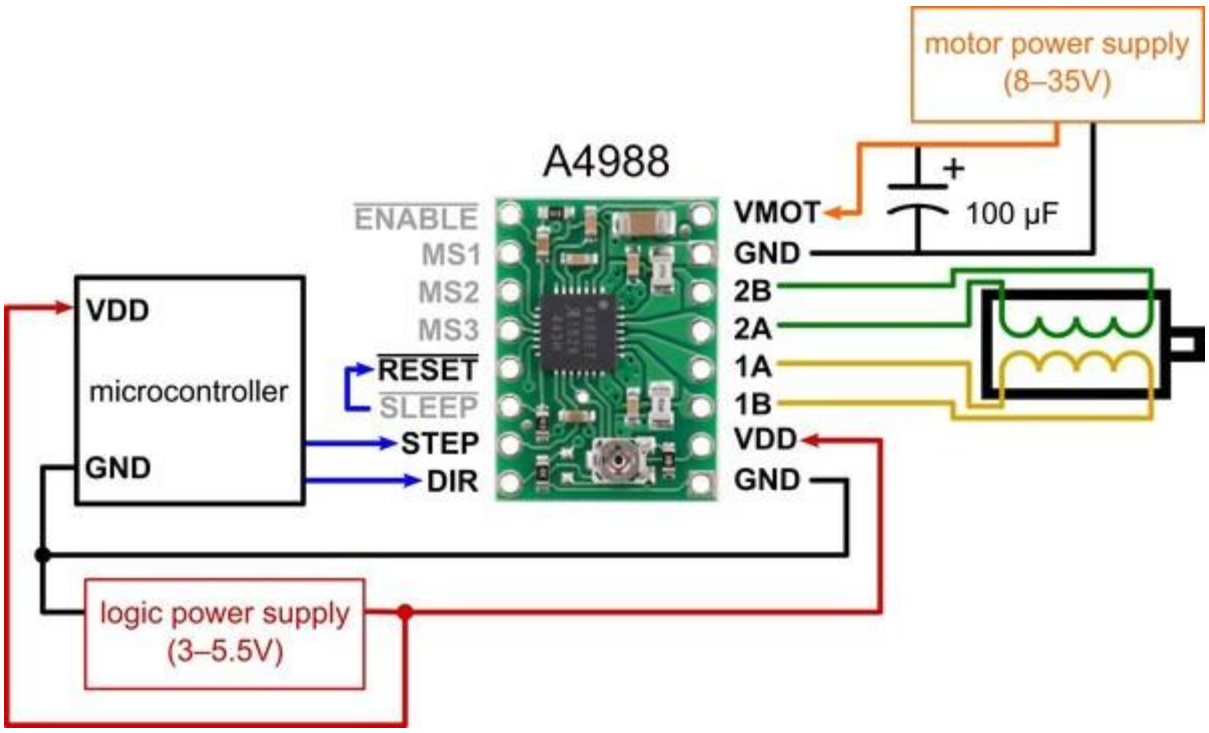
Stepper motor driver – documentation

The driver is designed and tested on the A4988 Stepper Motor Driver Carrier. Below are details related to the connection and features included in the driver

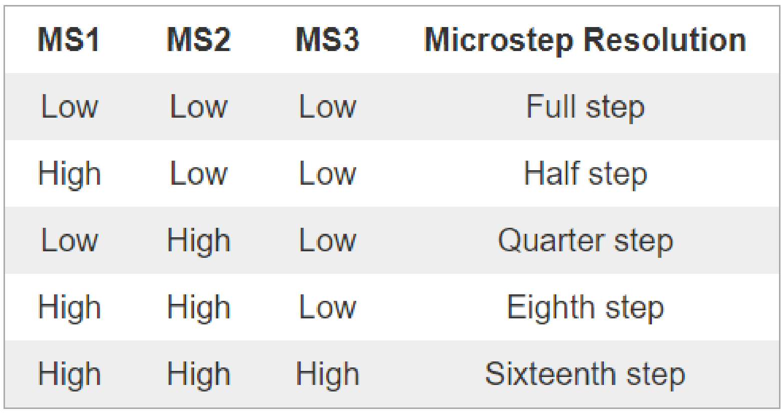
A4988 Stepper Motor Driver Carrier

Connection:



Stepper motors typically have a step size specification (e.g. 1.8° or 200 steps per revolution), which applies to full steps. A microstepping driver such as the A4988 allows higher resolutions by allowing intermediate step locations, which are achieved by energizing the coils with intermediate current levels. For instance, driving a motor in quarter-step mode will give the 200-step-per-revolution motor 800 microsteps per revolution by using four different current levels.

To use microstepping, follow the table below:



MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

Setting the current limit is described in the following link:

https://www.pololu.com/product/1182

Driver documentation and operating principle

Principle of Operation

A stepper motor operates by moving in discrete steps, which allows for precise control of its position. The Stepper class manages the motor by sending pulses to the step pin (m_Step) and setting the direction using the direction pin (m_Dir). The motor speed is controlled by adjusting the interval between pulses. The class uses a separate thread to handle motor operations asynchronously, ensuring smooth and continuous movement.

Constructor and Destructor

Stepper(PinName step_pin, PinName dir_pin, float initial_speed = 3.0f, int step_rev = 200)

Constructs a new Stepper motor object. Initializes the stepper motor with the specified step and direction pins, initial speed, and steps per revolution. Sets up the motor speed and direction, and initializes the ticker and thread for handling motor operations. Initial speed and step_rev are predefined, step_rev is the number of steps in one revolution (to be changed in case of having a different motor or enabled microstepping.

~Stepper()

Destroys the Stepper object, detaches the ticker, and terminates the thread to clean up resources.

Getter Functions

float getPosition()

Returns the current position of the stepper motor in steps. This function is useful for applications that require precise position control.

float getRotations()

Returns the current position of the stepper motor in rotations. This function converts the step count to rotations based on the steps per revolution.

float getSpeed()

Returns the current speed of the stepper motor in revolutions per second.

Setter Functions

void setDirection(bool direction)

Sets the direction of the stepper motor. The direction can be clockwise (CW) or counter-clockwise (CCW).

void setSpeed(float speed)

Sets the speed of the stepper motor in revolutions per second. The motor will automatically adjust its direction based on the sign of the speed.

void setAbsoluteZeroPosition()

Sets the current position as the zero position. This function is useful for resetting the motor's position reference.

void setAbsolutePosition(int absolute_pos)

Sets an absolute position for the stepper motor to move to, in steps. The motor will move to the specified position relative to the current zero position.

void setRelativePosition(int relative_pos)

Sets a relative position for the stepper motor to move to, in steps. The motor will move the specified number of steps from its current position.

void setAbsoluteRevolutions(float absolute_rev)

Sets an absolute position for the stepper motor to move to, in revolutions. The motor will move to the specified number of revolutions relative to the current zero position.

void setRelativeRevolutions(float relative_rev)

Sets a relative position for the stepper motor to move to, in revolutions. The motor will move the specified number of revolutions from its current position.

Control Functions

void startRotation()

Starts the continuous rotation of the stepper motor. This function sets the motor mode to VELOCITY.

void stopRotation()

Stops the rotation of the stepper motor. This function sets a flag to stop the motor in the handler function.

Enumerators

enum Direction

Defines the possible directions for the stepper motor:

- CW: Clockwise direction
- CCW: Counter-clockwise direction

enum MotorMode

Defines the possible operating modes for the stepper motor:

- STEPS: Step mode for moving to a specific position
- VELOCITY: Velocity mode for continuous rotation
- VOID: Void mode for stopping the motor
- STOP: Stop mode for detaching the ticker

Private Functions

void sendThreadFlag()

Sets a thread flag to trigger the motor handler. This function is called by the ticker at regular intervals.

void handler()

Handles the motor operations based on the current mode (STEPS, VELOCITY, VOID, STOP). This function is run in a separate thread to ensure smooth motor operation.

void step()

Executes a single step of the motor. This function is called by the handler function based on the current mode and position.

void enableDigitalOutput()

Enables the digital output by setting the step pin high. This function is used to generate step pulses.

void disableDigitalOutput()

Disables the digital output by setting the step pin low. This function is used to generate step pulses.

Hardware Objects

- DigitalOut m_Step: Controls the step pin.
- DigitalOut m_Dir: Controls the direction pin.
- Thread m_Thread: Manages the motor control thread.
- Ticker m_Ticker: Generates regular interrupts to trigger the motor handler.
- Timeout m_Timeout: Generates short delays for step pulses.
- ThreadFlag m_ThreadFlag: Manages thread synchronization.

Main.cpp configuration and additional info

1. Of the basic information to be considered is an additional element in the main file ie: DigitalOut enableStepper(PB_1), the PB_1 pin should be connected to the Enable pin on the breakout board (image above) this is used to activate or deactivate the driver. An important point is that when you want to activate the driver you should use the following command: enableStepper.write(0);
2. The speed is limited and depends on (in my opinion) the driver principle. It is hard to determine the speed range because it depends on the number of steps per revolution and the ranges will vary. If the speed is set too high, the motor will simply not rotate due to the frequency of the signal being sent too high. In the other case, if the speed is too low then the motor will make oscillating movements. To reduce the speed, you need to make changes in the driver configuration and change the number of steps per revolution.
3. To reduce the speed, you should:
   - connect the digital pin to the MS1 pin on the driver (picture above)
   - create a digital output object
   - set output to high
   - change the number of steps per revolution in the constructor (in our case we are doing half of the step so the number of steps for revolution is 400.

   Example code:

```
    Stepper stepper(PB_14, PC_4, 1.0f, 400);
    DigitalOut enableStepper(PB_1);
    DigitalOut miniStepper(PA_6);

    enableStepper.write(0);
    miniStepper.write(1);
```