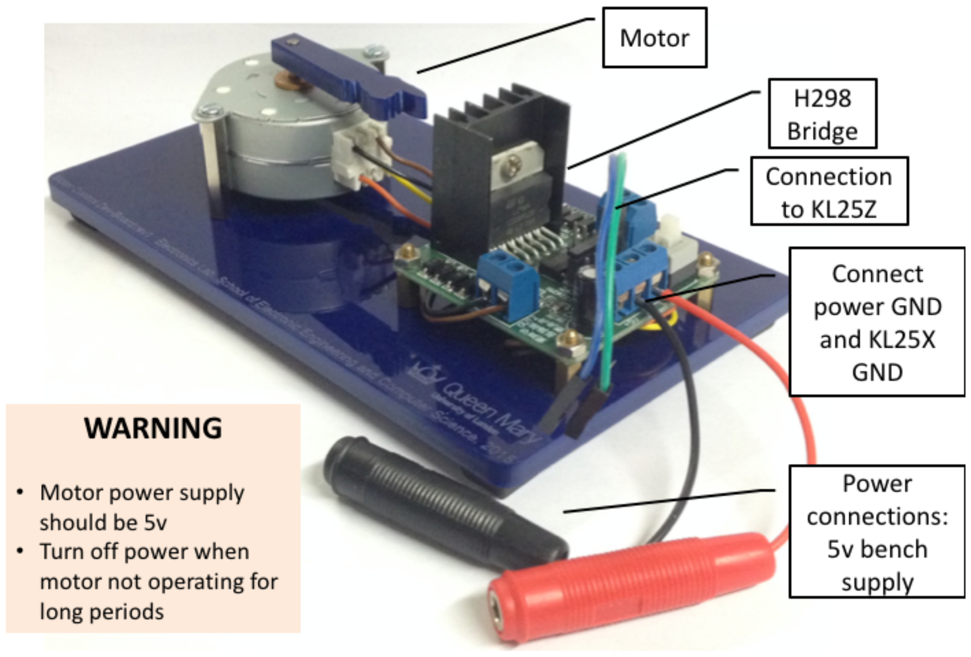
Demonstration Project
Stepper Motor

This project shows how to move the stepper motor. The demonstration project includes and uses a small API that can be included in your programs.

1. Principles of Stepper Motor Control

A stepper motor moves in a number of discrete steps. The stepper we will use has a step of 7.5 degrees, making 48 steps per rotation. The stepper moves in either direction.

The stepper is driven by an L298 H-bridge: this is all wired up on a base:


Motor
H298 Bridge
Connection to KL25Z
Connect power GND and KL25X GND
Power connections: 5v bench supply
WARNING
• Motor power supply should be 5v
• Turn off power when motor not operating for long periods

There are 4 connections from the motor to the KL25Z. These are numbered IP1, IP2, IP3 and IP4. In addition, there are two enable connection: EN1 and EN2. These should be connected to high (3.3v).

A common ground (0 v) must be connected: ensure the motor ground is connected to the KL25Z (there should be an additional ground wire to allow this).

1.1 Example Calculation of Frequency

Suppose that we wish to rotate the motor at 100 RPM. This means that:

- 48 x 100 = 4,800 steps in a minute
- 4,800 / 60 = 80 steps per second
- Time between steps = 1/80 second = 12.5 ms

2. Using the Stepper Software

The stepper motor software is in two files:

1. stepperMotor.c contains the code
2. stepperMotor.h is a header file

Add the .c to your project. This involves two steps: add it to the SRC directory and then use the microVision file management to 'add' it to the project. Include the header file in another .c file in which you wish to call the provided functions. It should not be necessary to modify the provided code but you can if you wish.

2.1 Overall Design

The overall design of the software has the following components:

- Definition of a MotorType: this structure holds the state of the motor. A variable of this type is declared, initialised (see below) and passed as a parameter to the other functions.
- API functions that update values in this structure, described below.
- A special update function that is called, at the desired interval, to move the stepper on the next step.

2.2 Functions

The following functions are provided:

Function	Description
`void initMotor( MotorId m)`	Initialise the motor. The motor identity must be created first: it points to a variable containing information including the 4 GPIO pins used. These are all assumed to be on the same port but any port can be used. The clock must be enabled to the port selected before this function is called.
`void moveSteps( MotorId m, uint16_t steps, bool clockwise)`	Move the motor the given number of steps. If this number is zero, the motor moves continuously. The direction is given by the boolean clockwise: the motor rotates anticlockwise if this is false. **Calling this function when the motor is moving may cause the motor to miss a step.**
`int32_t getSteps( MotorId m)`	Get the number of net steps rotated since the motor was initialised. Clockwise counts positive, anticlockwise negative so this value can be positive or negative. Note that the result is only reliable if the motor is stopped: if it is moving the number will continue to change.
`bool isMoving( MotorId m)`	Return true if the motor is moving (i.e. it has outstanding steps to make or is moving continuously); false if the motor has no outstanding steps.
`void stopMotor( MotorId m)`	Stop the motor. This function erases any outstanding steps requested by 'moveSteps'.
`updateMotor( MotorId m)`	Update the state of the motor. If the motor has outstanding steps, or is running continuously, calling this function causes the motor to step. **It is the user's responsibility to call this function at the correct frequency**. It is safe to call this function from an ISR. (The other functions should not be called from an ISR.) If there are no outstanding steps, then calling this function is harmless: no change occurs.

2.3 MotorType, MotorId and Initialisation

The following code shows how the motorType variable is created before the function initMotor can be called.

```
motorType mcb ;           // Motor type variable
MotorId m1 = & mcb ;      // Motor id variable
m1->port = PTE ;          // Set the port
m1->bitAp = MOTOR_IN1 ; // Set pin for IN1
m1->bitAm = MOTOR_IN2 ; // Set pin for IN2
m1->bitBp = MOTOR_IN3 ; // Set pin for IN3
m1->bitBm = MOTOR_IN4 ; // Set pin for IN4
```

The key features are:

- The MotorId variable is a pointer to the motorType structure
- The structure is initialised with the port and pins. Note that MOTOR_INx are macros that give names to the pin numbers used.
- All the pins must be on the same port.

3. Example Project

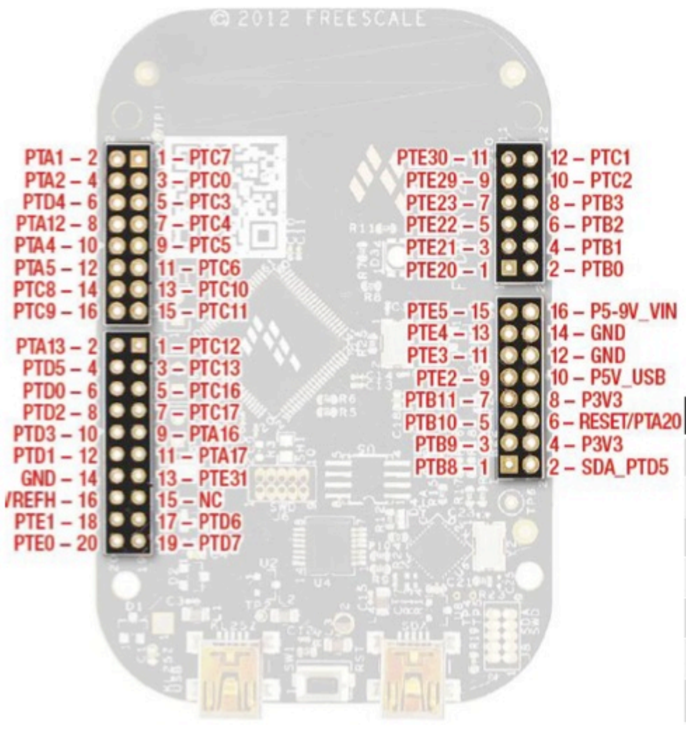
The sample project has a button and a motor. The behaviour is as follows:

1. Pressing the button causes the motor to rotate clockwise continuously, until …
2. The button is pressed again. The motor then rotates anticlockwise the same number of steps until it returns to its original position.
3. The system does not respond to further button presses until restarted.

The motor moves with two speeds. The speed is controlled by counting the number of 10ms major cycles in the cyclic system; this limit both the accuracy of the speed and the top speed.

3.1 KL25Z Connections

The following picture shows the connection to the KL25Z headers.



Function	Pin / Header
Button	PTD6 / J2 17
IN1	PTE30
IN2	PTE29
IN3	PTE23
IN4	PTE22
3.3 volts	J9, 4 or 8
GND	J9, 12 or 14