LPC824 Drivers for the Adafruit Motor Shield V2

The LPC82x Xpresso v2 board from NXP Semiconductors makes use of the standard Arduino pinout. The key benefit to this pin and board layout is that it is possible to combine the highly flexible and efficient ARM Cortex-M0 based LPC82x with a wide variety of third-party shields, meaning that you don't have to create your own custom boards to test out many common peripherals.

As an example of how third-party shields can be used with the LPC82x Xpresso boards, drivers have been created for version two of the **Adafruit Motor Shield**.

The shield can be ordered directly from Adafruit or one of Adafruit's many distributors, and a variety of learning guides are available for this shield.

Product Page: https://www.adafruit.com/product/1438
Board Schematic: https://learn.adafruit.com/assets/9536
Official Documentation: https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/overview

The Adafruit Motor Shield makes use of the PCA9685 from NXP Semiconductors, which is a 16-channel I2C-based PWM driver. The 16 PWM outputs are used to drive a variety of motors types through a standard H-Bridge, allowing up to 4 DC motors in the current configuration.

The motor shield also makes use of the PWM pins on the Arduino pinout to drive up to two servo motors. On the LPC82x, these PWM outputs are driven with the flexible State Configurable Timer (SCT).

Setting up the Adafruit Motor Shield Drivers with the LPC82x

The Adafruit Motor Shield can be stacked on top of the LPC82x Xpresso board as you would with an Arduino. Male header pins should be soldered on the appropriate header row, and the shields should be stacked on top of the LPC82x development board.

Powering the Shield

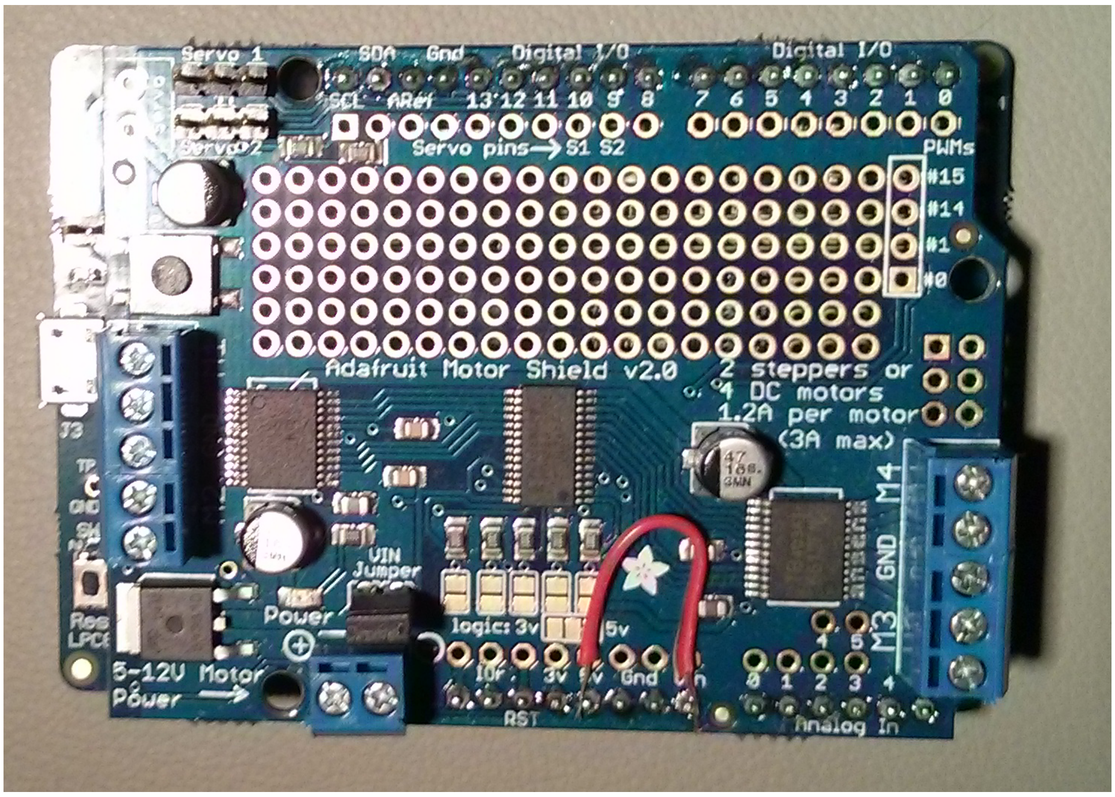
There are two options to power the motor shield. It can be powered from an external 5-12V DC power supply using the 2-pin terminal block labeled **POWER**, or it can be powered directly from the 5V USB VBUS supply on the LPC82x Xpresso board (within the limit of 500mA current for USB devices).

To power the motor shield from the USB VBUS, make sure that the **VIN Jumper** located beside the power terminal block is inserted.

One minor modification is required to the board if you wish to power the motor shield from the USB power supply. The Adafruit Motor Shield uses the **VIN** pin as the power supply, and VIN isn't connected on the LPC82x Xpresso board by default.

To enable power to the shield, you must solder a small jumper wire between the **5V** pin and the **VIN** pin, redirectling the VBUS supply to VIN, and routing power to the Motor Shield when the VIN jumper is closed.

The red wire in the image below was added to solve this problem:



Supported Motor Types

At present, the current drivers support a variety of **DC motors** using I2C and the PWM driver (ex.low cost hobby DC motors, vibration DC motors, etc.), or **servo motors** using the SCT, which generates a PWM output tailored for common servos.

DC Motors

Up to 4 DC motors can be controlled with the Adafruit Motor Shield, making use of the **M1, M2, M3** or **M4** two-pin terminal blocks. Motor speed and direction can be controlled using a variety of helper functions, described below in the API documentation.

Servo Motors

Up to two servo motors can be used (either continuous rotation or fixed range), although by default the PWM pin used by Servo 2 is also connected to the BLUE LED (P0_27-BLUE on the LPC82x Xpresso schematic), which may cause the LED to flicker during motor operation.

LPC82x LPCXpresso API

The following API has been created to driver to motor shield using the LPCXpresso IDE, available free of charge from NXP Semiconductors.

bool msInit(uint8_t void)

This function initializes the motor shield, setting up any pins required by the board and attempting to initialize the PCA9685 I2C PWM driver.

If the initialization was successful, the function will return **true**. If initialization failed, the function will return **false**.

DC Motor Functions

The following functions are used to interact with DC motors. The speed and direction of any DC motor can be controlled with these functions:

void msDCInit(uint8_t motorNum)

This function will attempt to initialized the specified DC motor, where **motorNum** can be between 1 and 4, corresponding the the **M1, M2, M3** or **M4** terminal blocks on the motor shield.

void msDCRun(uint8_t motorNum, misdirection_t direction, uint8_t speed)

This function will cause the specified DC motor (1..4) to advanced in the specified direction, at a speed controlled by an 8-bit PWM output, where 1 = slowest and 255 = fastest.

'direction' can be any of the following values:

```
typedef enum
{
  MS_DIR_FORWARD  = 1,
  MS_DIR_BACKWARD = 2,
  MS_DIR_RELEASE  = 3
} msDirection_t;
```

For example, the following code will move **M1** forward at reasonably high speed, and **M2** backward and ½ the speed of M1:

```
msDCRun(1, MS_DIR_FORWARD, 200);
msDCRun(2, MS_DIR_BACKWARD, 100);
```

Servo Functions

The following functions are used to control servo motors, using the SCT timer as a PWM generator on the two dedicated Servo headers on the motor shield.

void msServoInit(uint8_t motorNum)

This function will attempt to initialize the specified servo number (valid numbers are 1 or 2).

This function will configure the SCT peripheral as a PWM output, and depending on the motor number specified will also configure the appropriate pin as a PWM output. The PWM frequency is set to 50Hz, which is a fairly common frequency for low cost servo motors, though this may need to be adjusted for specific motors.

void msServoGetMinDutyCycle(void)

This helper function is used to provide the minimum number of clock ticks that can be used for the PWM duty cycle (based on a 50Hz clock and whatever frequency the LPC82x is being driven at).

void msServoGetMaxDutyCycle(void)

This helper function is used to provide the maximum number of clock ticks that can be used for the PWM duty cycle (based on a 50Hz clock and whatever frequency the LPC82x is being driven at).

void msServoSetDutyCycle(uint8_t motorNum, uint32_t dutyCycle)

This function adjusts the duty cycle for the PWM output that drives the servo, which will control the speed or the position of the servo motor depending on the motor type being used (continuous rotation or fixed range).

'motorNum' can be either 1 or 2, depending on the motor being used, and dutyCycle is specified in ticks on the PWM clock.

'dutyCycle' will normally be provided in conjunction with the **msServoGetMinDutyCycle** and **msServoGetMaxDutyCycle** helper functions, since the specific range that can be applied will depend on the core clock speed that the LPC82x is being run at, as well as the PWM frequency, which may need to be adjusted from one motor to the next.

Example Code

The following examples can be used to drive either DC motors or servos, and should serve as a starting point when working with the motor shield and the LPC82x Xpresso development board:

DC Motor Example

This example will drive two motors, constantly accelerating and decelerating the motors, then switching directions and repeating in an endless loop.

```
/* Initialise two DC motors (insert motors in M1 and M2) */
msDCInit(1);
msDCInit(2);

while(1)
{
    uint8_t i;

    /* Forward */
    for (i=0; i<255; i++)
    {
      msDCRun(1, MS_DIR_FORWARD, i);
      msDCRun(2, MS_DIR_BACKWARD, i);
      delay();
    }
    for (i=255; i!=0; i--)
    {
      msDCRun(1, MS_DIR_FORWARD, i);
      msDCRun(2, MS_DIR_BACKWARD, i);
      delay();
    }

    /* Backward */
    for (i=0; i<255; i++)
    {
      msDCRun(1, MS_DIR_BACKWARD, i);
      msDCRun(2, MS_DIR_FORWARD, i);
      delay();
    }
    for (i=255; i!=0; i--)
    {
      msDCRun(1, MS_DIR_BACKWARD, i);
      msDCRun(2, MS_DIR_FORWARD, i);
      delay();
    }

    /* Release */
    msDCRun(1, MS_DIR_RELEASE, 0);
    msDCRun(2, MS_DIR_RELEASE, 0);
}
```

Servo Motor Example

The following will drive a single servo motor (connected on the Servo 1 header pin) across the entire movement range of the device. In the case of a fixed range servo, this should move across the entire motion range. In the case of a continuous rotation servo, the motor should change direction at mid point, since the speed and direction of the motor depends on the specific PWM frequency being generated.

```
/* Initialise Servo 1 */
msServoInit(1);

while(1)
{
  uint32_t volatile dutyCycle;

  /* Cover the entire servo motion range */
  for(dutyCycle=msServoGetMinDutyCycle();
      dutyCycle<msServoGetMaxDutyCycle();
      dutyCycle++)
  {
    msServoSetDutyCycle(1, dutyCycle);
  }
  for(dutyCycle=msServoGetMaxDutyCycle();
      dutyCycle>msServoGetMinDutyCycle();
      dutyCycle--)
  {
    msServoSetDutyCycle(1, dutyCycle);
  }
}
```