Demonstration Project
Arduino 16x2 LCD Shield

This project shows how to use the Arduino LCD shield with the Freescale FDRM-KL25Z. This document introduces:

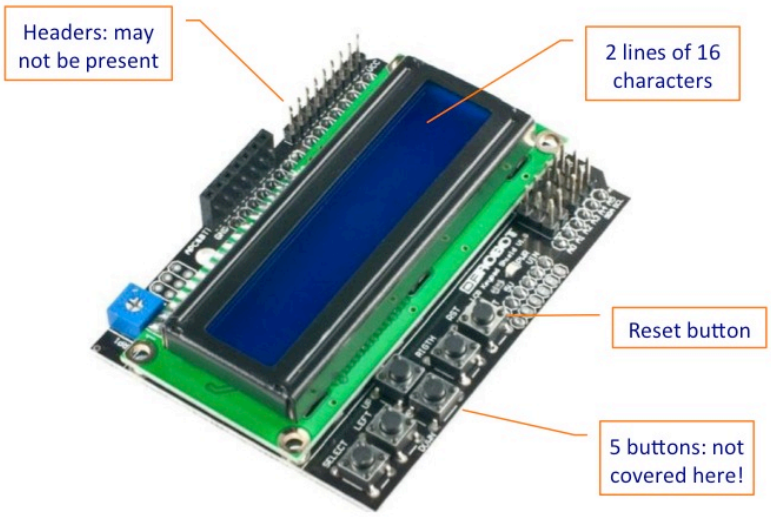
1. The Arduino LCD shield that can be used with the Freescale FRDM-KL25Z
2. The demonstration project
3. A software API for driving the LCD. This API is used in the demonstration project and is available for your use.

1. Principles

1.1 Introducing the Arduino Shield

The headers on the Freescale FRDM-KL25Z are intentionally pin compatible with the Arduino[1]. A shield is a circuit board that fits on top of the main processor board – so no wiring necessary. The design of the shield (the pins chosen) therefore fixes the ports that must be used to drive the circuit.

There is only one way to plug the shield on to the FRDM board: the touch pad should remain visible. The board is almost entirely covered.



1.2 Operation of The LCD

The LCD displays 2 lines of 16 characters. The LCD is controlled by an integrated circuit (IC) and interfaced to the KL25Z using a number of GPIO pins that are used as a databus. (All the necessary low-level coding has been done in the demonstration project).

Display Memory

The controller IC contains RAM (called display data RAM or DDRAM) that holds the characters shown on the display. Each character requires one byte, in ASCII[2]. However, rather than just 32 bytes, the DDRAM has 40 bytes for each line, of which only 16 are visible on the display.

Once data has been written to the DDRAM, the characters remain visible on the screen until the data is over-written or cleared.

[1] See https://www.arduino.cc/ if you want to know more about Arduino
[2] With a small number of exceptions: see below.

Data Entry Point

The address in the DDRAM at which new data is written can be set. This is known as the data entry point. In the API, the addresses are given by line (0 or 1) and character position (0 to 39).

Cursor

The LCD can optionally display a cursor. The cursor can be 'off', 'on', or 'blinking'. The cursor is often at the data entry point but it is possible for the two to be in different places.

Shifting

There are two varieties of shift. 1) The cursor can be shifted left or right; when this is done the entry points moves with it. If the cursor is shifted outside the visible part of the memory (usually positions 0 to 15) it can no longer be seen. Eventually, the addresses wrap around. 2) The visible region can be shifted right or left. This changes the part of the DDRAM that is displayed. For example, a shift right changes the visible part of the line from positions 0-15 to 1-16, then 2-17, 3-18 etc one step at a time. This appears to make the text move to the left (the opposite direction) but note that the content of the DDRAM does not change.

Note that both lines shift together. The shifting eventually wraps around and the lines wrap separately – that is, text from line 0 does not end up shown on line 1.

Entry Mode

Data is written to DDRAM in a particular entry mode: in the *increment (or decrement) mode*, the data entry point is incremented / decremented after each data transfer. This means that a sequence of data transfers are automatically placed in successive DDRAM locations. The usual mode is 'increment', so that the characters are written from left to write. In *shift increment (or decrement) mode*, the display position is shifted as well as the entry point being changed. With shift increment, the cursor stays in the same place on the display (the entry point increment and the screen shift produce opposite apparent movements) and the entered text scrolls to the left.

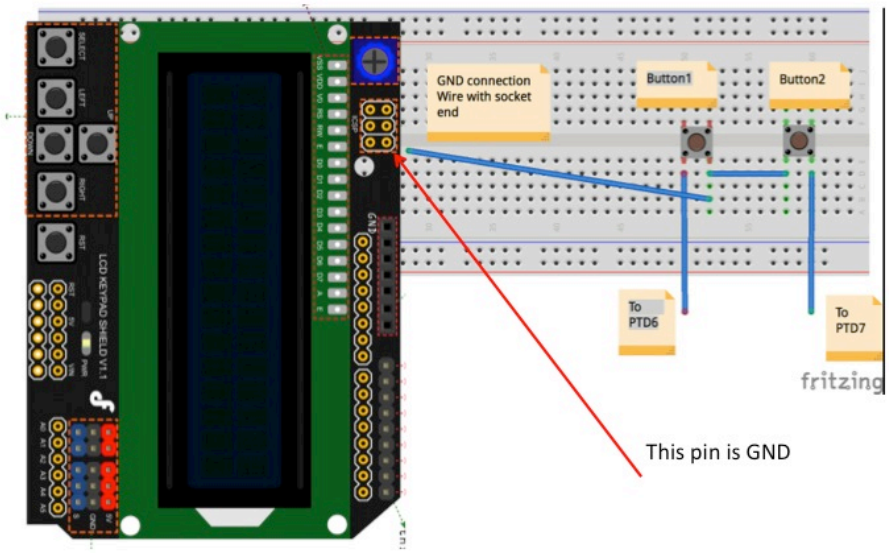
2. Demonstration Project

The demonstration project is intended to test the LCD API (see Section 3). You can

- Read the code of the demonstration project to see how the API is used.
- Run the demonstration project to understand the behaviour of the LCD, alongside the description here.

2.1 Hardware

The project uses an Arduino LCD shield and two buttons on the breadboard. The buttons on the shield are not used in this project.

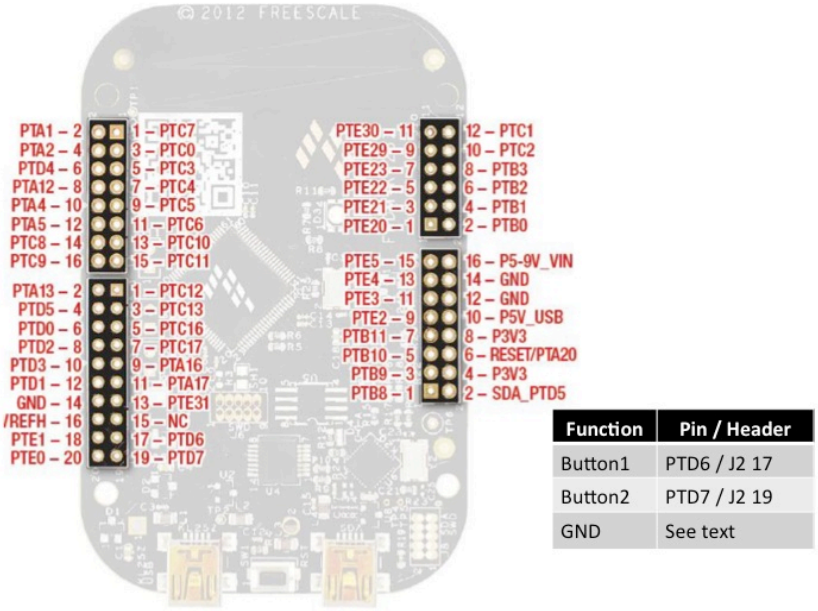


The buttons are connected as follows:

1. Button 1: PTD6, FRDM header J2, pin 17
2. Button 2: PTD7, FRDM header J2 pin 19

These connections are in the inner side of the FRDM connections: suitable wires need to inserted before the Arduino shield is plugged in, taking care not to leave any exposed copper that could contact the underside of the shield.

The GND connection for the button is also a problem as all the GND header pins are obscured by the shield. Fortunately, one of the pins on the shield (see diagram above) is connected to earth.



Function	Pin / Header
Button1	PTD6 / J2 17
Button2	PTD7 / J2 19
GND	See text

2.2 Functionality of the Project

The demonstration project has a cycle of 5 test cases. Button 1 cycles round the test cases. Line 0 of the display show the test case title. In each test case, button 2 demonstrates one of the LCD features: press button 2 as many times as you wish before cycling to the next test.

The test cases are:

Test		Description
1	Scroll text	Display text and, on button 2 presses, scroll right then left.
2	Scroll cursor	Display text and, on button 2 presses, scroll the cursor right then left
3	Increment entry mode	On each button 2 press enter characters in increment mode.
4	Shift entry mode	On each button 2 press, enter characters in shift entry mode. See how confusing this is.
5	Screen control	Cycle through screen and cursor off / on states with each button 2 press.

3. LCD Driver Software

The LCD driver software is the following files:

1. LCD.c : add this file to your project.
2. LCD.h: include this file wherever you wish to communicate with the LCD.

It should not be necessary to change any of the code.

3.1 Principles

The LCD driver is stateless and synchronous.

Stateless
The driver sends instructions and data to the LCD but does not hold any state data about the LCD. To the extent needed for a particular application, the API user needs to track the state of the LCD, including, for example, the cursor state, the entry point, the display shift, the entry mode.

Synchronous
The functions complete the data transfer to the LCD before returning. Unfortunately, the LCD device cannot run quite as fast as the KL25Z so it would be possible to send data too fast. To avoid this, the functions include some delays. Most LCD operations complete in c40 microseconds (around 800 KL25Z instructions), so only a small amount of time is wasted. With this design, it is quite safe to call the API functions as rapidly as you wish.

Two instructions take much longer: approximately 1.5ms. These instructions have a flag: if set a delay is included, wasting a lot of time. If you do not set this flag, then your program is responsible for making sure that no data is sent to the LCD device for at least 1.5ms after calling these functions. This could be done, for example, in a cyclic system but waiting until the next cycle or in an RTOS design by calling an RTOS delay function.

Using this approach, the data transfer is likely not happening as fast as it could. The LCD controller IC has a 'busy' signal that can be used for handshaking. Unfortunately, this function cannot be used on Arduino shield.

3.2 API Documentation

Initialisation
Initialisation at start-up or power-on.

```
void initLCD(void) ;
```

This initialisation function should be called as part of the initialisation sequence. All data is deleted. The entry point is line 0, position 0 with the cursor blinking at that location. There is no shift. The entry mode is 'increment'.

Entry Mode
The entry mode can be set.

```
typedef enum {M_Inc, M_Dec, M_IncShift, M_DecShift} LCDMode;
void lcdMode(LCDMode);
```

The entry mode can be set to increment or decrement the entry point (the DDRAM address) on each data transfer, with an optional shift.

Display and Cursor State
The display and the cursor can be switched on and off.

```
typedef enum {D_OFF, C_OFF, C_ON, C_BLINK} LCDState ;
void lcdCntrl(LCDState) ;
```

The display and cursor can be switched on or off: rather than providing independent control we assume that the cursor is off when the display is off. Value C_OFF turns the display on with cursor off; similarly C_ON and C_BLINK set the display on with different cursor states.

Reset

The LCD can be reset. There are two flavours and these functions take 1.5ms: see explanation about timing above.

```
void lcdClear(bool d) ;
void lcdHome(bool d) ;
```

The parameter d: when true, a 1.5ms delay is inserted after the call. These functions

- Both set the entry point and cursor to line 0, position 0.
- Clear: sets the DDRAM to 'space'.
- Reset: does not change DDRAM but resets any shift.

The behaviour concerning entry mode is not clear. My experience suggests it is best to assume that the entry mode needs to be set explicitly along with these calls.

Data Transfer

The following functions are used to write data to the DDRAM.

```
void setLCDAddress(uint8_t l, uint8_t p) ;
void writeLCDChar(char) ;
void writeLCDString(char *p) ;
```

The data entry address and data entered. Remember that the behaviour depends on the entry mode. Typically, you only need to set the start address and then can enter multiple characters. The 'write string' entry function repeated calls 'write char', for a maximum for 40 characters.

Shift

Both cursor or display can be shifted.

```
typedef enum {D_Left, D_Right} LCDDirection ;
void lcdShift(LCDDirection) ;
void cursorShift(LCDDirection) ;
```

The LCD shift is best understood by thinking of the display as a visible window on the DDRAM: the DDRAM does not change and the visible window shifts left and right. Shifting the window left makes the text appear to move right.