LCD Driver HAL design

This document is currently a Work In Progress

# Introduction

This document describes the designed Hardware Abstraction Layer for the LCD driver personal project. The reason and goal behind this HAL is to make interacting with the LCD display via the driver easier by abstracting the GPIO layout behind a set of mappings. This is partially to keep the GPIO layout configurable but also to keep the code more readable: By separating the logic of controlling the LCD from the logic of controlling the correct GPIO pins.

# Requirements

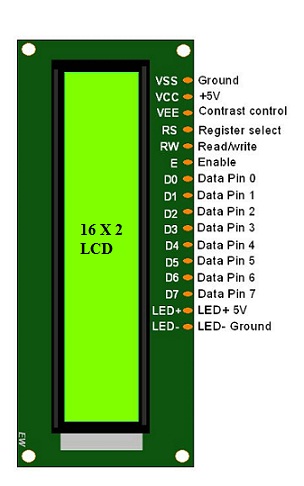
The following requirements were prioritised when designing the HAL:

The used GPIO pins must be configurable and changeable.

Driver uses should be able to write strings to the display by writing to the driver as if it were a file: Via a write system call.

# Hardware

The LCD display consists of 16 pins in total. Of these, the pins that are required for logic are pins RS to D7. In total, 11 GPIO pins are required.



The non-logic pins have the following functionality:

|  |  |
| --- | --- |
| Pin | Description |
| VSS | Ground pin for supply voltage |
| VCC | Source for 5V supply voltage |
| VEE | Contrast control. Controls contrast of displayed data. Usually connected to a pot-meter |
| LED+ | Source for 5V supply voltage for LED backlight |
| LED- | Ground for LED backlight |

The logic pins have the following functionality:

|  |  |
| --- | --- |
| Pin | Description |
| RS | Toggles between data modes. **Low** for instructions and **high** for characters |
| RW | Toggles between reading or writing to display. **Low** for writing and **high** for reading |
| E | Feed pin. Set to **high** after setting all data pins accordingly. |
| D0 | Data pin for bit 0 -> 0000000**x** |
| D1 | Data pin for bit 1 -> 000000**x**0 |
| D2 | Data pin for bit 2 -> 00000**x**00 |
| D3 | Data pin for bit 3 -> 0000**x**000 |
| D4 | Data pin for bit 4 -> 000**x**0000 |
| D5 | Data pin for bit 5 -> 00**x**00000 |
| D6 | Data pin for bit 6 -> 0**x**000000 |
| D7 | Data pin for bit 7 -> **x**0000000 |

The LCD display can receive 2-types of data on it’s data pins: characters, and instructions.

Characters are ASCII-encoded characters which are displayed on the screen when written. The characters are encoded as follows:

A picture containing text, crossword puzzle, receipt

Description automatically generated

Instructions are commands that can be send to the LCD. The instructions are encoded as follows:

A picture containing calendar

Description automatically generated

Note: The LCD is also capable of operating in a mode where only 4 of the 8 data bits are used. For this HAL design all 8 data bits are used.

# Low-level HAL design

Drivers should be able to handle the different logic pins like a 11-bit register. The low-level HAL layer should be responsible for translating a 11-bit value into high/low values on the different assigned GPIO pins.

The following table shows how each pin on the LCD is mapped to a bit in the 11-bit value. RS is the least significant bit and D7 is the most significant bit.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | E | RW | RS |

The user of the low-level HAL can specify a 11-bit value and feed this to the Set\_display\_pins *()* and  *Clear\_display\_pins()* functions. These functions will act as a mapping between the specific bits and connected GPIO pins, and set each used GPIO pin to a high or low value accordingly.

The *Set\_display\_pins ()* and *Clear\_display\_pins()* functions will take the 11-bit value and transform it into a 32-bit value which will then be written to the GPIO pin output set or GPIO pin output clear registers of the BCM2835 chip respectively:

*Set\_display\_pins(uint16\_t value)*: Sets the GPIO pins mapped to the first 11 bits to high accordingly. (only bits set to 1 will be set)

*Set\_display\_pins(uint16\_t value)*: Sets the GPIO pins mapped to the first 11 bits to low accordingly. (only bits set to 1 will be cleared)

To write a character to a display. The driver goes through the following steps:

1. Set the data, E, RW and RS pins to low (reset previous state)
2. Set the RS pin to high (character mode)
3. Set the RW pin to low (write mode)
4. Set the 8 data pins accordingly
5. Set the E pin to high (enable)

The low-level HAL will provide macros for some of these steps. Such as settings the RS, RW and E pins.

#define MODE\_CHARACTER 0b00000000001

#define ENABLE\_BIT 0b00000000100

#define DATA\_BITS\_OFFSET 3

For each of the above steps, the 11-bit value representing the state of the LCD pins will be mapped to a 32-bit value which is written into the GPIO pin output set or GPIO pin output clear registers.

These are then used by higher-level functions so that eventually user-space code will be able to write anything to the LCD driver by calling a “write” system call on the developed character device driver. The following higher-level functions are used for this:

*Write\_character(char a)*: Goes through the steps laid-out above to print a single character to the screen.

*Write\_string(char \*str)*: Iterates through the C-style string, calling Write\_character() on each individual character.

*Clear\_display()*: Clears the LCD screen.

# Resources:

<https://www.youtube.com/watch?v=hZRL8luuPb8>

<https://doc.lagout.org/electronics/lcd/instr.pdf>

# Research questions:

## is encoding used by lcd driver and linux for characters the same?

The following encoding scheme used by the LCD driver appears to be in ASCII-encoding:

<https://mil.ufl.edu/3744/docs/lcdmanual/characterset.html>

ASCII-encoding also appears to be easily usable in C++:

<https://linuxhint.com/use-ascii-cpp/>