F28HS Hardware-Software Interface: Systems Programming

Hans-Wolfgang Loidl

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh

Semester 2 — 2025/26

[0]No proprietary software has been used in producing these slides

Outline

HERIOT WATT UNIVERSITY

Tutorial 5: Programming an LCD Display

This tutorial will focus on programming a simple output device:

an 16x2 LCD display using an Hitachi HD44780U controller

This will be an exercise of controlling a device slightly more complicated than the LED and button devices so far.
The principles of programming are the same as before.

Overview

We will cover:

1. Connecting an LCD display to the RPi2
2. Low-level interface in assembler (`digital_write`)
3. Medium-level interface in C (`lcd_command`)
4. Sending characters and strings (`lcd_putchar`, `lcd_puts`)
5. Character data (defining own characters)

Acknowledgements

The code in this tutorial is mostly taken directly from the **wiringPi** library for the Raspberry Pi, by **Gordon Henderson**.
If you have downloaded the sources, you can look-up examples in the directory `wiringPi/examples` (e.g. `lcd.c`) and the code for the LCD functions in `wiringPi/devLib` (also `lcd.c`)

```
* wiringPi:
* Arduino look-a-like Wiring library for the Raspberry Pi
* Copyright (c) 2012-2015 Gordon Henderson
* Additional code for pwmSetClock by Chris Hall <chris@kchall.plus.com>
*
* Thanks to code samples from Gert Jan van Loo and the
* BCM2835 ARM Peripherals manual, however it's missing
* the clock section /grr/mutter/
*********************************************************************
* This file is part of wiringPi:
* https://projects.drogon.net/raspberry-pi/wiringpi/
```

The version of the LCD code on this slide set is by Adam T. Sampson and Hans-Wolfgang Loidl.

Acknowledgements

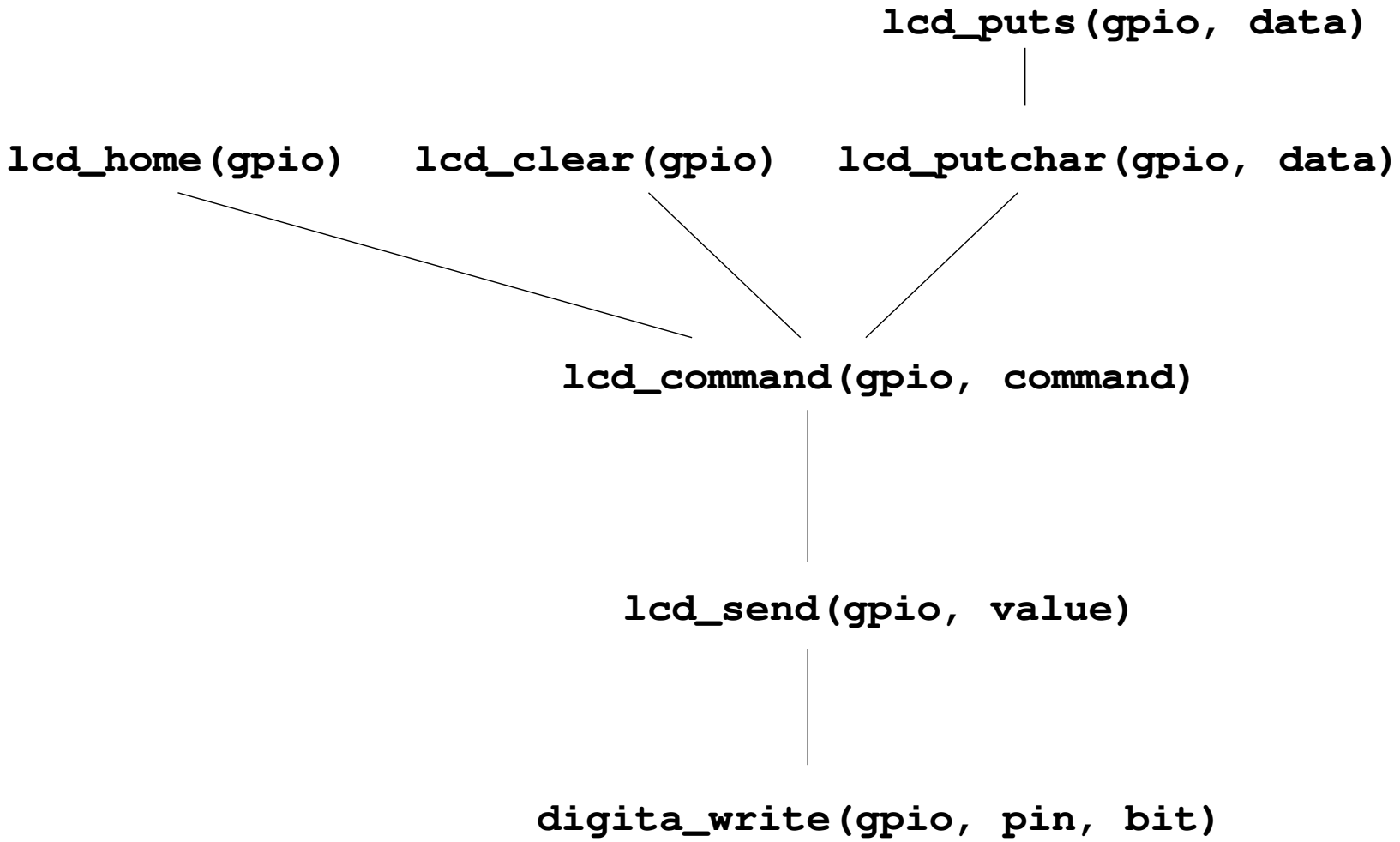
The code in this tutorial is mostly taken directly from the **wiringPi** library for the Raspberry Pi, by **Gordon Henderson**.
If you have downloaded the sources, you can look-up examples in the directory `wiringPi/examples` (e.g. `lcd.c`) and the code for the LCD functions in `wiringPi/devLib` (also `lcd.c`)

```
* wiringPi:
* Arduino look-a-like Wiring library for the Raspberry Pi
* Copyright (c) 2012-2015 Gordon Henderson
* Additional code for pwmSetClock by Chris Hall <chris@kchall.plus.com>
*
* Thanks to code samples from Gert Jan van Loo and the
* BCM2835 ARM Peripherals manual, however it's missing
* the clock section /grr/mutter/
*********************************************************************
* This file is part of wiringPi:
* https://projects.drogon.net/raspberry-pi/wiringpi/
```

The version of the LCD code on this slide set is by Adam T. Sampson and Hans-Wolfgang Loidl.

Function dependencies

Here is a simple picture of the dependencies of the API functions:



NB: only the lowest level, `digital_write` is in assembler, the rest is in C

LCD commands

We need some constant definitions and boilerplate code:
Here is a list of instructions for the Hitachi HD44780U controller:

```
#define LCD_CLEAR       0x01
#define LCD_HOME        0x02
#define LCD_ENTRY       0x04
#define LCD_CTRL        0x08
#define LCD_CDSHIFT     0x10
#define LCD_FUNC        0x20
#define LCD_CGRAM       0x40
#define LCD_DGRAM       0x80
```

[0] See Table 6 and Figure 11 in the HD4478 Technical Reference

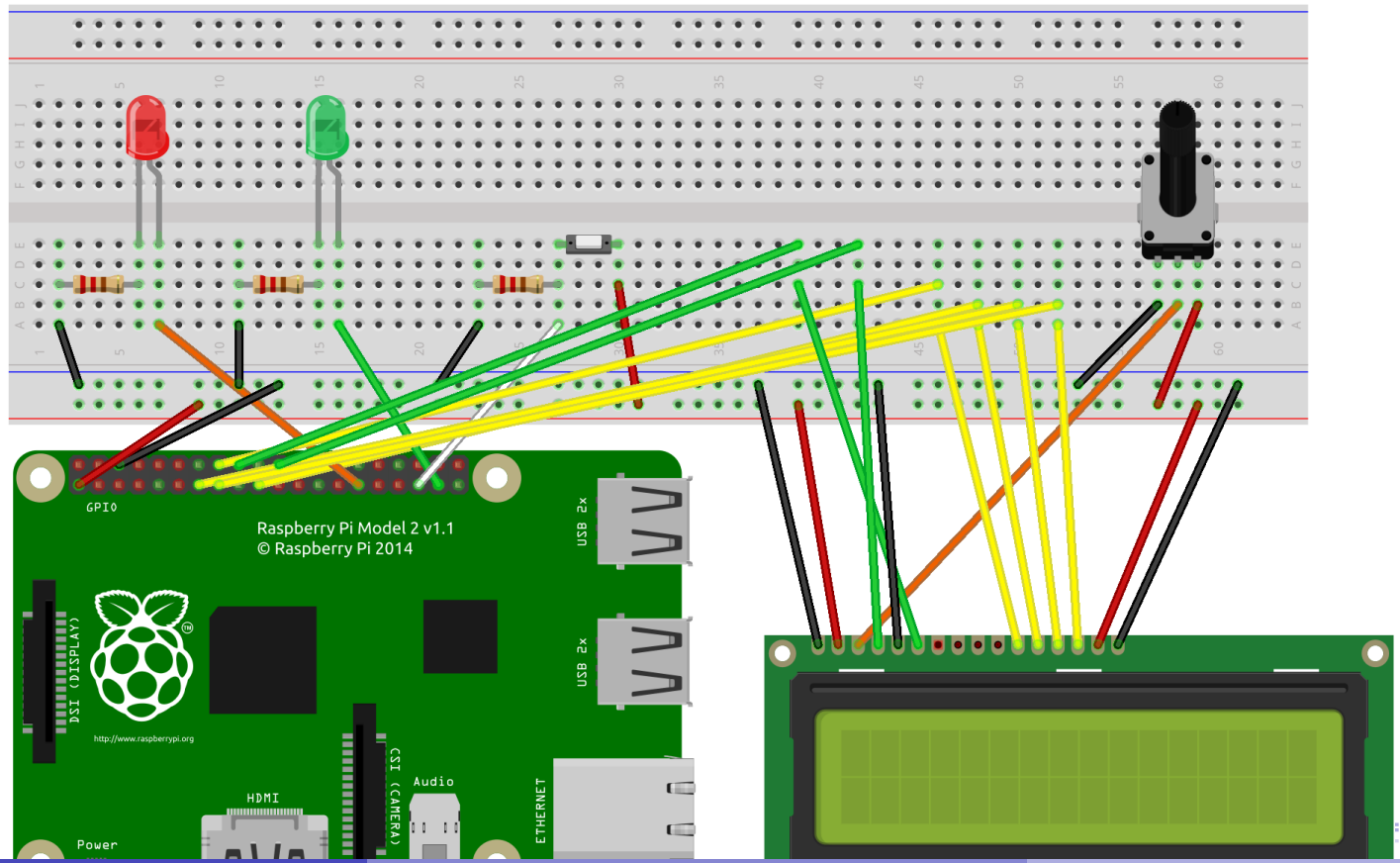
1. The wiring

Pin 19 (White) Button
Pin 26 (Green) green LED
Pin 5 (Orange) red LED

All power wires are red
All ground wires are black

LCD Data (Yellow)
GPIO 23 <-> LCD 11
GPIO 10 <-> LCD 12
GPIO 27 <-> LCD 13
GPIO 22 <-> LCD 14

LCD Control (Green)
GPIO 25 <-> LCD 4
GPIO 24 <-> LCD 6



The wiring: encoded

To encode this wiring in the program we define:

```
// GPIO pins used by the LCD
#define LCD_PIN_RS 25
#define LCD_PIN_E 24
#define LCD_PIN_D4 23
#define LCD_PIN_D5 10
#define LCD_PIN_D6 27
#define LCD_PIN_D7 22
```

2. Low-level Assembler interface

This code is essentially the same as in the blinking LED example, i.e. we want to “send” one bit to a pin that’s an argument to the interface:

- Set the mode of the pin to output (before calling the function)
- Identify the register and bit to write to
- Write one bit (1) into this location
- It is recommended that you use inline assembler to implement this function

2. Low-level Assembler interface

For CW2 you need to complete the code in `lcd-binary.c`:

```
void digital\_write (uint32_t *gpio, int pin, int value) {
 asm volatile(/* inline assembler version of setting/clearing LED to
     ouput" */
  "\tLDR␣R2,␣%[gpio]\n"
  /* COMPLETE inline assembler code  */
 : /* COMPLETE Output operands */
 : [gpio] "m" (gpio)
   /* COMPLETE Input operands */
 : "R2" /* COMPLETE Clobbers */ );
}
```

The above assembler code needs to implement the functionality of the C code from Tutorial LED (see `tut_led.c`).

2. Low-level Assembler interface

For CW2 you need to complete the code in `lcd-binary.c`:

```
void digital\_write (uint32_t *gpio, int pin, int value) {
 asm volatile(/* inline assembler version of setting/clearing LED to
     ouput" */
  "\tLDR␣R2,␣%[gpio]\n"
  /* COMPLETE inline assembler code  */
 : /* COMPLETE Output operands */
 : [gpio] "m" (gpio)
   /* COMPLETE Input operands */
 : "R2" /* COMPLETE Clobbers */ );
}
```

The above assembler code needs to implement the functionality of the C code from Tutorial LED (see `tut_led.c`).

3. Medium-level interface

Using a 4-pin wiring setup we send one “nibble” (a half-byte) along 4 pins using `digital_write`:

```
static void lcd_send4(volatile uint32_t *gpio, int value) {
    // Set the data lines
    digital_write(gpio, LCD_PIN_D7, (value >> 3) & 1);
    digital_write(gpio, LCD_PIN_D6, (value >> 2) & 1);
    digital_write(gpio, LCD_PIN_D5, (value >> 1) & 1);
    digital_write(gpio, LCD_PIN_D4, value & 1);

    // Pulse the E line
    digital_write(gpio, LCD_PIN_E, 1);
    usleep(50);
    digital_write(gpio, LCD_PIN_E, 0);
    usleep(50);
}
```

Send a byte

Sending a byte involves sending two nibbles:

```
static void lcd_send(volatile uint32_t *gpio, int value) {
    // Send the top four bits
    lcd_send4(gpio, (value >> 4) & 0xf);

    // Send the bottom four bits
    lcd_send4(gpio, value & 0xf);
}
```

Send a command

Sending a command works like sending a byte, except that we only need 4 bits to encode the command:

```
void lcd_command(volatile uint32_t *gpio, int command) {
    digital_write(gpio, LCD_PIN_RS, 0);
    lcd_send(gpio, command);

    if (command == LCD_HOME) {
        usleep(2000);
    }
}
```

Specific commands

This is an example of implementing the Clear command:

```
void lcd_clear(volatile uint32_t *gpio) {
  lcd_command(gpio, LCD_CLEAR);
}
```

Another example of implementing the Home command:

```
void lcd_home(volatile uint32_t *gpio) {
  lcd_command(gpio, LCD_HOME);
}
```

4. Sending characters and strings

Displaying a character is done by sending a byte after setting the register select (RS) signal to 1:

```
void lcd_putchar(volatile uint32_t *gpio, char data) {
    digital_write(gpio, LCD_PIN_RS, 1);
    lcd_send(gpio, data);
}
```

Display a string

Once we can send characters, we only need a loop on top of it to send entire strings:

```
void lcd_puts(volatile uint32_t *gpio, const char *data) {
    const char *p = data;
    while (*p != '\0') {
        lcd_putchar(gpio, *p++);
    }
}
```

Write to a row

Writing to a specific row of the display involves a bitwise or operation:

```
void lcd_write_row(volatile uint32_t *gpio, int row, const char *data)
    {
  if (row<0 || row>2) return;
  lcd_command(gpio, (row==1) ? LCD_DGRAM : LCD_DGRAM | 0x40);
  lcd_puts(gpio, data);
}
```

From the HD44780U manual (fig4, p11):

Display position	1	2	3	4	5	.....	39	40
DDRAM address (hexadecimal)	00	01	02	03	04	.....	26	27
	40	41	42	43	44	.....	66	67

Figure 4 2-Line Display

5. Putting things together

In the main function we:

- Memory-map the GPIO address into user space (`gpio`)
- Initialise the connection to the LCD display `lcd_init`
- Set-up the LCD display using `lcd_clear` and `lcd_home`
- Write “Hello World” using `lcd_puts`

Application code from `lcd-hello.c`

```
int main (int argc, char *argv[])
{
  ...
  gpio = (uint32_t *)mmap(0, BLOCK_SIZE, PROT_READ|PROT_WRITE,
      MAP_SHARED, fd, gpiobase) ;
  ...
  // initialise LCD display
  lcd_init(gpio);

  lcd_command(gpio, LCD_DGRAM);
  lcd_puts(gpio, "Hello␣world!");

  lcd_command(gpio, LCD_DGRAM | 0x40);
  lcd_puts(gpio, "␣␣␣␣␣␣␣␣␣␣␣␣");

  waitForEnter () ;
  lcd_command(gpio, LCD_CLEAR);   lcd_command(gpio, LCD_HOME);
```

[0] See the sample program `lcd-hello.c` in this gitlab repo

Conclusions

- Controlling an LCD display requires communication on several wires/pins.
- The lowest level (`digital_write`) is the same as for the LED.
- A software stack of functions enables writing strings to the LCD display.
- Implementation of the stack requires information from the hardware manual.
- The low-level code depends on the concrete driver.
- In your program you just need the top level of the hierarchy (`lcd_init`, `lcd_puts` etc).

[0] See the sample program `lcd-hello.c` in this gitlab repo