How to connect the LCD screen

The Group (Andrei Vacariu, Kevin Ang, Allan Kuan)

This guide will guide the reader through picking GPIO pins, setting their mode using device tree overlays, connecting the LCD screen, and running some example code (using our LCD control library).

1. Choosing pins

Many of the pins on the Beaglebone Black are used by other device trees, and there could be conflicts when trying to set modes on the chosen pins. We have found that a safe choice for connecting the LCD screen is using pins 11 - 16 on P8.

Although the pin information for the Beaglebone Black is available in its manual, the needed information is spread across multiple tables. There is a compiled table of all the useful information provided by Derek Molloy here:
https://github.com/derekmolloy/boneDeviceTree/tree/master/docs

For Pins 11 - 16, we need 2 pieces of information: GPIO number and offset. Looking at the table linked above, we get the following information:

Pin number	GPIO number	Offset
P8_11	45	0x034
P8_12	44	0x030
P8_13	23	0x024
P8_14	26	0x028
P8_15	27	0x03c
P8_16	46	0x038

The GPIO numbers are used for setting direction and values on the pins using the `/sys` filesystem, and the offsets are used to set pin modes in the device tree overlay.

2. Setting the pin modes

Once you've picked your pins, you need to set their mode to GPIO using a device tree overlay. These overlays are loaded at runtime by the kernel.

This is a sample overlay for the pins we have chosen above:

```
/*
 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

/dts-v1/;
/plugin/;

/ { compatible = "ti,beaglebone", "ti,beaglebone-black";
       /* identification */
       part-number = "LCD-1602-GPIO";
       version = "00A0";

       exclusive-use =
              /* the pin header uses */
              "P8.16",       /* LCD: DB7 */
              "P8.15",       /* LCD: DB6 */
              "P8.14",       /* LCD: DB5 */
              "P8.13",       /* LCD: DB4 */
              "P8.12",       /* LCD: RS */
              "P8.11";       /* LCD: E */

       fragment@0 {
       target = <&am33xx_pinmux>;
       __overlay__ {
              pinctrl_test: LCD_1602_GPIO_Pins {
              pinctrl-single,pins = <
                     0x038 0x17     /* P8_16, OUTPUT_PULLUP | MODE7 */
                     0x03c 0x17     /* P8_15, OUTPUT_PULLUP | MODE7 */
                     0x028 0x17     /* P8_14, OUTPUT_PULLUP | MODE7 */
                     0x024 0x17     /* P8_13, OUTPUT_PULLUP | MODE7 */
                     0x030 0x17     /* P8_12, OUTPUT_PULLUP | MODE7 */
                     0x034 0x17     /* P8_11, OUTPUT_PULLUP | MODE7 */
              >;
              };
       };
       };

       fragment@1 {
              target = <&ocp>;
              __overlay__ {
                     test_helper: helper {
                            compatible = "bone-pinmux-helper";
                            pinctrl-names = "default";
                            pinctrl-0 = <&pinctrl_test>;
                            status = "okay";
                     };
              };
       };
};
```

The `exclusive-use` section of the device tree states that this overlay will fail to load if there are other overlays who want exclusive use of the same pins. This is helpful in troubleshooting why your pin modes aren't being set properly since there will be a related error message in `/var/log/syslog` when you try to load the device tree.

Change the pin address offsets and numbers in the device tree overlay above to what you have chosen to use. The offsets are the first column under `pinctrl-single,pins`.

Compiling and loading the overlay

1. Save the overlay to `LCD-1602-GPIO-00A0.dts`
2. Execute the following command in a terminal to compile the device tree
   `dtc -O dtb -o LCD-1602-GPIO-00A0.dtbo -@ LCD-1602-GPIO-00A0.dts`
3. Copy the compiled file to `/lib/firmware`
   `cp LCD-1602-GPIO-00A0.dtbo /lib/firmware/`
4. Load the device tree (there should be no output from this command)
   `echo LCD-1602-GPIO > /sys/devices/bone_capemgr.9/slots`
5. Check if device tree was loaded
   `cat /sys/devices/bone_capemgr.9/slots`
6. If there are any issues, look in `/var/log/syslog` for clues. You might need to choose different pins.
7. Check if pins were set properly. This is an example using pins choices above. Adding an 8 in front is necessary because the values in this file are the full addresses not just the offsets we had in the device tree.
   `grep "83c" /sys/kernel/debug/pinctrl/44e10800.pinmux/pins`
   Output should be:
   `pin 15 (44e1083c) 00000017 pinctrl-single`

Loading the overlay at boot

The Debian install on the Beaglebone Black does not allow loading a device tree overlay using `/boot/uboot/uEnv.txt`, so we'll need to get the cape manager to load it as soon as possible after booting. We do this by adding `CAPE=LCD-1602-GPIO` to `/etc/default/capemgr`.

3. Connecting the LCD display

Now that we've got the pins set up to the correct modes, we'll need to connect the pins to the LCD screen and a potentiometer. Use the following table to connect the pins (adjusting for your pin choices).

LCD pin	Connect to
VSS	GND (P9_1)
VDD	5v (P9_5)
V0	Potentiometer (adjusts contrast)
RS	P8_12
RW	GND
E	P8_11
D4	P8_13
D5	P8_14
D6	P8_15
D7	P8_16
A	5v
K	GND

4. Running the example code

Save the following code in `lcd-example.c`. Use the GPIO numbers you have chosen in the first step for the pins in each struct. The first element of each struct is not currently used by the library, but is useful for keeping track of which pins the numbers are associated with.

```
#include <string.h>
#include <unistd.h>
#include "lcd.h"

int main() {
	struct pin pins[NUM_PINS] = {
		{"P_11", 45}, // E
		{"P_12", 44}, // RS
		{"P_13", 23}, // D4
		{"P_14", 26}, // D5
		{"P_15", 47}, // D6
		{"P_16", 46}  // D7
	};

	screen_init(pins);
	screen_clear(pins);

	char *to_write = "Hello, world!";
	screen_print(pins, to_write, strlen(to_write), 0);
```

```
        char *to_write2 = "test";
        screen_print(pins, to_write2, strlen(to_write2), 1);

        sleep(3);

        char *to_write3 = "something";
        screen_print(pins, to_write3, strlen(to_write3), 0);

        sleep(2);
        screen_clear(pins);
        screen_deinit(pins);

        return 0;
}
```

Compile the code against the provided library:

```
gcc -std=gnu99 -c lcd.c
gcc -std=gnu99 -c lcd-example.c
gcc -std=gnu99 -o lcd-example lcd-example.o lcd.o
```

At this point when you execute `lcd-example`, the code should print “Hello, world!” in the first line, “test” on the second line, wait 3 seconds, print “something” on the first line, wait 2 seconds, and then clear the screen.

Troubleshooting

- If you notice that the pin modes haven’t been set to 17, and there’s nothing suspicious in `/var/log/syslog` when loading the overlay, try different pins
- Check pin modes:
  `grep "83c" /sys/kernel/debug/pinctrl/44e10800.pinmux/pins`
  Output should be:
  `pin 15 (44e1083c) 00000017 pinctrl-single`
- If you get “no such file or directory” when trying to load the overlay, check `/var/log/syslog` for messages about pin conflicts
- Make sure you have the right pins in the `exclusive-use` section of the overlay. This is useful for detecting conflicts.
- Double-check your pin numbers and GPIO numbers (it’s quite likely to have typos)
- Do you have the pins in the right order when using the library? Look at the comments beside the pins in the example code to see which GPIO pin is for which LCD pin
- Double-check cables.
- Adjust contrast using potentiometer (maybe it’s too high or too low and you can’t see the characters)

Notes

The LCD control library is called `lcd.h` and is available in the zip file.

Copyright of document