LP5521/LP5523/LP55231/LP5562/LP8501 Common Driver

Authors: Milo(Woogyom) Kim <milo.kim@ti.com>

Description

LP5521, LP5523/55231, LP5562 and LP8501 have common features as below.

> Register access via the I2C Device initialization/deinitialization Create LED class devices for multiple output channels Device attributes for user-space interface Program memory for running LED patterns

The LP55xx common driver provides these features using exported functions.

> lp55xx_init_device() / lp55xx_deinit_device() lp55xx_register_leds() / lp55xx_unregister_leds() lp55xx_regsister_sysfs() / lp55xx_unregister_sysfs()

(Driver Structure Data)

In lp55xx common driver, two different data structure is used.

- lp55xx_led
  control multi output LED channels such as led current, channel index.
- lp55xx_chip
  general chip control such like the I2C and platform data.

For example, LP5521 has maximum 3 LED channels. LP5523/55231 has 9 output channels:

```
lp55xx_chip for LP5521 ... lp55xx_led #1
                           lp55xx_led #2
                           lp55xx_led #3

lp55xx_chip for LP5523 ... lp55xx_led #1
                           lp55xx_led #2
                                 .
                                 .
                           lp55xx_led #9
```

(Chip Dependent Code)

To support device specific configurations, special structure 'lpxx_device_config' is used.

- Maximum number of channels
- Reset command, chip enable command
- Chip specific initialization
- Brightness control register access
- Setting LED output current
- Program memory address access for running patterns
- Additional device specific attributes

(Firmware Interface)

LP55xx family devices have the internal program memory for running various LED patterns.

This pattern data is saved as a file in the user-land or hex byte string is written into the memory through the I2C.

LP55xx common driver supports the firmware interface.

LP55xx chips have three program engines.

To load and run the pattern, the programming sequence is following.

1. Select an engine number (1/2/3)
2. Mode change to load
3. Write pattern data into selected area
4. Mode change to run

The LP55xx common driver provides simple interfaces as below.

select_engine:
  Select which engine is used for running program
run_engine:

Start program which is loaded via the firmware interface

firmware:

Load program data

In case of LP5523, one more command is required, 'enginex_leds'. It is used for selecting LED output(s) at each engine number. In more details, please refer to 'leds-lp5523.txt'.

For example, run blinking pattern in engine #1 of LP5521:

```
echo 1 > /sys/bus/i2c/devices/xxxx/select_engine
echo 1 > /sys/class/firmware/lp5521/loading
echo "4000600040FF6000" > /sys/class/firmware/lp5521/data
echo 0 > /sys/class/firmware/lp5521/loading
echo 1 > /sys/bus/i2c/devices/xxxx/run_engine
```

For example, run blinking pattern in engine #3 of LP55231

Two LEDs are configured as pattern output channels:

```
echo 3 > /sys/bus/i2c/devices/xxxx/select_engine
echo 1 > /sys/class/firmware/lp55231/loading
echo "9d0740ff7e0040007e00a0010000" > /sys/class/firmware/lp55231/data
echo 0 > /sys/class/firmware/lp55231/loading
echo "000001100" > /sys/bus/i2c/devices/xxxx/engine3_leds
echo 1 > /sys/bus/i2c/devices/xxxx/run_engine
```

To start blinking patterns in engine #2 and #3 simultaneously:

```
for idx in 2 3
do
echo $idx > /sys/class/leds/red/device/select_engine
sleep 0.1
echo 1 > /sys/class/firmware/lp5521/loading
echo "4000600040FF6000" > /sys/class/firmware/lp5521/data
echo 0 > /sys/class/firmware/lp5521/loading
done
echo 1 > /sys/class/leds/red/device/run_engine
```

Here is another example for LP5523.

Full LED strings are selected by 'engine2_leds':

```
echo 2 > /sys/bus/i2c/devices/xxxx/select_engine
echo 1 > /sys/class/firmware/lp5523/loading
echo "9d80400004ff05ff437f0000" > /sys/class/firmware/lp5523/data
echo 0 > /sys/class/firmware/lp5523/loading
echo "111111111" > /sys/bus/i2c/devices/xxxx/engine2_leds
echo 1 > /sys/bus/i2c/devices/xxxx/run_engine
```

As soon as 'loading' is set to 0, registered callback is called. Inside the callback, the selected engine is loaded and memory is updated. To run programmed pattern, 'run_engine' attribute should be enabled.

The pattern sequence of LP8501 is similar to LP5523.

However pattern data is specific.

Ex 1) Engine 1 is used:

```
echo 1 > /sys/bus/i2c/devices/xxxx/select_engine
echo 1 > /sys/class/firmware/lp8501/loading
echo "9d0140ff7e0040007e00a001c000" > /sys/class/firmware/lp8501/data
echo 0 > /sys/class/firmware/lp8501/loading
echo 1 > /sys/bus/i2c/devices/xxxx/run_engine
```

Ex 2) Engine 2 and 3 are used at the same time:

```
echo 2 > /sys/bus/i2c/devices/xxxx/select_engine
sleep 1
echo 1 > /sys/class/firmware/lp8501/loading
echo "9d0140ff7e0040007e00a001c000" > /sys/class/firmware/lp8501/data
echo 0 > /sys/class/firmware/lp8501/loading
sleep 1
echo 3 > /sys/bus/i2c/devices/xxxx/select_engine
sleep 1
echo 1 > /sys/class/firmware/lp8501/loading
echo "9d0340ff7e0040007e00a001c000" > /sys/class/firmware/lp8501/data
echo 0 > /sys/class/firmware/lp8501/loading
sleep 1
echo 1 > /sys/class/leds/d1/device/run_engine
```

('run_engine' and 'firmware_cb')

The sequence of running the program data is common.

But each device has own specific register addresses for commands.

To support this, 'run_engine' and 'firmware_cb' are configurable in each driver.

run_engine:

Control the selected engine

firmware_cb:

The callback function after loading the firmware is done.

Chip specific commands for loading and updating program memory.

(Predefined pattern data)

Without the firmware interface, LP55xx driver provides another method for loading a LED pattern. That is 'predefined' pattern.

A predefined pattern is defined in the platform data and load it(or them) via the sysfs if needed.

To use the predefined pattern concept, 'patterns' and 'num_patterns' should be configured.

Example of predefined pattern data:

```
/* mode_1: blinking data */
static const u8 mode_1[] = {
              0x40, 0x00, 0x60, 0x00, 0x40, 0xFF, 0x60, 0x00,
              };

/* mode_2: always on */
static const u8 mode_2[] = { 0x40, 0xFF, };

struct lp55xx_predef_pattern board_led_patterns[] = {
      {
              .r = mode_1,
              .size_r = ARRAY_SIZE(mode_1),
      },
      {
              .b = mode_2,
              .size_b = ARRAY_SIZE(mode_2),
      },
}

struct lp55xx_platform_data lp5562_pdata = {
...
      .patterns      = board_led_patterns,
      .num_patterns  = ARRAY_SIZE(board_led_patterns),
};
```

Then, mode_1 and mode_2 can be run via through the sysfs:

```
echo 1 > /sys/bus/i2c/devices/xxxx/led_pattern    # red blinking LED pattern
echo 2 > /sys/bus/i2c/devices/xxxx/led_pattern    # blue LED always on
```

To stop running pattern:

```
echo 0 > /sys/bus/i2c/devices/xxxx/led_pattern
```