TM1628 7-Segment LED Driver Code Development Guide

Prepared By: Venkatesh M **Email:** venkatesh.muninagaraju@essae.com

TM1628 7-Segment LED Driver Code Development Guide 1
1. Introduction 2
2. 7-Segment Display Overview 2
3. Hardware Connection Details 3
4. Device Tree (DTS) Configuration 3
5. Build Configuration 4
5.1 Makefile Entry 4
5.2 Kernel Configuration (Kconfig) 4
6. Kernel Build and Deployment 4
6.1. Set Up the Environment 4
6.2. Configure the Kernel 5
Then run the configuration menu: 5
➢ ..... **Navigate through the menu as follows:** 5
6.3. Compile the Kernel/Module 5
6.4. Deploy the Module or Built-in Image 5
➢ ..... **If Built-In:** 6
7. Module Loading and Verification 6
8. Sysfs Interfaces and Usage 6
9. Driver Source Code Overview 7
9.1 Header and Global Declarations 7
9.2 Low-Level Functions 8

9.3 Display Functions 8
9.4 Key Scanning Section 8
9.5 Sysfs Attribute Wrappers 8
__9.6 Platform Driver Probe and Remove 8
10. Appendix: Complete Source Code 8
11. Conclusion 30

This document provides a detailed, step-by-step explanation of the development of a platform driver for the TM1628 7-segment LED display. It covers hardware connections, device tree settings, kernel build configuration, module loading, and driver usage via sysfs interfaces.

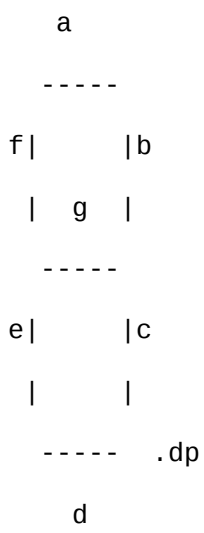
1. Introduction

The TM1628 is an LED driver that controls a multi-digit 7-segment display using a simple three-wire (STB, DIO, CLK) GPIO interface. This guide explains how to build and deploy a driver that:

- Drives the 7-segment display,
- Displays demo patterns and the current time,
- Supports a key-scanning feature with sysfs controls.

2. 7-Segment Display Overview

The 7-segment display consists of seven segments (labeled **a** through **g**) plus an optional decimal point (**dp**). The basic layout is illustrated below:



Each segment is controlled by a corresponding bit in the driver's digit map.

3. Hardware Connection Details

The TM1628 board is connected via an expansion connector. The following table details the mapping between the system's GPIO numbers, expansion connector pins, and TM1628 pins:kyuukukthfgbfvfvrrrrfref

System Signal	Expansion Connector Pin	TM1628 Pin
5V	4	5V
GPIO_IO18	12	STB
GPIO_IO19	35	DIN (DIO)
GPIO_IO21	40	CLK
GND	6	GND

Ensure that the power, ground, and control signals are connected as shown.

4. Device Tree (DTS) Configuration

Add a node for the TM1628 in your device tree. For example:

$ kernel_source_imx93/linux-imx/arch/arm64/boot/dts/freescale$ vi imx93-11x11-frdm.dts

```
/ {
    tm1628: tm1628@0 {
        compatible = "my,tm1628";
        stb-gpio = <&gpio2 18 GPIO_ACTIVE_HIGH>;
        dio-gpio = <&gpio2 19 GPIO_ACTIVE_HIGH>;
        clk-gpio = <&gpio2 21 GPIO_ACTIVE_HIGH>;
    };
};
```

This configuration tells the kernel which GPIOs are used to control the TM1628.

5. Build Configuration

5.1 Makefile Entry

Include the following line in your Makefile to compile the driver:

essae@rnd:~ kernel_source_imx93/linux-imx/drivers/leds$ vi Makefile

```
obj-$(CONFIG_LEDS_TM1628) += tm1628.o
```

5.2 Kernel Configuration (Kconfig)

Add a configuration entry to allow building the driver as a module or built-in:

essae@rnd:~ kernel_source_imx93/linux-imx/drivers/leds$ vi Kconfig

```
config LEDS_TM1628
    tristate "TM1628 LED driver"
    default m or y
    help
      This driver supports the TM1628 LED display via GPIO bit-banging.
      It will first display a demo pattern then update the display with the
current time.
```

6. Kernel Build and Deployment

Follow these steps to compile the kernel (or module) with the TM1628 driver and deploy it:

6.1. Set Up the Environment

```
$ . /opt/fsl-imx-wayland/6.6-scarthgap/environment-setup-armv8a-poky-linux
```

6.2. Configure the Kernel

```
$ make imx_v8_defconfig
```

Then run the configuration menu:

```
$ make menuconfig
```

➢ Navigate through the menu as follows:

- **General setup →**

  [*] Configure standard kernel features (expert users) – Enable this and exit.

- **Device Drivers →**

  GPIO Support →

  Enable /sys/class/gpio/... (sysfs int erface)

- **LED Support →**

  Select the TM1628 LED driver (choose as module <M> or built-in [*] as desired).

6.3. Compile the Kernel/Module

```
$ make -j$(nproc)
```

6.4. Deploy the Module or Built-in Image

➢ If Built as a Module:

Copy the compiled `tm1628.ko` to the target board's module directory:

```
$ sudo cp -r tm1628.ko /media/essae/root/lib/modules/6.6.36-lts-next-g3e42481c8760-dirty/kernel/drivers/leds/
```

➢ If Built-In:

Copy the kernel image to the boot partition and also update the device tree binary:

```
$ cp Image /path/to/boot/partition/
```

```
$ cp imx93-11x11-frdm.dtb /path/to/boot/partition/
```

Then sync the file system and reboot the board.

7. Module Loading and Verification

After building the driver, load it (if built as a module) using the modprobe command:

```
root@imx93frdm:~# modprobe /lib/modules/6.6.36-lts-next-g3e42481c8760-dirty/kernel/drivers/leds/tm1628.ko
```

Verify that the driver is loaded by checking the kernel log:

```
root@imx93frdm:~# dmesg | grep tm1628
```

8. Sysfs Interfaces and Usage

The driver creates several sysfs entries under `/sys/class/auxdisplay/`:

- **brightness**: Adjust display brightness (values 0–15).

Example:

- `echo 10 > /sys/class/auxdisplay/brightness`

- **time**: Enable or disable time mode.

Example:

- `echo on > /sys/class/auxdisplay/time`
- `echo off > /sys/class/auxdisplay/time`

- **display**: Write a custom string (e.g., "E.S.S.A.E.") to the display.

Example:

- `echo E.S.S.A.E. > /sys/class/auxdisplay/display`

- **displaymode_config**: Set display modes (e.g., "4x13", "5x12", "6x11", "7x10").

Examples:

- `echo "5x12" > /sys/class/auxdisplay/displaymode_config`
- `echo "7x10" > /sys/class/auxdisplay/displaymode_config`

- **keys**: Manage key scanning:
- **Reading** the file (using `cat`) returns one stored key at a time (FIFO style). When no keys remain, it returns “0”.
- **Writing** the command **"show"** displays the entire stored key sequence on the TM1628.
- **Writing** the command **"clear"** clears the key buffer.

- *Examples:*
- `cat /sys/class/auxdisplay/keys`
- `echo clear > /sys/class/auxdisplay/keys`
- `echo show > /sys/class/auxdisplay/keys`

9. Driver Source Code Overview

The driver source file (`tm1628.c`) is organized into several sections:

9.1 Header and Global Declarations

The file begins with necessary header inclusions and global variable definitions (e.g., GPIO descriptors, brightness, time mode, and display mode).

9.2 Low-Level Functions

- **GPIO Wrappers** – Functions to set GPIO values and create small delays.
- **tm1628_send_byte() and tm1628_send_command()** – Functions that send data/commands to the TM1628.

9.3 Display Functions

- **tm1628_display_pattern()** – Sends the pattern to the display.
- **tm1628_display_grids()** – Converts a string into a 6-byte pattern using a digit map.
- **tm1628_display_time()** – Displays the current time in HH.MM.SS format.

9.4 Key Scanning Section

- The key scanning routine reads key data using GPIO bit-banging.
- A global key buffer accumulates pressed keys.
- The sysfs **keys** attribute supports reading (which returns one key at a time) and writing commands ("clear" or "show").

9.5 Sysfs Attribute Wrappers

These functions allow user-space control of brightness, time mode, display text, display mode, and key operations.

9.6 Platform Driver Probe and Remove

- **tm1628_probe()**: Obtains the GPIOs, initializes the display, starts a kernel thread for key scanning, and creates the sysfs entries.
- **tm1628_remove()**: Cleans up by removing sysfs files, stopping the thread, and freeing resources.

For full details, refer to the complete source code included in the appendix.

10. Appendix: Complete Source Code

$ essae@rnd:~ kernel_source_imx93/linux-imx/drivers/leds$ vi tm1628.c

```
/*
 * tm1628.c - Platform driver for TM1628 LED display using GPIO bit-banging
 *
 * This driver uses three GPIOs (STB, DIO, CLK) to control the TM1628.
 * It obtains these GPIOs from the device tree.
 *
 * DTS sample:
 * / {
 *     tm1628: tm1628@0 {
 *         compatible = "my,tm1628";
 *         stb-gpio = <&gpio2 18 GPIO_ACTIVE_HIGH>;
 *         dio-gpio = <&gpio2 19 GPIO_ACTIVE_HIGH>;
 *         clk-gpio = <&gpio2 21 GPIO_ACTIVE_HIGH>;
 *     };
 * };
 *
 * Class attributes are created under /sys/class/auxdisplay/ for:
 *   - brightness        (RW)
 *   - time              (RW)
 *   - display           (RW, for showing text or amount)
 *   - displaymode_config (RW, to select the display mode)
 *   - keys              (RW, for key scanning and control)
```

```
/*
 * tm1628.c - Platform driver for TM1628 LED display using GPIO bit-banging
 *
 * This driver uses three GPIOs (STB, DIO, CLK) to control the TM1628.
 * It obtains these GPIOs from the device tree.
 *
 * DTS sample:
 * / {
 *     tm1628: tm1628@0 {
 *         compatible = "titanmec,tm1628";
 *         stb-gpio = <&gpio2 18 GPIO_ACTIVE_HIGH>;
 *         dio-gpio = <&gpio2 19 GPIO_ACTIVE_HIGH>;
 *         clk-gpio = <&gpio2 21 GPIO_ACTIVE_HIGH>;
 *     };
 * };
 *
 * Class attributes are created under /sys/class/auxdisplay/ for:
 *   - brightness        (RW)
 *   - time              (RW)
 *   - display           (RW, for showing text or amount)
 *   - displaymode_config (RW, to select the display mode)
 *   - display_raw       (RW, for writing 14 raw bytes to the display registers)
 *
 * Supported display modes:
 *   "4x13"  → 4 grids, 13 segments (mode command 0x00)
```

```
 *   "5x12"  → 5 grids, 12 segments (mode command 0x01)
 *   "6x11"  → 6 grids, 11 segments (mode command 0x02) [default]
 *   "7x10"  → 7 grids, 10 segments (mode command 0x03)
 *
 * Build this driver as a module or built-in.
 */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/of.h>
#include <linux/of_gpio.h>
#include <linux/platform_device.h>
#include <linux/gpio/consumer.h>
#include <linux/delay.h>
#include <linux/kthread.h>
#include <linux/timekeeping.h>
#include <linux/time.h>
#include <linux/device.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/jiffies.h>

#define DRIVER_NAME "tm1628"

/* GPIO descriptors obtained from DT */
static struct gpio_desc *gpiod_stb;
static struct gpio_desc *gpiod_dio;
static struct gpio_desc *gpiod_clk;
```

```
static struct task_struct *tm1628_thread;


/* Digit map for segments (for digits 0-9) */
static const unsigned char digit_map[10] = {
    0x3F, /* 0 */
    0x06, /* 1 */
    0x5B, /* 2 */
    0x4F, /* 3 */
    0x66, /* 4 */
    0x6D, /* 5 */
    0x7D, /* 6 */
    0x07, /* 7 */
    0x7F, /* 8 */
    0x6F  /* 9 */
};


/* For default 6x11 mode, grid addresses */
static const unsigned char grid_addresses[6] = { 0xC0, 0xC2, 0xC4, 0xC6, 0xC8,
0xCA };


/* Globals for sysfs control */
static int current_brightness = 10;
static int time_enabled = 0;


#define GRID_STR_SIZE 16
static char grids_str[GRID_STR_SIZE] = "000000";
```

```
/* Global for display mode command.
 * Supported values:
 *   0x00: 4x13, 0x01: 5x12, 0x02: 6x11 (default), 0x03: 7x10.
 */
static unsigned char mode_cmd = 0x02;  /* default 6x11 */

/* Sysfs class node for auxdisplay */
static struct class *auxdisplay_class;

/* Forward declaration */
static void tm1628_display_grids(const char *str);

/* --- Helper wrappers using GPIO descriptor APIs --- */
static inline void tm1628_gpio_set_desc(struct gpio_desc *desc, int value)
{
    gpiod_set_value(desc, value);
}

static inline void tm1628_delay_us(unsigned int us)
{
    udelay(us);
}

/* --- Low-Level TM1628 Functions --- */
static void tm1628_send_byte(unsigned char data)
{
    int i;
    for (i = 0; i < 8; i++) {
```

```
            tm1628_gpio_set_desc(gpiod_clk, 0);
            tm1628_delay_us(5);
            tm1628_gpio_set_desc(gpiod_dio, (data >> i) & 0x01);
            tm1628_delay_us(5);
            tm1628_gpio_set_desc(gpiod_clk, 1);
            tm1628_delay_us(5);
    }
}

static void tm1628_send_command(unsigned char command)
{
    tm1628_gpio_set_desc(gpiod_stb, 0);
    tm1628_delay_us(5);
    tm1628_send_byte(command);
    tm1628_delay_us(5);
    tm1628_gpio_set_desc(gpiod_stb, 1);
    tm1628_delay_us(5);
}

static void tm1628_set_brightness(unsigned char level)
{
    unsigned char cmd = 0x80 | (level & 0x0F);
    tm1628_send_command(cmd);
}

/* Initialize display with selected configuration */
static void tm1628_init_display(void)
{
```

```
    tm1628_send_command(mode_cmd);
    tm1628_send_command(0x40); /* Data command: auto-increment mode */
    tm1628_set_brightness(current_brightness);
}

/*
 * Display bitmap to clear or update all 14 registers.
 * The function writes 14 bytes of data starting at register 0xC0.
 */
static void tm1628_display_bitmap(const unsigned char bitmap[14])
{
    int i;
    unsigned char address = 0xC0;
    for (i = 0; i < 14; i++) {
          pr_info("tm1628: writing register 0x%02X = 0x%02X\n", address,
bitmap[i]);
          tm1628_gpio_set_desc(gpiod_stb, 0);
                tm1628_delay_us(5);
          tm1628_send_byte(address);
          tm1628_send_byte(bitmap[i]);
          tm1628_gpio_set_desc(gpiod_stb, 1);
          address++;
    }

    tm1628_delay_us(5);
}

/* Display a pattern on all 6 grids */
```

```
static void tm1628_display_pattern(const unsigned char pattern[6])
{
    int i;
    for (i = 0; i < 6; i++) {
          tm1628_gpio_set_desc(gpiod_stb, 0);
                tm1628_delay_us(5);
          tm1628_send_byte(grid_addresses[i]);
          tm1628_send_byte(pattern[i]);
          tm1628_gpio_set_desc(gpiod_stb, 1);
    }
    tm1628_delay_us(5);
}

/* Display a repeated digit with decimal point lit on all grids */
static void __maybe_unused tm1628_display_repeated_dp(unsigned char digit)
{
    unsigned char pattern[6];
    int i;
    for (i = 0; i < 6; i++)
          pattern[i] = digit_map[digit] | 0x80;
    tm1628_display_pattern(pattern);
}

/* Display current time in HH.MM.SS format */
static void __maybe_unused tm1628_display_time(void)
{
    unsigned char pattern[6];
    struct timespec64 ts;
```

```
    struct tm tm;
    time64_t time_sec;

    ktime_get_real_ts64(&ts);
    time_sec = ts.tv_sec;
    time64_to_tm(time_sec, 0, &tm);

    pattern[0] = digit_map[tm.tm_hour / 10];
    pattern[1] = digit_map[tm.tm_hour % 10] | 0x80;
    pattern[2] = digit_map[tm.tm_min / 10];
    pattern[3] = digit_map[tm.tm_min % 10] | 0x80;
    pattern[4] = digit_map[tm.tm_sec / 10];
    pattern[5] = digit_map[tm.tm_sec % 10];

    tm1628_display_pattern(pattern);
}

/*
 * Process an amount string (e.g., "6999.09") to display on the grids.
 * (Not used in key mode but available for future use.)
 */
static void __maybe_unused tm1628_display_amount(const char *amount_str)
{
    char digits[16];
    int i, j;

    j = 0;
    for (i = 0; amount_str[i] != '\0'; i++) {
```

```
      if (amount_str[i] != '.')
            digits[j++] = amount_str[i];
}
digits[j] = '\0';

{
      int len = strlen(digits);
      if (len > 5) {
            memmove(digits, digits + (len - 5), 5);
            digits[5] = '\0';
      } else if (len < 5) {
            char padded[6] = {0};
            int pad = 5 - len;
            for (i = 0; i < pad; i++)
                  padded[i] = '0';
            strcpy(padded + pad, digits);
            strcpy(digits, padded);
      }
}

{
      char final[7];
      final[0] = digits[0];
      final[1] = digits[1];
      final[2] = digits[2];
      final[3] = '.';
      final[4] = digits[3];
      final[5] = digits[4];
```

```
            final[6] = '\0';
            tm1628_display_grids(final);
    }
}

/*
 * Map a character (digit or letter A-Z) to a 7-segment pattern.
 */
static unsigned char tm1628_map_char(char c)
{
    if (c >= '0' && c <= '9')
            return digit_map[c - '0'];
    if (c >= 'a' && c <= 'z')
            c -= ('a' - 'A');
    switch(c) {
    case 'A': return 0x77;
    case 'B': return 0x7C;
    case 'C': return 0x39;
    case 'D': return 0x5E;
    case 'E': return 0x79;
    case 'F': return 0x71;
    case 'G': return 0x3D;
    case 'H': return 0x76;
    case 'I': return 0x06;
    case 'J': return 0x1E;
    case 'K': return 0x76;
    case 'L': return 0x38;
    case 'M': return 0x37;
```

```
    case 'N': return 0x54;
    case 'O': return 0x3F;
    case 'P': return 0x73;
    case 'Q': return 0x67;
    case 'R': return 0x50;
    case 'S': return 0x6D;
    case 'T': return 0x78;
    case 'U': return 0x3E;
    case 'V': return 0x3E;
    case 'W': return 0x2A;
    case 'X': return 0x76;
    case 'Y': return 0x6E;
    case 'Z': return 0x5B;
    default:  return 0x00;
    }
}

/*
 * Process an input string (which may include '.') and build a 6-byte pattern.
 */
static void tm1628_display_grids(const char *str)
{
    unsigned char pattern[6];
    int len = strlen(str);
    int i = 0, pat_index = 0;

    while (i < len && pat_index < 6) {
          if (str[i] == '.') {
```

```
                i++;
                continue;
            }
            {
                unsigned char seg = tm1628_map_char(str[i]);
                if ((i + 1 < len) && (str[i + 1] == '.')) {
                    seg |= 0x80;
                    i += 2;
                } else {
                    i++;
                }
                pattern[pat_index++] = seg;
            }
        }
        while (pat_index < 6)
            pattern[pat_index++] = 0x00;

        tm1628_display_pattern(pattern);
}

/* --- KEY SCANNING SECTION (Driver Version) --- */

#define KEY_BUFFER_SIZE 64
static char key_buffer_global[KEY_BUFFER_SIZE] = {0};
static int key_buffer_index_global = 0;

static unsigned char tm1628_read_byte_driver(void)
{
```

```
    int i;
    unsigned char byte = 0;
    for (i = 0; i < 8; i++) {
          tm1628_gpio_set_desc(gpiod_clk, 0);
          tm1628_delay_us(5);
          tm1628_gpio_set_desc(gpiod_clk, 1);
          tm1628_delay_us(5);
          {
                int bit = gpiod_get_value(gpiod_dio);
                if (bit < 0)
                      bit = 0;
                byte |= ((bit & 0x01) << i);
          }
          tm1628_delay_us(5);
    }
    return byte;
}

static void tm1628_read_keys_driver(unsigned char key_data[5])
{
    int i;
    tm1628_gpio_set_desc(gpiod_stb, 0);
    tm1628_delay_us(5);
    tm1628_send_byte(0x42);  /* Send key read command */
    tm1628_delay_us(5);

    gpiod_direction_input(gpiod_dio);
    for (i = 0; i < 5; i++) {
```

```
            key_data[i] = tm1628_read_byte_driver();
            tm1628_delay_us(5);
    }
    gpiod_direction_output(gpiod_dio, 1);

    tm1628_gpio_set_desc(gpiod_stb, 1);
    tm1628_delay_us(5);
}

struct key_pos {
    int byte;
    int bit;
    char key;
};

static struct key_pos key_map[10] = {
    {0, 0, '2'},
    {0, 1, '1'},
    {0, 3, '4'},
    {0, 4, '3'},
    {1, 0, '5'},
    {1, 1, '6'},
    {1, 3, '7'},
    {1, 4, '8'},
    {2, 0, '9'},
    {2, 1, '0'}
};
```

```
static char get_pressed_key_driver(unsigned char key_data[5])
{
    int i;
    for (i = 0; i < 10; i++) {
          if (key_data[key_map[i].byte] & (1 << key_map[i].bit))
                return key_map[i].key;
    }
    return '\0';
}

/* --- Kernel Thread Function with Key Scanning / Time Mode --- */
static int tm1628_thread_fn(void *data)
{
    int d;
    char disp_str[16];

    for (d = 0; d <= 9; d++) {
          snprintf(disp_str, sizeof(disp_str), "%c.%c.%c.%c.%c.%c",
                   '0' + d, '0' + d, '0' + d, '0' + d, '0' + d, '0' + d);
          tm1628_display_grids(disp_str);
          ssleep(1);
    }
    tm1628_display_grids("E.S.S.A.E.");
    ssleep(1);
    tm1628_display_grids("0.0.0.0.0.0");

    {
          unsigned char key_data[5];
```

```
        while (!kthread_should_stop()) {
            if (time_enabled) {
                tm1628_display_time();
                msleep(1000);
                continue;
            }

            memset(key_data, 0, sizeof(key_data));
            tm1628_read_keys_driver(key_data);
            {
                char key = get_pressed_key_driver(key_data);
                if (key != '\0') {
                    if (key_buffer_index_global < (KEY_BUFFER_SIZE - 1)) {
                        key_buffer_global[key_buffer_index_global++] =
key;
                        key_buffer_global[key_buffer_index_global] =
'\0';
                    }
                }
            }
            msleep(200);
        }
    }
    return 0;
}

/* --- Sysfs Attributes --- */

/*
```

```
 * keys attribute: returns the first stored key and removes it.
 */
static ssize_t keys_show(const struct class *cls,
                         const struct class_attribute *attr, char *buf)
{
    char out;
    if (key_buffer_index_global > 0) {
          out = key_buffer_global[0];
          memmove(key_buffer_global, key_buffer_global + 1,
key_buffer_index_global);
          key_buffer_index_global--;
          return sprintf(buf, "%c\n", out);
    } else {
          return sprintf(buf, "0\n");
    }
}


static ssize_t keys_store(const struct class *cls,
                          const struct class_attribute *attr,
                          const char *buf, size_t count)
{
    if (sysfs_streq(buf, "clear")) {
          memset(key_buffer_global, 0, sizeof(key_buffer_global));
          key_buffer_index_global = 0;
          tm1628_display_grids("0.0.0.0.0.0");
    } else if (sysfs_streq(buf, "show")) {
          tm1628_display_grids(key_buffer_global);
    }
    return count;
```

```
}
static CLASS_ATTR_RW(keys);

static ssize_t brightness_show(const struct class *cls,
                        const struct class_attribute *attr, char *buf)
{
    return sprintf(buf, "%d\n", current_brightness);
}

static ssize_t brightness_store(const struct class *cls,
                        const struct class_attribute *attr,
                        const char *buf, size_t count)
{
    unsigned long val;
    int ret = kstrtoul(buf, 10, &val);
    if (ret)
          return ret;
    if (val > 15)
          val = 15;
    current_brightness = val;
    tm1628_set_brightness(current_brightness);
    return count;
}
static CLASS_ATTR_RW(brightness);

static ssize_t time_show(const struct class *cls,
                const struct class_attribute *attr, char *buf)
{
```

```
    return sprintf(buf, "%s\n", time_enabled ? "on" : "off");
}

static ssize_t time_store(const struct class *cls,
                  const struct class_attribute *attr,
                  const char *buf, size_t count)
{
    if (sysfs_streq(buf, "on"))
          time_enabled = 1;
    else if (sysfs_streq(buf, "off")) {
          time_enabled = 0;
          tm1628_display_grids(grids_str);
    }
    return count;
}
static CLASS_ATTR_RW(time);

static ssize_t display_show(const struct class *cls,
                    const struct class_attribute *attr, char *buf)
{
    return sprintf(buf, "%s\n", grids_str);
}

static ssize_t display_store(const struct class *cls,
                     const struct class_attribute *attr,
                     const char *buf, size_t count)
{
    size_t len = min(count, (size_t)(GRID_STR_SIZE - 1));
```

```
    char tmp[GRID_STR_SIZE];
    memcpy(tmp, buf, len);
    tmp[len] = '\0';
    if (tmp[len - 1] == '\n')
          tmp[len - 1] = '\0';
    strncpy(grids_str, tmp, GRID_STR_SIZE - 1);
    grids_str[GRID_STR_SIZE - 1] = '\0';
    tm1628_display_grids(grids_str);
    return count;
}
static CLASS_ATTR_RW(display);

/*
 * display_raw attribute: Write 14 raw bytes to update all registers.
 * For reading, we simply return a help message.
 */
static ssize_t display_raw_show(const struct class *cls,
                                const struct class_attribute *attr,
                                char *buf)
{
    return sprintf(buf, "Write 14 bytes of raw data to update the display
registers\n");
}

static ssize_t display_raw_store(const struct class *cls,
                                 const struct class_attribute *attr,
                                 const char *buf, size_t count)
{
```

```
char tmp[28], bitdata[14];
size_t i, j;
if (count != 28)
    return -EINVAL;
for (i = 0, j = 0; i < 28; i++) {
   tmp[i] = *buf++;
if(tmp[i] >= '0' && tmp[i] <= '9') //numbers
   {
        tmp[i] -= '0';
   }
   else if(tmp[i] >= 'A' && tmp[i] <= 'F') //A TO F
   {
        tmp[i] = tmp[i] - 'A'+10;
   }
   else if(tmp[i] >= 'a' && tmp[i] <= 'f') //A TO F
   {
        tmp[i] = tmp[i] - 'a'+10;
   }
   else
   {
       return -EINVAL;
   }
   if(i&0x1)
   {
        bitdata[j++] = tmp[i-1] << 4 |  tmp[i] ;
   }

}
```

```
    tm1628_display_bitmap(bitdata);
    return count;
}

/* Updated attribute: Changed mode from 0666 to 0644 */
static struct class_attribute class_attr_display_raw =
    __ATTR(display_raw, 0644, display_raw_show, display_raw_store);

static ssize_t displaymode_config_show(const struct class *cls,
                            const struct class_attribute *attr, char *buf)
{
    const char *cfg;
    switch (mode_cmd) {
    case 0x00: cfg = "4x13"; break;
    case 0x01: cfg = "5x12"; break;
    case 0x02: cfg = "6x11"; break;
    case 0x03: cfg = "7x10"; break;
    default:   cfg = "unknown"; break;
    }
    return sprintf(buf, "%s\n", cfg);
}

static ssize_t displaymode_config_store(const struct class *cls,
                            const struct class_attribute *attr,
                            const char *buf, size_t count)
{
    if (sysfs_streq(buf, "4x13"))
          mode_cmd = 0x00;
```

```
    else if (sysfs_streq(buf, "5x12"))
          mode_cmd = 0x01;
    else if (sysfs_streq(buf, "6x11"))
          mode_cmd = 0x02;
    else if (sysfs_streq(buf, "7x10"))
          mode_cmd = 0x03;
    else
          return -EINVAL;
    tm1628_init_display();
    return count;
}
static CLASS_ATTR_RW(displaymode_config);

/* --- Platform Driver Probe and Remove --- */
static int tm1628_probe(struct platform_device *pdev)
{
    int ret;
    gpiod_stb = devm_gpiod_get(&pdev->dev, "stb", GPIOD_OUT_HIGH);
    if (IS_ERR(gpiod_stb)) {
          dev_err(&pdev->dev, "Failed to get STB GPIO\n");
          return PTR_ERR(gpiod_stb);
    }
    gpiod_dio = devm_gpiod_get(&pdev->dev, "dio", GPIOD_OUT_HIGH);
    if (IS_ERR(gpiod_dio)) {
          dev_err(&pdev->dev, "Failed to get DIO GPIO\n");
          return PTR_ERR(gpiod_dio);
    }
    gpiod_clk = devm_gpiod_get(&pdev->dev, "clk", GPIOD_OUT_HIGH);
```

```
if (IS_ERR(gpiod_clk)) {
      dev_err(&pdev->dev, "Failed to get CLK GPIO\n");
      return PTR_ERR(gpiod_clk);
}

tm1628_init_display();

tm1628_thread = kthread_run(tm1628_thread_fn, NULL, "tm1628_thread");
if (IS_ERR(tm1628_thread)) {
      dev_err(&pdev->dev, "Failed to create kernel thread\n");
      return PTR_ERR(tm1628_thread);
}

auxdisplay_class = class_create("auxdisplay");
if (IS_ERR(auxdisplay_class)) {
      dev_err(&pdev->dev, "Failed to create class\n");
      ret = PTR_ERR(auxdisplay_class);
      goto fail_class;
}
ret = class_create_file(auxdisplay_class, &class_attr_brightness);
if (ret)
      dev_err(&pdev->dev, "Failed to create brightness sysfs file\n");
ret = class_create_file(auxdisplay_class, &class_attr_time);
if (ret)
      dev_err(&pdev->dev, "Failed to create time sysfs file\n");
ret = class_create_file(auxdisplay_class, &class_attr_display);
if (ret)
      dev_err(&pdev->dev, "Failed to create display sysfs file\n");
```

```
    ret = class_create_file(auxdisplay_class, &class_attr_displaymode_config);
    if (ret)
          dev_err(&pdev->dev, "Failed to create displaymode_config sysfs file\n");
    ret = class_create_file(auxdisplay_class, &class_attr_keys);
    if (ret)
          dev_err(&pdev->dev, "Failed to create keys sysfs file\n");
    ret = class_create_file(auxdisplay_class, &class_attr_display_raw);
    if (ret)
          dev_err(&pdev->dev, "Failed to create display_raw sysfs file\n");

    dev_info(&pdev->dev, "TM1628 driver loaded successfully\n");
    return 0;

fail_class:
    class_destroy(auxdisplay_class);
    kthread_stop(tm1628_thread);
    return ret;
}

static int tm1628_remove(struct platform_device *pdev)
{
    class_remove_file(auxdisplay_class, &class_attr_brightness);
    class_remove_file(auxdisplay_class, &class_attr_time);
    class_remove_file(auxdisplay_class, &class_attr_display);
    class_remove_file(auxdisplay_class, &class_attr_displaymode_config);
    class_remove_file(auxdisplay_class, &class_attr_display_raw);
    class_remove_file(auxdisplay_class, &class_attr_keys);
    class_destroy(auxdisplay_class);
```

```
    if (tm1628_thread)
          kthread_stop(tm1628_thread);
    dev_info(&pdev->dev, "TM1628 driver unloaded\n");
    return 0;
}

static const struct of_device_id tm1628_of_match[] = {
    { .compatible = "titanmec,tm1628", },
    { },
};
MODULE_DEVICE_TABLE(of, tm1628_of_match);

static struct platform_driver tm1628_driver = {
    .driver = {
          .name = DRIVER_NAME,
          .of_match_table = tm1628_of_match,
    },
    .probe = tm1628_probe,
    .remove = tm1628_remove,
};

module_platform_driver(tm1628_driver);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
MODULE_DESCRIPTION("TM1628 LED Display Platform Driver with Integrated Key
Scanning and Time Mode");
```

11. Conclusion

This guide has walked you through the full development process for the TM1628 7-segment LED driver. You now have:

- A clear hardware connection diagram,
- DTS and build configurations,
- Detailed sysfs interfaces for brightness, time, display, and key control,
- Complete instructions for compiling and deploying the kernel/module, and
- The complete driver source code.