ML56-TK Module Linux Driver

Example Code Introduction for 64/32-bit NuMicro® Family

Document Information

Application	This example code shows how to adapt ML56-TK driver to Linux kernel.
BSP Version	Linux-5.4.x
Hardware	NuMaker-Base-MA35D1B1 V2.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. Overview

This example code shows how to insert ML56-TK kernel module into Linux, and use ‘input_test’ test program to decode an input event into human readable format.

In order for users to adapt ML56-TK into their application easily, the ML56-TK driver enables ML56-TK to act as an ’Input Device’ in the Linux, and enables a touch key to act as a ’keyboard’ and report key code (KEY_A, KEY_B, etc.) and key status (Press/Release) to the system. Users can define the key code via a device tree and enables a slider/wheel to act as a ’touch panel’ and convert raw data from ML56-TK into X-Y coordinates and report to system, where X presents the percentage and Y is used to identify the percentage that is belonged to the slider or wheel.

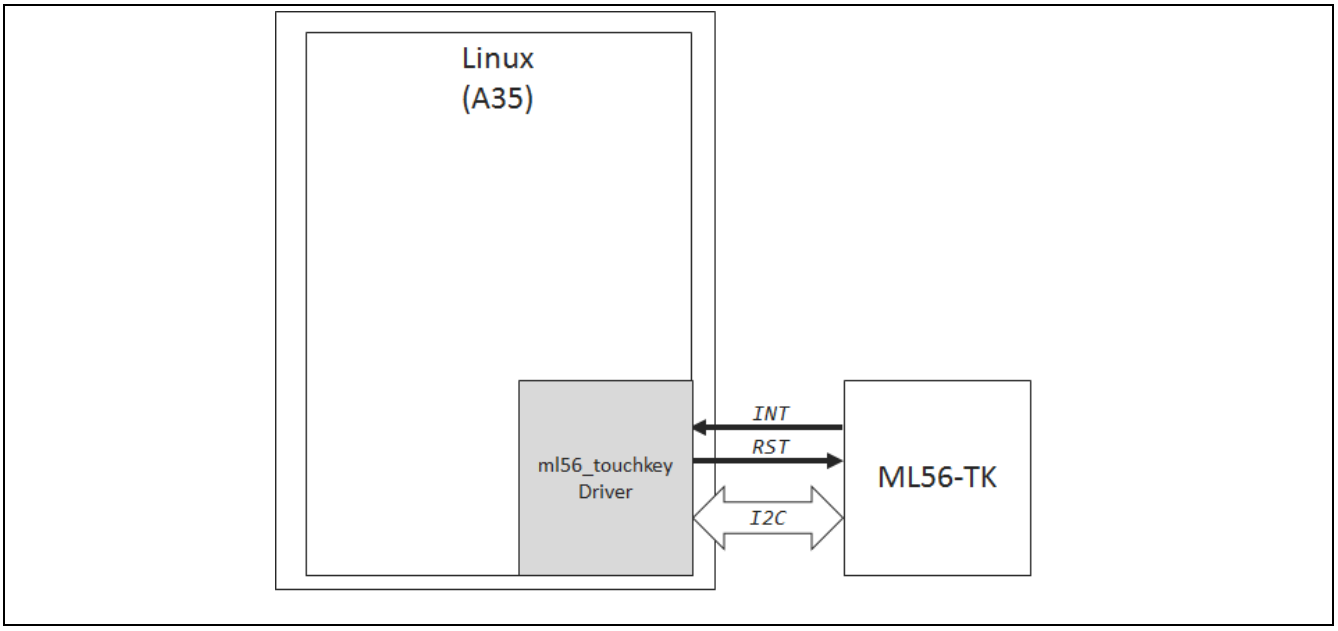

Figure 1-1 ML56-TK Linux Driver Architecture

1.1 Module Driver Setting

The device tree setting of ML56-TK kernel module is described below.

1.1.1 I²C Setting

The device tree setting of ML56-TK GPIO driving I²C protocol is as follows.

```
i2c_gpio0: i2c-gpio-0 {
    reg = <0 0 0 0>;
    compatible = "i2c-gpio";
    gpios = <&gpiom 14 0 /* sda */
      &gpiom 15 0 /* scl */
     >;
    i2c-gpio,delay-us = <20>;        /* ~100 kHz */
    #address-cells = <1>;
    #size-cells = <0>;
    status = "disabled";
};
```

1.1.2 Multi-function Pin Setting

The device tree setting of ML56-TK kernel module multi-function pins is as follows.

```
&i2c_gpio0 {
    status = "okay";
   ...
    ml56-tk@52 {
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_i2c_gpio0>;
        compatible = "nuvoton,ml56-tk";
        reg = <0x52>;
        reset-gpios = <&gpiom 1 GPIO_ACTIVE_LOW>;
        interrupt-parent = <&gpiok>;
        interrupts = <8 IRQ_TYPE_LEVEL_LOW>;
        irq-gpios = <&gpiok 8 0x00>;
        ml56-tk,keymap = <10 KEY_A>, <11 KEY_B>;
    };
};

/* Add ml56 module pin configuration to device tree's pinctl as below */

&pinctrl {
....
pcfg_ml56_interrupt: pcfg-pcfg_ml56_interrupt {
    slew-rate = <0>; /* 0:normal, 1:high */
    input-schmitt-disable;  /* input-schmitt-disable, input-schmitt-enable */
    bias-pull-up; /* bias-disable, bias-pull-up, bias-pull-down  */
    power-source = <3300>; /* 1800:1.8v , 3300:3.3v */
    drive-strength = <3>; /* 0 ~ 7 */
};

pcfg_ml56_scl: pcfg-pcfg_ml56_sck {
    slew-rate = <0>;        /* 0:normal, 1:high */
    input-schmitt-disable;  /* input-schmitt-disable, input-schmitt-enable */
```

```
    bias-disable;           /* bias-disable, bias-pull-up, bias-pull-down  */
    power-source = <3300>;  /* 1800:1.8v , 3300:3.3v */
    drive-strength = <3>;   /* 0 ~ 7 */
};

pcfg_ml56_sda: pcfg-pcfg_ml56_sda {
    slew-rate = <0>;        /* 0:normal, 1:high */
    input-schmitt-disable;  /* input-schmitt-disable, input-schmitt-enable */
    bias-disable;           /* bias-disable, bias-pull-up, bias-pull-down  */
    power-source = <3300>;  /* 1800:1.8v , 3300:3.3v */
    drive-strength = <3>;   /* 0 ~ 7 */
};

pcfg_ml56_reset: pcfg-pcfg_ml56_reset {
    slew-rate = <0>;        /* 0:normal, 1:high */
    input-schmitt-disable;  /* input-schmitt-disable, input-schmitt-enable */
    bias-pull-up;           /* bias-disable, bias-pull-up, bias-pull-down  */
    power-source = <3300>;  /* 1800:1.8v , 3300:3.3v */
    drive-strength = <3>;   /* 0 ~ 7 */
};

i2c_gpio0 {
      pinctrl_i2c_gpio0: i2c_gpio0grp {
       nuvoton,pins =
       <SYS_GPK_MFPH_PK8MFP_GPIO       &pcfg_ml56_interrupt>,
       <SYS_GPM_MFPL_PM1MFP_GPIO       &pcfg_ml56_reset>,
       <SYS_GPM_MFPH_PM15MFP_GPIO      &pcfg_ml56_scl>,
       <SYS_GPM_MFPH_PM14MFP_GPIO      &pcfg_ml56_sda>;
       };
   };
};
```

1.2 Demo Result

The demo result is shown below.

```
Wed Mar  2 03:50:39 2022.437039 type 0x0001; code 0x0030; value 0x00000001; Key 48 (0x30) press
Wed Mar  2 03:50:39 2022.437039 type 0x0000; code 0x0000; value 0x00000000;
Wed Mar  2 03:50:39 2022.657376 type 0x0001; code 0x0030; value 0x00000000; Key 48 (0x30) release
Wed Mar  2 03:50:39 2022.657376 type 0x0000; code 0x0000; value 0x00000000;
Wed Mar  2 03:50:41 2022.888186 type 0x0001; code 0x001e; value 0x00000001; Key 30 (0x1e) press
Wed Mar  2 03:50:41 2022.888186 type 0x0000; code 0x0000; value 0x00000000;
Wed Mar  2 03:50:42 2022.107135 type 0x0001; code 0x001e; value 0x00000000; Key 30 (0x1e) release
Wed Mar  2 03:50:42 2022.107135 type 0x0000; code 0x0000; value 0x00000000;

Wed Mar  2 03:51:20 2022.610418 type 0x0001; code 0x014a; value 0x00000001; Button 74 press
Wed Mar  2 03:51:20 2022.610418 type 0x0003; code 0x0000; value 0x00000004; Absolute X 4
Wed Mar  2 03:51:20 2022.610418 type 0x0003; code 0x0001; value 0x00000000; Absolute Y 0
Wed Mar  2 03:51:20 2022.807639 type 0x0003; code 0x0000; value 0x00000061; Absolute X 97
Wed Mar  2 03:51:20 2022.898640 type 0x0001; code 0x014a; value 0x00000000; Button 74 release

Wed Mar  2 03:53:18 2022.847879 type 0x0001; code 0x014a; value 0x00000001; Button 74 press
Wed Mar  2 03:53:18 2022.847879 type 0x0003; code 0x0000; value 0x00000000; Absolute X 0
```

```
Wed Mar  2 03:53:18 2022.847879 type 0x0003; code 0x0001; value 0x00000002; Absolute Y 2
Wed Mar  2 03:53:19 2022.544309 type 0x0003; code 0x0000; value 0x00000050; Absolute X 80
Wed Mar  2 03:53:19 2022.644725 type 0x0001; code 0x014a; value 0x00000000; Button 74 release
```

Figure 1-2 Demo Result

2. Code Description

The 'input_test' reads from /dev/input/event* and then converts events into human readable format.

```
    if (argc > 1) {
        sprintf(name, "/dev/input/event%d", atoi(argv[1]));
        if ((fd = open(name, O_RDWR, 0)) >= 0) {
            printf("%s: open, fd = %d\n", name, fd);
            for (i = 0; i < LED_MAX; i++) {
                event.time.tv_sec  = time(0);
                event.time.tv_usec = 0;
                event.type         = EV_LED;
                event.code         = i;
                event.value        = 0;
                write(fd, &event, sizeof(event));
            }

            while ((rc = read(fd, &event, sizeof(event))) > 0) {
                printf("%-24.24s.%06lu type 0x%04x; code 0x%04x;"
                       " value 0x%08x; ",
                       ctime(&event.time.tv_sec),
                       event.time.tv_usec,
                       event.type, event.code, event.value);
                switch (event.type) {
                case EV_KEY:
                    if (event.code > BTN_MISC) {
                        printf("Button %d %s",
                               event.code & 0xff,
                               event.value ? "press" : "release");
                    } else {
                        printf("Key %d (0x%x) %s",
                               event.code & 0xff,
                               event.code & 0xff,
                               event.value ? "press" : "release");
                    }
                    break;
                case EV_REL:
                    switch (event.code) {
                    case REL_X:      tmp = "X";       break;
                    case REL_Y:      tmp = "Y";       break;
                    case REL_HWHEEL: tmp = "HWHEEL";  break;
                    case REL_DIAL:   tmp = "DIAL";    break;
                    case REL_WHEEL:  tmp = "WHEEL";   break;
                    case REL_MISC:   tmp = "MISC";    break;
                    default:         tmp = "UNKNOWN"; break;
                    }
                    printf("Relative %s %d", tmp, event.value);
                    break;
                case EV_ABS:
                    switch (event.code) {
                    case ABS_X:        tmp = "X";        break;
                    case ABS_Y:        tmp = "Y";        break;
                    case ABS_Z:        tmp = "Z";        break;
```

```
                    case ABS_RX:       tmp = "RX";       break;
                    case ABS_RY:       tmp = "RY";       break;
                    case ABS_RZ:       tmp = "RZ";       break;
                    case ABS_THROTTLE: tmp = "THROTTLE"; break;
                    case ABS_RUDDER:   tmp = "RUDDER";   break;
                    case ABS_WHEEL:    tmp = "WHEEL";    break;
                    case ABS_GAS:      tmp = "GAS";      break;
                    case ABS_BRAKE:    tmp = "BRAKE";    break;
                    case ABS_HAT0X:    tmp = "HAT0X";    break;
                    case ABS_HAT0Y:    tmp = "HAT0Y";    break;
                    case ABS_HAT1X:    tmp = "HAT1X";    break;
                    case ABS_HAT1Y:    tmp = "HAT1Y";    break;
                    case ABS_HAT2X:    tmp = "HAT2X";    break;
                    case ABS_HAT2Y:    tmp = "HAT2Y";    break;
                    case ABS_HAT3X:    tmp = "HAT3X";    break;
                    case ABS_HAT3Y:    tmp = "HAT3Y";    break;
                    case ABS_PRESSURE: tmp = "PRESSURE"; break;
                    case ABS_DISTANCE: tmp = "DISTANCE"; break;
                    case ABS_TILT_X:   tmp = "TILT_X";   break;
                    case ABS_TILT_Y:   tmp = "TILT_Y";   break;
                    case ABS_MISC:     tmp = "MISC";     break;
                    default:           tmp = "UNKNOWN";  break;
                    }
                    printf("Absolute %s %d", tmp, event.value);
                    break;
                case EV_MSC: printf("Misc"); break;
                case EV_LED: printf("Led");  break;
                case EV_SND: printf("Snd");  break;
                case EV_REP: printf("Rep");  break;
                case EV_FF:  printf("FF");   break;
                    break;
                }
                printf("\n");
            }
            printf("rc = %d, (%s)\n", rc, strerror(errno));
            close(fd);
        }
    }
```

3. Software and Hardware Requirements

3.1 Software Requirements

- Linux version
  - Linux-5.4.x
- Module name
  - ml56-touchkey.ko

3.2 Hardware Requirements

- Circuit components
  - NuMaker-HMI-MA35D1 V2.0
  - ML56-TK Module
- Pin Connect

NuMaker-HMI-MA35D1		ML56-TK
3V3	↔	3V3
GND	↔	GND
I2C3_SDA (PM.14)	↔	SDA
I2C3_SCL (PM.15)	↔	SCL
PK.8	↔	RST
PA.1	↔	INT

Figure 3-1 Pin Connect

4. Directory Information

The directory structure is shown below.

MA35D1_ML56-TK_Driver_V1.00	
Src	Source files of ML56-TK device driver
Ko	Kernel object file of ML56-TK device driver
Input_test	The test program of input device
DT_Binding	Declares requirements on the contents of ML56-TK driver

Figure 4-1 Directory Structure

5. Example Code Execution

1. Configure the required pins (referring to the Module Driver Setting section).
2. Re-compile the kernel and module into the built environment.
3. Power on the NuMaker-HMI-MA35D1.
4. Copy ml56-touchkey.ko and input_test (test program) into the target board.
5. Install the kernel module.

```
# insmod ml56-touchkey.ko
```

The following screen will appear, the numbers in the red box are the input parameters of the application

```
[   61.425658] input: ml56-tk as /devices/platform/0.i2c-gpio-0/i2c-0/0-0052/input/input2
```

6. Run the test program.

```
# ./input_test 2
```

7. Touch the key on the ML56-TK.

```
Wed Mar  2 03:50:39 2022.437039 type 0x0001; code 0x0030; value 0x00000001; Key 48 (0x30) press
Wed Mar  2 03:50:39 2022.437039 type 0x0000; code 0x0000; value 0x00000000;
Wed Mar  2 03:50:39 2022.657376 type 0x0001; code 0x0030; value 0x00000000; Key 48 (0x30) release
Wed Mar  2 03:50:39 2022.657376 type 0x0000; code 0x0000; value 0x00000000;
Wed Mar  2 03:50:41 2022.888186 type 0x0001; code 0x001e; value 0x00000001; Key 30 (0x1e) press
Wed Mar  2 03:50:41 2022.888186 type 0x0000; code 0x0000; value 0x00000000;
Wed Mar  2 03:50:42 2022.107135 type 0x0001; code 0x001e; value 0x00000000; Key 30 (0x1e) release
Wed Mar  2 03:50:42 2022.107135 type 0x0000; code 0x0000; value 0x00000000;
```

8. Slide the slider/wheel on the ML56-TK.

```
Wed Mar  2 03:51:20 2022.610418 type 0x0001; code 0x014a; value 0x00000001; Button 74 press
Wed Mar  2 03:51:20 2022.610418 type 0x0003; code 0x0000; value 0x00000004; Absolute X 4
Wed Mar  2 03:51:20 2022.610418 type 0x0003; code 0x0001; value 0x00000000; Absolute Y 0
Wed Mar  2 03:51:20 2022.807639 type 0x0003; code 0x0000; value 0x00000061; Absolute X 97
Wed Mar  2 03:51:20 2022.898640 type 0x0001; code 0x014a; value 0x00000000; Button 74 release
```

6. Revision History

Date	Revision	Description
2022.09.27	1.00	Initial version.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.