ILI2XXX QUICK START GUIDE

Riverdi

Display Solution Experts

ILI2xxx Quick Start Guide

Linux implementation

Rev.1.0

2020-08-16

REVISION RECORD

REVNO.	REVDATE	CONTENTS	REMARKS
1.0	2020-08-16	Initial Release	

CONTENTS

1 INTRODUCTION

This document describes how to use ILI2xxx Touch panel controller in Linux OS environment. Document is mostly focused on usage with Raspberry Pi but same approach can be used on any other Linux OS.

Document requires basic knowledge of Linux operating system, and I2C bus usage on the Linux platform. It will go through the setup of the hardware and necessary software, integration of the ILITEK library and simple example which printout the X,Y position of the detected touch to the terminal.

Main components :

- Touch panel with IL2xxx controller
- ZIF breakout board
- 8 wire jumpers (Female-Female)
- USB cable (Type C for board supply)
- Raspberry Pi board with mirco SD card (Raspberrian OS flashed)

2 HARDWARE AND SOFTWARE SETUP

Setup of the Raspberry Pi is not in the scope of this document so we will consider that you already prepared the Raspberry Pi board and have at least ssh access. It is important to say that console is enough, meaning graphic interface is not necessary.

- Log into the your Raspberry Pi system
- Enable I2C peripherals on your Raspberry Pi system by executing

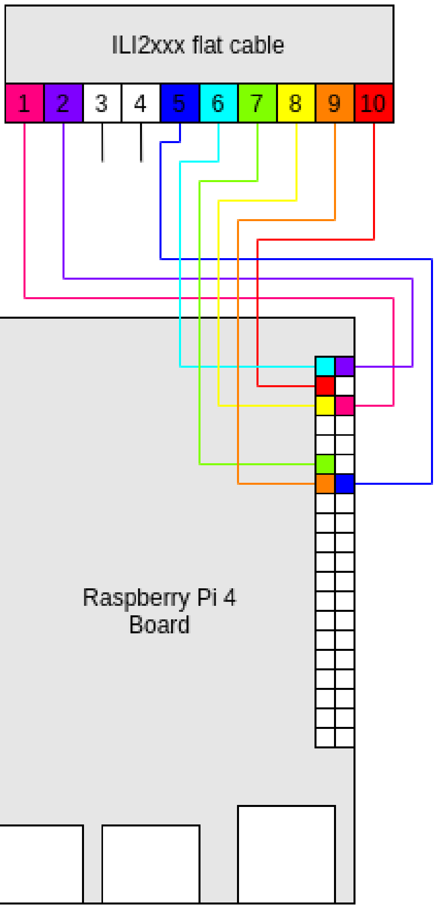
  ```
  sudo raspi-config
  ```

  - Navigate to "Interface Options"
  - Select I2C
  - Enable interface
  - Restart the system and login again
- Install i2c-tools and make to the your Raspberry Pi system by executing

  ```
  sudo apt-get install i2c-tools cmake
  ```

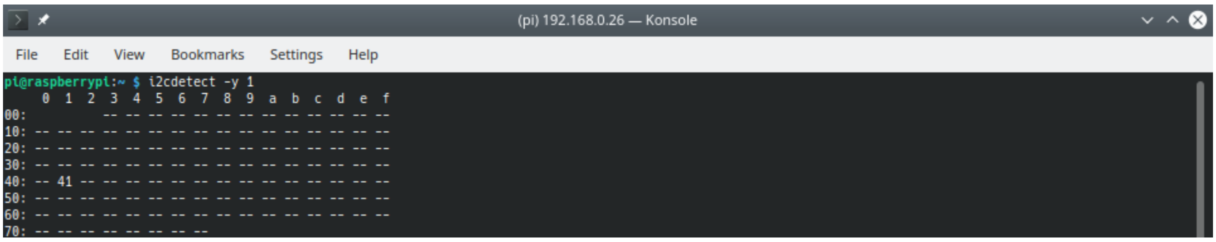
- Connect Touch panel (ILI2xxx) flat cable to the ZIF breakout board
- Connect ZIF breakout board headers to the Raspberry Pi headers

Figure 1. Raspberry Pi Connection



While `cmake` is installed to simplify the build, I2C tools is there as tool helpful to probe the devices connected to the bus. If everything went well I2C address scanning should report ILI2xxx address available. So execute `i2cdetect -y 1` and result should be something like this.

Figure 2. Ilitek address detection



```
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- 41 -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

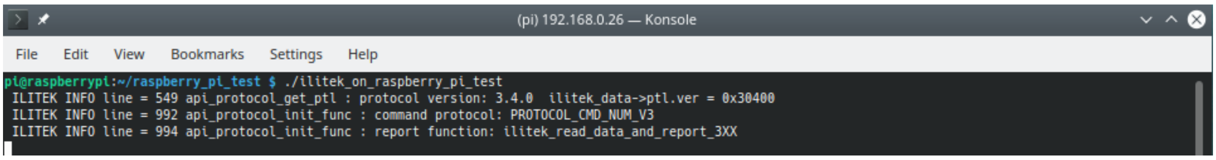
3 COMPILE AND RUN DEMO EXAMPLE

Demo application made for this "Getting started" purpose uses simplified version of the original library made for ILI2xxx, implemented in form of the kernel module. You can download this version of the library alongside with the demo from ours github repository.

- Unpack the content and place on desired place on your OS.
- Navigate inside unpacked folder and compile demo by executing `make`.
- (Optionally) Clean up build folder by calling `make clean`.

If everything went well you should see new binary `ilitek_on_raspberry_pi_demo` as output of the compilation inside root folder. Simply run it by executing `./ilitek_on_raspberry_pi_demo` The initial output should be something like this.

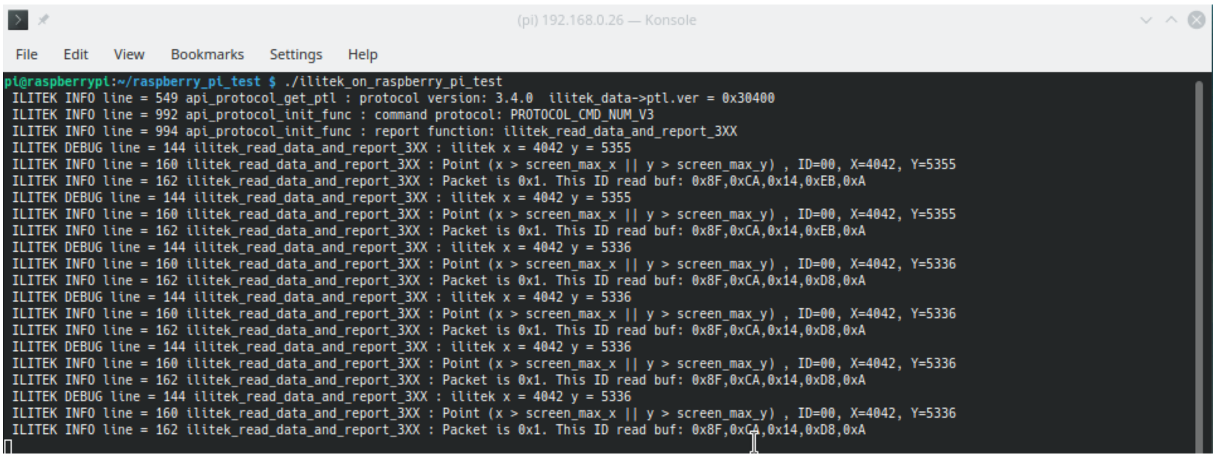
Figure 3. Initialization log



```
pi@raspberrypi:~/raspberry_pi_test $ ./ilitek_on_raspberry_pi_test
 ILITEK INFO line = 549 api_protocol_get_ptl : protocol version: 3.4.0  ilitek_data->ptl.ver = 0x30400
 ILITEK INFO line = 992 api_protocol_init_func : command protocol: PROTOCOL_CMD_NUM_V3
 ILITEK INFO line = 994 api_protocol_init_func : report function: ilitek_read_data_and_report_3XX
```

After that, each time you touch the panel information about touch position will be logged, continuously until you release the touch.

Figure 4. Touch log



```
pi@raspberrypi:~/raspberry_pi_test $ ./ilitek_on_raspberry_pi_test
 ILITEK INFO line = 549 api_protocol_get_ptl : protocol version: 3.4.0  ilitek_data->ptl.ver = 0x30400
 ILITEK INFO line = 992 api_protocol_init_func : command protocol: PROTOCOL_CMD_NUM_V3
 ILITEK INFO line = 994 api_protocol_init_func : report function: ilitek_read_data_and_report_3XX
 ILITEK DEBUG line = 144 ilitek_read_data_and_report_3XX : ilitek x = 4042 y = 5355
 ILITEK INFO line = 160 ilitek_read_data_and_report_3XX : Point (x > screen_max_x || y > screen_max_y) , ID=00, X=4042, Y=5355
 ILITEK INFO line = 162 ilitek_read_data_and_report_3XX : Packet is 0x1. This ID read buf: 0x8F,0xCA,0x14,0xEB,0xA
 ILITEK DEBUG line = 144 ilitek_read_data_and_report_3XX : ilitek x = 4042 y = 5355
 ILITEK INFO line = 160 ilitek_read_data_and_report_3XX : Point (x > screen_max_x || y > screen_max_y) , ID=00, X=4042, Y=5355
 ILITEK INFO line = 162 ilitek_read_data_and_report_3XX : Packet is 0x1. This ID read buf: 0x8F,0xCA,0x14,0xEB,0xA
 ILITEK DEBUG line = 144 ilitek_read_data_and_report_3XX : ilitek x = 4042 y = 5336
 ILITEK INFO line = 160 ilitek_read_data_and_report_3XX : Point (x > screen_max_x || y > screen_max_y) , ID=00, X=4042, Y=5336
 ILITEK INFO line = 162 ilitek_read_data_and_report_3XX : Packet is 0x1. This ID read buf: 0x8F,0xCA,0x14,0xD8,0xA
 ILITEK DEBUG line = 144 ilitek_read_data_and_report_3XX : ilitek x = 4042 y = 5336
 ILITEK INFO line = 160 ilitek_read_data_and_report_3XX : Point (x > screen_max_x || y > screen_max_y) , ID=00, X=4042, Y=5336
 ILITEK INFO line = 162 ilitek_read_data_and_report_3XX : Packet is 0x1. This ID read buf: 0x8F,0xCA,0x14,0xD8,0xA
 ILITEK DEBUG line = 144 ilitek_read_data_and_report_3XX : ilitek x = 4042 y = 5336
 ILITEK INFO line = 160 ilitek_read_data_and_report_3XX : Point (x > screen_max_x || y > screen_max_y) , ID=00, X=4042, Y=5336
 ILITEK INFO line = 162 ilitek_read_data_and_report_3XX : Packet is 0x1. This ID read buf: 0x8F,0xCA,0x14,0xD8,0xA
 ILITEK DEBUG line = 144 ilitek_read_data_and_report_3XX : ilitek x = 4042 y = 5336
 ILITEK INFO line = 160 ilitek_read_data_and_report_3XX : Point (x > screen_max_x || y > screen_max_y) , ID=00, X=4042, Y=5336
 ILITEK INFO line = 162 ilitek_read_data_and_report_3XX : Packet is 0x1. This ID read buf: 0x8F,0xCA,0x14,0xD8,0xA
```

4 ADJUST LIBRARY AND DEMO FOR OTHER LINUX PLATFORMS

The library and demo for Linux systems is written according to ANSI-C standard so you should be able to compile it with any version of the GNU C/C++ compiler. It is depended only on standard GCC libraries and there should be no portability issues. However this demo is strictly written for the particular platform and there are some points which should be considered when switching to the other platform.

4.1 SYSTEM INTERFACE

The best way to start are probably easiest functions :

- `void ilitek_delay ( uint32_t msec )`
- `void ilitek_sleep ( uint32_t msec )`

On the Linux systems you should both implement to make delay for provided amount of millisecond, so we can use Linux `<time.h>` library for this particular purpose. So implementation might looks like this.

```
void ilitek_delay ( uint32_t msec )
{
    int ret;
    struct timespec start_time;
    struct timespec curr_time;
    uint32_t start_msec;
    uint32_t curr_msec;

    ret = clock_gettime( CLOCK_REALTIME, &start_time );
    if ( ret != 0 )
    {
        printf( "Error in clock_gettime !\r\n" );
        return;
    }

    start_msec =
            ( start_time.tv_sec * 1000 ) + ( start_time.tv_nsec / 1000 );

    //  Block here until msec expire.

    for ( ;; )
    {
        ret = clock_gettime( CLOCK_REALTIME, &curr_time );
        if ( ret != 0 )
        {
            printf( "Error in clock_gettime !\r\n" );
            return;
        }

        curr_msec =
            ( curr_time.tv_sec * 1000 ) + ( curr_time.tv_nsec / 1000 );

        if ( ( start_msec + msec ) < curr_msec )
        {
            break;
        }
    }
}
```

```
void ilitek_sleep ( uint32_t msec )
{
      ilitek_delay( msec );
}
```

In case of real usage, probably, this is not the best way to implement the delay inside the driver or program - but we are using it here just for testing purposes so this is kind of the easiest way to get it. Read current time and block the program until the desired delay expires. Also you can see that sleep function just wrapped delay inside itself.

The more important point for the Linux OS is the "initialization" of the I2C. Actually this is not real initialization - this is just setup and the opening of the i2c bus, implementation of the `void ilitek_interface_init( void )` in your case might looks like this.

```
void ilitek_interface_init ( void )
{
    //  Initialize I2C bus.

    if ( ( fd = open ( "/dev/i2c-1", O_RDWR ) ) < 0 )
    {
        printf( "I2C Open error" );
    }

    if ( ioctl ( fd, I2C_SLAVE, 0x41 ) < 0 )
    {
        printf( "I2C Slave setup error" );
    }

    //  Initialize RST.
    //  Initialize INT.
    //  (Optional) Initialize serial port for printout.
}
```

As you can see we just have opened the file descriptor - in other cases `"/dev/i2c-1"` location may vary so it is necessary to update it depending on the system. Then we have configured the I2C slave address with `ioctl` call. We will not use GPIO pins for this demo due to fact that it is not necessary and might be little bit more complex to do it on Linux OS, so there is not initialization for the GPIOs. Also we will use standard output and `printf` to provide readable output, meaning there is nothing to initialize in relation to that.

4.2 GPIO INTERFACE

As we already said both GPIO pins are not necessary and not used in this demo. You can leave both functions related to the GPIOs empty.

```
void ilitek_gpio_reset_set ( uint8_t value )
{

}

uint8_t ilitek_gpio_int_get ( void )
{
      return 0;
}
```

4.3 I2C INTERFACE

There are two functions as interface to the I2C. :

- `int ilitek_i2c_read( uint8_t * data, int read_len )`
- `int ilitek_i2c_rw( uint8_t * cmd, int write_len, int delay, uint8_t * data, int read_len )`

Considering the previous code snippet - we have to use same file descriptor to read from the I2C bus and particular device. So implementation of the read function for the driver might looks like this.

```
int ilitek_i2c_read( uint8_t * data, int read_len )
{
    if ( rdlen != read( fd, data, rdlen ) )
    {
        return -1;
    }

    return 0;
}
```

The second function is very similar:

```
int ilitek_i2c_rw( uint8_t * cmd, int write_len, int delay,
 uint8_t * data, int read_len )
{
    if ( wrlen != write( fd, cmd, wrlen ) )
    {
        return -1;
    }

    if ( del > 0 )
    {
        ilitek_delay( del );
    }

    if ( rdlen != read( fd, data, rdlen ) )
    {
        return -1;
    }

    return 0;
}
```

The implementation of the functions will be probably always the same due the fact that there is no any dependency on the platform stuff here. It is just important to use proper file descriptor.

4.4 SERIAL INTERFACE

As we already said - we will use standard output to provide readable output from the library. So implementation of the `void ilitek_print ( const char * fmt, ... )` should be nothing more then forward of the content to the `printf`. So it might looks like this:

```
void ilitek_print ( const char * fmt, ... )
{
    char str[1024];
    va_list arg;

    va_start( arg, fmt );
    vsprintf( str, fmt, arg );
    va_end( arg );

    printf( str );
}
```

4.5 MAIN APPLICATION

When all of the previous functions are done, implementation of the code which should be placed in main should be quiet simple. An example how it looks like can be found inside `main.c.template` file.

What is happening there is that first we initialize system and driver and then inside infinite poll the touch data. Then we just print it out in human readable format to the terminal. If we take a closer look to the `ilitek_read_data_and_report_3XX` function call which is called from infinite loop we will realize that everything happens in there - all readout of existing touch data.

4.5.1 DATA READOUT

The main function for data readout it pretty long - there are lot of internal checks so we will try to focus just on the main points of interest. Also for better understanding it is important to say that all volatile data is placed inside `struct ilitek_ts_data ilitek_ctx` structure. This means that after each readout function call you can check the content of the structure to get desired information. This structure is huge, each information which can be read from the controller is placed inside separated field so we will go just through the most important ones related to touch data readout.

The first operation during touch data read is readout of the status register which carries information about number of touch points detected, depending on number of points detected the appropriate flag for each touch index will be set.

```
ilitek_data->touch_flag[i] = 1;
```

This means after the functions is executed, we can check `touch_flag` by index to check number of touches detected, for example:

```
int i;

for ( i = 0; i < ILITEK_SUPPORT_MAX_POINT; ++i )
{
      if ( ilitek_data->touch_flag[i] == 1 )
      {
            // Touch on index i is detected.
      }
}
```

Then for each index where flag is set we will store information about X and Y coordinates inside `tp` field. As you can see `tp` field is also structure so information about X coordinate is stored inside `x` field and info about Y coordinate is stored inside `y` filed. This means that we can use which fields in combination with `touch_flag` to get complete info about each detected point, so previous code snippet might be improved like this:

```
int i;
int x_coord;
int y_coord;

for ( i = 0; i < ILITEK_SUPPORT_MAX_POINT; ++i )
{
      if ( ilitek_data->touch_flag[i] == 1 )
```

```
        {
                x_coord = ilitek_data->tp[i].x;
                y_coord = ilitek_data→tp[i].y;

                // Do whatever you need to do with (X,Y).
        }
}
```

5 FURTHER STEPS

This was just an basic introduction to the ILI2xxx on Linux platforms. This adopted version of the original driver implementation is just simplified version which should provide you fast introduction to the interfacing the ILI2xxx device on the Linux platforms.

The next step is probably considering the original Linux driver which is developed in form of the kernel module. Also the improvement can be made by introducing the usage of the GPIO pins.

While this approach we used is more suitable for development - the kernel module is more suitable for real usage. The original library can be downloaded from ours github repository. In case of any question related to original library and documentation you must contact manufacturer (Ilitek) directly.