ILI2XXX QUICK START GUIDE

ILI2xxx Quick Start Guide

NoOS 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 "No operating system" (NoOS) implementation of the software. Document is mostly focused on usage with Arduino but same approach can be used on any other MCU.

Document requires basic knowledge of Arduino platform, GPIO and I2C bus. It will go through the setup of the hardware and software, integration of the ILITEK library and simple example which printout the X,Y position of the detected touch to the serial port. Main components are:

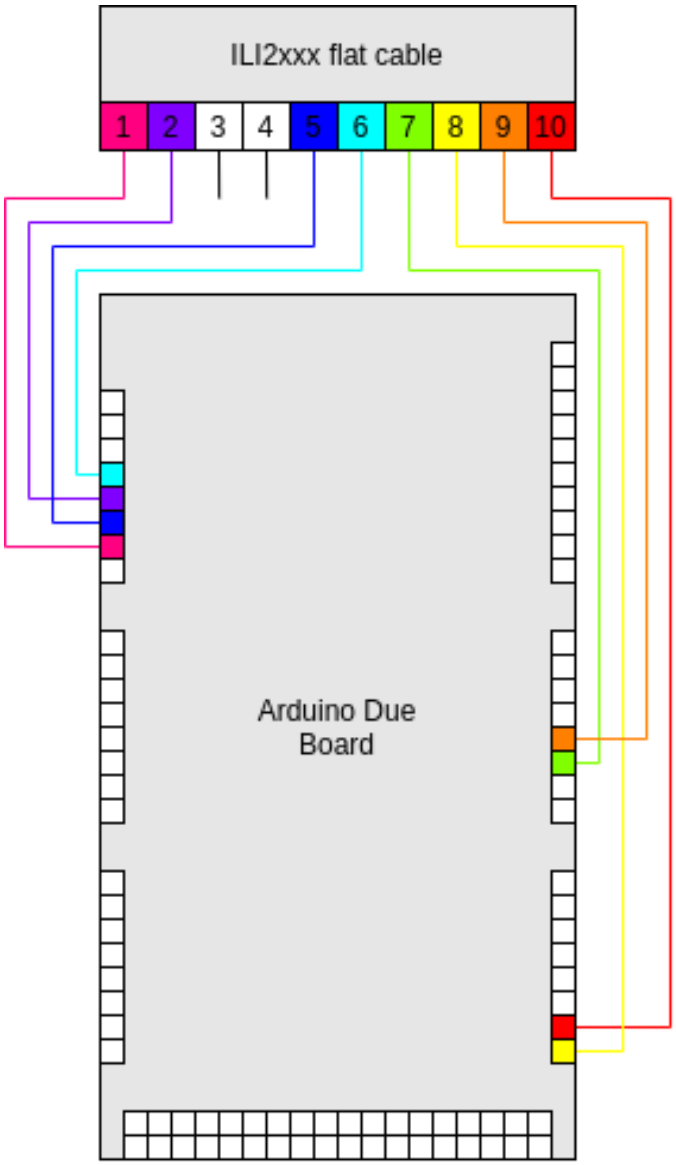
- Arduino due board
- Touch panel with IL2xxx controller
- ZIF breakout board
- Micro USB cable (Type B) with USB data support
- 8 wire jumpers (Male-Female)
- Arduino IDE installed on the PC (Windows/Linux)

Alongside with this it would be always nice to have breadboard which always give additional flexibility during work.

2 HARDWARE AND SOFTWARE SETUP

- Install and setup `Arduino IDE` on your PC
- Connect your Arduino board with PC using USB micro cable
- Connect Touch panel (ILI2xxx) flat cable to the breakout board
- Connect ZIF breakout board headers to the Arduino headers

Figure 1. Arduino Connection



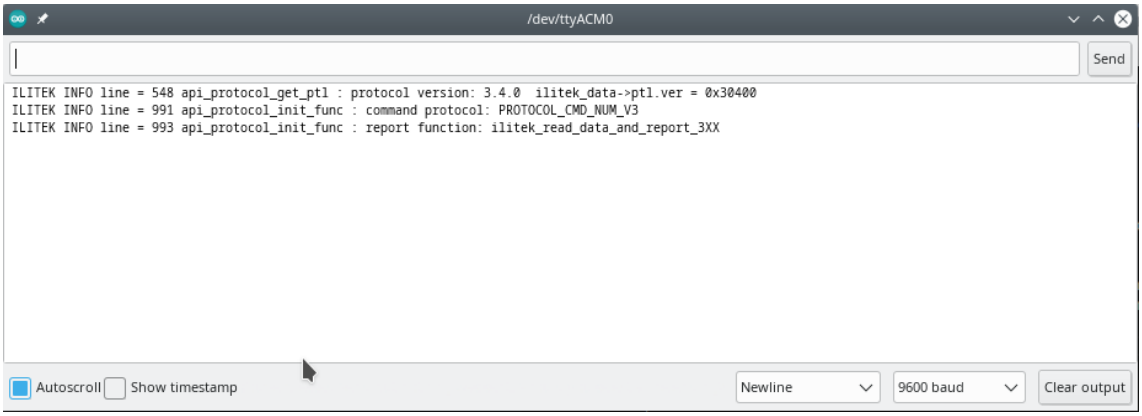
3 COMPILE AND RUN DEMO EXAMPLE

Demo application made for this "Getting started" purpose uses simplified version of the original library made for ILI2xxx on Linux OS. You can download this version of the library alongside with the demo from the ours github repository.

- Unpack the content and place on desired place on your OS.
- Start Arduino IDE and the go
  - File / Open
  - Inside unpacked folder from previous step, select `arduino_demo.ino` and open
  - Compile it program to the board
  - Turn on Arduino IDE serial monitor to track the logged information

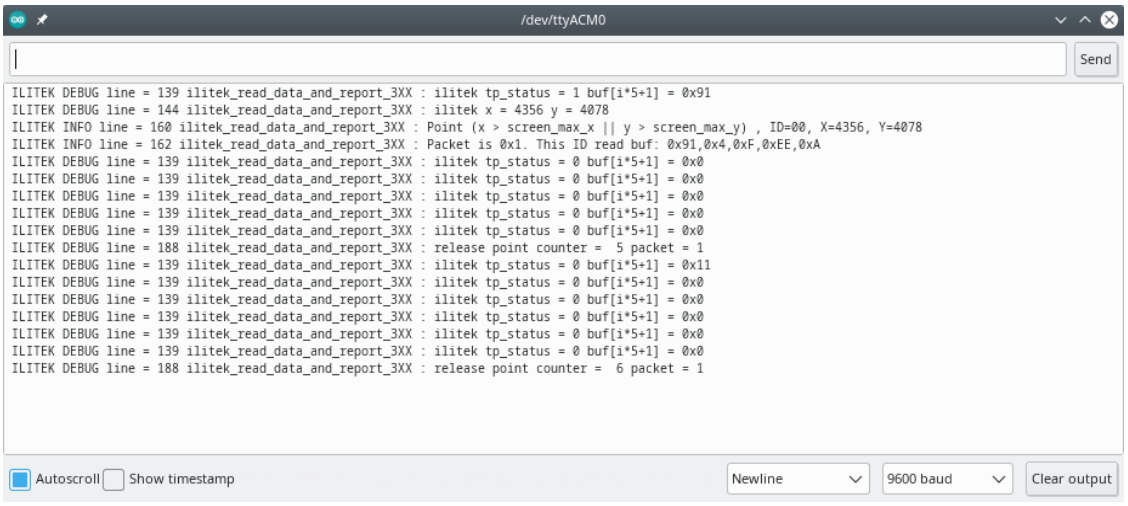
If everything went well you should see basic information like version of the firmware programmed to the ILI2xxx.

Figure 2. Initialization log



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

Figure 3. Touch log



4 PORT LIBRARY TO OTHER ARCHITECTURES

The library for no OS systems is written according to ANSI-C standard so you should be able to use it with almost any ANSI-C compliant C/C++ compiler. Inside this section we will go through all necessary functions to be implemented to have library and demo running on target platform/architecture. As example we will use STM32 HAL libraries but approach is the same for any other case.

The first of two steps for successful porting of the library is implementation of all functions declared in `port.h` file. There is already file `port.c.template` which can be used as start point but you can use your own file, it is just important that file is involved to the compilation.

There are already instructions what each of function should perform but no matter that we will go through each of them. We will consider that you are familiar with STM32Cube MX or STM32Cube IDE and that you have already generated initialization of the peripherals generated. This means that all provided functions can be placed inside `main.c` generated by STM32 project generation.

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 )`

for no OS systems you can both implement to make delay for provided amount of millisecond, so implementation for both function might looks like this.

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

If you are using project generator then implementation of the system initialization (System clock, GPIO. Communication peripherals) is probably already generated and placed inside main but anyway we will go through :

- `void ilitek_interface_init ( void )`

This function is place where it should be implemented in some cases, and in case you using `main.c.template` as start point you implementation of this function might looks like this.

```
void ilitek_interface_init ( void )
{
	MX_GPIO_Init( );
	MX_I2C1_Init( );
	MX_UART1_Init( );
}
```

4.2 GPIO INTERFACE

Now let's go with functions used for GPIO. As you probably realized there are only two GPIO pins used by ILI2xxx. RST and INT pin, let's implement two functions which are used for that purpose :

- `void ilitek_gpio_reset_set ( uint8_t value )`
- `uint8_t ilitek_gpio_int_get ( void )`

We will consider that RST pin is on PORTA pin 0 and INT pin is routed to the PORTA pin 1. Function for RST pin should set particular GPIO to the provided state while function for INT should return the state of the INT pin, so implementation might looks like this.

```
void ilitek_gpio_reset_set ( uint8_t value )
{
	if (value)
	{
		HAL_GPIO_WritePin( GPIOA, GPIO_PIN_0, GPIO_PIN_SET );
	}
	else
	{
		HAL_GPIO_WritePin( GPIOA, GPIO_PIN_0, GPIO_PIN_RESET );
	}
}

uint8_t ilitek_gpio_int_get ( void )
{
	if (GPIO_PIN_RESET == HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1))
	{
		return 0;
	}
	else
	{
		return 1;
	}
}
```

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 )`

We will consider that ILI2xxx if connected to the I2C1 module and that is previously initialized on appropriate speed. The first function should just read data from the I2C slave (in this case ILI2xxx), so implementation might looks like this:

```
int ilitek_i2c_read( uint8_t * data, int read_len )
{
	if ( HAL_OK != HAL_I2C_Master_Receive( &i2c1_handle,
	                                       0x41 « 1,
	                                       data,
	                                       (uint16_t) read_len,
	                                       1000 ))
	{
		return 1;
	}
```

```
	else
	{
		return 0;
	}
}
```

The second function is little bit more complex compared to this one due to fact that it performs write and then read operation on I2C bus. Also, delay might appear between operations and that condition should be implemented too, so it might looks like this :

```
int ilitek_i2c_rw( uint8_t * cmd, int write_len, int delay,
 uint8_t * data, int read_len )
{
	if ( HAL_OK != HAL_I2C_Master_Transmit( &i2c1_handle,
						0x41 « 1,
						cmd,
						(uint16_t) write_len,
						1000 ))
	{
		return 1;
	}

	if ( delay )
	{
		HAL_Delay( delay );
	}

	if ( HAL_OK != HAL_I2C_Master_Receive( &i2c1_handle,
						0x41 « 1,
						data,
						(uint16_t) read_len,
						1000 ))
	{
		return 1;
	}
	else
	{
		return 0;
	}
}
```

4.4 SERIAL INTERFACE

Serial interface is not must to have but it is good to have it for debug purposes. It is only used in single direction so only:

- `void ilitek_print ( const char * fmt, ... )`

This function is necessary to be implemented for test and debug and 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 );
	( void ) HAL_USART_Transmit ( &uart1_handle,
```

```
                                        str,
                                        (uint16_t) strlen( str ),
                                        1000 );
}
```

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 loop we waiting for INT pin goes low. That means that touch is detected and we can readout information from the ILI2xxx. Then we just print it out in human readable format to the serial port.

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 a basic introduction to library for ILI2xxx. This adopted version for NoOS implementation is just simplified version of the original library developed for Linux OS and it is made in manner with as few changes as possible to get it running in NoOS environment.

If you are really familiar with "Bare Metal" applications you will realize that lot can be additionally optimized as NoOS environment requires, for example, huge structure which carries all available information about controller.

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.