QuickLogic® ArcticLink® III VX/BX CSSPs – Software Integration Guide for MTK Platform, MIPI Command Mode

QuickLogic Application Note 99

Introduction

This application note describes the changes that must be made to the MediaTek (MTK) platform video driver for the QuickLogic ArcticLink III VX/BX Customer Specific Standard Product (CSSP) initialization, panel initialization and interface timing. The VxApp, a daemon for adapting VEE to ambient light and backlight changes, and the associated Android Java client, a graphical user interface (GUI), are also discussed.

This document includes the following sections:

NOTE: The following example code applies to VX3B2B silicon variant.

Video Driver Porting

ArcticLink III VX/BX CSSP Initialization

The ArcticLink III VX/BX CSSP must be initialized to pass video data to the bootloader and display the boot logo. To initialize the ArcticLink III VX/BX CSSP:

1. Apply **`sysclk`** (optional).
2. Toggle the reset.
3. Initialize the ArcticLink III VX/BX CSSP registers.
4. Make the following bootloader code changes.
   a. Add **`ql_init()`**.

```
Void ql_init()
{
…
      ret = i2c1_v1_init();
      DMS_LCD("Init I2C1 = %d\n",ret);
….
mt_set_gpio_mode(GPIO_VX3B2B_CLK,GPIO_VX3B2B_CLK_MODE);
mt_set_clock_output(GPIO_VX3B2B_CLK_PIN_CLK, GPIO_VX3B2B_PIN_FREQ,2);
mdelay(10);
```

```
mt_set_gpio_mode(GPIO_VX3B2B_RST,GPIO_VX3B2B_RST_MODE);
mt_set_gpio_out(GPIO_VX3B2B_RST,GPIO_OUT_ONE);
mdelay(10);
mt_set_gpio_out(GPIO_VX3B2B_RST,GPIO_OUT_ZERO);
mdelay(20);
mt_set_gpio_out(GPIO_VX3B2B_RST,GPIO_OUT_ONE);
….

  for(i = 0; i <
sizeof(ql_initialization_setting)/sizeof(ql_initialization_setting[0]); i
++)
  {
    result = ql_i2c_write
 (ql_initialization_setting[i].address, ql_initialization_setting[i].data,
11);

      DMS_LCD("result[%d] = 0x%x\n",i,result);
  }

  ql_Release_I2c();

  mdelay (100);
…
}
```

b. Call **`ql_init()`** in **`dsi_init()`**.

```
static DISP_STATUS dsi_init(UINT32 fbVA, UINT32 fbPA, BOOL isLcmInited)
{
….

#ifdef BUILD_UBOOT
#ifdef MTK_QLVX3B2B_SUPPORT
      LCD_SetResetSignal(1);
      LCD_SetResetSignal(0);
      mdelay(5);
      LCD_SetResetSignal(1);
      mdelay(50);
       ql_init();
//stop bootloader for Diolan board setup.
       //char c =serial_getc();
       //putc(c);
#endif
#endif
…..
}
```

5. In some bootloader versions the I²C clock is fixed at 1.2 MHz. If so, modify the bootloader I²C driver to 400 kHz as follows:

```
U32 i2c0_v1_set_speed (unsigned long clock, I2C_SPD_MODE mode, unsigned long
khz)
{
…
    if (mode == HS_MODE) {
        tmp  = __raw_readw(MT_I2C_HS) & ((0x7 << 12) | (0x7 << 8));
        tmp  = (sample_cnt_div & 0x7) << 12 | (step_cnt_div & 0x7) << 8 | tmp;
        __raw_writew(tmp, MT_I2C_HS);
        I2C_SET_HS_MODE(1);
    }
    else {
        tmp  = __raw_readw(MT_I2C_TIMING) & ~((0x7 << 8) | (0x3f << 0));

//      tmp  = (0/*sample_cnt_div*/ & 0x7) << 8 | (5/*step_cnt_div*/ & 0x3f)
<< 0 | tmp;
        tmp  = (sample_cnt_div & 0x7) << 8 | (step_cnt_div & 0x3f) << 0 | tmp;

        __raw_writew(tmp, MT_I2C_TIMING);
        I2C_SET_HS_MODE(0);
    }
….
}
```

6. The original I²C write supports 8 bytes of data. However, the ArcticLink III VX/BX CSSP needs 10 bytes of data.

```
U32 i2c1_v1_write (U8 chip, U8 *buffer, int len)
{
    ….
    /* polling mode : see if transaction complete */
    while (1) {
        status = I2C_INTR_STATUS;

        if ( status & I2C_TRANSAC_COMP) {
            ret = 0;
            ret_code = I2C_OK;
            break;
        }
    ….
        else if (time_out_val > 100000) {
            ret = 3;
            ret_code = I2C_WRITE_FAIL_TIMEOUT;
            printf("[i2c1_write] transaction timeout:%d\n", time_out_val);
```

```
                break;
            }
            time_out_val++;
            if( (time_out_val == 5000) && (data_left)){
        i2c1_v1_write_left_data((char *)&buffer[I2C_FIFO_SIZE],data_left);
           }
        }
    ….
        return ret_code;
    }
```

7. Initialize the ArcticLink III VX/BX CSSP registers through the I²C bus in kernel.

   a. Apply for RGB input use cases.

      - RGB to MIPI/DPI
      - RGB to LVDS

   b. Register the I²C address. The ArcticLink III VX/BX CSSP I²C address is 0x64 (depending on GPIO pin). Define the **`i2c_board_info`** structure and call **`i2c_register_board_info()`** with the correct bus number.

```
static struct i2c_board_info __initdata i2c_quickvx={
I2C_BOARD_INFO("quickvx", (0x64))};
```

   c. Use standard I²C driver initialization in the **`mtkfb.c`**.

```
static struct i2c_device_id i2c_quickvx_idtable[] = {
  { "i2c_quickvx", 0 },
  { }
};
MODULE_DEVICE_TABLE(i2c, i2c_quickvx_idtable);
static struct i2c_driver i2c_quickvx_driver = {
  .driver = {
    .owner = THIS_MODULE,
    .name  = "i2c_quickvx",
  },
  .id_table = i2c_quickvx_idtable,
  .probe    = i2c_quickvx_probe,
};
int __init mtkfb_init(void)
{
…..
```

```
#ifdef MTK_QLVX3B2B_SUPPORT
     i2c_register_board_info(1, &i2c_quickvx, 1);
     r= i2c_add_driver(&i2c_quickvx_driver);
     QL_DBG("i2c_add_driver ret = %d\n", r);
     if (r) {
       printk(KERN_ERR
         "%s: i2c_add_driver failed!\n", __func__);
       return r;
     }
#endif

…..
}
```

d. I²C read/write routines are provided by QuickLogic.

```
static struct i2c_client *I2cClient;
int ql_i2c_read(uint32 addr, uint32 *val, uint32 data_size);
int ql_i2c_write(long addr, long val, int data_size);
```

e. Send initialization data through the I²C interface.

```
struct chip_init_data vx3_init_data[] = {
//from init script
{ 0x700,0x30900080},
…..
{ 0x608,0x50F},
};
```

8. Initialize the ArcticLink III VX/BX CSSP registers through the MIPI DSI interface in kernel.

a. Apply for MIPI input use cases.

- MIPI/DPI to LVDS
- MIPI/DPI to MIPI/DPI
- MIPI/DBI to MIPI/DBI

b. The QuickLogic-specific generic DSI send routine **`DSI_set_cmdq_ql()`** sends out a DSI command in a generic packet. It is similar to the **`DSI_set_cmdq_V2()`**.

For example: An ArcticLink III VX/BX CSSP register write using the **`DSI_set_cmdq_ql()`**.

```
void ql_send_mipi_cmd()
{
….

        param[0] = 0x01;
        param[1] = 0x40;
        param[2] = ql_mipi_cmd_buf[i].reg & 0xff;
        param[3] = (ql_mipi_cmd_buf[i].reg & 0xff00) >> 8;
        param[4] =  ql_mipi_cmd_buf[i].data & 0x000000ff;
        param[5] =  (ql_mipi_cmd_buf[i].data & 0x0000ff00) >> 8;
        if(ql_mipi_cmd_buf[i].data_size == 4){
```

```
                param[6] =  (ql_mipi_cmd_buf[i].data & 0x00ff0000) >> 16;
                param[7] =  (ql_mipi_cmd_buf[i].data & 0xff000000) >> 24;
        }
        ql_data_len = ql_mipi_cmd_buf[i].data_size;
        DSI_set_cmdq_ql(0x5, ql_data_len+4, &param[0], 1);
…..
        }
```

c. For MIPI-DBI, tighten the MIPI send command to the frame update, so that the new changes (such as VEE strength) are effective on the first frame.

```
static void mtkfb_update_screen_impl(void)
{
….
#ifdef MTK_QLVX3B2B_SUPPORT
        ql_send_mipi_cmd();
#endif

        DISP_CHECK_RET(DISP_UpdateScreen(0, 0, fb_xres_update,
fb_yres_update));
….
}
```

Panel Data Structures

MIPI-specific Panel Data Structure is defined in the display driver **`lcm_get_params()`**. Use this section to change parameters such as the DSI packet size and display porch values.

```
static void lcm_get_params(LCM_PARAMS *params)
{
        memset(params, 0, sizeof(LCM_PARAMS));

        params->type   = LCM_TYPE_DSI;

        params->width  = FRAME_WIDTH;
        params->height = FRAME_HEIGHT;
…
        // Highly depends on LCD driver capability.
        #ifdef MTK_QLVX3B2B_SUPPORT
        params->dsi.packet_size=240;
        #else
        params->dsi.packet_size=256;
        #endif
…
        // Video mode setting
        params->dsi.PS=LCM_PACKED_PS_24BIT_RGB888;

        params->dsi.word_count=480*3;
        params->dsi.vertical_sync_active=2;
        params->dsi.vertical_backporch=2;
        params->dsi.vertical_frontporch=2;
        params->dsi.vertical_active_line=800;
```

```
        params->dsi.line_byte=2180;// 2256 = 752*3
        params->dsi.horizontal_sync_active_byte=26;
        params->dsi.horizontal_backporch_byte=206;
        params->dsi.horizontal_frontporch_byte=206;
        params->dsi.rgb_byte=(480*3+6);

        params->dsi.horizontal_sync_active_word_count=20;
        params->dsi.horizontal_backporch_word_count=200;
        params->dsi.horizontal_frontporch_word_count=200;
        ….
// Bit rate calculation
#ifdef MTK_QLVX3B2B_SUPPORT
params->dsi.pll_div1=24;// fref=26MHz, fvco=fref*(div1+1)(div1=0~63,
fvco=500MHZ~1GHz)
#else
params->dsi.pll_div1=29;// fref=26MHz, fvco=fref*(div1+1)(div1=0~63,
fvco=500MHZ~1GHz)
#endif

params->dsi.pll_div2=1;// div2=0~15: fout=fvo/(2*div2)

…
}
```

Panel Initialization

Initialization commands for the panels pass through the ArcticLink III VX/BX CSSP. For example:

- MIPI/DPI to MIPI/DPI
- MIPI/DBI to MIPI/DBI

Use the standard **`DSI_set_cmdq_V2()`** to send out the commands, but if the initialization commands have a non-standard DCS command, call **`DSI_set_cmdq_ql()`** to send it out with a generic command.

Backlight Control

Apply for a system that uses the QuickLogic PWM or IBC. Modify the backlight control routine in the video driver.

Suspend/Resume

If power to the ArcticLink III VX/BX CSSP stops, re-initialize the ArcticLink III VX/BX CSSP registers upon resume. The following is a MIPI to MIPI use case example:

On suspend:

1. Send MIPI a command to Set Backlight to 0.
2. Send the MIPI panel a command (such as Set Display Off (0x28) and Enter Sleep Mode (0x10)).
3. Power down the ArcticLink III VX/BX CSSP.

4. Stop **sysclk** (optional).

```
static void qlvx_early_suspend(struct early_suspend *h)
{
….
   hwPowerDown(MT65XX_POWER_LDO_VCAM_IO,"quickvx");
   hwPowerDown(MT65XX_POWER_LDO_VM12_2,"quickvx");
   mdelay(10);
   mt_set_gpio_mode(GPIO_VX3B2B_CLK,GPIO_VX3B2B_GPIO_MODE);
   mt_set_gpio_out(GPIO_VX3B2B_CLK, GPIO_OUT_ZERO);
…
}
```

On resume:

1. Power on the ArcticLink III VX/BX CSSP.
2. Start **sysclk**.
3. Toggle the reset.
4. GPIO.
5. Re-initialize the ArcticLink III VX/BX CSSP registers.
6. Send the MIPI panel a command (such as Exit Sleep Mode (0x11) or Set Display On (0x29)).
7. Set backlight to normal.

```
static void qlvx_late_resume(struct early_suspend *h)
{
….

   hwPowerOn(MT65XX_POWER_LDO_VM12_2, VOL_1200,"quickvx");
   hwPowerOn(MT65XX_POWER_LDO_VCAM_IO, VOL_1800,"quickvx");
   mdelay(10);

   mt_set_gpio_mode(GPIO_VX3B2B_CLK,GPIO_VX3B2B_CLK_MODE);
   mt_set_clock_output(GPIO_VX3B2B_CLK_PIN_CLK, GPIO_VX3B2B_PIN_FREQ,2);
   mdelay(10);

   mt_set_gpio_mode(GPIO_VX3B2B_RST,GPIO_VX3B2B_RST_MODE);
   mt_set_gpio_out(GPIO_VX3B2B_RST,GPIO_OUT_ONE);
   mdelay(10);
   mt_set_gpio_out(GPIO_VX3B2B_RST,GPIO_OUT_ZERO);
   mdelay(20);
….
}
```

To register suspend/resume handling in the I^2C driver or video driver:

```
static int __devinit i2c_quickvx_probe(struct i2c_client *client,
                                const struct i2c_device_id *id)
{
```

```
…
#ifdef CONFIG_HAS_EARLYSUSPEND
    obj->early_drv.level    = EARLY_SUSPEND_LEVEL_DISABLE_FB + 1,
    obj->early_drv.suspend  = qlvx_early_suspend,
    obj->early_drv.resume   = qlvx_late_resume,
    register_early_suspend(&obj->early_drv);
#endif
…
}
```

To make changes to **`dsi_enable_power()`** for suspend/resume:

```
static DISP_STATUS dsi_enable_power(BOOL enable)
{
    disp_drv_dsi_init_context();

    if(lcm_params->dsi.mode == CMD_MODE) {

       if (enable) {
….. 
          DSI_Reset();
#ifdef MTK_QLVX3B2B_SUPPORT
#ifdef BUILD_UBOOT
          ql_reinit();
                mdelay(20);
          LCD_CHECK_RET(LCD_PowerOn());
#else
          quickvx_ReInit();
          mdelay(20);
          LCD_CHECK_RET(LCD_PowerOn());
#endif
#else
          LCD_CHECK_RET(LCD_PowerOn());
#endif
       } else {
….
}
```

VxApp Interface

The VxApp interface is for a system that does not have I^2C connected to the ArcticLink III VX/BX CSSP. For example, MIPI/DPI to MIPI/DPI or a system that does not support Linux kernel I^2C development interface.

To support a specific operation expose the **`Sysfs`** structure. For example, the ArcticLink III VX/BX CSSP register read/write access:

```
static struct device_attribute mipi_quickvx_attributes[] = {
    __ATTR(driver_info, 0444, show_mipi_quickvx_driver_info, NULL),
    __ATTR(read_reg, 0644, show_mipi_quickvx_read, store_mipi_quickvx_read),
    __ATTR(write_reg, 0644, NULL, store_mipi_quickvx_write),
    __ATTR(write_bulk, 0644, NULL, store_mipi_quickvx_write_bulk),
    __ATTR(i2c_raw, 0644, show_i2c_raw, store_i2c_raw),
```

```
    __ATTR(dsi_raw, 0644, show_dsi_raw, store_dsi_raw),
    __ATTR(resume_count, 0444, show_resume_count, NULL),
    __ATTR(force_screen_update, 0644, show_force_screen_update,
store_force_screen_update),
};
```

QuickLogic provides function implementation details. For example, to displays the version number:

```
static ssize_t show_mipi_quickvx_driver_info(struct device *dev,
struct device_attribute *attr, char *buf)
{
    return snprintf(buf, PAGE_SIZE, "%s \n", QL_MIPI_DEVICE_NAME);
}
```

Example Codes

The following reference codes are located in the **driver source code** directory.

- RGB to MIPI/DPI
- RGB to LVDS
- MIPI/DPI to LVDS
- MIPI/DPI to MIPI/DPI
- MIPI/DBI to MIPI/DBI

NOTE: THE FOLLOWING SECTIONS APPLY TO VX VARIANTS OF THE ARCTICLINK III VX ONLY.

VxApp

VxApp Features

The VxApp has the following features:

- Communicates with the ArcticLink III VX/BX CSSP through I^2C dev or video drivers in the system.
- Communicates with Android light sensor and backlight control services.
  - Receives events from Android on ambient light and display brightness level changes and adjusts VEE operation.
  - Changes the display brightness level through Android.
  - Polls the ambient light level at a defined interval through I^2C drivers independent of Android light sensor service.
- Implements a socket server; any Android application can talk to VxApp through a socket client interface.
- Ability to read/write registers in the ArcticLink III VX/BX CSSP using the Command line mode.
  - Good hardware debugging tool.
  - Useful during the calibration process.

VxApp Porting

Most of the device-dependent variables are defined in the device-specific header files including:

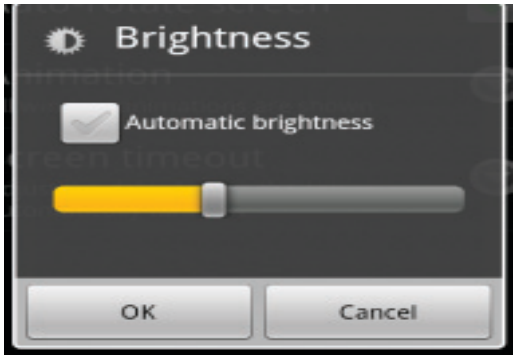
- ArcticLink III VX/BX CSSP specific – Type of ArcticLink III VX/BX CSSP in the system, such as vx3b3b, vx5a1d, etc.
- I^2C interface for ArcticLink III VX/BX CSSP communication – Linux kernel I^2C dev-interface file (bus) where the ArcticLink III VX/BX CSSP is located and the I^2C system address.
- Video driver **sysfs** interface for ArcticLink III VX/BX CSSP communication – Video driver **sysfs** path. For example **#define MIPI_SYSFS_PATH "/sys/devices/platform/mipi_quickvx.513/"**.
- System ambient light sensor – Ambient light sensor **sysfs** path, device driver direct access, or input event path (i.e., **#define AL_INPUT_EVENT_PATH "/dev/input/event4"**) and **min/dark/indoor/outdoors/max lux** value.
- System backlight control – Backlight control **sysfs** path and min/max value for read/write to the **sysfs** path.
- Display specifications information – Image width and height, Hsync front porch, width and back porch, Vsync front porch, width and back porch, MIPI video/command mode and RGB mode (RGB565/666/888)
- Calibration table – Calibrated on-site by QuickLogic.

Java Application

An Android Java application is used to enable/disable the auto brightness and VEE strength control, and to support internet connection sharing (ICS). There are two versions:

- The lite version uses the Standard Android Brightness Preference GUI. The changes to the system are minimal.
- The powerful version uses the QuickLogic GUI which offers more control, but involves more changes to the system.

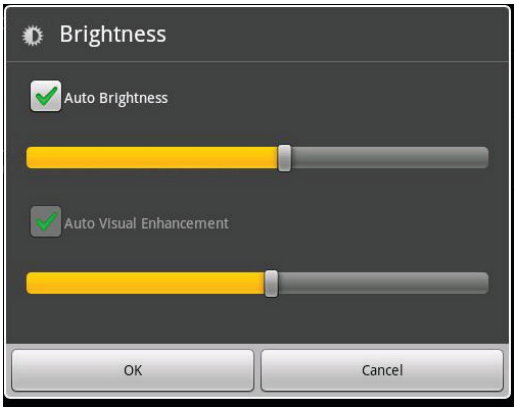
Standard Android Brightness Preference GUI – Lite Version



To use the Standard Android Brightness Preference GUI:

1. Disable the Android automatic brightness control in **`PowerManagerService.java`**.
2. Load the background application **`VEEServiceLite.java`**.
3. Monitor the Android global variable **`Settings.System.SCREEN_BRIGHTNESS_MODE`**.
4. If set, send a command to VxApp to start controlling backlight and VEE strength for different ambient light.
5. If clear, stop all backlight and VEE strength controls.

QuickLogic GUI – Powerful Version



To use the QuickLogic GUI:

The **`src`** folder contains all the files needed for the changes. Some files are for replacing an entire file in the Android source tree (**`packages/apps/Settings`**), while some are for modifying the original file.

1. Add new intents in **`AndroidManifest.xml.`**

```
<service android:enabled="true" android:name="VEEService">
    <intent-filter>
       <action android:name = "com.android.settings.VEEService"/>
    </intent-filter>
</service>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

```
<receiver android:name="VEEReceiver">
    <intent-filter>
       <action android:name ="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

2. Use the **BrightnessPreference.java** class to replace the original class.
3. Use the **VeeAppControl.java** class to control communication to VxApp daemon.
4. Use the **NavtiveVxControl.java** class to do native socket communication to VxApp daemon.
5. Use the **VEEService.java** service class to receive the ambient light sensor event.
6. Use the **VEEReceiver.java** receiver for the **BOOT_COMPLETED** event to start the **VEEService** on boot.

Comparable to the Standard Android Brightness Preference GUI, the QuickLogic GUI includes the following features.

- Implementation for **settings.apk**.
  - Auto Brightness automatically adjusts the backlight based on the current ambient light.
  - Provides individual user preference setting in addition to the Auto Brightness feature, adjusting brightness even with Auto Brightness enabled.
  - Auto Visual Enhancement automatically adjusts VEE compensation based on current ambient light and backlight level.
  - Provides individual user preference setting in addition to the auto VEE compensation, adjusting VEE compensation even with Auto Visual Enhancement enabled.
  - Auto Visual Enhancement with manual backlight control can be selected (with Auto Brightness deselected). In this mode, VEE compensation is adjusted based on the current ambient light level and user's selected backlight setting.
- Implements a socket client to communicate user selections with VxApp daemon.

Contact Information

Phone: (408) 990-4000 (US)
(647) 367-1014 (Canada)
+(44) 1932-21-3160 (Europe)
+(886) 26-603-8948 (Taiwan)
+(86) 21-5179-8474 (China)
E-mail: info@quicklogic.com
Sales: America-sales@quicklogic.com
Europe-sales@quicklogic.com
Asia-sales@quicklogic.com
Japan-sales@quicklogic.com
Support: www.quicklogic.com/support
Internet: www.quicklogic.com

Revision History

Revision	Date	Originator and Comments
A	September 2012	Sunny Lai and Kathleen Bylsma

Notice of Disclaimer

QuickLogic is providing this design, product or intellectual property "as is." By providing the design, product or intellectual property as one possible implementation of your desired system-level feature, application, or standard, QuickLogic makes no representation that this implementation is free from any claims of infringement and any implied warranties of merchantability or fitness for a particular purpose. You are responsible for obtaining any rights you may require for your system implementation. QuickLogic shall not be liable for any damages arising out of or in connection with the use of the design, product or intellectual property including liability for lost profit, business interruption, or any other damages whatsoever. QuickLogic products are not designed for use in life-support equipment or applications that would cause a life-threatening situation if any such products failed. Do not use QuickLogic products in these types of equipment or applications.

QuickLogic does not assume any liability for errors which may appear in this document. However, QuickLogic attempts to notify customers of such errors. QuickLogic retains the right to make changes to either the documentation, specification, or product without notice. Verify with QuickLogic that you have the latest specifications before finalizing a product design.

Copyright and Trademark Information