Samsung

ARTIK™ Modules

Driver Developer's Guide

TABLE OF CONTENTS

VERSION HISTORY

Revision	Date	Description	Maturity
V1.0	October 19, 2016	ARTIK Driver Developers Guide	Release

TARGET AUDIENCE

This document is intended for developers who do have previous experience writing or modifying kernel drivers.

During the course of this document assumptions will be made that are obvious to the experienced kernel driver developer, but probably seem out of place for developers without this experience.

For developers who do not have this background but want to start developing kernel level drivers, there are many tutorials available on the internet that can help guide you in the right direction.

This document does describe driver development techniques that can be applied to all our Linux® based Samsung ARTIK™ platforms, however throughout this document we use the ARTIK 710 Module as our platform of choice.

BUILDING THE KERNEL

Before we start our discussions about how to write a kernel driver it is good to first describe how to build a full kernel recovery image for the Samsung ARTIK™ 710 Module. In order to do this our first step is to install the DFU Tool.

DFU TOOL

The DFU tool is a generic firmware upgrade tool that allows for uploading and downloading firmware into devices like the ARTIK 710 Module using USB. The first step is to install 'dfu-util' either on a Windows® based or a Linux® based host PC.

WINDOWS

Installation on Windows happens in 4 steps namely:

1. Download the latest version from '*http://dfu-util.sourceforge.net/releases/*'.
2. Extract the zip file 'dfu-util-0.9-win64.zip'.
3. Download and install a firmware driver, for instance 'libusb32' from '*http://zadig.akeo.ie/*', for your host PC.
4. Connect a micro-USB cable between your host PC and the ARTIK 710 Module.

Once installed you are ready to do a 'dfu' based kernel image upgrade using your Windows PC.

LINUX

For installation on a Linux host PC use:

```
$ sudo apt-get install dfu-util
```

And add a 'udev' rule using:

```
$ vi /etc/udev/rules.d/51-android.rules
SUBSYSTEM=="usb",ATTR{idVendor}=="04e8",MODE="0666"
```

Once installed you are ready to do a 'dfu' based kernel image upgrade using your Linux PC.

AARCH64 TOOLCHAIN

The second step is to install the 'aarch64' toolchain using:

```
$ sudo apt-get install gcc-aarch64-linux-gnu gcc-arm-linux-gnueabihf device-tree-compiler kpartx
android-tools-fsutils
```

As an alternative you can also use the 'linaro' toolchain from *https://releases.linaro.org/components/toolchain/binaries/latest-5/aarch64-linux-gnu/*. Make certain that you properly set the 'PATH' environment variable pointing to the custom toolchain.

BOARD SUPPORT PACKAGE

Next we will download the BSP source files from GitHub *https://github.com/SamsungARTIK* and extract them using:

```
$ mkdir artik710
$ cd artik710
$ tar xf /opt/u-boot-artik7-0710GC0F-41F-01Q5.tar.gz
$ tar xf /opt/linux-artik7-0710GC0F-41F-01Q5.tar.gz
$ tar xf /opt/build-artik-0710GC0F-41F-01Q5.tar.gz
$ tar xf /opt/boot-firmwares-artik710-0710GC0F-41F-01Q5.tar.gz
```

EMMC RECOVERY IMAGE

Once you extracted the BSP files from GitHub you can build your own recovery image on eMMC using:

```
$ cd build-artik
$ ./release.sh -c config/artik710.cfg
```

Once created you will find the image 'artik710_sdcard-UNRELEASED-20160824.133640.img' under:

```
$ cd output/images/artik710/UNRELEASED/20160824.133640/
```

SD-Card Bootable Image

As an alternative you can create a bootable image on an SD card using:

```
$ cd build-artik
$ ./release.sh -c config/artik710.cfg -m
```

Once created you will find the image 'artik710_sdcard-UNRELEASED-20160824.133640.img' under:

```
$ cd output/images/artik710/UNRELEASED/20160824.133640/
```

Once you have written the image using 'dd' you can expand the root file system. If the sdcard device mount is '/dev/sd[X]' use:

```
$ ./expand_rootfs.sh /dev/sd[X]
```

Kernel Image

You can start with setting your customer kernel configurations using:

```
$ cd linux-artik7
$ make ARCH=arm64 artik710_raptor_defconfig
$ make ARCH=arm64 menuconfig
< - Edit your configurations - >
$ make ARCH=arm64 savedefconfig
$ mv defconfig arch/arm64/configs/artik710_raptor_defconfig
```

Once you have set your configurations build the kernel image using:

```
$ cd build-artik
$ ./build_kernel.sh -b artik710
$ ./mkbootimg.sh -b artik710
```

The build image can be found at 'output/images/artik710'. Once the kernel image has been created you can flash the kernel using 3 alternate methods, namely:

1. Fusing the image from the host PC using 'fastboot'.
2. Fusing the image from the host PC using 'dfu'.
3. Fusing the image from the host PC using 'ums'.

FASTBOOT

Enter the 'fastboot' mode from your ARTIK 710 Module using:

```
$ fastboot 0
```

Now from your host PC fuse the following images:

```
$ sudo fastboot flash partmap boot-firmwares-artik710/partmap_emmc.txt
$ sudo fastboot flash boot build-artik/output/images/artik710/boot.img
$ sudo fastboot flash modules build-artik/output/images/artik710/modules.img
$ sudo fastboot reboot
```

DFU

To install the Device Firmware Upgrade (DFU) tool see section *Before we start* our discussions about how to write a kernel driver it is good to first describe how to build a full kernel recovery image for the Samsung ARTIK™ 710 Module. In order to do this our first step is to install the DFU Tool.

DFU Tool. Now enter the DFU mode from the u-boot shell ('artik710#') on your ARTIK 710 Module using:

```
artik710# dfu 0 mmc 0
```

Now you can install the new kernel images from your host PC using:

```
$ sudo dfu-util -a 12 -D build-artik/output/images/artik710/boot.img
$ sudo dfu-util -a 13 -D build-artik/output/images/artik710/modules.img
```

UMS

First mount your eMMC storage device using 'dfu/ums' and enter the 'ums' mode from your ARTIK 710 Module u-boot shell ('artik710#') using:

```
$ ums 0 mmc 0
```

Next mount the images as loop device and copy all files into the 'ums' mount point using:

```
$ mkdir boot_mnt modules_mnt
$ sudo mount -o loop build-artik/output/images/artik710/boot.img boot_mnt
$ sudo rm -rf /media/{userid}/boot/*
$ sudo cp -rf boot_mnt /media/{userid}/boot
$ sudo umount boot_mnt
$ sudo mount -o loop build-artik/output/images/artik710/modules.img modules_mnt
$ sudo rm -rf /media/{userid}/modules/*
$ sudo cp -rf modules_mnt/* /media/{userid}/modules
$ sudo umount modules_mnt
```

Now stop the 'ums' mode by typing 'ctrl+c' on your keyboard and reboot from the u-boot shell using:

```
artik710# reset
```

EXPLORING THE DISPLAY DRIVER

INTRODUCTION

This section will describe how to add your own panel driver into the DRM/KMS graphics stack, using the DRM panel driver format.

DRM PANEL DRIVER

First familiarize yourself with the DRM panel base driver source files that can be found in GitHub under:

- 'drivers/gpu/drm/drm_panel.c'
- 'include/drm/drm_panel.h'

The panel driver works using a familiar callback mechanism, in this particular case the 'drm_panel_funcs' callback function needs modification. The structure of the callback function is given below:

```
struct drm_panel_funcs {
    int (*disable)(struct drm_panel *panel);
    int (*unprepare)(struct drm_panel *panel);
    int (*prepare)(struct drm_panel *panel);
    int (*enable)(struct drm_panel *panel);
    int (*get_modes)(struct drm_panel *panel);
    int (*get_timings)(struct drm_panel *panel, unsigned int num_timings,
               struct display_timing *timings);
};
```

The descriptions of the various functions covered in the callback structure are given below:

```
* @disable: disable panel (turn off back light, etc.)
* @unprepare: turn off panel
* @prepare: turn on panel and perform set up
* @enable: enable panel (turn on back light, etc.)
* @get_modes: add modes to the connector that the panel is attached to and
* return the number of modes added
* @get_timings: copy display timings into the provided array and return
* the number of display timings available
```

A typical panel session distinguishes the following steps:

```
Initialize the driver and add panel driver using:
    • drm_panel_init
    • drm_panel_add
Query the panel modes using:
    • get_modes
Connect panel to DPMS ("on") using:
    • prepare
    • enable
Disconnect panel from DPMS ("off") using:
    • disable
    • unprepare
Remove panel driver using:
    • drm_panel_remove
```

LVDS PANEL DRIVER

The ARTIK 710 Module supports 6x LVDS output channels (5 data channels and 1 clock channel) adhering to JEIDA and VESA data formats.

In this section we describe how to use the ARTIK 710 Development board with the Donguan's LVDS panel (gst7d0038 7" 1024x600 LVDS panel).

Most of the LVDS panels use a simple panel driver because the initialization sequences are relatively simple. You can find which panels are already supported from the 'simple_panel' driver found under '*drivers/gpu/drm/panel/panel-simple.c*'.

A supported list of panel drivers is given below:

```
.compatible = "ampire,am800480r3tmqwa1h",
.compatible = "auo,b101aw03",
.compatible = "auo,b101ean01",
.compatible = "auo,b101xtn01",
.compatible = "auo,b116xw03",
.compatible = "auo,b133htn01",
.compatible = "ampire,am800480r3tmqwa1h",
.compatible = "auo,b101aw03",
.compatible = "auo,b101ean01",
.compatible = "auo,b101xtn01",
.compatible = "auo,b116xw03",
.compatible = "auo,b133htn01",
.compatible = "auo,b133xtn01",
.compatible = "avic,tm070ddh03",
.compatible = "chunghwa,claa101wa01a",
.compatible = "chunghwa,claa101wb01",
.compatible = "edt,et057090dhu",
.compatible = "edt,et070080dh6",
.compatible = "edt,etm0700g0dh6",
.compatible = "foxlink,fl500wvr00-a0t",
.compatible = "giantplus,gpg482739qs5",
.compatible = "hannstar,hsd070pww1",
.compatible = "hit,tx23d38vm0caa",
.compatible = "innolux,at043tn24",
.compatible = "innolux,g121i1-l01",
.compatible = "innolux,n116bge",
.compatible = "innolux,n156bge-l21",
.compatible = "innolux,zj070na-01p",
.compatible = "lg,lp129qe",
.compatible = "ortustech,com43h4m85ulc",
.compatible = "samsung,ltn101nt05",
.compatible = "samsung,ltn140at29-301",
.compatible = "shelly,sca07010-bfn-lnn",
.compatible = "dongguan,gst7d0038",
```

To add your own LVDS panel driver follow the steps below:

1. Add your data into the data structures 'drm_display_mode' and 'panel_desc':

```
static const struct drm_display_mode dongguan_gst7d0038_mode = {
    .clock = 51206,
    .hdisplay = 1024,
    .hsync_start = 1024,
    .hsync_end = 1024 + 1 + 319,
    .htotal = 1024 + 1 + 319,
    .vdisplay = 600,
    .vsync_start = 600,
    .vsync_end = 600 + 1 + 34,
    .vtotal = 600 + 1 + 34,
    .vrefresh = 60,
};

static const struct panel_desc dongguan_gst7d0038 = {
    .modes = &dongguan_gst7d0038_mode,
    .num_modes = 1,
    .size = {
        .width = 154,
        .height = 85,
```

```
    },
};
```

The 'drm_display_mode' data structure is defined under 'include/drm/drm_modes.h'.

2. Once the structures are defined add a device tree into the 'platform_of_match' structure that can be found in the 'panel-simple.c' file:

```
static const struct of_device_id platform_of_match[] = {
    {
        ...
    }, {
        .compatible = "dongguan,gst7d0038",
        .data = &dongguan_gst7d0038,
    }, {
    }, {
        /* sentinel */
    }
};
MODULE_DEVICE_TABLE(of, platform_of_match);
```

3. Next add the LVDS device tree node. You can find the LVDS panel node document from 'Documentation/devicetree/bindings/drm/nexell/lvds_panel.txt'. Add the node into the 'arch/arm64/boot/dts/nexell/s5p6818-artik710-raptor-common.dtsi' file:

```
&dp_drm_lvds {
    status = "ok";

    remote-endpoint = <&lvds_panel>;

    dp_control {
        clk_src_lv0 = <3>;
        clk_div_lv0 = <16>;
        clk_src_lv1 = <7>;
        clk_div_lv1 = <1>;
        out_format = <3>;
    };
};
```

4. Associate your 'lvds_panel' node with the remote end point using:

```
panel: panel-simple {
        compatible = "dongguan,gst7d0038";
        enable-gpios = <&gpio_e 30 0>;
        backlight = <&backlight>;
        status = "okay";

        port {
            lvds_panel: endpoint {
            };
        };
    };
```

The panel requires a GPIO to turn on or off. In this case the panel does not require any regulator to provide power. If your panel does require a regulator, you have to add a regulator property in the panel node.

Backlight Control

When you want to adjust the brightness levels of your panel, you can add the backlight control driver. There are many backlight drivers. You can find them under 'drivers/video/backlight'. The ARTIK 710 Development board is using PWM-based backlight control. For PWM backlight control to work you will need to add a backlight node that is PWM backlight compatible. You can find the documentation of the backlight driver from 'Documentation/devicetree/bindings/video/backlight/pwm-backlight.txt'.

An example of a backlight structure is given below:

```
backlight: pwm-backlight {
        compatible = "pwm-backlight";
        pwms = <&pwm 0 20000 0>;
        pwm-names = "pwm-backlight";
        brightness-levels = < 0  1  2  3  4  5  6  7  8  9
                     10 11 12 13 14 15 16 17 18 19
                     20 21 22 23 24 25 26 27 28 29
                     30 31 32 33 34 35 36 37 38 39
                     40 41 42 43 44 45 46 47 48 49
                     50 51 52 53 54 55 56 57 58 59
                     60 61 62 63 64 65 66 67 68 69
                     70 71 72 73 74 75 76 77 78 79
                     80 81 82 83 84 85 86 87 88 89
                     90 91 92 93 94 95 96 97 98 99
                    100>;
        default-brightness-level = <60>;
        status = "okay";
    };
```

MIPI DSI Panel Driver

Unlike the LVDS panel that usually has simple initialization sequences you may have to write your own panel driver because of more complex initialization requirements. Even if you include your own panel driver you can still add the description into the 'simple-panel' driver. The following steps should be followed when creating your own driver:

1. Add your configuration in 'drivers/gpu/drm/panel/Kconfig'

```
config DRM_PANEL_S6E8FA0
    tristate "S6E8FA0 DSI video mode panel"
    depends on OF
    select DRM_MIPI_DSI
    select VIDEOMODE_HELPERS
```

2. Run Make to build 'drivers/gpu/drm/panel/Makefile'

```
obj-$(CONFIG_DRM_PANEL_S6E8FA0) += panel-s6e8fa0.o
```

3. Create your own panel file, an example is given under 'drivers/gpu/drm/panel/panel-s6e8fa0.c':

```
ctx->is_power_on = false;
    dsi->lanes = 4;
    dsi->format = MIPI_DSI_FMT_RGB888;
    dsi->mode_flags = MIPI_DSI_MODE_VIDEO | MIPI_DSI_MODE_VIDEO_BURST
        | MIPI_DSI_MODE_VIDEO_HFP | MIPI_DSI_MODE_VIDEO_HBP
        | MIPI_DSI_MODE_VIDEO_HSA | MIPI_DSI_MODE_VSYNC_FLUSH;
```

   The mode flags and format can be found under 'include/drm/drm_mipi_dsi.h'. You have to configure the settings according to your panel configurations.
   Some of the parameters are available in the device tree. Please refer to the 's6e8fa0_parse_dt' function for more information.

4. Get voltage regulator naming from device tree and fill ctx:

```
ctx->supplies[0].supply = "vdd3";
    ctx->supplies[1].supply = "vci";
    ret = devm_regulator_bulk_get(dev, ARRAY_SIZE(ctx->supplies),
                        ctx->supplies);
```

5. Enable GPIO functionality and fill ctx:

```
ctx->reset_gpio = of_get_named_gpio(dev->of_node, "reset-gpio", 0);
```

```
if (ctx->reset_gpio < 0) {
    dev_err(dev, "cannot get reset-gpios %d\n",
        ctx->reset_gpio);
    return ctx->reset_gpio;
}

ret = devm_gpio_request(dev, ctx->reset_gpio, "reset-gpio");
if (ret) {
    dev_err(dev, "failed to request reset-gpio\n");
    return ret;
}
```

6. Initialize drm panel using:

```
drm_panel_init(&ctx->panel);
ctx->panel.dev = dev;
ctx->panel.funcs = &s6e8fa0_drm_funcs;

ret = drm_panel_add(&ctx->panel);
if (ret < 0) {
    backlight_device_unregister(ctx->bl_dev);
    return ret;
}
```

7. Attach MIPI DSI layer using:

```
ret = mipi_dsi_attach(dsi);
```

Information on the interface can be found under 'drivers/gpu/drm/drm_mipi_dsi.c' and 'include/drm/drm_mipi_dsi.h'.

8. You also have to implement callbacks:

```
static const struct drm_panel_funcs s6e8fa0_drm_funcs = {
    .disable = s6e8fa0_disable,
    .unprepare = s6e8fa0_unprepare,
    .prepare = s6e8fa0_prepare,
    .enable = s6e8fa0_enable,
    .get_modes = s6e8fa0_get_modes,
};
```

9. The 'get_modes' callback has the responsibility to fill the 'drm_display_mode' structure. The 'drm_mode_probed_add' will add the mode into the drm probed mode list:

```
struct drm_display_mode *mode;

    mode = drm_mode_create(connector->dev);
    if (!mode) {
        DRM_ERROR("failed to create a new display mode\n");
        return 0;
    }

    drm_display_mode_from_videomode(&ctx->vm, mode);
    mode->width_mm = ctx->width_mm;
    mode->height_mm = ctx->height_mm;
    connector->display_info.width_mm = mode->width_mm;
    connector->display_info.height_mm = mode->height_mm;

    mode->type = DRM_MODE_TYPE_DRIVER | DRM_MODE_TYPE_PREFERRED;
    drm_mode_probed_add(connector, mode);
```

During DPMS on (power on), prepare -> enable will be called from the drm connector driver. So, you have to provide the proper power and init-sequences.

During DPMS off (power off), disable -> unprepare will be called from the drm connector driver. So, you have to provide the proper power-off sequences.

10. Backlight control, some panels like an AMOLED display can adjust brightness through gamma control. You can add the backlight control to your driver using:

```
ctx->bl_dev = backlight_device_register("s6e8fa0", dev, ctx,
                        &s6e8fa0_bl_ops, NULL);
    if (IS_ERR(ctx->bl_dev)) {
        dev_err(dev, "failed to register backlight device\n");
        return PTR_ERR(ctx->bl_dev);
    }

    ctx->bl_dev->props.max_brightness = MAX_BRIGHTNESS;
    ctx->bl_dev->props.brightness = DEFAULT_BRIGHTNESS;
    ctx->bl_dev->props.power = FB_BLANK_POWERDOWN;
```

11. Provide proper callback functions such as:

```
static const struct backlight_ops s6e8fa0_bl_ops = {
    .get_brightness = s6e8fa0_get_brightness,
    .update_status = s6e8fa0_set_brightness,
};
```

DTS Tree

Once the driver is finished, the Device Tree needs to be configured. You can change/add a MIPI DSI node from the 'arch/arm64/boot/dts/nexell/s5p6818-artik710-raptor-common.dtsi' file. Set the "panel@0" node according to your panel configurations:

```
&dp_drm_mipi {
    hs_bitrate = <960>;
    lp_bitrate = <100>;
    status = "ok";

    panel@0 {
        compatible = "samsung,s6e8fa0";
        reg = <0>;
        vdd3-supply = <&VCC_LDO0>;
        vci-supply = <&VCC_LDO1>;
        reset-gpio = <&gpio_e 30 0>;
        power-on-delay= <50>;
        reset-delay = <100>;
        init-delay = <100>;
        display-timings {
            timing-0 {
                clock-frequency = <130361520>;
                hactive = <1080>;
                vactive = <1920>;
                hfront-porch = <32>;
                hback-porch = <16>;
                hsync-len = <12>;
                vfront-porch = <12>;
                vback-porch = <3>;
                vsync-len = <1>;
            };
        };
    };

    dp_control {
        clk_src_lv0 = <3>;
```

```
        clk_div_lv0 = <6>;
        clk_src_lv1 = <7>;
        clk_div_lv1 = <1>;
        out_format = <3>;
        vs_start_offset = <480>;
        ev_start_offset = <480>;
        vs_end_offset = <0>;
        ev_end_offset = <0>;
    };
};
```

More information on this process can be found under: 'Documentation/devicetree/bindings/drm/nexell/mipi_panel.txt'.

Testing the Video Driver

The drm driver can be tested using the 'modetest' utility contained in the 'libdrm' package. It provides functionality to provides draw patterns onto the screen and run page flip events.

The LVDS/MIPI-DSI/HDMI interfaces are exposed to the "connector" device as part of the DRM framework. The DRM framework has four display layers:

- CRTC Display Layer
- Plane Display Layer
- Encoder Display Layer
- Connector Display Layer

The CRTC will read pixel data from the frame buffer and generate the desired video timing. The plane is a representation of the display buffer for CRTC. The Encoder encodes the video signals from the Plane layer to accommodate for the dedicated formats present.

CRTC and Plane

You can check what CRTC devices and Planes using the 'modetest -M nexell –p' command:

```
$ modetest -M nexell -p
CRTCs:
id    fb      pos     size
22      40      (0,0)   (1024x600)
  1024x600 60 1024 1024 1344 1344 600 600 635 635 flags: ; type:
  props:
31      40      (0,0)   (720x576)
  720x576 50 720 732 796 864 576 581 586 625 flags: nhsync, nvsync; type: driver
  props:

Planes:
id    crtc    fb      CRTC x,y        x,y     gamma size      possible crtcs
17      0       0       0,0             0,0     0               0x00000001
  formats: YUYV YVYU UYVY VYUY NV12 NV21 NV16 NV61 YUV9 YVU9 YU11 YV11 YU12 YV12 YU16 YV16 YU24 YV24
  props:
        5 type:
                flags: immutable enum
                enums: Overlay=0 Primary=1 Cursor=2
                value: 0
        18 colorkey:
                flags: range
                values: 0 4294967295
                value: 0
19      22      40      0,0             0,0     0               0x00000001
  formats: RG16 BG16 RG24 BG24 XR24 XB24 AR24 AB24
  props:
        5 type:
```

```
                flags: immutable enum
                enums: Overlay=0 Primary=1 Cursor=2
                value: 1
        20 transcolor:
                flags: range
                values: 0 4294967295
                value: 0
        21 alphablend:
                flags: range
                values: 0 4294967295
                value: 0
23      0       0       0,0             0,0     0               0x00000001
  formats: RG16 BG16 RG24 BG24 XR24 XB24 AR24 AB24
  props:
        5 type:
                flags: immutable enum
                enums: Overlay=0 Primary=1 Cursor=2
                value: 0
        24 transcolor:
                flags: range
                values: 0 4294967295
                value: 0
        25 alphablend:
                flags: range
                values: 0 4294967295
                value: 0
26      0       0       0,0             0,0     0               0x00000002
  formats: YUYV YVYU UYVY VYUY NV12 NV21 NV16 NV61 YUV9 YVU9 YU11 YV11 YU12 YV12 YU16 YV16 YU24 YV24
  props:
        5 type:
                flags: immutable enum
                enums: Overlay=0 Primary=1 Cursor=2
                value: 0
        27 colorkey:
                flags: range
                values: 0 4294967295
                value: 0
28      31      40      0,0             0,0     0               0x00000002
  formats: RG16 BG16 RG24 BG24 XR24 XB24 AR24 AB24
  props:
        5 type:
                flags: immutable enum
                enums: Overlay=0 Primary=1 Cursor=2
                value: 1
        29 transcolor:
                flags: range
                values: 0 4294967295
                value: 0
        30 alphablend:
                flags: range
                values: 0 4294967295
                value: 0
```

ENCODER DEVICES

You can check what encoder devices are present using the 'modetest -M nexell –e' command:

```
$ modetest -M nexell -e
Encoders:
id      crtc    type    possible crtcs  possible clones
32      22      LVDS    0x00000003      0x00000000
```

```
34      0       DSI     0x00000003      0x00000000
36      31      none    0x00000003      0x00000000
```

CONNECTOR DEVICES

You can check which connector devices are present on your device using the 'modetest -M nexell –c' command:

```
$ modetest -M nexell -c
Connectors:
id      encoder status          name            size (mm)       modes   encoders
33      32      connected       LVDS-1          154x85          1       32
  modes:
        name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
  1024x600 60 1024 1024 1344 1344 600 600 635 635 flags: ; type:
  props:
        1 EDID:
                flags: immutable blob
                blobs:

                value:
        2 DPMS:
                flags: enum
                enums: On=0 Standby=1 Suspend=2 Off=3
                value: 0
35      34      disconnected    DSI-1           0x0             0       34
  props:
        1 EDID:
                flags: immutable blob
                blobs:

                value:
        2 DPMS:
                flags: enum
                enums: On=0 Standby=1 Suspend=2 Off=3
                value: 0
37      36      disconnected    HDMI-A-1        0x0             0       36
  props:
        1 EDID:
                flags: immutable blob
                blobs:

                value:
        2 DPMS:
                flags: enum
                enums: On=0 Standby=1 Suspend=2 Off=3
                value: 0
```

As can be seen in above printout each connector has its own id (3 connector devices). In this example LVDS id-number=33, DSI id-number=35 and HDMI id-number=37.

SETTING DISPLAY PROPERTIES

To set the resolution of the LVDS display (with id:33) use:

```
$ modetest -M nexell -s 33@22:1024x600
setting mode 1024x600-60Hz@XR24 on connectors 33, crtc 22
```

To set page flipping use:

```
$ modetest -M nexell -v -s
33@22:1024x600
setting mode 1024x600-60Hz@XR24 on connectors 33, crtc 22
```

```
freq: 58.75Hz
freq: 58.59Hz
freq: 58.59Hz
```

To read the current brightness level use:

```
$ cat /sys/class/backlight/pwm-backlight/brightness
60
```

To adjust the brightness level to maximum use:

```
$ cat /sys/class/backlight/pwm-backlight/max_brightness
100
$ echo 100 > /sys/class/backlight/pwm-backlight/brightness
```

EXPLORING THE CAMERA SENSOR DRIVER

INTRODUCTION

This section will describe how to create your own camera sensor driver and incorporate the driver into the ARTIK 710 Module software structure.

CAMERA SENSOR DRIVER

The first step into creating a dedicated camera sensor driver for your particular hardware is to write a camera sensor driver. It is assumed that the camera sensor is controlled using I^2C. For this reason, the camera sensor driver was implemented as an I^2C media driver. As with all other source files throughout this document, these driver files can also be found in GitHub. More specifically a copy of the reference driver, that we use here as an example, can be found under 'drivers/media/i2c/s5k4ecga.c'.

As stated before the camera sensor system is controlled using I^2C, for this we need to prepare the 'i2c_driver' structure as follows:

```
int sensor_4ec_probe(struct i2c_client *client,
        const struct i2c_device_id *id);
static int sensor_4ec_remove(struct i2c_client *client);
static const struct i2c_device_id sensor_4ec_idt[] = {
        { SENSOR_NAME, 0 },
};


static struct i2c_driver sensor_4ec_driver = {
        .driver = {
                .name   = SENSOR_NAME,
                .owner  = THIS_MODULE,
        },
        .probe  = sensor_4ec_probe,
        .remove = sensor_4ec_remove,
        .id_table = sensor_4ec_idt
};
```

A minimum set of functions (v4l2), that is required to enable a camera sensor into the existing software structure are:

- Core operations : init, s_ctrl, g_ctrl
- Video operations : s_stream, s_param, s_mbus_fmt
- Pad operations : set_fmt

```
static struct v4l2_subdev_pad_ops pad_ops = {
        .set_fmt                = sensor_4ec_s_fmt,
};

static const struct v4l2_subdev_core_ops core_ops = {
        .init           = sensor_4ec_init,
        .s_ctrl         = sensor_4ec_s_ctrl,
        .g_ctrl         = sensor_4ec_g_ctrl
};

static const struct v4l2_subdev_video_ops video_ops = {
        .s_stream = sensor_4ec_s_stream,
        .s_parm = sensor_4ec_s_param,
        .s_mbus_fmt = sensor_4ec_s_format
};

static const struct v4l2_subdev_ops subdev_ops = {
        .core = &core_ops,
        .video = &video_ops,
```

```
        .pad    = &pad_ops,
};
```

Note that in our case the power on the camera sensor will be enabled right before calling 's_stream on', and the power will be disabled right after calling 's_stream off'. Therefore, all I^2C communications has to be done after 's_stream on' and before 's_stream off' is called.

DTS TREE

Once the Camera Sensor Driver is complete a next step is to incorporate the new camera information into the existing DTS Tree. For this edit the 'arch/arm64/boot/dts/nexell/s5p6818-artik710-raptor-common.dtsi' file and incorporate the data below:

```
Clipper clip vip input data and give data to decimator and
save clipped data to memory.
The DT definitions can be found in include/dt-bindings/media/nexell-vip.h

Required properties:
- compatible    : must contain "nexell,nx-clipper"
- module        : vip module number
- interface_type        : NX_CAPTURE_INTERFACE_PARALLEL or NX_CAPTURE_INTERFACE_MIPI
                          parallel interface is 8bit(656/601)
- pinctrl-names : must contain "default"
- pinctrl-0     : the pinctrl reference to vid
- port          : vip input port, each vip has two input port and can use only one
- external_sync : if 1, this is 601, else 656
- data_order    : YUV 422 data order from camera sensor
- interlace     : if 1, interlace input data
- sensor: type  : must contain NX_CAPTURE_SENSOR_I2C
- sensor: i2c_name      : camera sensor i2c type(name)
- sensor: i2c_adapter   : camera sensor i2c adapter number
- sensor: addr          : camera sensor i2c address

Optional properties:
- power: enable_seq     : camera sensor power on sequence
                          [ACTION_START_TAG][ACTION_TYPE][SEQUENCE][ACTION_END_TAG]
                          ACTION_START_TAG : NX_ACTION_START
                          ACTION_END_TAG : NX_ACTION_END
                          ACTION_TYPE : NX_ACTION_TYPE_PMIC, NX_ACTION_TYPE_GPIO,
NX_ACTION_TYPE_CLOCK
                          SEQUENCE for ACTION_TYPE_GPIO: gpionum, value, delay_ms, value, delay_ms
...
                          SEQUENCE for ACTION_TYPE_PMIC: regulator_name, value, delay_ms
                          SEQUENCE for ACTION_TYPE_CLOCK: enable/disable, delay_ms
- power: disable_seq    : camera sensor power off sequence
                          Action defines same to above
- clock: pwms   : reference to pwm that gives master clock to camera sensor
```

A code snippet of a typical DTS Tree structure is as given below:

```
mipi_csi:mipi_csi@c00d0000 {
        data_lane = <2>;
        pllval = <750>;
        hssettle = <7>;
        status = "okay";
};

clipper_0:clipper0@c0063000 {
        interface_type = <NX_CAPTURE_INTERFACE_MIPI_CSI>;
        gpios = <&gpio_a 17 0 &gpio_a 3 0 &gpio_a 6 0>;
        data_order = <NX_VIN_Y0CBY1CR>;
        status = "okay";
```

```
        sensor {
                type = <NX_CAPTURE_SENSOR_I2C>;
                i2c_name = "S5K4ECGA";
                i2c_adapter = <0>;
                addr = <0x56>;
        };
        power {
                enable_seq = <
                        NX_ACTION_START NX_ACTION_TYPE_GPIO 2 1 2 NX_ACTION_END
                        NX_ACTION_START NX_ACTION_TYPE_GPIO 1 1 1 NX_ACTION_END
                        NX_ACTION_START NX_ACTION_TYPE_GPIO 0 1 1 NX_ACTION_END
                >;

                disable_seq = <
                        NX_ACTION_START NX_ACTION_TYPE_GPIO 0 0 1 NX_ACTION_END
                        NX_ACTION_START NX_ACTION_TYPE_GPIO 1 0 1 NX_ACTION_END
                        NX_ACTION_START NX_ACTION_TYPE_GPIO 2 0 2 NX_ACTION_END
                >;
        };
};
```

KERNEL CONFIGURATION

Once the DTS Tree has been filled out with proper driver definitions of the camera sensor, the following camera interface configurations have to be enabled:

```
Device Drivers --->
  <*> Multimedia support --->
    [*] V4L platform devices --->
        --- V4L platform devices
        [*] Nexell S5Pxx18 SoC series V4l2 Subsystem driver
        [*]   Nexell S5Pxx18 SoC series camera interface driver
        [*]     Enable capture clipper video device
        [*]     Enable capture decimator video device
```

A typical example of a Kernel configuration file for a camera sensor looks like:

```
Device Drivers --->
<*> Multimedia support --->
    Encoders, decoders, sensors and other helper chips --->
     *** Camera sensor devices ***
       <*> Samsung S5K5K4ECGA sensor support
```

As a next step edit the 'drivers/media/i2c/Kconfig' file to create the right configuration for your new sensor driver. An example of a typical Kconfig file is as given below:

```
config VIDEO_S5K4ECGA
        tristate "Samsung S5K5K4ECGA sensor support"
        depends on I2C && VIDEO_V4L2 && VIDEO_V4L2_SUBDEV_API
        ---help---
          This is a V4L2 sensor-level driver for Samsung S5K4ECGA 5M
          camera sensor with an embedded SoC image signal processor.
          This sensor is revised version of S5K4ECGX.
```

In addition edit the I^2C 'drivers/media/i2c/Makefile' by adding your sensor driver like:

```
obj-$(CONFIG_VIDEO_AK881X)              += ak881x.o
obj-$(CONFIG_VIDEO_IR_I2C)  += ir-kbd-i2c.o
obj-$(CONFIG_VIDEO_ML86V7667)   += ml86v7667.o
obj-$(CONFIG_VIDEO_OV2659)      += ov2659.o
obj-$(CONFIG_VIDEO_SP2518)      += sp2518.o
obj-$(CONFIG_VIDEO_S5K4ECGA)    += s5k4ecga.o
```

Testing the Camera Sensor Driver

To test whether the camera sensor driver has been probed successfully, read the 'sysfs' entry of the camera sensor. If done correctly you should see something like:

```
$ cat /sys/devices/platform/camerasensor0/info
is_mipi:1,name:S5K4ECGA 0-0056
```

A simple preview function can be obtained using 'gstreamer' like:

```
gst-launch-1.0 -e v4l2src device=/dev/video6 ! video/x-raw,format=I420,framerate=30/1,width=1280,
height=720 ! nxvideosink
```

The various parameters used in 'gstreamer' have a meaning as explained in the table below:

Option	Description
v4l2src	v4l2 source gstreamer plugin : Reads frames from a Video4Linux2 device including your camera sensor
device=/dev/video6	Device node of MIPI camera 0
video/x-raw,format=I420	Format of video : I420
framerate=30/1	Frame rate : 30 fps
width=1280	Width of the image : 1280
height=720	Height of the image : 720
nxvideosink	nxvideosink gstreamer plugin : H/W Video Renderer for LCD or HDMI

EXPLORING THE AUDIO DRIVER

INTRODUCTION

This section will describe the steps required to create a dedicated audio driver, the example used is based on the Realtek ALC5658 SoC using an I^2C control mechanism.

AUDIO CODEC FEATURES

The Audio codec (ALC5658) has the following features:

- 2x 24bit/8kHz~192kHz I^2S/PCM/TDM Interfaces
- Support for S/PDIF-Out with a 24-bit Format up to 192kHz sampling rates
- Configurable Multi-band Digital Equalizer
- DAC with 7+7 bands for L/R path (1x Low Pass Filter + 2x High Pass Filters + 3x Band Pass Filters + 1x Biquad Filter)
- ADC with 6+6 bands for L/R path (1x Low Pass Filter + 1x High Pass Filter + 4x Band Pass Filters)
- 2x Digital Microphone Interfaces for 4-channel recording
- I^2C Control Interface up to 400kbps with a 16-bit address control interface

AUDIO KERNEL DRIVER IN MENUCONFIG

The 'menuconfig' file in your Linux kernel environment shows your particular hardware configuration. Below you see an example on what is supported on the ARTIK 710 Module, next to support for an ALC5658 Audio Codec, there is additional audio support for 3x channel I^2S provided by the Nexell S5P6818 SoC.

```
Device Drivers --->
    <*> Sound card support --->
        <*> Advanced Linux Sound Architecture --->
            <*> ALSA for SoC audio support --->
                <*> Nexell SoC Audio
                    [*] I2S 0
                    [*] I2S 1
                    [ ] I2S 2
                <*> SPDIF transceiver (PCM)
                <*> ALC5658 I2S audio codec.
```

USER CODEC FOR AUDIO DRIVER

The Realtek codec driver code for the ALC5658 can be found under 'rt5658.[c|h]' in GitHub. The 'rt5659.[c|h]' driver code that is released as part of the stable-kernel is not suitable for the ARTIK 710 Module codec driver. You have to use the 'rt5659.[c|h]' files that are contained as part of the ARTIK 710 Module kernel source code. These files can be found in GitHub under:

Realtek audio Driver can be found under:

- 'sound/soc/codec/rt5659.c'
- 'sound/soc/codec/rt5659.h'
- 'include/sound/rt5659.h'

The various Nexell audio drivers can be found under:

- 'sound/soc/nexell'

More specifically the following driver files need to be modified to gain full control of the audio hardware in your Linux environment:

```
nexell-alc5658.c              --> alc5658 sound card driver
nexell-i2s.c                  --> AP I2S interface driver
nexell-pcm.c                  --> PCM Data transaction driver
nexell-spdif-transceiver.c    --> Sound SPDIF transceiver driver
nexell-spdiftx.c              --> Sound SPDIF tx driver
```

As an example the sound card driver that is available under GitHub is shown below:

```
# sound/soc/nexell-alc5658.c

static struct platform_driver alc5658_driver = {
    .driver = {
        .name = "alc5658-audio",
        .owner = THIS_MODULE,
        .pm = &snd_soc_pm_ops, /* for suspend */
        .of_match_table = of_match_ptr(nx_alc5658_match),
    },
    .probe = alc5658_probe,
    .remove = alc5658_remove,
};

module_platform_driver(alc5658_driver);

# sound/soc/codec/rt5659.c

struct i2c_driver rt5659_i2c_driver = {
    .driver = {
        .name = "rt5659",
        .owner = THIS_MODULE,
        .of_match_table = rt5659_of_match,
        .acpi_match_table = ACPI_PTR(rt5659_acpi_match),
    },
    .probe = rt5659_i2c_probe,
    .remove = rt5659_i2c_remove,
    .shutdown = rt5659_i2c_shutdown,
    .id_table = rt5659_i2c_id,
};

module_i2c_driver(rt5659_i2c_driver);
```

DTS TREE

Below is the DTS tree:

```
dts file : arch/arm64/boot/dts/nexell/s5p6818-artik710-raptor-common.dtsi (For ARTIK710)



        spdiftx@c0059000 {           : For ALC5658 HDMI audio output setting.
            pcm-bit = <16>;
            sample_rate = <48000>;
            status = "okay";
        };


        i2s_0:i2s@c0055000 {
            master-mode = <1>;         : Select master/slave mode (0:Slave, 1:master)
            mclk-in = <0>;             : Select mclk ext in / mclk out (0:mclk out, 1:ext in)
            trans-mode = <0>;          : Select trans mode (0:i2s, 1:right-justified, 2:left
justified)
            frame-bit = <32>;          : Select frame bit (16, 32, 48)
            sample-rate = <48000>;     : Select default sample-rate
            pre-supply-mclk = <1>;
            status = "okay";
        };

        i2c_gpio6:i2c@6 {
            compatible = "i2c-gpio";
            gpios = < &gpio_a 5 0  /*sda*/
```

```
                    &gpio_a 4 0  /*scl*/
                  >;
            i2c-gpio,delay-us = <10>;
            #address-cells = <1>;
            #size-cells = <0>;

            alc5658: alc5658@1a {
                compatible = "realtek,rt5658";      : Define alc5658 compatible arguments.
                reg = <0x1a>;                       : Define gpio register information.
            };
        };

    sound {
        compatible = "nexell,nexell-alc5658";
        ch = <0>;
        sample-rate = <48000>;                  : Select default sample-rate.
        format = "S16";                         : Select default audio format.
        hpin-support = <0>;
        nexell,i2s-controller = <&i2s_0>;       : Select i2s channel for sound card
        audio-codec = <&alc5658>;               : Select audio codec for sound card
    };
```

ADDING AUDIO FEATURES TO 'AMIXER'

The 'amixer' is a command-line mixer for the ALSA soundcard driver that is available on the ARTIK 710 Module. The 'amixer' can support various codec features such as turn-on/off audio, adjust volume, select audio device, etc. To see the various features, that are currently supported use:

```
$ amixer
Simple mixer control 'Headphone',0
  Capabilities: pvolume
  Playback channels: Front Left - Front Right
  Limits: Playback 0 - 31
  Mono:
  Front Left: Playback 0 [0%] [-23.25dB]
  Front Right: Playback 0 [0%] [-23.25dB]
Simple mixer control 'Speaker',0
  Capabilities: pvolume
  Playback channels: Front Left - Front Right
  Limits: Playback 0 - 39
  Mono:
  Front Left: Playback 17 [44%] [-21.00dB]
  Front Right: Playback 17 [44%] [-21.00dB]
Simple mixer control 'Mono',0
…….
```

To allow your driver to tap into these features you have to create a 'snd_kcontrol_new' structure, in this case using 'sound/soc/rt5659.c', with the various features you want to add:

```
# sound/soc/rt5659.c
static const struct snd_kcontrol_new rt5659_snd_controls[] = {
static const struct snd_kcontrol_new rt5659_sto1_adc_l_mix[] = {
static const struct snd_kcontrol_new rt5659_sto1_adc_r_mix[] = {
static const struct snd_kcontrol_new rt5659_mono_adc_l_mix[] = {
static const struct snd_kcontrol_new rt5659_mono_adc_r_mix[] = {
static const struct snd_kcontrol_new rt5659_dac_l_mix[] = {
static const struct snd_kcontrol_new rt5659_dac_r_mix[] = {
static const struct snd_kcontrol_new rt5659_sto_dac_l_mix[] = {
static const struct snd_kcontrol_new rt5659_sto_dac_r_mix[] = {
static const struct snd_kcontrol_new rt5659_mono_dac_l_mix[] = {
static const struct snd_kcontrol_new rt5659_mono_dac_r_mix[] = {
static const struct snd_kcontrol_new rt5659_rec1_l_mix[] = {
```

```
static const struct snd_kcontrol_new rt5659_rec1_r_mix[] = {
static const struct snd_kcontrol_new rt5659_rec2_l_mix[] = {
static const struct snd_kcontrol_new rt5659_rec2_r_mix[] = {
static const struct snd_kcontrol_new rt5659_spk_l_mix[] = {
static const struct snd_kcontrol_new rt5659_spk_r_mix[] = {
static const struct snd_kcontrol_new rt5659_monovol_mix[] = {
static const struct snd_kcontrol_new rt5659_out_l_mix[] = {
static const struct snd_kcontrol_new rt5659_out_r_mix[] = {
static const struct snd_kcontrol_new rt5659_spo_l_mix[] = {
static const struct snd_kcontrol_new rt5659_spo_r_mix[] = {
static const struct snd_kcontrol_new rt5659_mono_mix[] = {
static const struct snd_kcontrol_new rt5659_lout_l_mix[] = {
static const struct snd_kcontrol_new rt5659_lout_r_mix[] = {
static const struct snd_kcontrol_new rt5659_dac_l2_mux =
static const struct snd_kcontrol_new rt5659_dac_r2_mux =
static const struct snd_kcontrol_new rt5659_sto1_adc1_mux =
static const struct snd_kcontrol_new rt5659_sto1_adc_mux =
static const struct snd_kcontrol_new rt5659_sto1_adc2_mux =
static const struct snd_kcontrol_new rt5659_sto1_dmic_mux =
static const struct snd_kcontrol_new rt5659_mono_adc_l2_mux =
static const struct snd_kcontrol_new rt5659_mono_adc_l1_mux =
static const struct snd_kcontrol_new rt5659_mono_adc_l_mux =
static const struct snd_kcontrol_new rt5659_mono_adc_r_mux =
static const struct snd_kcontrol_new rt5659_mono_dmic_l_mux =
static const struct snd_kcontrol_new rt5659_mono_adc_r2_mux =
static const struct snd_kcontrol_new rt5659_mono_adc_r1_mux =
static const struct snd_kcontrol_new rt5659_mono_dmic_r_mux =
static const struct snd_kcontrol_new rt5659_dac_r1_mux =
static const struct snd_kcontrol_new rt5659_dac_l1_mux =
static const struct snd_kcontrol_new rt5659_dig_dac_mixl_mux =
static const struct snd_kcontrol_new rt5659_dig_dac_mixr_mux =
static const struct snd_kcontrol_new rt5659_alg_dac_l1_mux =
static const struct snd_kcontrol_new rt5659_alg_dac_r1_mux =
static const struct snd_kcontrol_new rt5659_alg_dac_l2_mux =
static const struct snd_kcontrol_new rt5659_alg_dac_r2_mux =
static const struct snd_kcontrol_new rt5659_if2_adc_in_mux =
static const struct snd_kcontrol_new rt5659_if3_adc_in_mux =
static const struct snd_kcontrol_new rt5659_pdm_l_mux =
static const struct snd_kcontrol_new rt5659_pdm_r_mux =
static const struct snd_kcontrol_new rt5659_spdif_mux =
static const struct snd_kcontrol_new rt5659_rx_adc_dac_mux =
static const struct snd_kcontrol_new spkvol_l_switch =
static const struct snd_kcontrol_new spkvol_r_switch =
static const struct snd_kcontrol_new monovol_switch =
static const struct snd_kcontrol_new outvol_l_switch =
static const struct snd_kcontrol_new outvol_r_switch =
static const struct snd_kcontrol_new spo_switch =
static const struct snd_kcontrol_new mono_switch =
static const struct snd_kcontrol_new hpo_l_switch =
static const struct snd_kcontrol_new hpo_r_switch =
static const struct snd_kcontrol_new lout_l_switch =
static const struct snd_kcontrol_new lout_r_switch =
static const struct snd_kcontrol_new pdm_l_switch =
static const struct snd_kcontrol_new pdm_r_switch =
```

This approach will allow you to use the 'amixer' with your dedicated implementation.

Testing the Audio Driver

To verify if the Audio Driver has indeed loaded in kernel mode check the log during boot. You should see something like:

```
Kernel booting log
[    4.110000] ALSA device list:
```

```
[    4.110000]   #0: I2S-ALC5658
[    4.110000]   #1: SPDIF-Transceiver
```

You can see in above log file that both the Audio Codec (ALC5658) and the SPDIF Transceiver driver are successfully loaded. This means audio can be played back using either the HDMI output or the audio headphone jack.

To see if your new driver is supported after booting use:

```
$ cat /proc/asound/cards
 0 [I2SALC5658     ]: I2S-ALC5658 - I2S-ALC5658
                      I2S-ALC5658
 1 [SPDIFTransceive]: SPDIF-Transceiv - SPDIF-Transceiver
                      SPDIF-Transceiver
```

By default the 'alsa.conf' library configuration file defines device id numbers. These can be changed using:

```
$ vi /usr/share/alsa/alsa.conf

#
defaults.ctl.card 0 -> 1
defaults.pcm.card 0 -> 1
defaults.pcm.device 0
defaults.pcm.subdevice -1
```

To test if your driver is properly implemented use:

```
$ aplay <wav file>
```

LEGAL INFORMATION

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH THE SAMSUNG ARTIK™ DEVELOPMENT KIT AND ALL RELATED PRODUCTS, UPDATES, AND DOCUMENTATION (HEREINAFTER "SAMSUNG PRODUCTS"). NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. THE LICENSE AND OTHER TERMS AND CONDITIONS RELATED TO YOUR USE OF THE SAMSUNG PRODUCTS ARE GOVERNED EXCLUSIVELY BY THE SAMSUNG ARTIK™ DEVELOPER LICENSE AGREEMENT THAT YOU AGREED TO WHEN YOU REGISTERED AS A DEVELOPER TO RECEIVE THE SAMSUNG PRODUCTS. EXCEPT AS PROVIDED IN THE SAMSUNG ARTIK™ DEVELOPER LICENSE AGREEMENT, SAMSUNG ELECTRONICS CO., LTD. AND ITS AFFILIATES (COLLECTIVELY, "SAMSUNG") ASSUMES NO LIABILITY WHATSOEVER, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES, AND SAMSUNG DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, ARISING OUT OF OR RELATED TO YOUR SALE, APPLICATION AND/OR USE OF SAMSUNG PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATED TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

SAMSUNG RESERVES THE RIGHT TO CHANGE PRODUCTS, INFORMATION, DOCUMENTATION AND SPECIFICATIONS WITHOUT NOTICE. THIS INCLUDES MAKING CHANGES TO THIS DOCUMENTATION AT ANY TIME WITHOUT PRIOR NOTICE. THIS DOCUMENTATION IS PROVIDED FOR REFERENCE PURPOSES ONLY, AND ALL INFORMATION DISCUSSED HEREIN IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OF ANY KIND. SAMSUNG ASSUMES NO RESPONSIBILITY FOR POSSIBLE ERRORS OR OMISSIONS, OR FOR ANY CONSEQUENCES FROM THE USE OF THE DOCUMENTATION CONTAINED HEREIN.

Samsung Products are not intended for use in medical, life support, critical care, safety equipment, or similar applications where product failure could result in loss of life or personal or physical harm, or any military or defense application, or any governmental procurement to which special terms or provisions may apply.

This document and all information discussed herein remain the sole and exclusive property of Samsung. All brand names, trademarks and registered trademarks belong to their respective owners. For updates or additional information about Samsung ARTIK™, contact the Samsung ARTIK™ team via the Samsung ARTIK™ website at www.artik.io.