Application Report

SPRACD8–June 2019

Integrating New Cameras With Video Input Port on DRA7xx SoCs

Nikhil Devshatwar

ABSTRACT

This application report describes how to integrate a new camera with the DRA7xx software ecosystem. It documents all the software changes needed to bring up a new camera on a custom board using DRA7xx running Processor SDK Linux automotive software.

Contents

1 Introduction 2
2 Video Input Port and Possible Video Sources 2
3 Kernel Changes to Integrate Camera Devices 2

List of Figures

1 Possible Video Sources to be Interfaced With VIP 2
2 Block Diagram of Multichannel Video Source Integration 5
3 Block Diagram of LVDS Camera Integration 6

List of Tables

1 Port Names 3

Trademarks

All trademarks are the property of their respective owners.

Copyright © 2019, Texas Instruments Incorporated

1 Introduction

The DRA7xxx family of System-on-Chips (SoCs) have Video Input Port (VIP) as one of the essential module used for video capture in automotive use cases. The processor SDK supports a few cameras out of the box on the DRA7xx EVMs, but depending on the customer need, different cameras need to be integrated with the DRA7xx SoCs. Sometimes, the video that needs to be captured may not be directly from a camera; it may be the output of an analog video decoder or an HDMI receiver as well.

The VIP driver supports different types of cameras:

- Parallel port cameras with discrete sync signals
- BT656 video with embedded sync signals
- 8-bit/16-bit YUV video
- 24-bit RGB video
- Multiple video sources time multiplexed embedded sync video

2 Video Input Port and Possible Video Sources

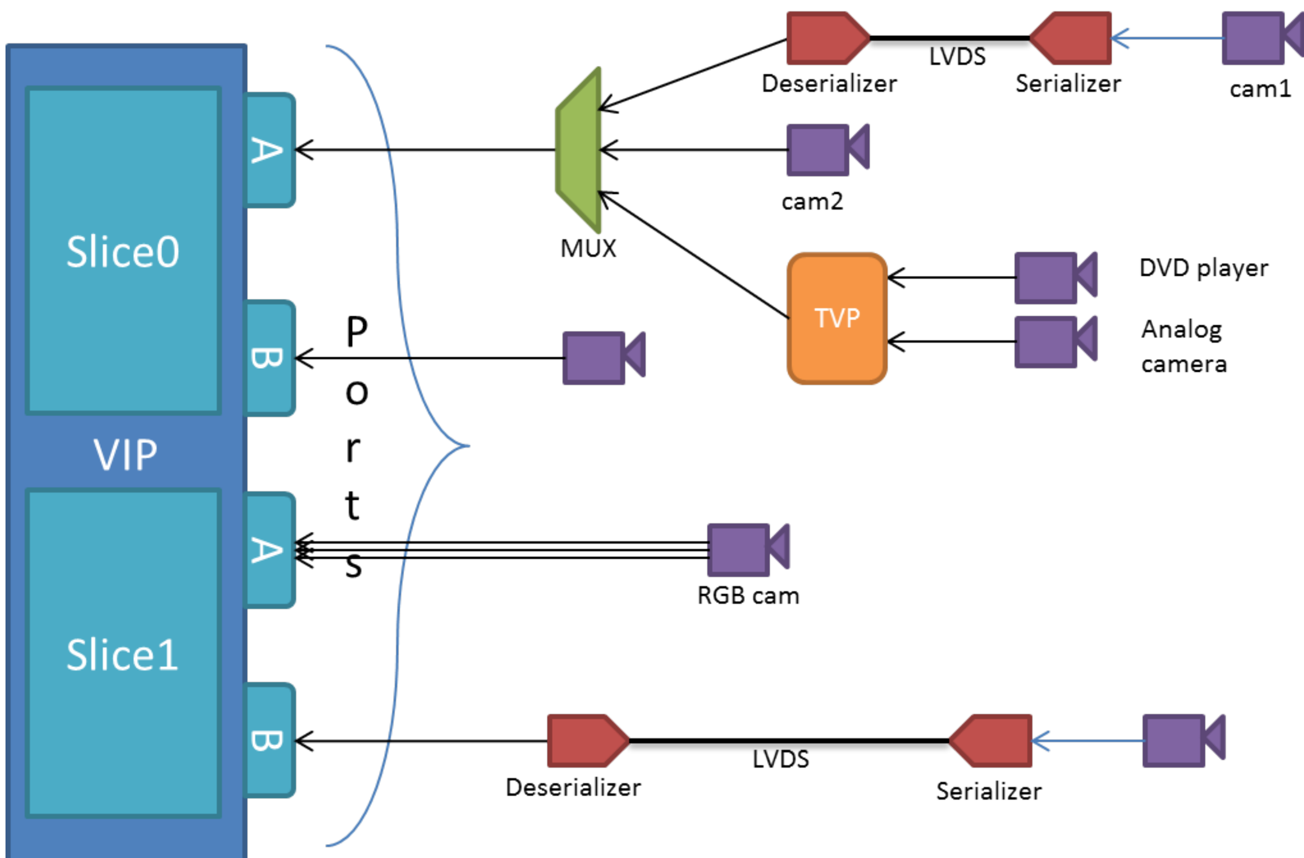


Figure 1. Possible Video Sources to be Interfaced With VIP

3 Kernel Changes to Integrate Camera Devices

The VIP driver is a V4L2 capture driver that registers /dev/videoX devices in userspace, which are used for performing video capture via standard V4L2 API. The V4L2 specification allows for SoC driver to be written independent of the camera/video source. There is one V4L2 driver for the VIP instance and each camera device may have a separate V4L2 subdevice driver. The camera driver is responsible for configuring the camera as described in the device node and implement some of the media bus operations. Theses operations are often used by the VIP driver whenever the camera needs to perform specific tasks (for example, start/stop camera, get/set fmt, and so forth).

Copyright © 2019, Texas Instruments Incorporated

3.1 V4L2 Endpoint Framework

Different camera/video sources have different configuration parameters when interfacing with the VIP video ports. Common interfacing properties like Horizontal Sync (Hsync), Vertical Sync (Vsync), Pclk polarities can be different across different devices. V4L2 endpoint, also known as v4l2_fwnode_endpoint, describes these as part of the device tree definition. This makes the VIP driver generic enough to have no dependency on the camera device. It also provides the flexibility to work with new cameras by doing simple device tree modifications.

The following example showcases the DT entries of VIP device node and its usage when interfacing different video sources.

3.1.1 VIP Device Definition

```
vip1 {
    #address-cells = <1>;
    #size-cells = <0>;
    ports {
        vin1a: port@1A {
            reg = <0>;
            #address-cells = <1>;
                endpoint@0 {
                    remote-endpoint = <&cam1>;
                };
         };
        vin2a: port@1A {
            ...
        };
        ...
   };
};
```

The above snippet describes the SoC VIP1 instance. It creates endpoint nodes for each of the video ports available for that instance. By default, all the port nodes are empty, so it is not associated with any cameras. Certain port nodes need to be populated when a new camera gets interfaced to that port. Both endpoint nodes are cross referenced in the device tree.

Table 1 describes exact names of the ports (this also matches the Technical Reference Manual (TRM) terminology) for each instance.

Table 1. Port Names

Endpoint Name	VIP Instance	Slice	Port Name
vin1a	VIP1	slice0	port A
vin1b	VIP1	slice0	port B
vin2a	VIP1	slice1	port A
vin2b	VIP1	slice1	port B
vin3a	VIP1	slice0	port A
vin3b	VIP1	slice0	port B
vin4a	VIP1	slice1	port A
vin4b	VIP1	slice1	port B
vin5a	VIP1	slice0	port A
vin6a	VIP1	slice1	port A

Copyright © 2019, Texas Instruments Incorporated

3.1.2 Camera Device Definition

```
ov10635@35{
    compatible = "ovti,ov10633";
    reg = <0x35>;
    mux-gpios = <&pcf_hdmi 3 GPIO_ACTIVE_LOW>;
    ...
    port {
        cam1: endpoint {
            remote-endpoint = <&vin1a>;
            hsync-active = <1>;
            vsync-active = <1>;
            pclk-sample = <0>;
            bus-width = <8>;
        };
    };
};
```

The above snippet describes the camera device. Standard properties about video interface can be described with the endpoint device nodes itself. This minimizes the communication between the SoC driver and the camera driver. This example describes:

- An 8-bit camera device connected to the vin1a port of the VIP device
- Hsync/Vsync signals are active HIGH
- Data needs to be sampled at falling edge of pixel clock.

3.2 Interfacing a Multichannel Video Source (TVP5158)

Most automotive use cases need to interface multiple cameras to the SoC. This can be achieved by connecting multiple cameras to the different ports. The problem with this approach is that, for every camera connected, at least 10 pads are used (assuming 8-bit interface). Most of the pads are muxed between video ports and other interfaces like ethernet, audio, gpmc, and so forth. When connecting multiple cameras this way, some of the functionality needs to be compromised. This can be avoided when using multichannel capture; In that, multiple video sources are time multiplexed and sent via only one video port. For example, TVP5158 is a TI analog video decoder chip that supports multiplexing. The VIP parser is capable of de-multiplexing up to 16 different channels from one video port. This is done using embedded sync BT656 protocol.

V4L2 framework does not support multichannel capture. There is no option to select a specific channel via a video device. Also, the application might become complex when it has to handle all the channels through the same video device. Current implementation of multichannel capture support for VIP driver registers one video device per channel. For four channel capture, the driver registers /dev/video1, /dev/video2, /dev/video3, /dev/video4. In this case, driver maps all the video devices to the same port and each of them can be used independently. Another advantage with this approach is that this allows multiple applications to handle each channel separately.

Copyright © 2019, Texas Instruments Incorporated

Figure 2 shows how the 4 channel capture is realized for the VIP and TVP5158 on DRA7xx-EVM.

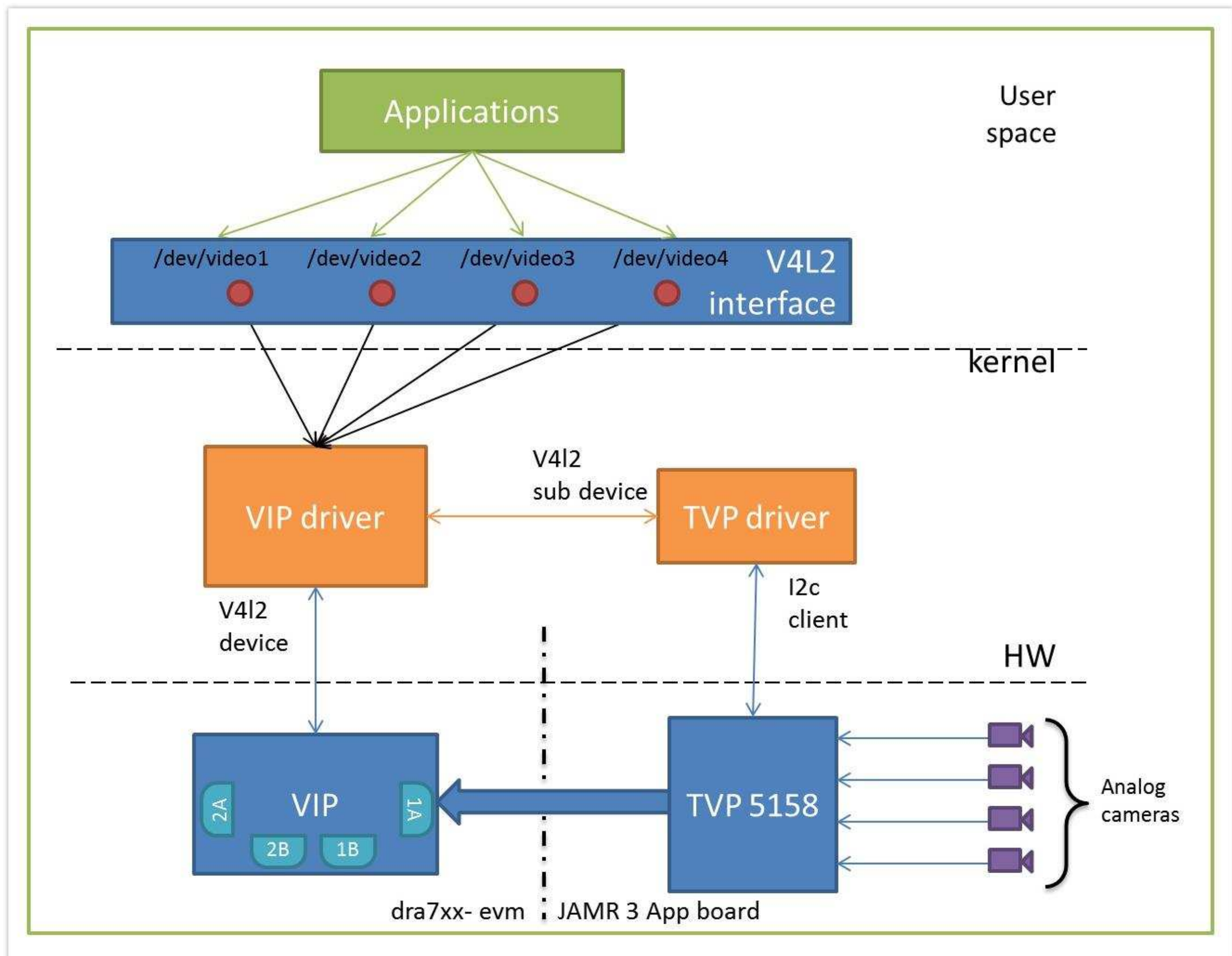


Figure 2. Block Diagram of Multichannel Video Source Integration

3.3 *Interfacing a Camera Over LVDS Serializer Deserializer*

When the camera is physically located far from the SoC, the parallel port video cannot be sent longer distances. For lossless reception, video data can be sent over linearly variable differential signaling (LVDS) cable. This means the LVDS interface has to be much faster and the video data at camera and SoC needs some way to serialized and deserialized, respectively.

Copyright © 2019, Texas Instruments Incorporated

3.3.1 I2C Address Remapping

When connecting a LVDS camera, all remote Inter-Integrated Circuit (I2C) devices are not directly connected. Serdes together acts as switch to forward the I2C messages back and forth. For this, each remote device has an alias address that is used by the local I2C master to communicate with the remote slave.

Figure 3 shows an example of interfacing LVDS camera and address translation.

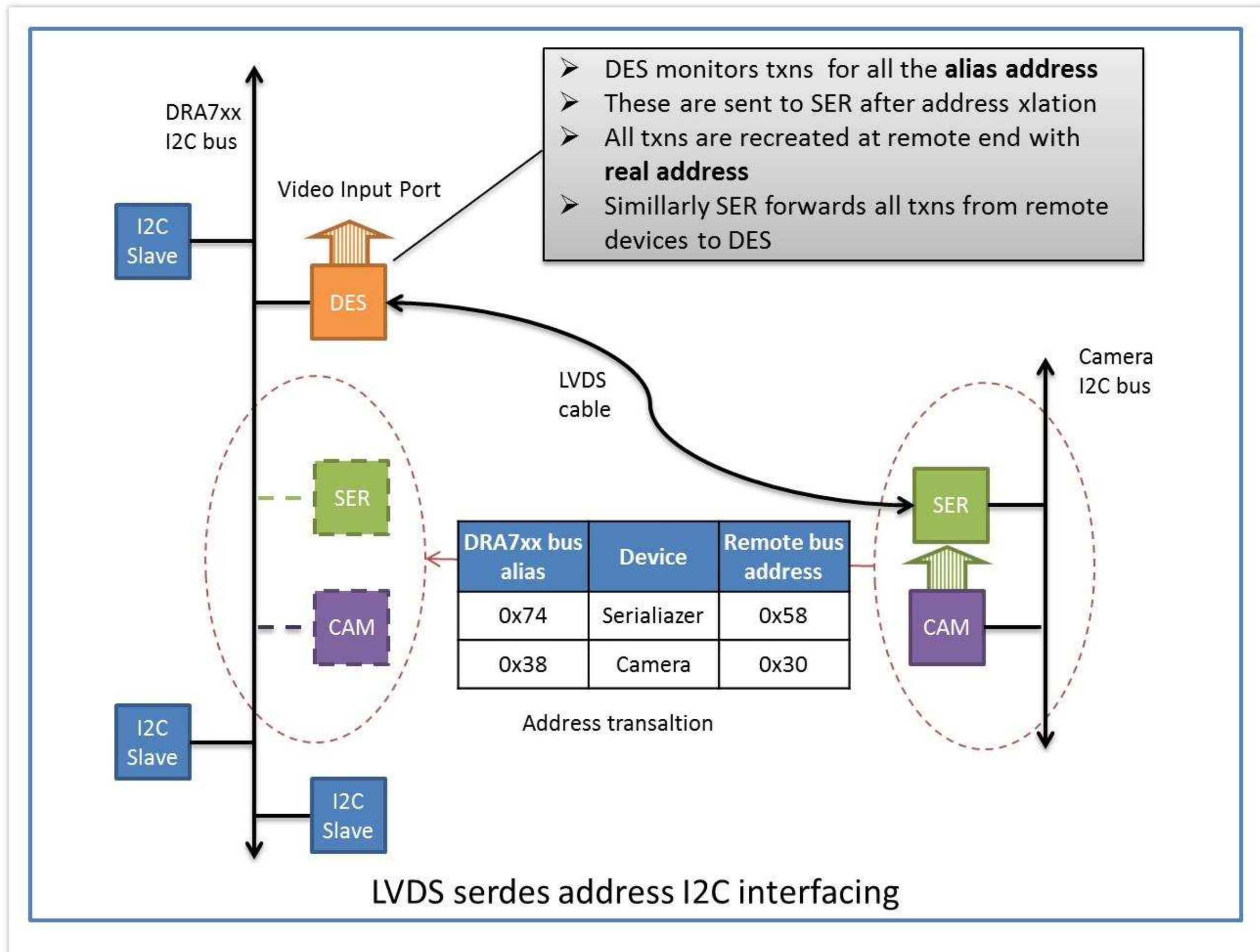


Figure 3. Block Diagram of LVDS Camera Integration

Copyright © 2019, Texas Instruments Incorporated

3.3.2 Serializer/Deserializer Configuration

Each serializer/deserializer device is an I2C slave from the Linux perspective. For cameras, typically the serializers will be on a remote I2C bus and the de-serializers will be on local I2C bus.

Note that along with the job of serialzing/de-serializing video data, most TI FPDLink3 devices have General Purpose Input/Output (GPIO) capabilities to drive few GPIO lines. This is useful when a remotely connected camera needs to toggle few power or Vsync lines through kernel driver

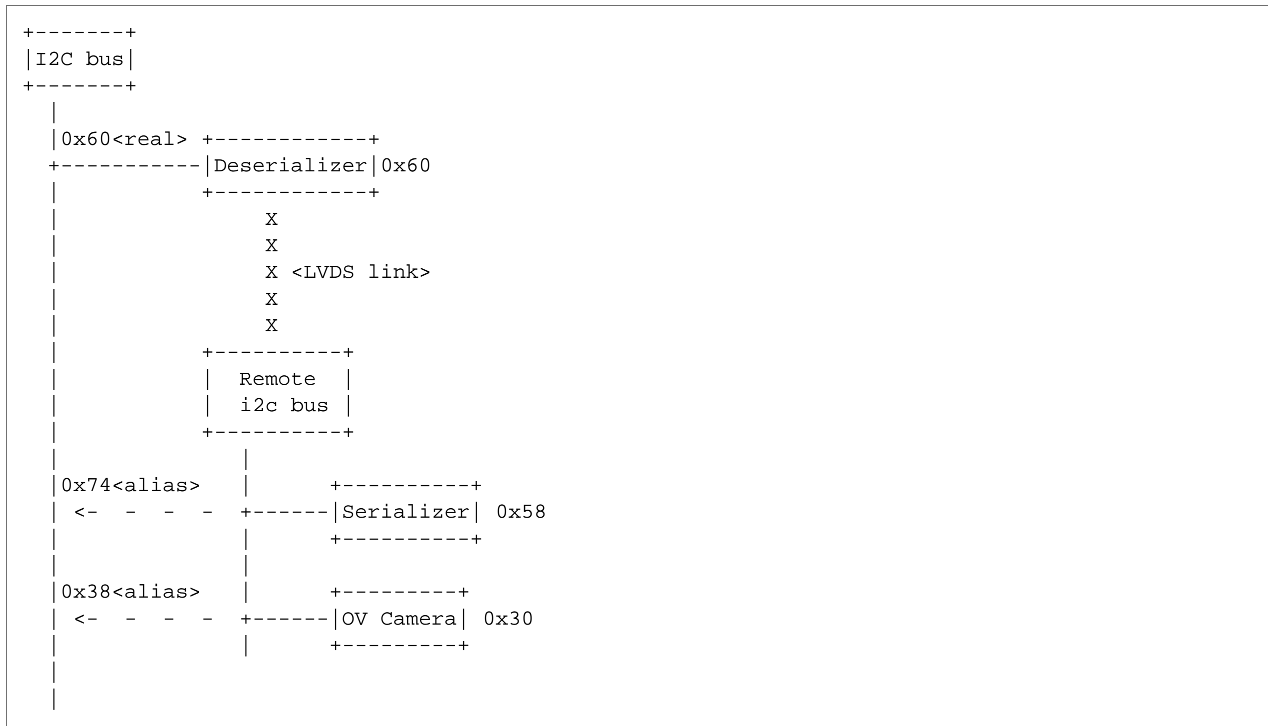
The following explains the device-tree bindings for describing the Serdes devices:

```
Texas instruments video serializer/de serializer
================================================
Required Properties:
  - compatible: should be one of the following.
    - "ti,ds90ub913aq": For TI FPDlink3 12bit video serializer
    - "ti,ds90ub914aq": For TI FPDlink3 12bit video de serializer
    - "ti,ds90uh925q": For TI FPDlink3 24bit video serializer
    - "ti,ds90uh928q": For TI FPDlink3 24bit video de serializer
  - reg: I2C slave address
    This would be the alias adress for remote device. The CPU side
    ser/des would address the remote device using this address.
Optional Properties:
  - gpio-controller: Marks the device node as a gpio controller.
  - #gpio-cells: Should be 1. The first cell is the GPIO number.
  - ranges: This is the address translation table for mapping i2c devices
    on the remote bus to the i2c address(alias) on parent bus.
    The first entry in the ranges property has to of the corresponding
    remote ser/des device.
    For dynamic address mapping, keep the remote slave address as 0x0
    The corresponding alias would be used if the remote slave doesn't
    have an address already mapped.
  - slave-mode: This property marks the ser/des as remote device.
    For a device where 'slave-mode' property is absent, it is considered
    as master device and 'ranges' and 'i2c-bus-num' properties are
    compulsory.
```


Copyright © 2019, Texas Instruments Incorporated

3.3.3 Serdes Device Definition

The following example demonstrates device node structure for a camera connected using FPDlink3 ser/des. Here, the deserializer and serializer are connected only via the LVDS link. The camera is connected on the serializer I2C bus. The serializer and camera are not connected to the system I2C bus, but it can be accessed from the system I2C bus. The deserializer maps each of the remote slave onto the system I2C bus and acts as a bridge to transfer any messages addressed to the remote devices.



Copyright © 2019, Texas Instruments Incorporated

The topology in the example above can be described in the device tree as shown below. Note that the **i2cbus** here is a local I2C bus connected to the SoC. The **lvds_des** is just a virtual I2C bus representing the I2C bus at the remote end. That is the reason why the camera is described under the virtual bus.

```
i2cbus {
     lvds_des: deserializer@60 {
         compatible = "ti,ds90ub914aq";
         reg = <0x60>;
         gpio-controller;
         #gpio-cells=1;
         i2c-bus-num = <5>;
     };
 };
 &lvds_des {
     ranges = <0x58 0x74>,
          <0x38 0x30>;
     lvds_ser: serializer@58 {
         compatible = "ti,ds90ub913aq";
         reg = <0x58>;
         remote-device = <&lvds_des>;
         gpio-controller;
         #gpio-cells=1;
         slave-mode;
     };
     lvds_cam: camera@30 {
         compatible = "ov10635";
         reg = <0x30>;
         gpios = <&lvds_ser 0>;
         /* power pin controlled by serializer local gpio */
     };
 };
```

Also, note that the range property decides the address mapping from one bus to other. Address referred to as 0x74 on the remote bus will be aliased on address 0x58 on the local bus.

The advantage with this kind of device modeling is that the kernel driver controlling the camera need not know about the aliases, serializer and de-serializer. Once the Serdes link is ready and aliases are setup, the camera driver works independently without even noticing the I2C transactions getting rerouted. The same driver works on a both cameras connected directly or connected through a Serdes pair.

3.4 Setting up Pinmux and IODELAY

Once the devices are defined in the device tree, it is important to configure the pinmux for the pads used for the video port. To ensure timings on DRA7xx SoCs, it is recommended to perform IODELAY configuration while in isolation in the first stage bootloader. Due to this, all the pinmux and IODELAY configuration is handled in the u-boot. Kernel does not handle any pinmux or IODELAY settings for video ports.

The following patch is an example of adding pinmux data to be configured in the u-boot. This adds entries for using **VIN1A** pads in the **muxmode 6**, which connects the signals to the **vin3a** video port and adds **manual mode** timing seed values for **falling edge pclk**.

Copyright © 2019, Texas Instruments Incorporated

```
diff --git a/board/ti/dra7xx/mux_data.h b/board/ti/dra7xx/mux_data.h
index c2b557f..fcd07fd 100644
--- a/board/ti/dra7xx/mux_data.h
+++ b/board/ti/dra7xx/mux_data.h
@@ -538,6 +538,54 @@ const struct pad_conf_entry dra74x_core_padconf_array[] = {
  {MCASP4_ACLKX, (M4 | PIN_INPUT_PULLUP)},/* mcasp4_aclkx.i2c4_sda */
  {MCASP4_FSX, (M4 | PIN_INPUT_PULLUP)},  /* mcasp4_fsx.i2c4_scl */
+#ifdef CONFIG_TARGET_DRA7XX_EVM_VISION
+ { VIN1B_CLK1, (M6 | PIN_INPUT | MANUAL_MODE) }, /* vin1b_clk1.vin3a_clk0 */
+ { VIN1A_D16, (M6 | PIN_INPUT | MANUAL_MODE) },  /* vin1a_d16.vin3a_d0 */
+ { VIN1A_D17, (M6 | PIN_INPUT | MANUAL_MODE) },  /* vin1a_d17.vin3a_d1 */
+ { VIN1A_D18, (M6 | PIN_INPUT | MANUAL_MODE) },  /* vin1a_d18.vin3a_d2 */
+ { VIN1A_D19, (M6 | PIN_INPUT | MANUAL_MODE) },  /* vin1a_d19.vin3a_d3 */
+ { VIN1A_D20, (M6 | PIN_INPUT | MANUAL_MODE) },  /* vin1a_d20.vin3a_d4 */
+ { VIN1A_D21, (M6 | PIN_INPUT | MANUAL_MODE) },  /* vin1a_d21.vin3a_d5 */
+ { VIN1A_D22, (M6 | PIN_INPUT | MANUAL_MODE) },  /* vin1a_d22.vin3a_d6 */
+ { VIN1A_D23, (M6 | PIN_INPUT | MANUAL_MODE) },  /* vin1a_d23.vin3a_d7 */
+ { VIN2A_D22, (M5 | PIN_INPUT | MANUAL_MODE) },  /* vin2a_d22.vin3a_hsync0 */
+ { VIN2A_D23, (M5 | PIN_INPUT | MANUAL_MODE) },  /* vin2a_d23.vin3a_vsync0 */
+#endif
 };

#ifdef CONFIG_IODELAY_RECALIBRATION
@@ -603,6 +651,53 @@ const struct iodelay_cfg_entry dra742_es1_1_iodelay_cfg_array[] = {
  {0x0174, 1904, 1471},   /* CFG_GPMC_A17_IN */
  {0x0188, 1690, 0},      /* CFG_GPMC_A18_OUT */
  {0x0374, 0, 0},         /* CFG_GPMC_CS2_OUT */
+#ifdef CONFIG_TARGET_DRA7XX_EVM_VISION
+ { 0x0A2C,     0,     0 }, /* CFG_VIN1B_CLK1_IN : VIN3A_CLK0 - VIP2_MANUAL2 */
+ { 0x0930,  2805,   459 }, /* CFG_VIN1A_D16_IN : VIN3A_D0 - VIP2_MANUAL2 */
+ { 0x093C,  2904,   360 }, /* CFG_VIN1A_D17_IN : VIN3A_D1 - VIP2_MANUAL2 */
+ { 0x0948,  2857,   527 }, /* CFG_VIN1A_D18_IN : VIN3A_D2 - VIP2_MANUAL2 */
+ { 0x0954,  2861,   517 }, /* CFG_VIN1A_D19_IN : VIN3A_D3 - VIP2_MANUAL2 */
+ { 0x096C,  2855,   344 }, /* CFG_VIN1A_D20_IN : VIN3A_D4 - VIP2_MANUAL2 */
+ { 0x0978,  2908,   248 }, /* CFG_VIN1A_D21_IN : VIN3A_D5 - VIP2_MANUAL2 */
+ { 0x0984,  2843,   191 }, /* CFG_VIN1A_D22_IN : VIN3A_D6 - VIP2_MANUAL2 */
+ { 0x0990,  2683,     0 }, /* CFG_VIN1A_D23_IN : VIN3A_D7 - VIP2_MANUAL2 */
+ { 0x0AEC,  1606,     0 }, /* CFG_VIN2A_D22_IN : VIN3A_HSYNC0 - VIP2_MANUAL2 */
+ { 0x0AF8,  1673,     0 }, /* CFG_VIN2A_D23_IN : VIN3A_VSYNC0 - VIP2_MANUAL2 */
+#endif
 };

 const struct iodelay_cfg_entry dra742_es2_0_iodelay_cfg_array[] = {
@@ -667,6 +762,42 @@ const struct iodelay_cfg_entry dra742_es2_0_iodelay_cfg_array[] = {
  {0x0174, 2533, 980},    /* CFG_GPMC_A17_IN */
  {0x0188, 590, 0},       /* CFG_GPMC_A18_OUT */
  {0x0374, 0, 0},         /* CFG_GPMC_CS2_OUT */
+#ifdef CONFIG_TARGET_DRA7XX_EVM_VISION
+ { 0x0930,  2244,  1202 }, /* CFG_VIN1A_D16_IN : VIN3A_D0 - VIP2_MANUAL2 */
+ { 0x093C,  2321,  1116 }, /* CFG_VIN1A_D17_IN : VIN3A_D1 - VIP2_MANUAL2 */
+ { 0x0948,  2280,  1288 }, /* CFG_VIN1A_D18_IN : VIN3A_D2 - VIP2_MANUAL2 */
+ { 0x0954,  2282,  1281 }, /* CFG_VIN1A_D19_IN : VIN3A_D3 - VIP2_MANUAL2 */
+ { 0x096C,  2284,  1090 }, /* CFG_VIN1A_D20_IN : VIN3A_D4 - VIP2_MANUAL2 */
+ { 0x0978,  2324,  1000 }, /* CFG_VIN1A_D21_IN : VIN3A_D5 - VIP2_MANUAL2 */
+ { 0x0984,  2278,   915 }, /* CFG_VIN1A_D22_IN : VIN3A_D6 - VIP2_MANUAL2 */
+ { 0x0990,  2423,   398 }, /* CFG_VIN1A_D23_IN : VIN3A_D7 - VIP2_MANUAL2 */
+ { 0x0A2C,     0,     0 }, /* CFG_VIN1B_CLK1_IN : VIN3A_CLK0 - VIP2_MANUAL2 */
+ { 0x0AEC,  1641,     0 }, /* CFG_VIN2A_D22_IN : VIN3A_HSYNC0 - VIP2_MANUAL2 */
+ { 0x0AF8,  1748,     0 }, /* CFG_VIN2A_D23_IN : VIN3A_VSYNC0 - VIP2_MANUAL2 */
+#endif
 };
```


Copyright © 2019, Texas Instruments Incorporated

3.4.1 Getting Pinmux and IODELAY Values

If you are starting with a new board design, see the PCT tool or Pinmux tool to select the right pinmux, IO-sets for the video ports you wish to use.

For getting iodelay values for the specific silicon revision, see the device-specific data manual. For video ports, depending on the whether the data needs to be parsed at rising edge of clock v/s falling edge, different set of IODELAY values are required.

You can also use the IOdelay Python tool to easily generate IODELAY values for the video ports in different formats.

3.5 Setting Up Board Muxes

Depending on the custom board designs, it is possible that the camera interfaces are multiplexed with other peripheral IO and often controlled via board muxes with GPIOs acting as selection lines. Typical camera drivers are written such that the GPIOs to be controlled are descibed with names in the device tree. For example, reset-gpio=**phandle** or power-gpio=**phandle**. This makes sense if the device needs fixed number of GPIOs. These are part of the camera and makes sense to be handled in the camera driver. On the other hand, board muxes are not part of the camera, and the programming changes from each board to the other. The board muxes were set in a generic way by handling a special property called mux-gpios=**list of phandles**.

The following example demonstrates how the same TVP5158 device connected on different boards can be described such that the generic driver can setup all the board muxes required for that instance.

```
----------------------------------------------------------------------------
arch/arm/boot/dts/dra7-evm.dts
----------------------------------------------------------------------------
&tvp_5158{
        mux-gpios = <&pcf_hdmi 3 GPIO_ACTIVE_HIGH>,     /*CAM_FPD_MUX_S0*/
                    <&pcf_jamr3_21 8 GPIO_ACTIVE_LOW>;  /*SEL_TVP_FPD*/
};

----------------------------------------------------------------------------
arch/arm/boot/dts/dra72-evm-revc.dts
----------------------------------------------------------------------------
&tvp_5158{
        mux-gpios = <&pcf_hdmi 2 GPIO_ACTIVE_LOW>,      /*VIN2_S0*/
                    <&pcf_jamr3_21 8 GPIO_ACTIVE_LOW>,  /*SEL_TVP_FPD*/
                    <&pcf_hdmi 6 GPIO_ACTIVE_HIGH>;     /*VIN2_S2*/
};

----------------------------------------------------------------------------
arch/arm/boot/dts/dra76-evm.dts
----------------------------------------------------------------------------
&tvp_5158{
        mux-gpios = <&pcf_jamr3_21 8 GPIO_ACTIVE_LOW>;  /*SEL_TVP_FPD*/
};
```


Copyright © 2019, Texas Instruments Incorporated

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

Copyright © 2022, Texas Instruments Incorporated