Device-Tree changes and Patches required for the 2020.1 Zynq UltraScale+ MPSoC VCU TRD for a ZCU106 board

Device-Tree Changes

SOC devices generally have static .dts/.dtsi files, but when it comes to the FPGA there can be many complicated designs, in which the peripheral logic (PL) IPs may vary or might have different configurations.

Device-Tree Generator (DTG), a part of the Xilinx PetaLinux toolset, dynamically generates device tree file for FPGA components.

Once the device-tree is generated for a hardware design using the PetaLinux tool, the components folder contains astatically configured Device-Tree for the board PS files and Device-Tree files generated for FPGA components.

- **pl.dtsi:** Contains the Device-Tree node for FPGA PL components.
- **pcw.dtsi:** Contains dynamic properties of PS peripherals.
- **system-top.dts:** Contains the memory information, early console and the boot arguments.
- **zynqmp.dtsi:** Contains all of the PS peripheral information and also the CPU information.
- **zynqmp-clk-ccf.dtsi:** Contains all of the clock information for the peripheral IPs.
- **board.dtsi:** Based on the board, DTG generates this file under the same output directory.

In some cases, the Device-Tree generator cannot generate the input and output port information for the video pipelines as there is no information from the design. For these IPs you will need to manually update the port information inside the video pipeline IP nodes.

The following sections describe the video pipelines for which the DTG tool cannot generate complete pipeline information, and you will need to manually update some of the device-tree properties and port information.

HDMI-Rx Audio Pipeline

To create a HDMI-Rx Capture media pipeline, you must update the following pipeline DT nodes.

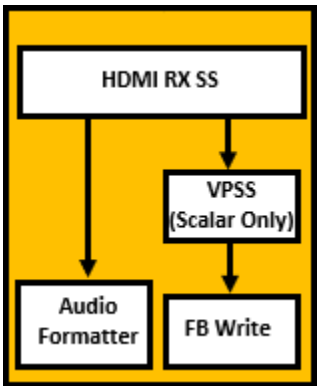

Figure-1: HDMI-Rx and Audio Pipeline Block Diagram

- **HDMI-Rx Audio DT Node**
  To create a media graph on a HDMI-Rx subsystem DT node, you must update the clocks, Video PHY lanes, and ports information.

```
&hdmi_input_v_hdmi_rx_ss_0 {
   phy-names = "hdmi-phy0", "hdmi-phy1", "hdmi-phy2";
   phys = <&vphy_lane0 0 1 1 0>, <&vphy_lane1 0 1 1 0>, <&vphy_lane2 0 1 1 0>;
   xlnx,audio-enabled ;
   xlnx,edid-ram-size = <0x100>;
   xlnx,input-pixels-per-clock = <2>;
   xlnx,max-bits-per-component = <8>;
   xlnx,snd-pcm = <&audio_ss_0_audio_formatter_1>;
   hdmirx_ports: ports {
      #address-cells = <1>;
      #size-cells = <0>;
      hdmirx_port: port@0 {
         /* Fill the fields xlnx,video-format and xlnx,video-width based on user
         requirement */
         reg = <0>;
         xlnx,video-format = <0>;
         xlnx,video-width = <10>;
         hdmi_rx_out: endpoint {
            remote-endpoint = <&vpss_scaler_in>;
         };
      };
   };
};
```

- **Scalar DT Node**

  To create a media pipeline on a Video Processing Sub System (VPSS) scalar node, you must update the information of the compatible string to select the v4l-based scalar driver, along with the reset-gpio and port information.

```
&hdmi_input_v_proc_ss_0 {
   reset-gpios = <&gpio 98 1>;
   compatible = "xlnx,v-vpss-scaler-2.2";
   vpss_ports: ports {
      #address-cells = <1>;
      #size-cells = <0>;
      vpss_port0: port@0 {
         /* For xlnx,video-format user needs to fill as per their requirement */
         reg = <0>;
         xlnx,video-format = <3>;
         xlnx,video-width = <8>;
         vpss_scaler_in: endpoint {
            remote-endpoint = <&hdmi_rx_out>;
         };
      };
      vpss_port1: port@1 {
         /* For xlnx,video-format user needs to fill as per their requirement */
         reg = <1>;
         xlnx,video-format = <3>;
         xlnx,video-width = <8>;
         vpss_scaler_out: endpoint {
            remote-endpoint = <&scd_in>;
```

```
        };
      };
   };
};
```

- **Framebuffer Write**
  On a Framebuffer Write DT node, you must update the reset-gpio, and dma-align information. The dma-align information helps to align DMA data with the VCU encoder.

```
hdmi_input_v_frmbuf_wr_0 {
   reset-gpios = <&gpio 93 1>; /* GPIO information for resetting the IP */
   xlnx,dma-align = <32>; /* Configure 32 byte align data from framebuffer write as
   encoder also works on this 32 byte align data */
};
```

- **Create HDMI-Rx Capture Video Pipeline**
  Create the v4l2 video capture pipeline by adding DMA and port information to the HDMI Capture DT node. The video pipeline is created based on this node.

```
vcap_hdmi {
   compatible = "xlnx,video"; dma-names = "port0";
   dmas = <&hdmi_input_v_frmbuf_wr_0 0>; /* Map Frame write dma as video capture
   dma driver */
   vcap_hdmi_ports: ports {
      #address-cells = <1>;
      #size-cells = <0>;
      vcap_hdmi_port: port@0 {
         direction = "input";
         reg = <0>;
         vcap_hdmi_in: endpoint {
            remote-endpoint = <&vpss_scaler_out>; /* Map vpss scalar out to dma driver
            */
         };
      };
   };
};
```

HDMI-Tx Audio Pipeline

To create a HDMI-Tx media pipeline, you need to update the following pipeline DT nodes.

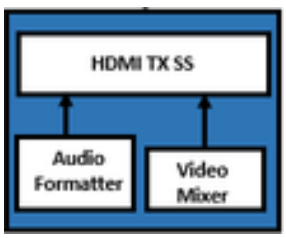

Figure-2: HDMI-Tx and Audio Pipeline Block Diagram

- **HDMI-Tx Audio DT Node**

On a HDMI-Tx subsystem DT node, you must update the clocks, Video PHY lanes, and ports information.

```
&hdmi_output_v_hdmi_tx_ss_0{
   phy-names = "hdmi-phy0", "hdmi-phy1", "hdmi-phy2";
   phys = <&vphy_lane0 0 1 1 1>, <&vphy_lane1 0 1 1 1>, <&vphy_lane2 0 1 1 1>;
   reg = <0x0 0xa0020000 0x0 0x20000>;
   clock-names = "s_axi_cpu_aclk", "link_clk", "s_axis_audio_aclk", "video_clk",
   "s_axis_video_aclk","txref-clk", "retimer-clk";
   clocks = <&zynqmp_clk 71>, <&misc_clk_0>, <&misc_clk_1>, <&misc_clk_2>,
   <&zynqmp_clk 72>, <&si5319 0>, <&dp159>;
   reg-names = "hdmi-txss";
   xlnx,audio-enabled ;
   xlnx,input-pixels-per-clock = <2>;
   xlnx,max-bits-per-component = <8>;
   xlnx,snd-pcm = <&audio_ss_0_audio_formatter_1>;
   xlnx,xlnx-hdmi-acr-ctrl = <&hdmi_input_hdmi_acr_ctrl_0>;
   hdmitx_ports: ports {
      #address-cells = <1>;
      #size-cells = <0>;
      encoder_hdmi_port: port@0 {
         reg = <0>;
         hdmi_encoder: endpoint {
            remote-endpoint = <&mixer_crtc>;
         };
      };
   };
};
```

- **Video Mixer DT Node**

  To create a media pipeline on a Video Mixer DT node, you must update the clocks, reset-gpio, ports, and overlay layers information.

```
&hdmi_output_v_mix_0 {
   reset-gpios = <&gpio 98 1>;
   /delete-property/ clock-names;
   clocks = <&si5319 0>;
   xlnx,dma-addr-width = <64>;
   crtc_mixer_port: port@0 {
   reg = <0>;
      mixer_crtc: endpoint {
         remote-endpoint = <&hdmi_encoder>;
      };
   };
   xx_mix_master: layer_0 {
      xlnx,layer-id = <0>;
      xlnx,layer-max-height = <2160>;
      xlnx,layer-max-width = <3840>;
      xlnx,layer-primary ;
      xlnx,vformat = "BG24";
   };
   xx_mix_overlay_1: layer_1 {
```

```
      xlnx,layer-id = <1>;
      xlnx,layer-max-width = <1920>;
      xlnx,vformat = "NV12";
   };
   xx_mix_overlay_2: layer_2 {
      xlnx,layer-id = <2>;
      xlnx,layer-max-width = <1920>;
      xlnx,vformat = "NV12";
   };
   xx_mix_overlay_3: layer_3 {
      xlnx,layer-id = <3>;
      xlnx,layer-max-width = <1920>;
      xlnx,vformat = "NV12";
   };
   xx_mix_overlay_4: layer_4 {
      xlnx,layer-id = <4>;
      xlnx,layer-max-width = <1920>;
      xlnx,vformat = "NV12";
   };
   xx_mix_overlay_5: layer_5 {
      xlnx,layer-id = <5>;
      xlnx,layer-max-width = <1920>;
      xlnx,vformat = "NV12";
   };
   xx_mix_overlay_6: layer_6 {
      xlnx,layer-id = <6>;
      xlnx,layer-max-width = <1920>;
      xlnx,vformat = "NV12";
   };
   xx_mix_overlay_7: layer_7 {
      xlnx,layer-id = <7>;
      xlnx,layer-max-width = <1920>;
      xlnx,vformat = "NV12";
   };
   xx_mix_overlay_8: layer_8 {
      xlnx,layer-id = <8>;
      xlnx,layer-max-width = <1920>;
      xlnx,vformat = "NV12";
   };
   xx_mix_logo: logo {
      xlnx,layer-id = <9>;
      xlnx,logo-height = <64>;
      xlnx,logo-width = <64>;
   };
};
```

- **HDMI I2C clocks**

  On HDMI I2c clock node, you need to update the si5319 and dp159 clock nodes used by the HDMI-Tx and Video Mixer DT node.

```
&mpsoc_ss_hdmi_ctrl_iic {
   clocks = <&vid_s_axi_clk>;
```

```
    /* Si5319 i2c clock generator */
    si5319: clock-generator@68 {
        status = "okay";
        compatible = "silabs,si5319";
        reg = <0x68>;
        #address-cells = <1>;
        #size-cells = <0>;
        #clock-cells = <1>;

        /* input clock(s); the XTAL is hard-wired on the ZCU102 board */
        clocks = <&refhdmi>;
        clock-names = "xtal";

        /* output clocks */
        clk0 {
            reg = <0>;
            /* HDMI TX reference clock output frequency */
            clock-frequency = <27000000>;
        };
    };

    /* DP159 exposes a virtual CCF clock. Upon .set_rate() */
    /* it adapts its retiming/driving behavior */
    dp159: hdmi-retimer@5e {
        status = "okay";
        compatible = "ti,dp159";
        reg = <0x5e>;
        #address-cells = <1>;
        #size-cells = <0>;
        #clock-cells = <0>;
    };
};
```

- **Framebuffer Read**
  On a Framebuffer Read DT node, you need to update the reset-gpio information.

```
&hdmi_output_v_frmbuf_rd_0 {
    reset-gpios = <&gpio 79 1>;
};
```

HDMI Audio Pipeline

- **HDMI Audio Formatter**
  On an Audio Formatter DT node, you must update the clocks, and input/output audio devices information.

```
&audio_ss_0_audio_formatter_1 {
    clocks = <&zynqmp_clk 71>, <&si570_1>, <&si570_1>, <&zynqmp_clk 71>;
    xlnx,rx = <&hdmi_input_v_hdmi_rx_ss_0>;
    xlnx,tx = <&hdmi_output_v_hdmi_tx_ss_0>;
```

```
};
```

SDI-Rx Audio Pipeline

To create a media pipeline for an SDI Audio capture pipeline, you must update the following pipeline DT nodes.

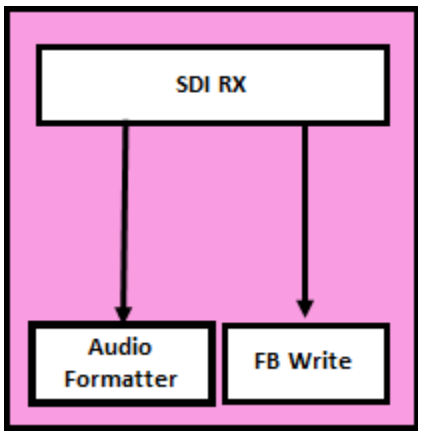

Figure-3: SDI-Rx and Audio Pipeline Block Diagram

- **SDI-Rx Audio DT Node**

  On a SDI-Rx with Audio Capture DT node, update the clocks, reset GT GPIOs, and ports information.

```
&sdi_rx_input_v_smpte_uhdsdi_rx_ss {
    clock-names = "video_out_clk", "sdi_rx_clk", "s_axi_aclk";
    clocks = <&zynqmp_clk 72>, <&si570_2>, <&zynqmp_clk 71>;
    reset_gt-gpios = <&axi_gpio_0 0 0 1>;
    sdirx_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        sdirx_port: port@0 {
            /* Fill the fields xlnx,video-format and xlnx,video-width based on user
            requirement */
            reg = <0>;
            xlnx,video-format = <0>;
            xlnx,video-width = <10>;
            sdi_rx_out: endpoint {
                remote-endpoint = <&vcap_sdirx_in>;
            };
        };
    };
};
```

- **Framebuffer Write**

  On a Framebuffer Write DT node, you must update the reset-gpio, and dma-align information. The dma-align information helps to align DMA data with the VCU encoder.

```
&sdi_rx_input_v_frmbuf_wr_0 {
    reset-gpios = <&gpio_resets_axi_gpio_resets 1 0 1>;
```

```
    xlnx,dma-align = <32>;
};
```

- **Create SDI Video Capture Pipeline**
  Add the DMA and port information to create the v4l2 video capture pipeline.

```
vcap_sdirx {
    compatible = "xlnx,video";
    dma-names = "port0";
    dmas = <&sdi_rx_input_v_frmbuf_wr_0 0>;
    vcap_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        vcap_port: port@0 {
            direction = "input";
            reg = <0>;
            vcap_sdirx_in: endpoint {
                remote-endpoint = <&sdi_rx_out>;
            };
        };
    };
};
```

SDI-Tx Audio Pipeline

For creating the SDI-Tx media pipeline, you must update the following pipeline DT nodes.

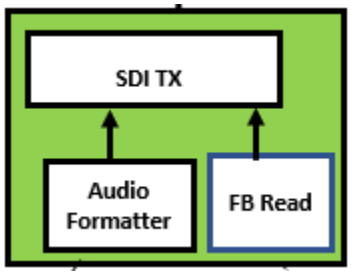

Figure-4: SDI-Tx and Audio Pipeline Block Diagram

- **SDI-Tx Audio DT Node**
  On a SDI-Tx subsystem node, you must update the reset-gpio and SDI ports information.

```
&sdi_tx_output_v_smpte_uhdsdi_tx_ss {
    clock-names     =     "sdi_tx_clk",     "video_in_clk",
    "s_axi_aclk";
    clocks     =     <&si570_2>,     <&zynqmp_clk     72>,
    <&zynqmp_clk 71>;
    reset-gpios = <&axi_gpio_0 0 0 0>;
    picxo_rst-gpios = <&axi_gpio_0 2 0 0>;

    sditx_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
```

```
      encoder_sdi_port: port@0 {
         reg = <0>;
         sdi_encoder: endpoint {
            remote-endpoint = <&pl_disp_crtc>;
         };
      };
      sdi_audio_port: port@1 {
         reg = <1>;
         sdi_audio_sink_port: endpoint {
            remote-endpoint                              =
            <&sditx_audio_embed_src>;
         };
      };
   };
};
```

- **Framebuffer Read**
  On a Framebuffer Read DT node, you must update the reset-gpio information.

```
&sdi_tx_output_v_frmbuf_rd_0 {
   reset-gpios = <&gpio_resets_axi_gpio_resets 0 0 1>;
};
```

- **Display CRTC DT Node**
  You can create the video playback pipeline by adding the information of the compatible string to select the display driver, along with the DMA and port information.

```
v_pl_disp: drm-pl-disp-drv {
   /* Fill the field xlnx,vformat based on user requirement */
   compatible = "xlnx,pl-disp";
   dma-names = "dma0";
   dmas = <&sdi_tx_output_v_frmbuf_rd_0 0>;
   xlnx,vformat = "YUYV";
   pl_display_port: port@0 {
      reg = <0>;
      pl_disp_crtc: endpoint {
         remote-endpoint = <&sdi_encoder>;
      };
   };
};
```

SDI Audio Pipeline

- **SDI Audio formatter**
  On an Audio Formatter node, you must update the input/output audio devices information.

```
&audio_ss_audio_formatter_0 {
   xlnx,rx = <&audio_ss_v_uhdsdi_audio_1_extract>;
   xlnx,tx = <&audio_ss_v_uhdsdi_audio_0_embed>;
};
```

Audio Embed and Audio Extract IP nodes are required to embed/extract audio in the SDI Tx and Rx protocols respectively. SDI embed contains an output port to a remote endpoint of the SDI video Tx node.

- **Audio Embed DT Node**

```
&audio_ss_v_uhdsdi_audio_0_embed {
   xlnx,snd-pcm = <&audio_ss_audio_formatter_0>;
   sdi_av_port: port@0 {
      reg = <0>;
      sditx_audio_embed_src: endpoint {
         remote-endpoint = <&sdi_audio_sink_port>;
      };
   };
};
```

- **Audio Extract**

```
&audio_ss_v_uhdsdi_audio_1_extract {
   xlnx,sdi-rx-video = <&sdi_rx_input_v_smpte_uhdsdi_rx_ss>;
   xlnx,snd-pcm = <&audio_ss_audio_formatter_0>;
};
```

I2S-Rx Pipeline

On a I2S-Rx DT node, you must need to update the PCM/DMA driver node.

Figure-5: I2S-Rx Audio block diagram

- **I2S-Rx DT Node**

```
&audio_ss_0_i2s_transmitter_0 {
      xlnx,snd-pcm = <&audio_ss_0_audio_formatter_2>;
};
```

I2S-Tx Pipeline

On a I2S-Tx DT node, you need to update the PCM/DMA driver node.

Figure-6: I2S-Tx Audio block diagram

- **I2S-Tx DT Node**

```
&audio_ss_0_i2s_receiver_0 {
	xlnx,snd-pcm = <&audio_ss_0_audio_formatter_2>;
};
```

I2S Audio

On an Audio Formatter node, you must update the input/output audio devices information.

- **I2S Audio formatter DT Node**

```
&audio_ss_0_audio_formatter_2 {
	xlnx,rx = <&audio_ss_0_i2s_receiver_0>;
	xlnx,tx = <&audio_ss_0_i2s_transmitter_0>;
};
```

MIPI CSI-Rx Pipeline

To create a Mobile Industry Processor Interface (MIPI) Camera Serial Interface (CSI)-Rx Capture media pipeline, you must update the following pipeline DT nodes.

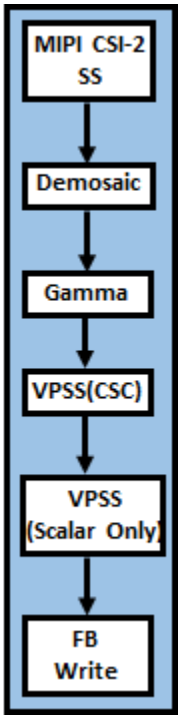

Figure-7: MIPI CSI-Rx Pipeline Block Diagram

- **LI-IMX274MIPI Sensor DT Node**

  On an IMX274 MIPI sensor node you must update the compatible string, reset-gpios and remote-endpoint information.

```
&sensor_iic_0 {
    clocks = <&vid_s_axi_clk>;
    imx274: sensor@1a{
        compatible = "sony,imx274";
        reg = <0x1a>;
        #address-cells = <1>;
        #size-cells = <0>;
        reset-gpios = <&gpio 90 0>;

        port@0 {
            reg = <0>;
            sensor_out: endpoint {
                remote-endpoint = <&csiss_in>; /* IMX274 <-> CSI-2 Rx */
            };
        };
    };
};
```

- **MIPI CSI-2 SS DT Node**
  For the MIPI CSI-2 subsystem DT node, you need to update the information in the compatible string to select the MIPI CSI 2 capture driver along with the CSI ports, for creating a media pipeline.

```
&mipi_csi2_rx_mipi_csi2_rx_subsystem_0 {
   compatible = "xlnx,mipi-csi2-rx-subsystem-2.0";
   csiss_ports: ports {
      #address-cells = <1>;
      #size-cells = <0>;
      csiss_port0: port@0 {
         /* Fill cfa-pattern=rggb for raw data types, other fields video format and video-
         width user needs to fill */
         reg = <0>;
         xlnx,cfa-pattern = "rggb";
         xlnx,video-format = <12>;
         xlnx,video-width = <8>;
         csiss_out: endpoint {
            remote-endpoint = <&demosaic_in>;
         };
      };
      csiss_port1: port@1 {
         /* Fill cfa-pattern=rggb for raw data types, other fields video format, video-width
         user needs to fill */
         /* User need to add something like remote-endpoint=<&out> under the node
         csiss_in:endpoint */
         reg = <1>;
         xlnx,cfa-pattern = "rggb";
         xlnx,video-format = <12>;
         xlnx,video-width = <8>;
         csiss_in: endpoint {
            data-lanes = <1 2 3 4>;
            remote-endpoint = <&sensor_out>;
         };
      };
   };
};
```

- **Demosaic DT Node**
  On a Demosaic DT node, you must update the reset-gpio, and demosaic ports information .

```
&mipi_csi2_rx_v_demosaic_0 {
   reset-gpios = <&gpio 85 1>;
   xlnx,max-height = <2160>;
   xlnx,max-width = <3840>;
   demosaic_ports: ports {
      #address-cells = <1>;
      #size-cells = <0>;
      demosaic_port0: port@0 {
         /* For cfa-pattern=rggb user needs to fill as
         per BAYER format */
         reg = <0>;
```

```
            xlnx,cfa-pattern = "rggb";
            xlnx,video-width = <8>;
            demosaic_in: endpoint {
               remote-endpoint = <&csiss_out>;
            };
         };
         demosaic_port1: port@1 {
            /* For cfa-pattern=rggb user needs to fill as
            per BAYER format */
            reg = <1>;
            xlnx,cfa-pattern = "rggb";
            xlnx,video-width = <8>;
            demosaic_out: endpoint {
               remote-endpoint = <&gamma_in>;
            };
         };
      };
};
```

- **Gamma DT Node**

  On a Gamma DT node, you must update the reset-gpio, and gamma ports information.

```
&mipi_csi2_rx_v_gamma_lut_0 {
   reset-gpios = <&gpio 86 1>;
   gamma_ports: ports {
      #address-cells = <1>;
      #size-cells = <0>;
      gamma_port0: port@0 {
         reg = <0>;
         xlnx,video-width = <8>;
         gamma_in: endpoint {
            remote-endpoint = <&demosaic_out>;
         };
      };
      gamma_port1: port@1 {
         reg = <1>;
         xlnx,video-width = <8>;
         gamma_out: endpoint {
            remote-endpoint = <&csc_in>;
         };
      };
   };
};
```

- **VPSS CSC DT Node**

  To create the media pipeline on a Video Processing Sub System (VPSS) Color Space Converter (CSC) DT node, you must update the information of the compatible string to select the v4l-based scalar driver, along with the reset-gpio, and SCS port information.

```
&mipi_csi2_rx_v_proc_ss_csc {
```

```
    compatible = "xlnx,v-vpss-csc";
    reset-gpios = <&gpio 84 1>;
    csc_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        csc_port0: port@0 {
            /* For xlnx,video-format user needs to fill as per their requirement */
            reg = <0>;
            xlnx,video-format = <3>;
            xlnx,video-width = <8>;
            csc_in: endpoint {
                remote-endpoint = <&gamma_out>;
            };
        };
        csc_port1: port@1 {
            /* For xlnx,video-format user needs to fill as per their
            requirement */
            reg = <1>;
            xlnx,video-format = <3>;
            xlnx,video-width = <8>;
            csc_out: endpoint {
                remote-endpoint = <&scaler_in>;
            };
        };
    };
};
```

- **Scaler DT Node**

  To create the media pipeline on a vproc subsystem scalar node, you must update the information of the compatible string to select the v4l-based scalar driver, along with the reset-gpio and port information.

```
&mipi_csi2_rx_v_proc_ss_scaler {
    compatible = "xlnx,v-vpss-scaler-2.2";
    reset-gpios = <&gpio 82 1>; /* Reset GPIO connected to scalar IP */
    scaler_ports: ports { /* Connect the scalar input to CSC output node */
        #address-cells = <1>;
        #size-cells = <0>;
        scaler_port0: port@0 {
            /* For xlnx,video-format user needs to fill as per their requirement */
            reg = <0>;
            xlnx,video-format = <3>;
            xlnx,video-width = <8>;
            scaler_in: endpoint {
                remote-endpoint = <&csc_out>;
            };
        };
        scaler_port1: port@1 { /* connect scaler output node to capture csi node */
        /* For xlnx,video-format user needs to fill as per their requirement */
            reg = <1>;
            xlnx,video-format = <3>;
            xlnx,video-width = <8>;
```

```
            scaler_out: endpoint {
                remote-endpoint = <&vcap_csi_in>;
            };
        };
    };
};
```

- **Framebuffer Write**
  On a Framebuffer Write DT node, you must update reset-gpio, and dma-align information. The dma-align information helps to align DMA data with the VCU encoder.

```
&mipi_csi2_rx_v_frmbuf_wr_0 {
    reset-gpios = <&gpio 80 1>;
    xlnx,dma-align = <32>;
};
```

- **Create MIPI-CSI2 Rx Capture Video Pipeline**
  To create a CSI video capture pipeline, add DMA and port information

```
vcap_csi {
    compatible = "xlnx,video";
    dma-names = "port0";
    dmas = <&mipi_csi2_rx_v_frmbuf_wr_0 0>;
    vcap_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        vcap_port: port@0 {
            direction = "input";
            reg = <0>;
            vcap_csi_in: endpoint {
                remote-endpoint = <&scaler_out>;
            };
        };
    };
};
```

TPG Pipeline

To create a media pipeline for the Test Pattern Generator (TPG) pipeline, you must update the following pipeline DT nodes:

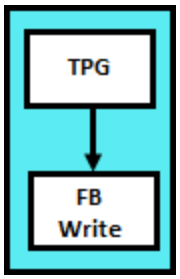

Figure-8: TPG Pipeline Block Diagram

- **TPG DT Node**
  On a TPG node, you must update the reset-gpio and ports information.

```
&tpg_input_v_tpg_1{
   reset-gpios = <&gpio 96 1>;
   xlnx,vtc = <&tpg_input_v_tc_1>;
   tpg_ports0: ports {
   #address-cells = <1>;
   #size-cells = <0>;
   tpg_port0: port@0 {
         /* Fill the field xlnx,video-format based on user requirement */
         reg = <0>;
         xlnx,video-format = <0x3>;
         xlnx,video-width = <8>;
         tpg_out0: endpoint {
            remote-endpoint = <&vcap_dev_in0>;
         };
      };
   };
};
```

- **Framebuffer Write**
  On a Framebuffer Write DT node, you must update the reset-gpio, and dma-align information. The dma-align information helps to align DMA data with the VCU encoder.

```
&tpg_input_v_frmbuf_wr_0 {
   reset-gpios = <&gpio 94 1>;
   xlnx,dma-align = <32>;
};
```

- **Create TPG Pipeline**
  Add the DMA and port information to create a TPG pipeline.

```
vcap_tp0 {
   compatible = "xlnx,video";
   dma-names = "port0";
   dmas = <&tpg_input_v_frmbuf_wr_0 0>;
   v_ports0: ports {
         #address-cells = <1>;
         #size-cells = <0>;
         v_port0: port@0 {
            direction = "input";
            reg = <0>;
            vcap_dev_in0: endpoint {
               remote-endpoint = <&tpg_out0>;
            };
         };
      };
   };
};
```

Streaming-Based SCD

For creating a HDMI-Streaming based SCD media pipeline, you must update the following pipeline DT nodes.

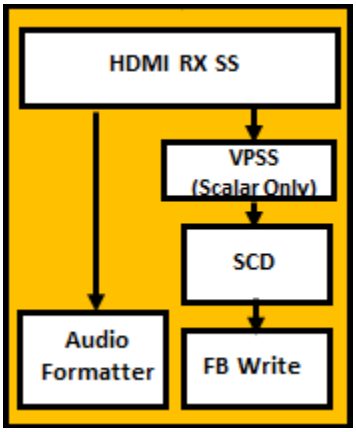

Figure-9: Streaming-Based SCD Block Diagram

- **HDMI Rx Stream-based SCD Pipeline**

  On a HDMI-Rx stream-based SCD node, you must update Video PHY lanes, audio formatter, and ports information.

```
&hdmi_input_v_hdmi_rx_ss_0 {
   phy-names = "hdmi-phy0", "hdmi-phy1", "hdmi-phy2";
   phys = <&vphy_lane0 0 1 1 0>, <&vphy_lane1 0 1 1 0>, <&vphy_lane2 0 1 1 0>;
   xlnx,audio-enabled ;
   xlnx,edid-ram-size = <0x100>;
   xlnx,input-pixels-per-clock = <2>;
   xlnx,max-bits-per-component = <8>;
   xlnx,snd-pcm = <&audio_ss_0_audio_formatter_1>;
   hdmirx_ports: ports {
      #address-cells = <1>;
      #size-cells = <0>;
      hdmirx_port: port@0 {
         /* Fill the fields xlnx,video-format and xlnx,video-width based on user
         requirement */
         reg = <0>;
         xlnx,video-format = <0>;
         xlnx,video-width = <10>;
         hdmi_rx_out: endpoint {
            remote-endpoint = <&vpss_scaler_in>;
         };
      };
   };
};
```

- **VPROC SS DT Node**

  To create the media pipeline on a VPSS scaler node, you must perform the following steps:

1. Update the information of the compatible string to select the v4l-based scalar driver.
2. Update the reset-gpio, and port information.

```
&hdmi_input_v_proc_ss_0 {
   reset-gpios = <&gpio 98 1>; /* Reset GPIO connected to VPSS scalar IP */
   compatible = "xlnx,v-vpss-scaler-2.2"; /* V4L2 scalar selection using the compatible
   string */
   vpss_ports: ports {
      #address-cells = <1>;
      #size-cells = <0>;
      vpss_port0: port@0 { /* Connect the scalar input to HDMI-Rx output node */
         /* For xlnx,video-format user needs to fill as per their requirement */
         reg = <0>;
         xlnx,video-format = <3>;
         xlnx,video-width = <8>;
         vpss_scaler_in: endpoint {
            remote-endpoint = <&hdmi_rx_out>;
         };
      };
      vpss_port1: port@1 { /* connect VPSS output node to capture video node */
      /* For xlnx,video-format user needs to fill as per their requirement */
         reg = <1>;
         xlnx,video-format = <3>;
         xlnx,video-width = <8>;
         vpss_scaler_out: endpoint {
         remote-endpoint = <&scd_in>;
         };
      };
   };
};
```

- **SCD DT Node**

  To create a media pipeline on an SCD DT node, you must update the reset GPIO and SCD ports information.

```
&hdmi_input_v_scenechange_0 {
   reset-gpios = <&gpio 88 1>;
   scd_ports: scd {
      #address-cells = <1>;
      #size-cells = <0>;
      scd_port0: port@0 {
         reg = <0>;
         scd_in: endpoint {
            remote-endpoint = <&vpss_scaler_out>;
         };
      };
      scd_port1: port@1 {
         reg = <1>;
         scd_out: endpoint {
            remote-endpoint = <&scd_hdmi_in>;
         };
      };
```

```
    };
};
```

- **HDMI Audio Formatter**
  On an Audio Formatter DT node, you must update the clocks and input/output audio devices information.

```
&audio_ss_0_audio_formatter_1 {
    clocks = <&zynqmp_clk 71>, <&si570_1>, <&si570_1>, <&zynqmp_clk 71>;
    xlnx,rx = <&hdmi_input_v_hdmi_rx_ss_0>;
    xlnx,tx = <&hdmi_output_v_hdmi_tx_ss_0>;
};
```

- **Framebuffer Write**
  On a Framebuffer Write DT node, you must update reset-gpio and dma-align information. The dma-align information helps to align DMA data with the VCU encoder.

```
&hdmi_input_v_frmbuf_wr_0 {
    reset-gpios = <&gpio 78 1>;
    xlnx,dma-align = <32>;
};
```

- **Create the HDMI Stream-based SCD Video Pipeline**
  Add DMA and SCD port information to a HDMI capture DT node to create a v4l2 video capture pipeline with a stream-based SCD video pipeline.

```
vcap_hdmi {
    compatible = "xlnx,video";
    dma-names = "port0";
    dmas = <&hdmi_input_v_frmbuf_wr_0 0>;
    scd_hdmi_ports: ports {
        #address-cells = <1>;
        #size-cells = <0>;
        scd_hdmi_port: port@0 {
            direction = "input";
            reg = <0>;
            scd_hdmi_in: endpoint {
                remote-endpoint = <&scd_out>;
            };
        };
    };
};
```

Memory-Based SCD

The Video SCD IP is a configurable IP core that can read up to eight video streams in Memory mode. In Memory mode, input is read from the memory mapped AXI4 interface.

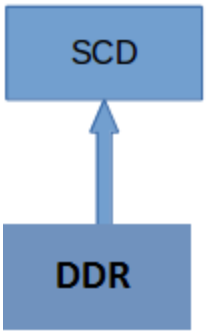

Figure-10: Memory-based SCD block diagram

For memory-based SCD, there is nothing to update manually. The SCD node is in pl.dtsi, which is an autogenerated .dtsi using PetaLinux.

You might need to add reset-gpio information in Scene Change node.

```
&v_scenechange_0 {
   reset-gpios = <&gpio 168 1>;
};
```

Enabling PL DDR for VCU

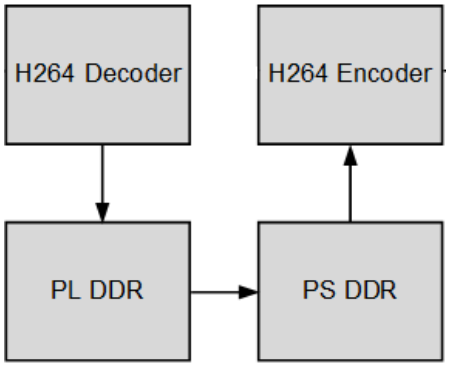

Figure-11: VCU PL DDR block diagram

The PL DDR design is a proposed new design approach to use PL_DDR for decoding and PS_DDR for encoding, so that the DDR bandwidth would be enough to achieve transcoding at 4k@60fps. The figure below explains the transcoding pipeline. The decoder completes the decoding process and writes the data to PL_DDR. The same data is copied to PS_DDR and from there the encoder consumed the data. The buffer copy from PS_DDR to PL_DDR is achieved by DMA transfers.

- The UC2 design requires two Frame Buffer IP cores for DMA transfers. The video_m2m device node constructs the video mem2mem pipeline. The .dtsi changes are as follows:

```
&amba_pl {
   video_m2m {
      compatible = "xlnx,mem2mem";
      dmas = <&v_frmbuf_rd_0 0>, <&v_frmbuf_wr_0 0>;
      dma-names = "tx", "rx";
   };
```

```
};
```

- Update the device tree node to have a reserved memory from the PL DDR for the VCU decoder.

```
/ {
   reserved-memory {
      #address-cells = <0x2>;
      #size-cells = <0x2>;
      ranges;
      plmem_vcu: vcu_dma_mem_region {
         compatible = "shared-dma-pool";
         no-map;
         reg = <0x48 0x0 0x0 0x70000000>;
      };
   };
};
```

- The VCU device-tree node requires changes to allocate memory from PL_DDR. If the decoder is connected to PL-DDR:

```
&decoder {
   memory-region = <&plmem_vcu>;
};
```

Enabling APM for VCU

You will need to provide the VCU APM node manually to get encoder and decoder bandwidth measurement because the APM is internal to the VCU and its base address starts from the middle of the VCU base address. As a result, the DTG tool is not able to generate this node manually.

```
vcu_apm: apm@0xa0140000 {
   #address-cells = <0x1>;
   #size-cells = <0x1>;
   compatible = "generic-uio";
   reg = <0x0 0xa0140000 0x0 0x10000>;
   reg-names = "generic";
};
```

HDMI Dummy pipeline

Having an AXI broadcaster after the scaler into the HDMI-Rx pipeline will allow you to replicate the original HDMI-Rx data to multiple framebuffers. In order to create a capture pipeline for the V4L2 subsystem we have developed a HDMI dummy driver which registers a dummy HDMI node to the V4L2 subsystem using the following DT node.

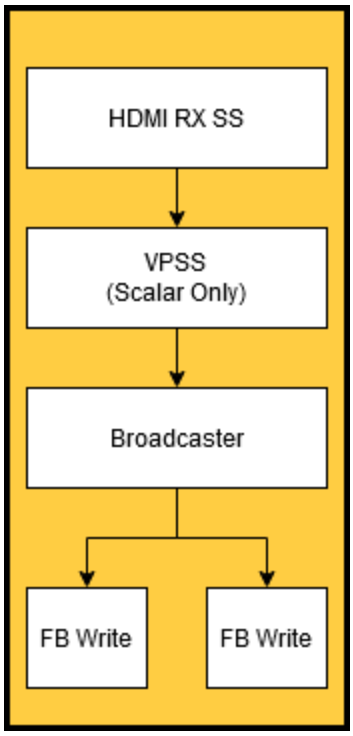

Figure-12: HDMI-Rx and dummy pipeline block diagram

- **HDMI dummy node**

```
dummy {
   compatible = "xlnx,dummy";
   #address-cells = <0x1>;
   #size-cells = <0x0>;
   clocks = <&vid_stream_clk>;
   reg = <0x0 0xa02f0000 0x0 0x100>;

   ports {
      #address-cells = <0x1>;
      #size-cells = <0x0>;

      port@0 {
         reg = <0x0>;
         xlnx,video-format = <0x0>;
         xlnx,video-width = <0x8>;

         hdmi_input_2:endpoint {
            remote-endpoint = <&vcap_hdmi_in_2>; /* HDMI dummy node <->
            Framebuffer write DMA */
         };
      };
   };
};
```

- **V4L2 node to create HDMI dummy pipeline**

```
vcap_hdmi_2 {
   compatible = "xlnx,video";
   dmas = <&hdmi_input_v_frmbuf_wr_1 0>; /* Map Framebuffer write DMA */
   dma-names = "port0";

   ports {
      #address-cells = <0x1>;
      #size-cells = <0x0>;

      port@0 {
         reg = <0x0>;
         direction = "input";

         vcap_hdmi_in_2:endpoint {
            remote-endpoint = <&hdmi_input_2>; /* HDMI dummy node <->
            Framebuffer write DMA */
         };
      };
   };
};
```

Video PHY Controller DT Node

For the Video PHY controller DT node, you need to update the clock property according to the design.

- **For 10G and Audio designs**

```
&vid_phy_controller {
   clock-names = "mgtrefclk0_pad_p_in", "mgtrefclk0_pad_n_in", "mgtrefclk1_pad_p_in",
   "mgtrefclk1_pad_n_in",          "gtsouthrefclk1_in",          "gtsouthrefclk1_odiv2_in",
   "vid_phy_tx_axi4s_aclk",          "vid_phy_rx_axi4s_aclk",          "vid_phy_sb_aclk",
   "vid_phy_axi4lite_aclk", "drpclk","dru-clk";
   clocks = <&misc_clk_4>, <&misc_clk_4>, <&misc_clk_4>, <&misc_clk_4>,
   <&misc_clk_5>, <&misc_clk_5>, <&misc_clk_0>, <&misc_clk_0>, <&zynqmp_clk
   71>, <&zynqmp_clk 71>, <&zynqmp_clk 71>,<&si5328 0>;
};
```

- **Si5328 clock DT node for 10G and Audio designs dru-clk**

```
&si5328 {
   status = "okay";
   compatible = "silabs,si5328";
   reg = <0x69>;
   #address-cells = <1>;
   #size-cells = <0>;
   #clock-cells = <1>;

   /* input clock(s); the XTAL is hard-wired on the ZCU106 board */
   clocks = <&refhdmi>;
   clock-names = "xtal";
```

```
    /* output clocks */
    clk0 {
        reg = <0>;
        /* VPHY DRU reference clock output frequency */
        clock-frequency = <148500000>;
    };
};
```

- **For Other designs**

```
&vid_phy_controller {
    clock-names = "vid_phy_axi4lite_aclk", "dru-clk";
    clocks = <&vid_s_axi_clk>, <&si570_2>;
};
```

10G Ethernet DT node

On the 10G Ethernet DT node, you need to update local MAC address information.

```
/ {
    aliases {
        ethernet0 = &gem3;
        ethernet1 = &eth_sys_xxv_ethernet_0;
    };
};

&eth_sys_xxv_ethernet_0 {
    local-mac-address = [00 0a 35 03 04 ff];
};
```

PCIe Subsystem DT node

On the PCIe Subsystem DT node, you need to update the interrupt-parent, interrupts and interrupt-names properties.

```
&PCIe_Subsystem_pcie_reg_space_v1_0_0 {
    interrupt-parent = <&gic>;
    interrupts = <0x0 89 0x4 0x0 90 0x4 0x0 91 0x4 >;
    interrupt-names = "read", "write", "close";
};
```

AXI Interrupt controller DT node

On the AXI Interrupt controller DT node you need to update the GIC interrupt connected to the AXI interrupt controller.

```
&axi_intc_0 {
    interrupt-parent = <&gic>;
    interrupts = <0 108 4>;
```

```
};
```

UART1 DT node

To disable UART1 below you will need set it in the respective DT file.

```
&uart1 {
   status = "disabled";
};
```

Bootargs

For VCU TRD use-cases, the below bootargs need to be set in the respective DT file.

```
/ {
   chosen {
      bootargs = "earlycon clk_ignore_unused consoleblank=0 cma=1700M
      uio_pdrv_genirq.of_id=generic-uio";
      stdout-path = "serial0:115200n8";
   };
};
```

Kernel Changes

The below kernel patches need to be applied on the released 2020.1 linux-xlnx for the VCU TRD use-cases to work.

#	Patches	Comments
1	0001-media-xilinx-TPG-Add-IOCTL-to-set-PPC.patch	This patch provides an interface to set the pixels per clock. User can call the IOCTL to set PPC for TPG. This patch will enable the TPG to run at 60fps.
2	0002-Add-2nd-HDMI-support-by-adding-a-dummy-driver.patch	This patch adds a dummy driver to support dummy HDMI devices. In the design we have added a broadcaster to support multiple HDMI sources from the single source to showcase Multi-stream feature of the VCU. The dummy driver will provide an interface driver for creating multiple video devices.
3	0003-drm-xlnx_mixer-Dont-enable-primary-plane-by-default.patch	This patch disables the mixer primary plane by default. Enabling primary by default is causing bandwidth issue with the TRD serial use-cases. This patch disables primary which will avoid bandwidth issues
4	0004-drm_atomic_helper-Supress-vblank-timeout-warning-mes.patch	This patch suppresses the drm vblank timeout warning message from the drm atomic framework. This patch is used to avoid warning messages from the atomic drm framework.
5	0006-drivers-misc-add-support-for-interrupt-based-PCIe-en.patch	This patch is to transfer data between the HOST system and a ZCU106 board through PCIe. This patch adds support to access the user-space bar map registers of PCIe from the Xilinx PCIe endpoint driver in order to do read and write transfers.
6	0007-xilinx_pci_endpoint-Add-resolution-use-case-and-64-b.patch	This patch is added to get resolution, use case type, and FPS information from the HOST system and handle large files (> 4GB).

7	0008-Added-ioctl-to-support-different-formats.patch	This patch is added to get format and profile information from the PCIe Host application to the PCIe endpoint application.
8	0001-sdirxss-update-the-gt-clock-only-when-framerate-is-c.patch	This patch provides fractional frame support for SDI-Rx. To support fractional and integer frame rates in SDI, the GT clock should switch to 148.35 MHZ and 148.5 MHZ. In SDI-Rx, the GT clock will be updated only when there is a change in the frequency compared to the previous frequency.
9	0001-sdi-tx-Add-fractional-framerate-support-for-SDI-Tx.patch	This patch provides fractional frame rate support for the SDI-Tx by switching the clock to 148.35 MHZ. When there is a change in the clock, GT lanes and PICXO need to be reset.
10	0001-dma-mapping-avoid-calling-memset-unless-gfp-flag-set_20.1.patch	During pipeline initialization, the decoder allocates large raw DMA buffers. During allocation, by default kernel memset sets those buffers to 0 which adds an overhead on the initialization part causing frame drops. This patch disables the memset to decrease the allocation time thus reducing overall pipeline initialization time. This patch is only applicable for LLP2 PL DDR HDMI and LLP2 PL DDR SDI designs.