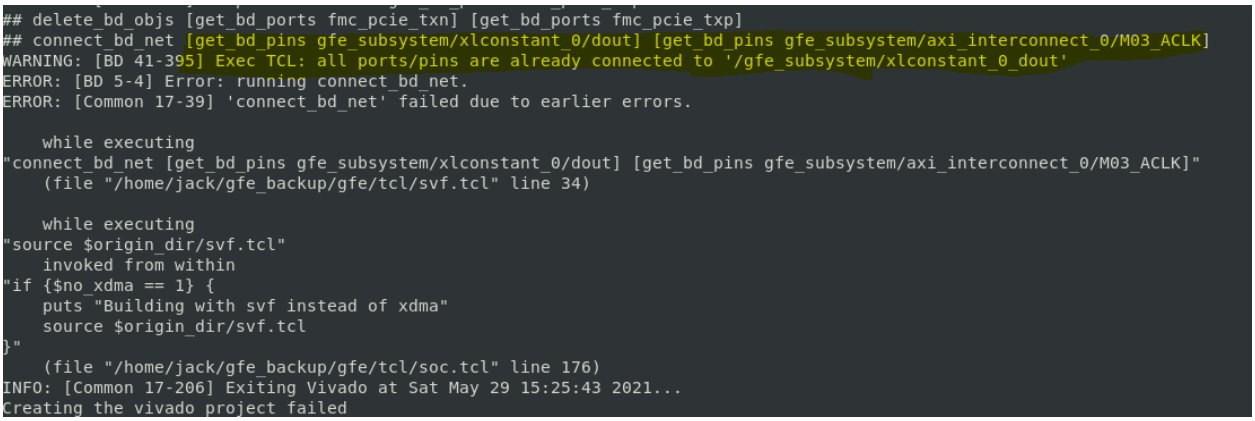
Appendix A - Issues

1.0 GFE Integration

Accomplishing changes to the Block Design using the Vivado IP integrator flow could be done, but generating a block design tcl files caused conflicts depending on load order. A processor is typically created with `./setup_soc_project <processor>`, causing the following calling order

```
setup_soc_project.sh
    tcl/soc.tcl
        tcl/soc_bd.tcl
        tcl/svf.tcl         #if no_xdma == 1
        tcl/soc_bd_video.tcl     #new, if en_frame_buffer == 1
```

Our generated block design file could be generated in vivado with the tcl command `write_bd_tcl soc_bd_video.tcl`. However when we created the initial project, it was with no_xdma=1. This caused the SVF to be added as part of the block diagram and got included within `soc_bd_video.tcl`. This issue in turn causes Vivado to throw critical warnings when creating a new project, as `soc_bd_video.tcl and svf.tcl` contain similar ports and nets.

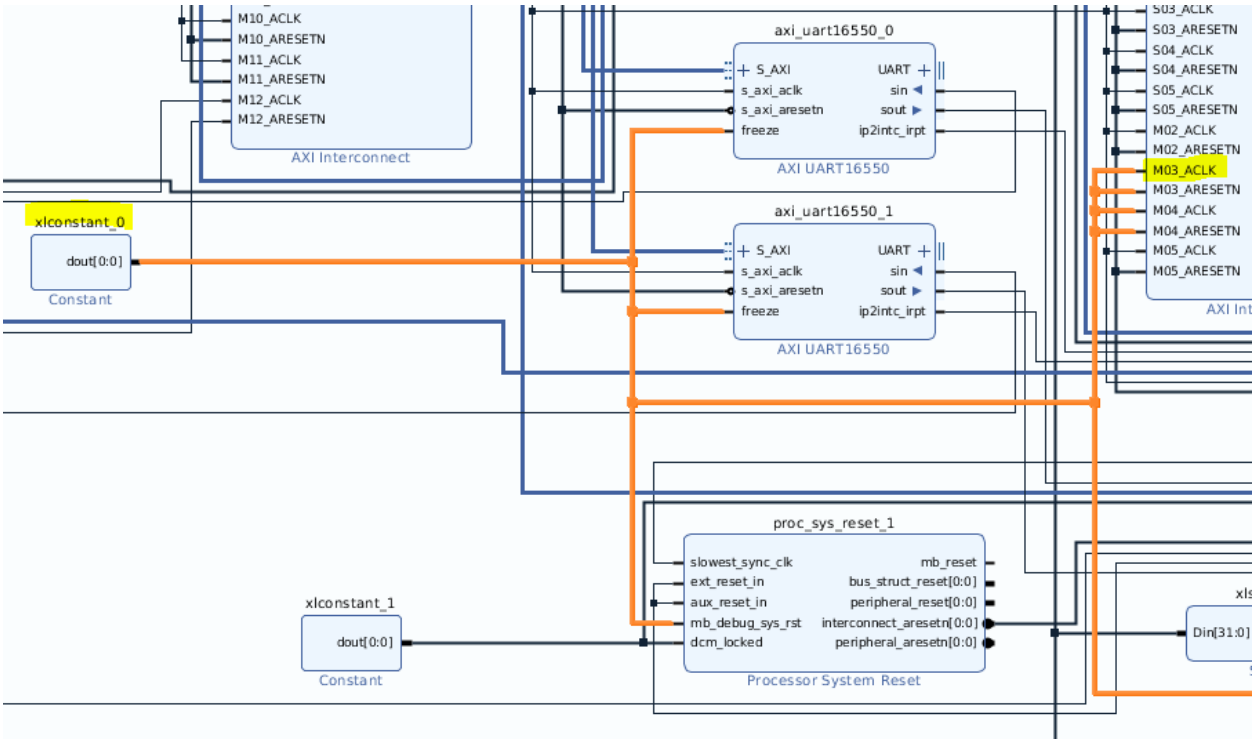


```
## delete_bd_objs [get_bd_ports fmc_pcie_txn] [get_bd_ports fmc_pcie_txp]
## connect_bd_net [get_bd_pins gfe_subsystem/xlconstant_0/dout] [get_bd_pins gfe_subsystem/axi_interconnect_0/M03_ACLK]
WARNING: [BD 41-395] Exec TCL: all ports/pins are already connected to '/gfe_subsystem/xlconstant_0_dout'
ERROR: [BD 5-4] Error: running connect_bd_net.
ERROR: [Common 17-39] 'connect_bd_net' failed due to earlier errors.

    while executing
"connect_bd_net [get_bd_pins gfe_subsystem/xlconstant_0/dout] [get_bd_pins gfe_subsystem/axi_interconnect_0/M03_ACLK]"
    (file "/home/jack/gfe_backup/gfe/tcl/svf.tcl" line 34)

    while executing
"source $origin_dir/svf.tcl"
    invoked from within
"if {$no_xdma == 1} {
    puts "Building with svf instead of xdma"
    source $origin_dir/svf.tcl
}"
    (file "/home/jack/gfe_backup/gfe/tcl/soc.tcl" line 176)
INFO: [Common 17-206] Exiting Vivado at Sat May 29 15:25:43 2021...
Creating the vivado project failed
```

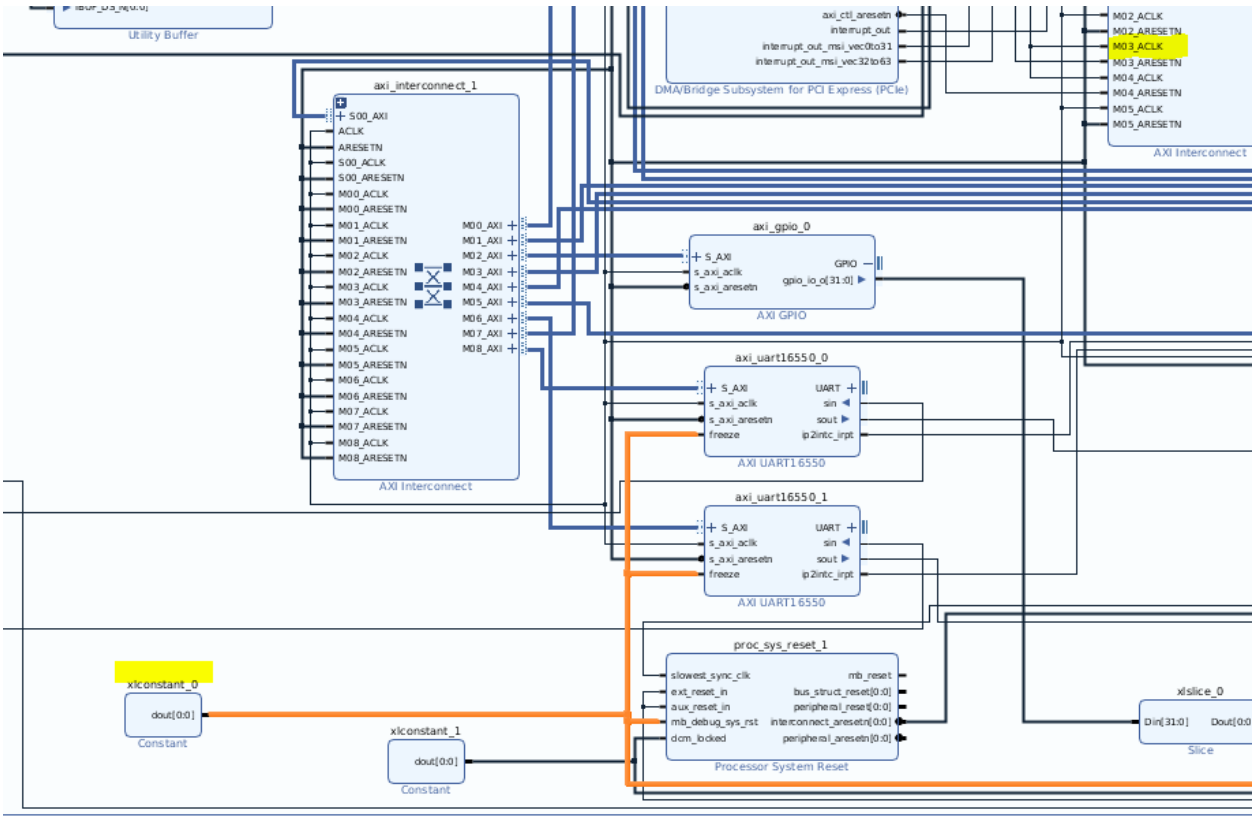
Error when generating new Vivado project with video output. [Link to conflicting lines in tcl/svf.tcl](1)

A fix would be to build the project with no_xdma=0, whenever en_frame_buffer=1. This will create the project with the frame buffer as well as the SVF within the block diagram. The optimal solution, given more time, would be to rebuild the frame buffer video output IP with a clean copy with no_xdma=0.

1 https://gitlab-ext.galois.com/ssith/gfe/-/blob/feature/video-output/tcl/svf.tcl#L34



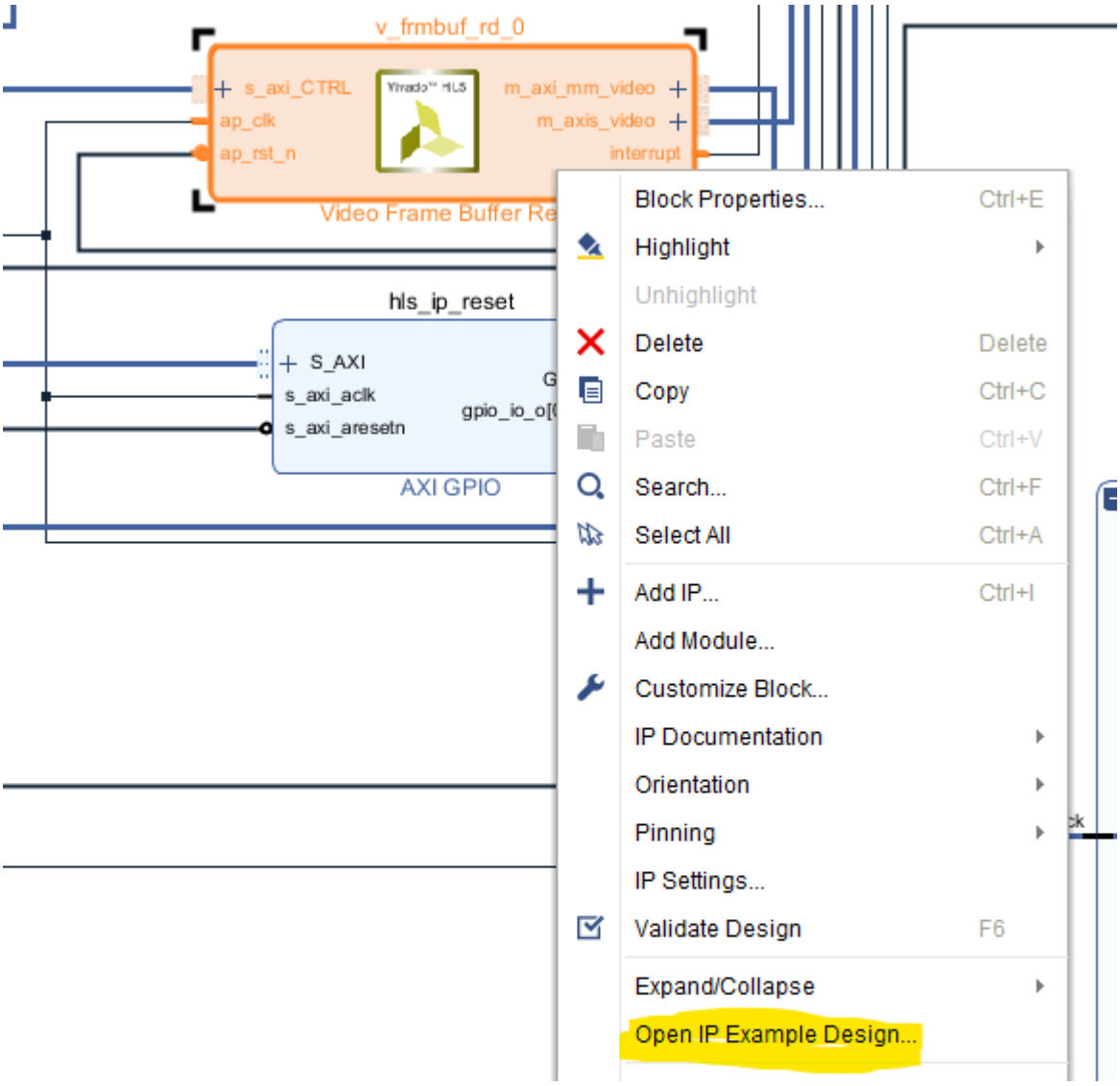
Result in Vivado IP Integrator



Intended configuration with SVF disabled

2.0 Driver and Xilinx

As part of implementing hardware changes, software changes (i.e. drivers) are expected to be done to use hardware. We chose our hardware IP to be the Xilinx Video Processing IP modules, specifically the Video Frame Buffer Read IP alongside the Video Timing Controller and the AXI4 Stream to Video Out. These were chosen because an example Vivado project for the Video Frame Buffer Read is provided (see image below) alongside drivers for the example[2].



Opening an example design of the Video Frame Buffer Read IP.

Hardware changes to the GFE subsystem would be based on this example. The idea would then be to load the provided drivers onto the VCU118, with a prewritten memory image. This was tested on an Arty A7 35T locally with a MicroBlaze soft microprocessor core — as provided

2 https://github.com/Xilinx/embeddedsw/tree/master/XilinxProcessorIPLib/drivers/v_frmbuf_rd/examples

in the example — and achieved successful results as seen in Figures 13 and 14. This driver utilizes

However, in porting this over to the GFE, using the Xilinx SDK to establish a connection and install this driver was not viable since the SDK required the Vivado GUI which runs incredibly slowly over a remote connection.

What the intended route should have been is to add these IP modules into the device tree in `bootrom/devicetree.dts`, build the relevant operating system according to the provided instructions in the GFE repository, and add any driver and/or userspace program over the established serial connection/console.

3.0 VCU118 and PMOD to DVI Board

As part of our initial test of the hardware, we took an RTL model from [an example repository for our PMOD to DVI adapter][3], generated a bitstream, and examined any issues that would show up on our local FPGA development boards and monitors. Upon examination, our tests appeared to work properly; expected test patterns were shown on our local monitors over a DVI-D (aka HDMI) cable.

Color palette test pattern from example Vivado project

When a bitstream for this project was generated and uploaded onto the VCU118, nothing appeared on the monitor outside of a notification which implied no valid signal was found. After some debugging and documentation, we discovered that this issue is due to an older version of the PMOD specification that the VCU118 was designed on[4]. This specification requires pullup

3 Developed by Kevin Hubbard from Black Mesa Labs in the iCEBreaker repository. https://github.com/icebreaker-fpga/icebreaker-examples/tree/master/dvi-12bit

4 https://forums.xilinx.com/t5/Xilinx-Evaluation-Boards/VCU118-Rev-2-0-PMOD0-U41-board-bug-not-fixed/td-p/940193

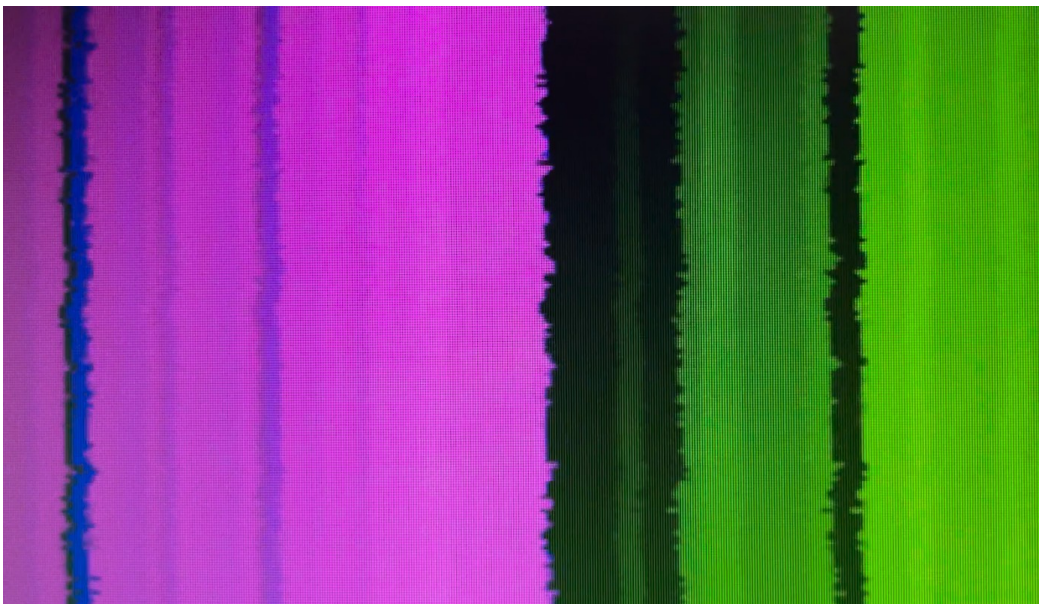
resistors to 3.3V to be on the daughter card instead of on the VCU118.

The solution to this problem, as was also described in the forum post, would be to add pull ups as part of the PMOD expansion board. After inquiring, we noticed this was done before on a schematic about a microSD card reader over PMOD interface and developed into a PCB[5].

Due to limited time for this project, our fix would be to enable video output on the VCU118 would be to hand solder pull ups via a PMOD 12-pin Test Point Header board, deliver it to the remote machine, and plug in the test point headers in between the jumper cables attached to the VCU118 and the PMOD DVI board. This setup can be seen in Figure 9 of the Final Report.

After enabling video output via a simple test, the VCU118 showed some fuzziness on the screen which was not evident on our local tests. While we're not sure what the root cause of this error is due to the lack of time for investigation, we suspect it's one of the following issues.

- Increased inductance due to jumper cables and/or the soldered PMOD pullups. This could be fixed by adding bypass capacitors in between 3.3V and GND.
- Timing control of this display test was done using a clocking wizard. Perhaps the clock is not clean.
- This was developed using an RTL mode. This may not be an issue with the AXI4-Stream to Video Out IP



Fuzziness on screen from display test on the monitor connected to the VCU118

4.0 Xilinx embeddedsw Frame Buffer Driver Linux Compilation

The driver we used for local development was one provided by Xilinx. While building on a

5 https://github.com/GaloisInc/BESSPIN-Voting-System-Demonstrator-2019/blob/master/hardware/sbb-shield/R4-TA1/bvs2019-r4-sch.pdf

standalone/bare-metal system works fine, there is an issue if building for a Linux system. This is due to a compile flag syntax error from within one of the header files for the driver for the Video Frame Buffer Read IP[6]

In `src/xv_frmbuf_rd.h`, there is a definition for if one is compiling for Linux on line 34 for some typical typedefs. However it appears that this #ifdef was improperly closed, as line 44 for a struct does not get defined when building for Linux. This struct is used throughout the driver. Therefore we believe the fix for this would be make sure that this struct is defined no matter the system one is compiling to.

```
32
33  /************************** Type Definitions *****************************/
34  #ifdef __linux__
35  typedef uint8_t u8;
36  typedef uint16_t u16;
37  typedef uint32_t u32;
38  #else
39
40  /**
41  * This typedef contains configuration information for the frame buffer read core
42  * Each core instance should have a configuration structure associated.
43  */
44  typedef struct {
45    u16 DeviceId;             /**< Unique ID of device */
46    UINTPTR BaseAddress;      /**< The base address of the core instance. */
47    u16 PixPerClk;            /**< Samples Per Clock */
48    u16 MaxWidth;             /**< Maximum columns supported by core instance */
49    u16 MaxHeight;            /**< Maximum rows supported by core instance */
50    u16 MaxDataWidth;         /**< Maximum Data width of each channel */
51    u16 AXIMMDataWidth;       /**< AXI-MM data width */
52    u16 AXIMMAddrWidth;       /**< AXI-MM address width */
53    u16 RGBX8En;              /**< RGBX8      support */
```

src/xv_frmbuf_rd.h

6 https://github.com/Xilinx/embeddedsw/tree/master/XilinxProcessorIPLib/drivers/v_frmbuf_rd

5.0 Vivado Environment Issues

Often during development, there were not enough system resources to run Vivado especially during memory intensive bitstream generation. In some instances, Vivado would cause a segfault due to using up too much memory[7]. This could be fixed by increasing the stack size (e.g. `vivado -stack 2000`), however a full solution would be to allocate more memory such as increasing the swap file size by several GB on an external SSD.

Additionally, the Vivado install alongside other dependencies took up large amounts of storage. An instance of a virtual machine running Debian, a virtual storage drive of 160 GB[8] was allocated to hold the installs for Vivado, RISC-V toolchain, and other GFE dependencies. While the gross storage of these files ended up being ~80 GB, extra storage was needed as part of the installation and download process, on top of any swap file usage.

Storage speeds were also a concern; some team members were still running HDDs as their main drive and slowed virtual machines to a crawl. Due to these reasons, in addition to memory capacity issues and failing drives, many team members purchased new hardware to upgrade their systems such as new SSDs and new memory modules.

Additional issues with Vivado occasionally came through errors being `IllegalArgumentException: Window must not be zero`. This issue is related to the version of Java that runs Vivado and a patch can be downloaded and applied using `vivado -jvm -patch-module=java.desktop=<path-to-jar>/AR72614.jar`[9]. Note that this patch is automatically applied to 2020.x or newer.

Other minor issues may occur while running Vivado or Vivado installer with sudo in which any GUI will not open and the command defaults to a command prompt due to X-forwarding issues. This can be fixed by running the following:

```
$ xhost +
Access control disabled, clients can connect from any host
# From here, one gui for Vivado or Vivado Installer can function
$ sudo vivado
```

7 https://www.xilinx.com/support/answers/64434.html
8 While 160 GB were allocated, the size of the virtual drive bloated to 346 GB on the host machine due to constant Vivado reinstalls and media expansion.
9 https://www.xilinx.com/support/answers/72614.html

6.0 SDK Development and Solution

As mentioned in the Integration section of the report, the Xilinx SDK was used for driver development and usage. The idea was that this allowed for easier implementation of a provided example driver.

Unfortunately, this pipeline relied heavily with both the Vivado GUI and the Xilinx SDK GUI. With the connection to the remote host making any X-forwarded application nearly impossible to use, this means that our standard pipeline could not be used to install drivers or modify memory maps via the SDK.

However, it is possible to connect to a hardware target over a remote connection and run a debug sessions using the SDK on the host machine[10]. While we did not replicate these steps due to a lack of time during our integration processes, we predict that these would be the steps needed if we went down this route.

10 https://www.xilinx.com/support/answers/64759.html

7.0 Linux and PetaLinux Device Tree

Our development process focused on progressing local development as much as possible. By running PetaLinux, we mimic the Linux environment we would have expected on the SoC. In particular, we needed to add our video output hardware to the device tree. We would base our process with an AR describing how to accomplish this on PetaLinux with a similar hardware setup[11].

Additionally, after adding the hardware device to the device tree, the video output hardware would need to be probed. Just like with device tree changes, an AR was found which provided references on what needed to be done to probe for our device[12].

Due to our local changes to our device tree, the following changes would be necessary to the device tree in `bootrom/devicetree.dts`. Items in brackets denote device specific values.

```
        v_frmbuf_rd@[IP Address]{
                reset-gpios = <&hls_ip_reset 0 0 1>;
                clocks = <&[Clock]>;
                clock-names = "[Clock Name]";
                memory-region = <&[Memory Controller]>;

        };
```

```
        v_tc@44a20000 {
                xlnx,pixels-per-clock = <1>;
                clocks = <&[Clock]>;
                clock-names = "[Clock Name]";
        };
```

11 https://forums.xilinx.com/t5/Video-and-Audio/PetaLinux-FrameBuffer-Read-Device-Tree-Generation/m-p/953370
12 https://forums.xilinx.com/t5/Embedded-Linux/Video-Frame-Buffer-Linux-Driver-not-Probing/td-p/964714

After setting up the device, changes are needed in the kernel to drive the device and generate a terminal output. An implementation of a kernel module was found, although ultimately not used for our tests and attempted demonstration, in another AR[13].

Additionally to set up a console terminal output, Linux has a specified pipeline via a DRM stack to deal with any video rendering. However, this would require Linux to have hardware access to the video encoder[14]. For our project however, the encoder is offboard on the PMOD to DVI board. Therefore a dummy encoder would be needed to finish the Linux pipeline and allow console output. We could also attempt to modify the DRM stack, however the AR also pointed out that the rendering manager itself is very difficult to understand. There was simply not enough development time to implement this feature, which were only needed for demonstration purposes.

13 https://forums.xilinx.com/t5/Embedded-Linux/Video-Frame-Buffer-Read-Linux-Test-Kernel-Module-Implementation/td-p/929905
14 https://forums.xilinx.com/t5/Embedded-Linux/Getting-simple-output-with-Xilinx-DRM-KMS-framework/td-p/892275