RISC-V Secure Hardware Video Output

Sponsored by Dan Zimmerman and Michal Podhradsky, Galois Inc.

Team 7

Ahmad Alothaimin, Hector Soto, Jack Chen, Ross Wegter, Ryan Nand

Advisor: Tom Schubert

June 9, 2021

Rev. 1.0.0

Table of Contents

Introduction

Executive Summary

At Portland State University, our team of five senior electrical and computer engineering

students were given a project by our sponsors from Galois Inc. This project involves Galois' custom System on Chip (SoC) integrated on an XCVU9P Field Programmable Gate Array (FPGA) on a VCU118 development board. An FPGA is hardware that can be programmed and our goal in this project is to code changes to the FPGA to support video output on this SoC[1].

The VCU118 development board does not natively support video output. Our sponsors suggested either an I/O expansion adapter via the Peripheral Module (PMOD)[6] interface or a standard graphics card connected to the board's PCIe expansion. We decided to choose a PMOD to Digital Video Interface (DVI) adaptor, implemented as an HDMI port[8], as it can support higher resolutions of video output. Our choice influenced the hardware development path for the SoC. We also needed to modify or create a driver on the SoC to interface with the hardware. To verify our solution works, we must display an image or live Linux terminal from the VCU118.

The project was completely remote due to COVID-19. Our sponsors could not lend us the FPGA due to contractual obligations from their overall project[2]. Therefore, our approach was to rapidly develop on local FPGAs and integrate our display systems to the SoC on the VCU118 via a remote host. Any physical alterations to the VCU118 is accomplished by our sponsors.

The project was a partial success. We successfully developed a video display system through modifications to the SoC and added physical hardware to output video onto a monitor. However, due to complications with integration processes to the FPGA, we were unable to demonstrate a working Linux terminal on a monitor.

Problem Background

This project was assigned to Galois by their contractor, the Defense Advanced Research Projects Agency (DARPA) of the United States Department of Defense (DoD), as part of the

System Security Integration Through Hardware and Firmware (SSITH) program. Galois received Government Furnished Equipment (GFE) as part of its contract in the form of the VCU118 FPGA development board. Galois implemented an SoC on this FPGA for the SSITH program. Currently, ths SoC only communicates with a serial console, and having a display connected directly on the VCU118 would be more convincing for demonstration applications of the novel security features within this SoC.

Need Statement

Our sponsors prefered video output over serial communication. Our team needed to implement video output capabilities to the VCU118 FPGA development board. This can be accomplished by adding external attachments to the VCU118's PMOD connectors or utilizing a graphics card via a PCIe expansion. Regardless of the method chosen, changes to this FPGA came in the form of changes to the Vivado block design to allow video output through either method. Once that was done, we needed to implement a Linux terminal or any kind of dynamic output by modifying or creating a Linux driver for the FPGA.

Objective Statement

The objective for this capstone project is to research and develop video output capability onto the VCU118 FPGA. This video capability would accept information (some examples may be a frame, text, or an image) to be displayed from the Linux kernel being run on the SoC. Then the video capability will take that information and display it onto a standard monitor by outputting a valid video signal. In the end, the VCU118 board must be able to connect to a screen showing an image or a live terminal window.

Requirements

The following table is a list of requirements created to identify the goals of this project. These requirements were based on a few relevant specifications given by Galois. Those being the VCU118's CPU clock rate of around 100 MHz, its ability to run Linux + FreeBSD, and suggested solutions utilizing PMOD or PCIe connections. Deliverables based on these requirements can be seen in the Results section.

Requirements	
Functionality	SoC's video hardware can control video output through a connection (PMOD or PCIe + graphics card) to a screen. Driver can drive hardware to display video on a screen.
Performance	Video is clear enough that applications such as a console are easily interpretable. Video output has reasonable frame rate or responsiveness when changes to display are requested.
Reliability	SoC's video hardware processes video with no errors so that it's correctly displayed on a screen. Driver consistently outputs given video data to the screen when requested.

Our Approach

As a requirement due to stay at home orders during the COVID-19 pandemic, this entire project must be done remotely. Therefore we approached this project with as much local development as possible on any FPGAs that we had available. Access to the VCU118 FPGA development board was provided by the sponsors over a remote host. However the usability for

the Graphical User Interface (GUI) elements of Vivado was non-existent due to network delay and the set up process for this remote host took a significant amount of time for the sponsors to set up. Therefore our plan in the beginning would be to research as much as possible about video signal generation and build general enough solutions that can be tested locally. The following is a simplified version of this schedule.

Timeline	Process
December - January	Research and project proposal
February - April	Begin prototyping
April - May	Integration
May - June	Execute test plan
Mid May - June	Documentation and final report

During the prototyping phase, we had split into two main development groups. One for creating hardware changes via HDLs and Xilinx IP, the other for software driver development. For full details of the schedule, please refer to the schedule found in the proposal or ProjectLibre XML file[1]. For roles and responsibilities, please refer to the Contributions section.

1 Proposal, pg. 10-11 https://github.com/jackchen0226/ECE-Capstone-proj-7/blob/main/Project%20Design%20Specification%20and%20Timeline.pdf or https://github.com/jackchen0226/ECE-Capstone-proj-7/blob/main/Schedule.xlsx

Design

Overview

Our design process was focused into two different aspects for the SoC. Hardware changes to the XCVU9P FPGA on the VCU118, and software changes via kernel driver development to support the new hardware.

For hardware changes, our initial design focuses on building a subsystem for video output to attach to the main SoC (hereby referred to as the GFE subsystem). This subsystem takes inputs as an image or frame loaded to the onboard DDR4 memory and outputs a requisite video signal to display the input image or frame onto a monitor. Software changes involve creating the driver which writes to memory and communicates with the hardware. All changes would be added as part of the main GFE subsystem, and can be generated during compile time via a flag.

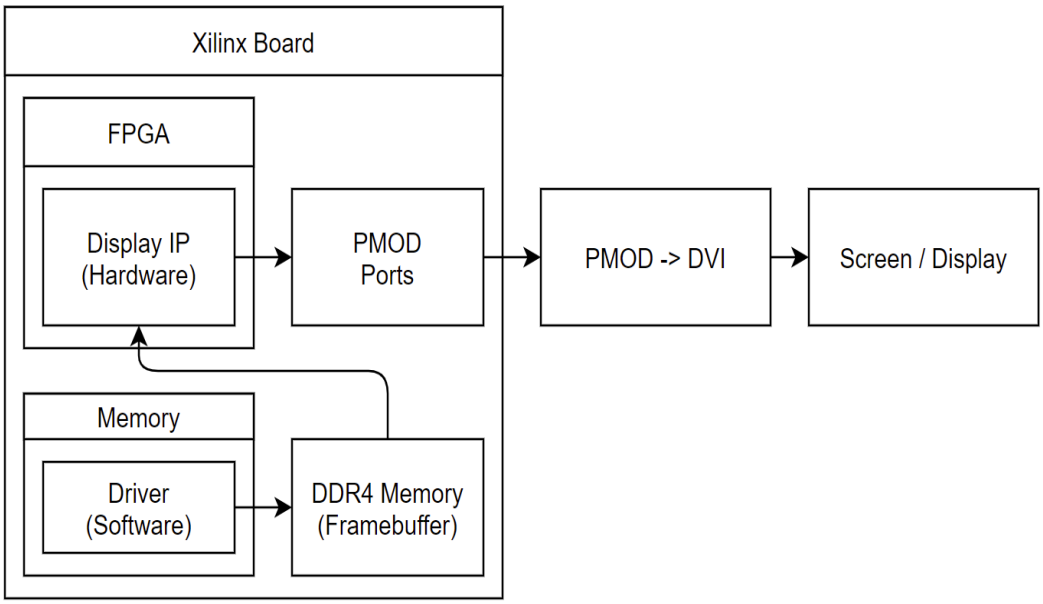

Figure 1: Overview Level 0 Diagram

While the VCU118 development board has significant I/O options, interfaces for video output such as VGA connectors or HDMI ports are not present[13]. The VCU118 does have a pair of PMOD interfaces, one as a right angle female receptacle and the other as a set of male pin headers, seen in Figure 2. These headers can be used in conjunction with a PMOD DVI daughterboard to fully enable video output and generate a DVI-D signal — otherwise known as HDMI signal — over an HDMI connector.

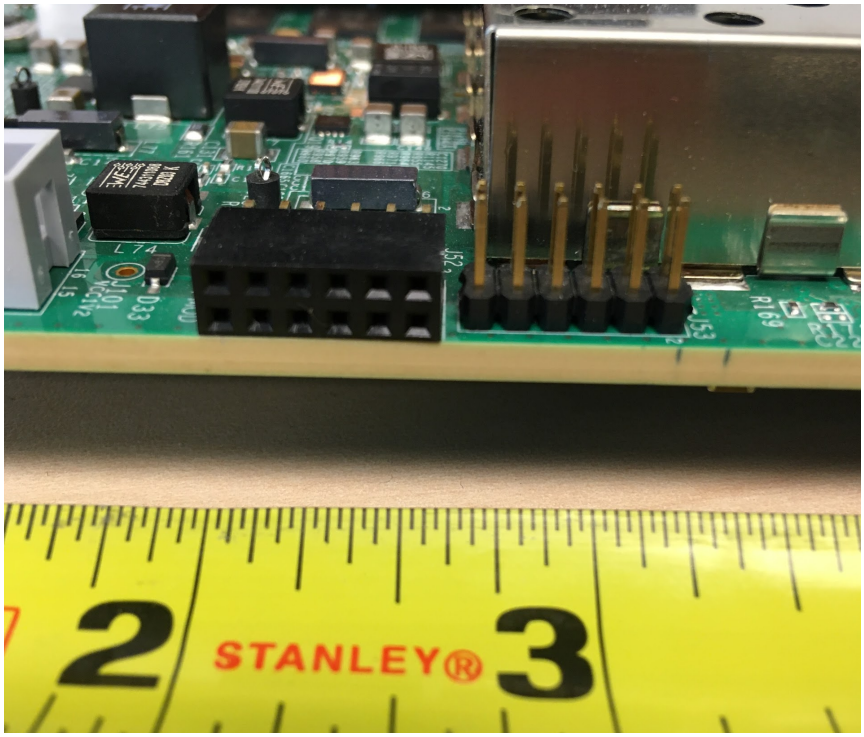

Figure 2: PMOD headers on the VCU118

PMOD to DVI board

The chosen daughterboard is the “PMOD Digital Video Interface, 12 bits per pixel” manufactured and sold by 1BitSquared. This board is based off of a previous design by Black

Mesa Labs[8] which consists of 2 PMOD connectors based on the Digilent PMOD specification[6]. For this board, these PMOD connectors accept a 12 bit RGB color VGA signal (4 bits resolution per color) and horizontal and vertical sync pulses, typical as part of the VGA specification[12]. Other pins atypical of a standard VGA interface — though some are present in specifications such as HDMI[14] — are also used, being an active high signal when the screen is drawing pixels and a pixel clock. All signals are traced to an onboard TFP410 to convert to an HDMI signal via Transition-Minimized Differential Signal[9]. Figure 4 is the schematic of this PMOD to DVI board, Figure 5 is a representation of the connections needed on the PMOD connections.

Since the PMOD pins require full VGA output, this means horizontal and vertical sync (labelled “hs” and “vs” in Figure 5) need to be generated. Additional signals are required, being “de” which is set high whenever active pixel data is being written and low during the blanking interval, as well as “ck” being the pixel clock. Figure 6 shows a more in-depth timing diagram for describing the active region and the status of the horizontal and vertical sync.

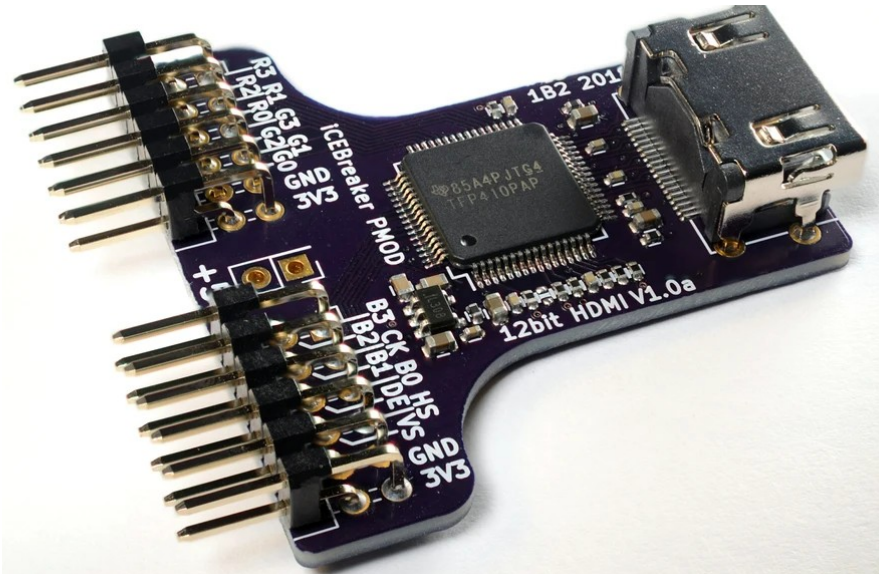

Figure 3: Image of the PMOD Digital Video Interface (PMOD to DVI), 12 bits per pixel daughterboard by 1BitSquared

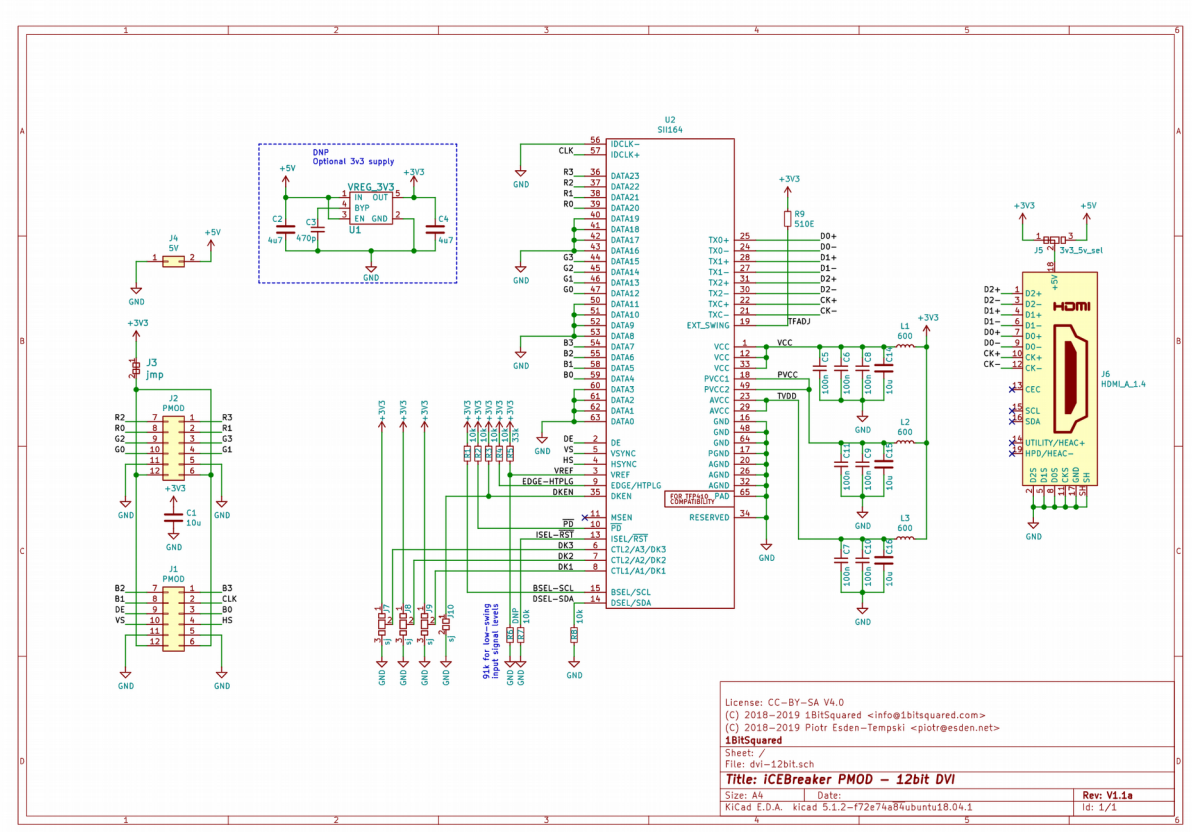

Figure 4: Schematic of the 12 bit PMOD to DVI board by 1BitSquared from their icebreaker-pmod repository[7]. Link here

```
-- 12b Module - Facing module pins
--       ------------------------------      ------------------------------
--      | 1-R3 2-R1 3-G3  4-G1 GND 3V |      | 1-B3 2-ck 3-B0  4-HS GND 3V |
--      | 7-R2 8-R0 9-G2 10-G0 GND 3V |      | 7-B2 8-B1 9-DE 10-VS GND 3V |
--   ___|_____________________________|______|_____________________________|__
-- |        BML HDMI 12b color PMOD board                                    |
--  ------------------------------------------------------------------------
--        pmod_*_*<1> =  r[3]                  pmod_*_*<1> =  b[3]
--        pmod_*_*<2> =  r[1]                  pmod_*_*<2> =  ck
--        pmod_*_*<3> =  g[3]                  pmod_*_*<3> =  b[0]
--        pmod_*_*<4> =  g[1]                  pmod_*_*<4> =  hs
--        pmod_*_*<7> =  r[2]                  pmod_*_*<7> =  b[2]
--        pmod_*_*<8> =  r[0]                  pmod_*_*<8> =  b[1]
--        pmod_*_*<9> =  g[2]                  pmod_*_*<9> =  de
--        pmod_*_*<10> = g[0]                  pmod_*_*<10> = vs
```

Figure 5: Modified pinout of the 12 bit PMOD DVI board from the icebreaker examples repository[10]. PMOD pins are numbered based on PMOD spec, top row pins 1-4 are used for GPIO, bottom row pins 7-10 are for GPIO. Pins 5 and 11 are ground, 6 and 12 are 3.3V

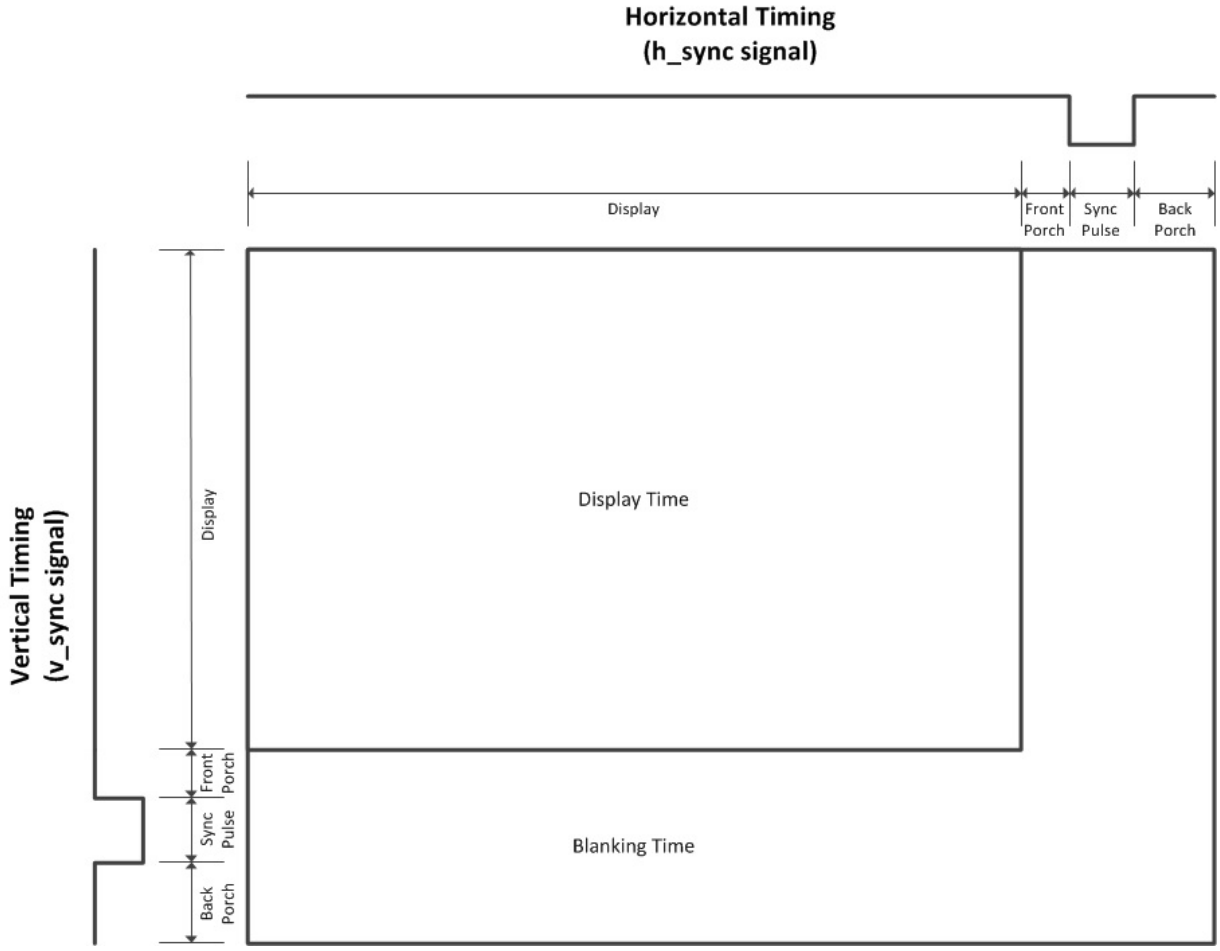

Figure 6: VGA signal timing diagram from DigiKey eewiki[11].

Drivers and Software

With the necessary hardware changes via Xilinx IPs designed, relevant drivers would need to be written to allow an operating system to communicate with the hardware, as well as a user program to draw images or write to a section in memory. We based our driver design from an example driver provided by Xilinx for their Video Frame Buffer Read IP[2]. In this example driver, the expected environment is in a bare metal or standalone system environment, ideally using the MicroBlaze soft microprocessor core. While there exists flags for a Linux build, there is currently a preprocessor compilation bug which prevents one from doing so. For more information, refer to Appendix A, Section 4.0.

The main reason we designed based on this driver is due to our use of the example Vivado project given by Xilinx. It should be noted that the driver is also based on the same Video Frame Buffer Read example project and to use this driver would require running the Xilinx SDK[3] alongside a generated bitstream of said project. Within the SDK, a simple command using the Xilinx Software Command Line Tool can be given to load data to a memory location, specifically the starting address of the Video Frame Buffer Read. This load includes writing from a provided file, such as an image file.

2 Acquired from Xilinx's embeddedsw Github repository in `drivers/v_frmbuf_rd/examples`, https://github.com/Xilinx/embeddedsw/tree/master/XilinxProcessorIPLib/drivers/v_frmbuf_rd/examples[17].
3 For Vivado 2020 or newer, the Xilinx SDK is replaced by the Vitis software platform.

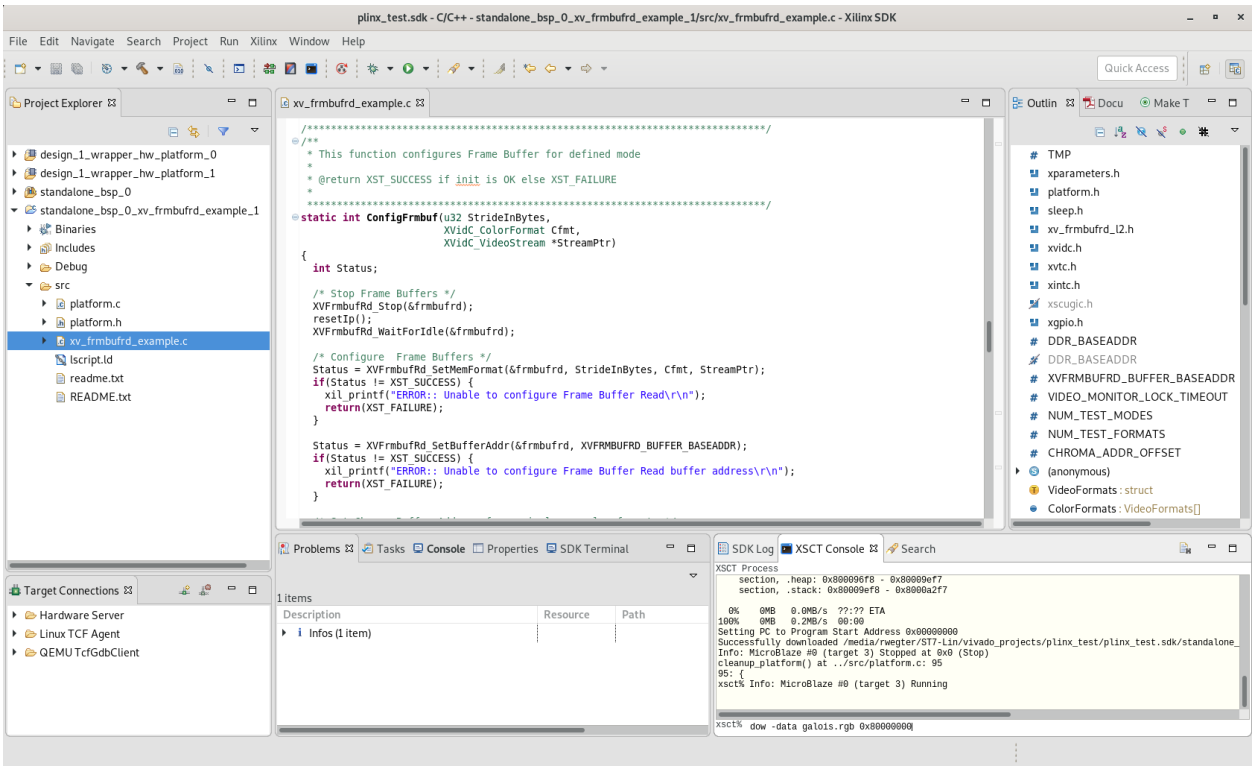

Figure 7: Image of Xilinx SDK in use

While this driver design was the one used, an experimental driver was also developed using the Linux kernel[4]. The inspiration for this driver was due to our knowledge of being able to run a Debian distribution, albeit compiled to a RISC-V architecture, on the SoC. Additionally, we could install a version of Linux for embedded systems using Xilinx PetaLinux which allows for a closer emulation of the full SoC. Considering the hardware should be as simple as reading from a memory location, driver development should be a simple memory mapping process and initialization sequence. Ultimately this design was not used as there was not enough time to integrate and run this driver on a VCU118 and spend time debugging and testing the software. This driver was however tested on a host machine, with a simple userspace program using SDL2 to display video output.

4 Source within group repository: https://github.com/jackchen0226/ECE-Capstone-proj-7/tree/dev/driver

Implementation

Physical Hardware and PMOD

With only remote access to the VCU118, we would not have easy physical access to the FPGA development board. And the PMOD DVI board must be attached to the VCU118 to allow for video output. This is where our sponsors attached the daughterboard to the development board, done so using female-female jumpers for the PMOD male pin headers and male-female jumpers for the PMOD right angle female receptacle as seen in Figure 8.

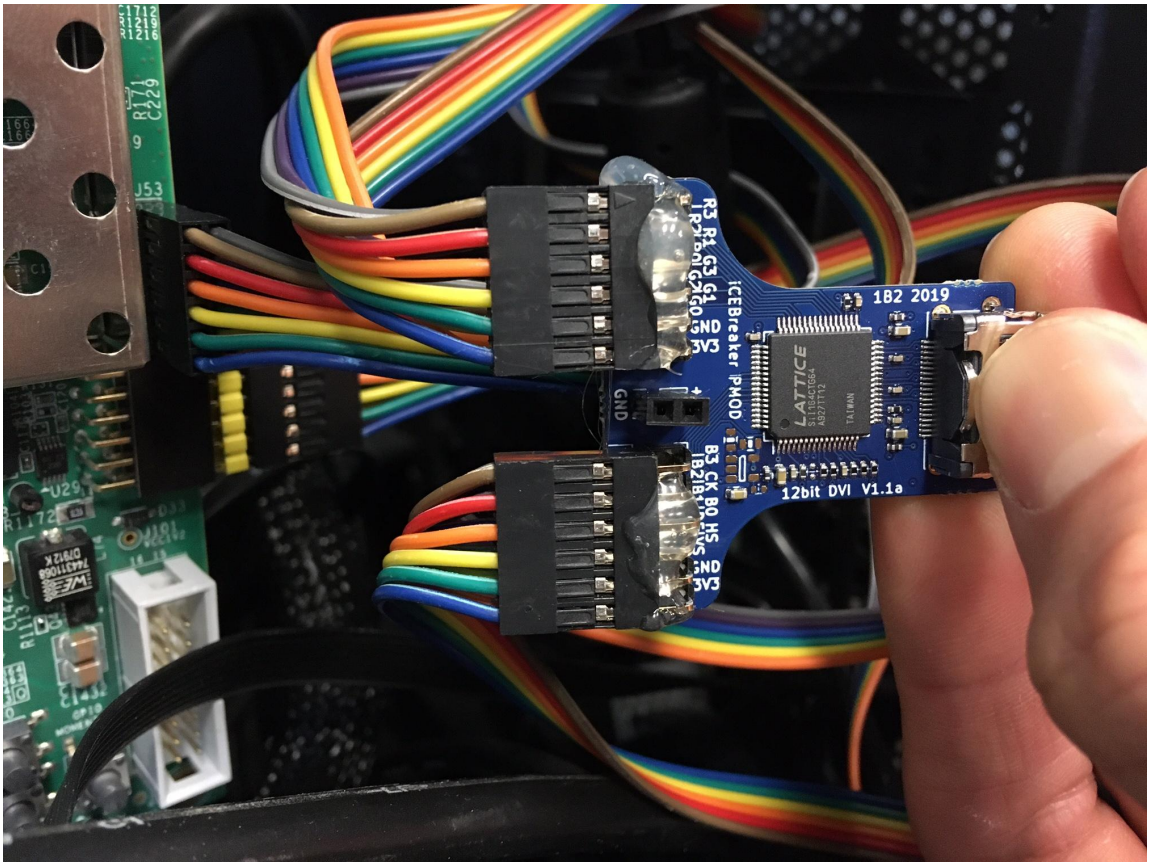

Figure 8: Jumper cables from the VCU118 PMOD headers (left) to the PMOD DVI board (right)

Additionally, due to the version of the PMOD specification that the VCU118 was designed around[6][16], there is also a requirement for pullup resistors to 3.3V on all PMOD pins. For our implementation, a fix using test point header boards and soldered on 10k ohm resistors were used and delivered to our sponsors to attach to the VCU118. The resulting product can be seen in Figure 9.

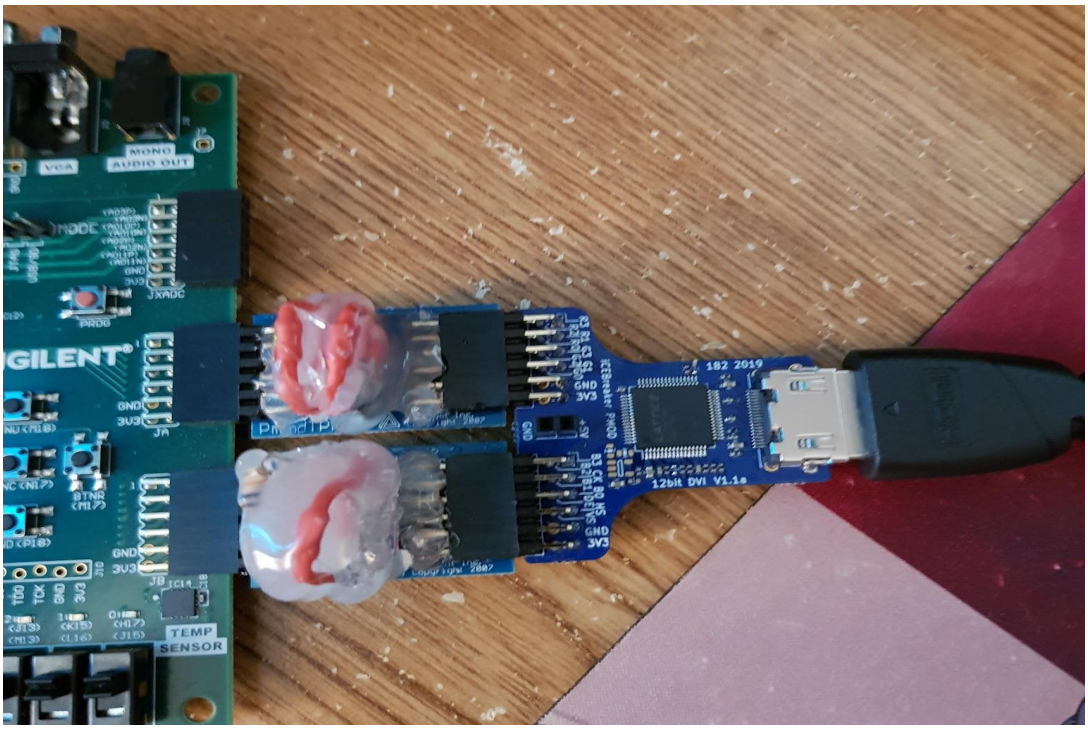

Figure 9: Temporary fix of pullup resistors tied to 3.3V for all PMOD GPIO pins

Xilinx IP and Vivado project

Implementing hardware changes required modifications to the Vivado project holding the GFE subsystem within the FPGA. This is where our subsystem for video output would reside. Our final design primarily utilizes the Xilinx IP blocks Video Frame Buffer Read, Video Timing Controller, and AXI-4 Stream to Video Out as seen in Figure 10. This design is based off of the example design for the Video Frame Buffer Read IP provided by Xilinx. Steps to generate this example are in Appendix A, Section 2.0

This Video Frame Buffer Read IP can request memory via AXI-4 stream from it's m_axi_m_video interface connected to the DDR4 Memory Interface Generator; a block within the GFE subsystem. This requested memory would be from a configurable starting address using Vivado's Address Editor, for our case it would be 0x8000_0000, which contains the pixel data of the frame. This data then gets output as an AXI-4 stream. The Video Timing Controller outputs an interface of timing constraints including horizontal and vertical syncs, blanking periods, and active signals. Both outputs of the Video Frame Buffer Read and the Video Timing Controller get sent to the AXI4-Stream to Video Out to get a VGA signal.

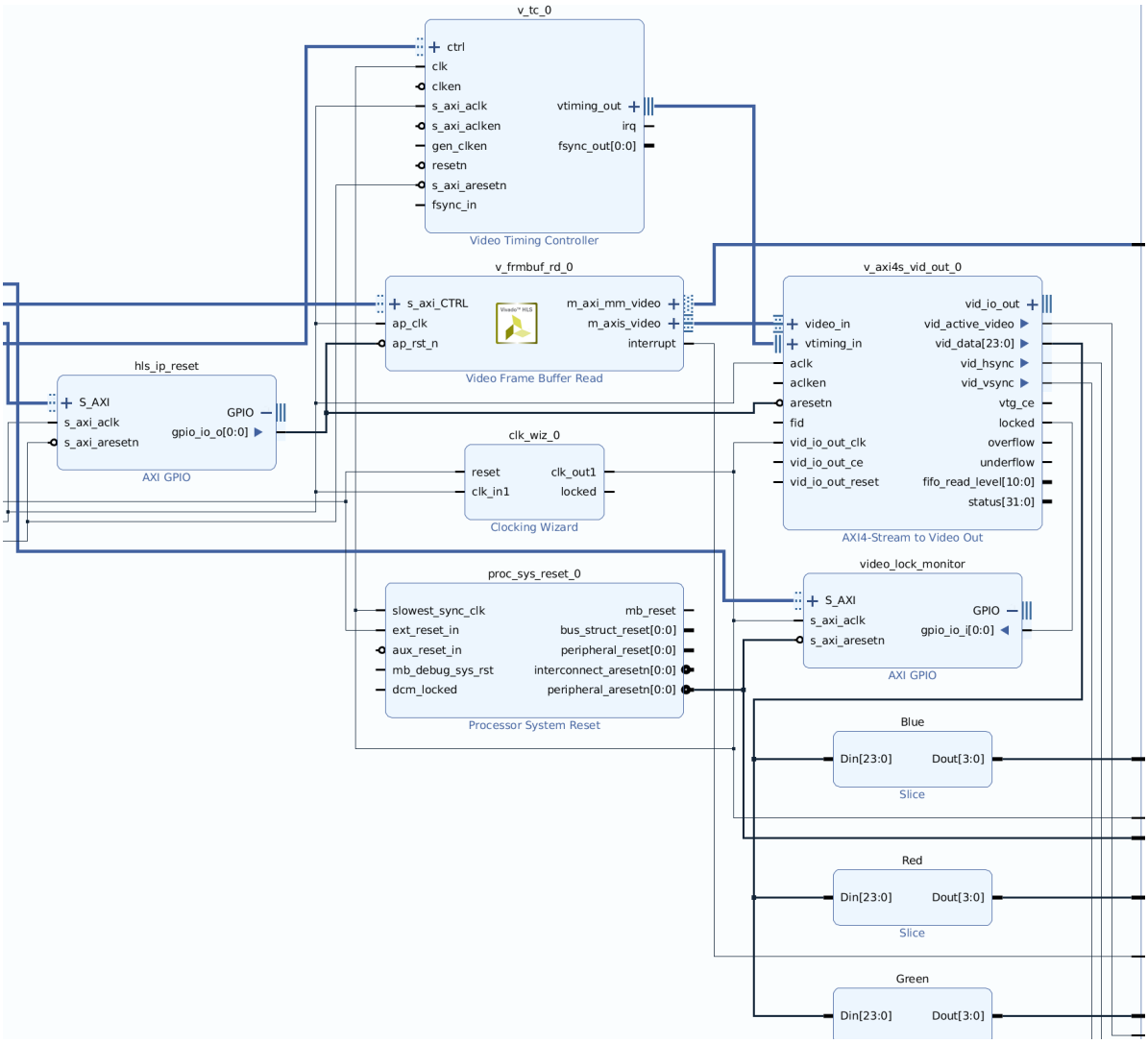

Figure 10: Video Frame Buffer Read subsystem, part of the GFE subsystem.

Leading up to this design was done in stages due to the remote nature of our development process. First a simple test of the PMOD to DVI board was done by converting example RTL from the icebreaker examples repository to a block design within Vivado[10]. This test is then loaded onto our local FPGA development boards for rapid prototyping and testing. This design can be seen in Figure 11 as a VGA core taken from the icebreaker examples repository which generates 24 bit wide pixel color data — 4 bits of which are used for each color — as well as the

horizontal sync, vertical sync, and active region signals. For inputs, this core takes an enable for 3 bit color, an enable for the display mode for either demos or a simple test pattern, a pixel clock, and a seed for a random number generator.

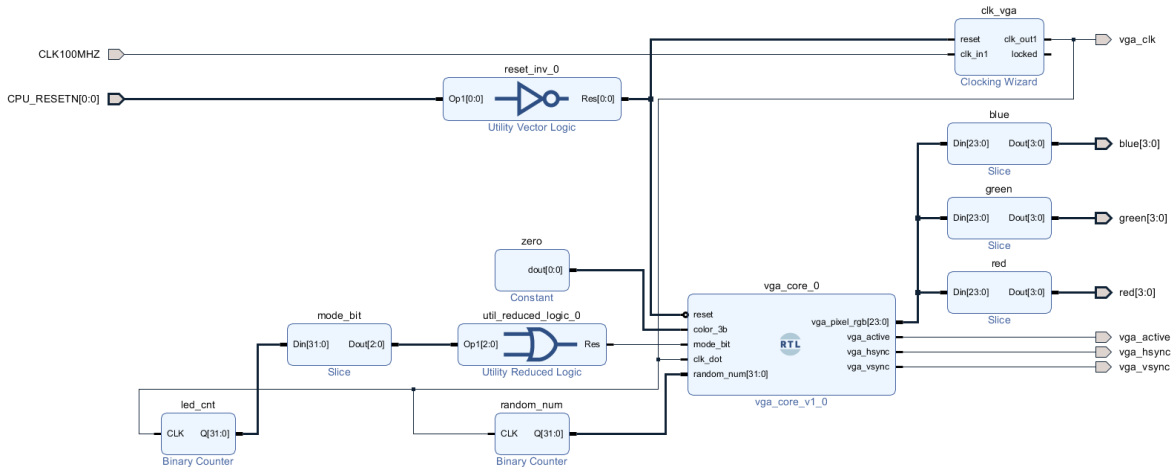

Figure 11: Display test using simple RTL for VGA output to observe the behavior of the PMOD to DVI board and requisite output signals.

From there, to design for integration to the SoC, we created a version using Xilinx IP and a MicroBlaze soft microprocessor core instead of using RTL. The IPs used are the Video Test Pattern Generator, the Video Timing Controller, and an AXI4-Stream to video out. In short, the Video Test Pattern Generator outputs an AXI-4 stream of the test pattern. The AXI4-Stream to Video Out takes both of these outputs and generates the requisite VGA signals. Figure 12 shows these IPs and their attached connections.

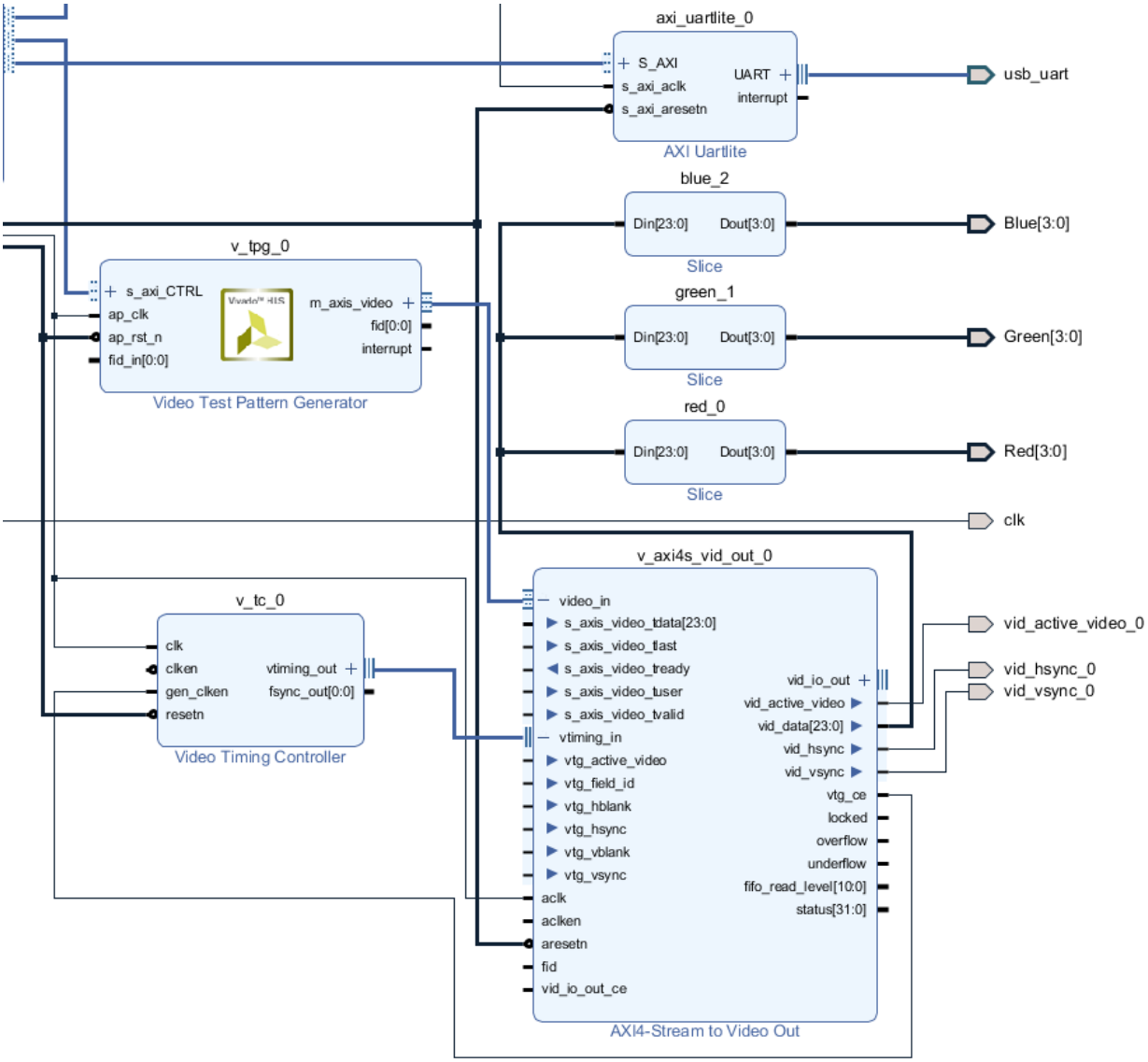

Figure 12: Video subsystem using Xilinx IPs

Xilinx SDK and Driver

This driver ran through a straightforward process to set up the IPs, first resetting all IP blocks, then initializing interrupts. Afterwards a sanity check occurs to ensure that the video is locked. Then color format and timing parameters are set if defaults are not used. After which the Video Timing Controller is configured if defaults were not used. Lastly the Video Frame Buffer Read IP gets configured, sets the buffer address, enables interrupt, and starts the frame buffer.

To use this driver, a bitstream of the Vivado project must be generated and used as part of the Xilinx SDK. From there, one can load data into an address in memory with pixel data or an image via the Xilinx Software Command Line Tool (XSCT) with the `command dow -data <file> <address>`. When the example driver is built and run on the local FPGA, an image will display on the connected monitor and seen in Figures 13 and 14.

Figure 13: An image loaded into memory of the Galois Inc. logo.

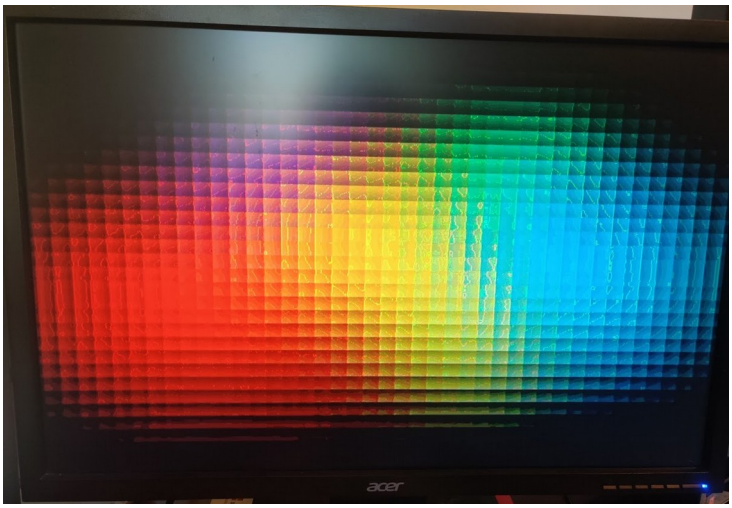

Figure 14: An image of a variety of colors to observe the video's color space

While this method works on our local development FPGAs, the implementation of this driver to the VCU118 could not be done reliably with the Xilinx SDK GUI due to the remote connection available. The alternative would have been to rely solely on using the XSCT and issuing commands to load, compile, and run the driver.

Operating Systems

To more closely mimic the SoC's environment, we attempt to run Linux on our local FPGAs and test the experimental driver. This was done using Xilinx PetaLinux Tools via an embedded Linux Software Development Kit (SDK) targeting FPGA-based SoC designs. The tool contains a Yocto Extensible SDK (eSDK), XSCT toolchains, and PetaLinux Command Line Interface (CLI) tools. Various Linux configuration tools are also provided to customize and generate bootloaders, Linux kernel, file system, and a variety of other configurations. The idea would be to include the experimental driver within Linux, load a userspace program, and see dynamic output on a screen. Unfortunately, due to time constraints, we were unable to set this up fully locally and instead focused efforts in running any OS on the SoC.

The operating systems that the SoC supports are the Debian distribution of Linux, FreeRTOS[15], and a software suite which interfaces with Unix utilities known as BusyBox. All of these OSes are compiled to RISC-V as a memory image which can be loaded onto the VCU118 over a serial connection. One can interact with the OS via a serial terminal from the host machine. Our approach was to build the Debian distribution and use either our experimental driver or the example Xilinx driver as both support Linux. Unfortunately, Debian could not be built on our local development environment, and the end result would have been to switch operating systems such as FreeRTOS. Full details of which are described in the Post-Mortem section.

Outcomes

Test Plan

Due to the current circumstances of having to work remotely, the majority of the testing would best be done on local developments boards. This in turn meant that all integration testing would be done on the VCU118, which would reveal if the bare minimum requirements were being met. Such as, video output of an image being displayable on a standard monitor using the remote VCU118 build.

On the local boards, however, the test plan consisted of several tests which fall under the following categories: module testing, functional testing, and acceptance testing. Detailed test cases were planned and written out for the display IP block test and the DVI PMOD functional test. The purpose of the display IP block test was to verify that the IP block can generate a valid VGA signal. The purpose of the DVI PMOD functional test was to measure the output qualitatively by varying the input. Steps to these test cases as well as expected results can be found in the test case tables found within the project test plan document.

Please refer to the test plan document for more detail on the tests conducted in this project. Information such as personnel, equipment, and objectives can be found within the document. The document can be found within our group repository[5].

5 https://github.com/jackchen0226/ECE-Capstone-proj-7/blob/main/Project%20Test%20Plan.pdf

Results

The results of this project were mixed. While the goal of creating a subsystem which was properly integrated into the main SoC was not achieved, we were able to demonstrate a simplified version of our video output on local boards with a soft microprocessor core. Ultimately the deliverables that were achieved are the following:

Requirements	Outcome
Must	
Develop HDL to output video to a screen.	Accomplished
Change the firmware on the SoC on the VCU118 FPGA to output an image to a monitor.	Partially accomplished. Hardware changes were implemented, allowing hardwired image output. Software for dynamic images did not get fully implemented.
Push any development and changes to the project onto the feature/video-output branch in the GFE repository.	Accomplished
Document our development process.	Accomplished
Requirements	**Outcome**
Should	
Display a terminal from the Linux kernel over	Did not get past integration and testing to

our video output capability.	work on this.
Have weekly meetings to go over the direction of the project and address issues.	Accomplished
May	
Display a desktop environment.	Did not get past integration and testing to work on this.
Implement multiple video output methods (VGA, DVI-A, DVI-D, PCIe to GPU, etc.)	Did not get past integration and testing to work on this.
Have dynamic resolution for the video output.	Did not get past integration and testing to work on this.

Post Mortem

With most projects, unexpected issues will inevitably crop out. Some of the most significant issues and design dead-ends that we encountered were spending time troubleshooting workarounds for local development and testing due to the remote nature of this project, issues in building the provided repository of the SoC for integration, and unreliable specifications and dependencies due to an atypical development environment.

Early on in the project, it was decided that as much of the development that could be done locally on individual FPGA boards should be done so. This was due to several reasons, including the necessity to work remotely due to the COVID-19 pandemic, but a local development platform with some necessary hardware allows much significantly faster workflow for programming and unit testing. For example, testing the PMOD to DVI board is done significantly faster if it is on hand, allowing for any HDL changes when the unit test fails. Having rapid feedback using debugging tools on hand allowed for the development of most of this project.

Switching to this development method does have significant ramifications, particularly the main SoC project is simply too large to fit in our local FPGAs. In compromise, we developed using Xilinx IPs and a 3rd party microprocessor core in the hopes that this implementation plan is generic enough to be integrated in the main SoC project on the VCU118. We also attempted to run PetaLinux, an embedded system version of Linux, on this microprocessor to emulate the RISC-V Debian Linux environment found on the SoC. One of the main issues with this method is we spent a significant amount of time setting up this local development environment with overconfidence with a generic, adaptable solution. This, alongside other issues, caused there to

be less time spent for integration than expected.

Additionally, a significant amount of our device driver experience is based off of Linux. The SoC project can run RISC-V versions of the Debian distribution of Linux which is familiar to us, as well as the less familiar FreeRTOS operating systems[15]. Our sponsors at Galois did briefly mention that the RISC-V Debian Linux build requires some configuration to properly build early on in development. But this issue was made more evident to us late into integration of our local work to the main SoC project through many build errors. The root cause of which, according to our sponsors, was due to the various dependencies that this RISC-V fork of Debian required being updated and incompatible with other dependencies. Resolving this issue would have been an incredible hassle as one would freeze certain dependencies to get previous versions. Ultimately the workaround had to be to produce our device driver in FreeRTOS instead. Since we had also spent the bulk of our investigation and development of device drivers in the Linux kernel, we discovered this issue late in development and could not develop FreeRTOS drivers in time for a demonstration. In hindsight, we should have properly investigated our sponsor's Linux development environment to catch this issue in the build process of the RISC-V based Debian Linux kernel earlier in this project. This would provide us more time to learn FreeRTOS and develop relevant drivers for that operating system.

Another significant issue which caused delays were rooted in outdated specifications for some of the hardware and software dependencies used. One in particular were the VCU118 PMOD specifications in which this development board was created on a version of the PMOD specification which stated that any external daughterboard should have pullup resistors to 3.3V[16]. This caused the default logic high voltage to be roughly 1.2V or 1.8V, and in turn unable to drive the PMOD DVI board. The solution would be to attach additional headers with

the necessary pullups, in our case with 10k ohm resistors, in between the PMOD headers and the PMOD DVI board. This caused about 2 weeks spent debugging, establishing tests, and creating and delivering a fix. While objectively this was not a significant amount of time spent on a fix, this led to less time for integration and testing which led to a non-working driver and lack of demonstration.

Conclusion

In conclusion, we developed in a remote environment, creating hardware and software changes to our FPGAs for local testing and debugging of dynamic video output. These changes would then be integrated into Galois' SoC project via a VCU118 FPGA development board which we only had access to over a remote host. By having the SoC support video output, Galois could then further demonstrate the capabilities of their novel computer security features as part of their overarching SSITH project. While we were not completely successful in creating a working demonstration due to a lack of working drivers, we were still able to implement the necessary hardware changes to allow for the SoC to output some form of video.

Contributions and Lessons Learned

Name	Contributions	Lessons Learned
Ahmad	Local unit testing, remote testing, schematic design, trello board management, sponsor communication	Have better team communication, and have a better approach to this project. The remote aspect was very challenging, and I would have personally tried to work alongside the physical hardware.
Jack	Integration, local unitsting, local hardware development, project scheduling	Make sure to test all dependencies early on, including any repository given, to make sure integration goes much more smoothly.
Ryan	Project documentation and organization, research	How important emulators are, I personally thought the local developments would not help as much as it did. Next time round, I would definitely invest in an FPGA so that I could be more helpful in integration and testing specifically, but throughout the project as well. In addition, personally this project made me realize how much I like working alongside physical hardware which gives that rewarding feedback of accomplishment. Something I had missed in this project.
Hector	Research, documentation, experimental driver development	How vital consistent team communication is when tackling a project with many different aspects. Integration is an important step in development and should be properly prepared for during all previous stages of development.
Ross	Research, RTL development, functional testing, debugging, integration	The importance of initial evaluation of a project to better understand each component and step towards a deliverable from many scopes. More specifically, questioning how this project could have been completed with no FPGA development boards.

Collaboration Site and Repositories

Project documentation can be found within the group repository:

https://github.com/jackchen0226/ECE-Capstone-proj-7

Our contribution Galois' main SoC project can be found in the `feature/video-output` branch in this repository:

https://github.com/GaloisInc/BESSPIN-GFE/tree/470e7e7dd2fe08bdea1c3197ccdd2c8ff7e8051d

References

[1] D. Zimmerman and M. Podhradsky, “RISC-V Secure Hardware Video Output,” Available: http://web.cecs.pdx.edu/~faustm/capstone/projectdescriptions/2021/cd5cb5a5-62ab-4e60-b976-045751974f5b/ECE%20Capstone%20RISCV%20secure%20hardware%20video%20output%20.pdf [Accessed Jan. 22, 2021]

[2] K. Rebello, “System Security Integration Through Hardware and Firmware (SSITH),” Available: https://www.darpa.mil/program/ssith [Accessed Jan. 22, 2021]

[3] L. C. Williams, “DARPA's new hardware proves tough to crack,” *Federal Computer Week,* 24 Aug. 2020. Available: https://fcw.com/articles/2020/08/24/williams-darpa-cyber-hardware.aspx

[4] DARPA, “DARPA FETT Bug Bounty Program,” Available: https://fett.darpa.mil/

[5] DARPA, “FETT Bug Bounty Helps Strengthen SSITH Hardware Defenses,” 28 Jan. 2021, Available: https://www.darpa.mil/news-events/2020-01-28

[6] Digilent, “Digilent Pmod™ Interface Specification 1.2.0,” 8 October 2017

[7] P. Esden-Tempski et. al., “iCEBreaker PMOD Collection,” Available: https://github.com/icebreaker-fpga/icebreaker-pmod [Accessed 2 Jun. 2021]

[8] K. Hubbarb, “BML DVI DIGITAL VIDEO FOR FPGAS OVER PMOD,” 15 Dec. 2017, Available: https://blackmesalabs.wordpress.com/2017/12/15/bml-hdmi-video-for-fpgas-over-pmod/

[9] Texas Instruments, “TFP410 TI PanelBus™ Digital Transmitter (Rev. C),” Dec. 2014.

[10] P. Esden-Tempski et. al., “iCEBreaker examples,” Available: https://github.com/icebreaker-fpga/icebreaker-verilog-examples [Accessed 2 Jun. 2021]

[11] DigiKey, “VGA Controller (VHDL),” Mar. 2021, Available: https://forum.digikey.com/t/vga-controller-vhdl/12794

[12] IBM, “IBM VGA XGA Technical Reference Manual,” May 1992

[13] Xilinx, “VCU118 Evaluation Board User Guide UG1224 (v1.4),” 17 Oct. 2018, Available: https://www.xilinx.com/support/documentation/boards_and_kits/vcu118/ug1224-vcu118-eval-bd.pdf

[14] Wikipedia Contributors, “HDMI,” Available: https://en.wikipedia.org/wiki/HDMI. [Accessed 20 Dec. 2021]

[15] Galois Inc., “GFE Rev. 5.2 System Description,” 2 Apr. 2020, Available: https://github.com/GaloisInc/BESSPIN-GFE/blob/470e7e7dd2fe08bdea1c3197ccdd2c8ff7e8051d/GFE_Rel5.2_System_Description.pdf

[16] M. Waugaman, “VCU118 Rev 2.0 PMOD0 (U41) board bug not fixed,” 15 Feb. 2019, Available: https://forums.xilinx.com/t5/Xilinx-Evaluation-Boards/VCU118-Rev-2-0-PMOD0-U41-board-bug-not-fixed/td-p/940193

[17] Xilinx, “Xilinx Embedded Software (embeddedsw) Development,” Available: https://github.com/Xilinx/embeddedsw, [Accessed 10 Jun 2021]

[18] J. Kiniry, “BESSPIN (Balancing Evaluation of System Security Properties with Industrial Needs)”, Available: https://galois.com/project/besspin/, [Accessed 22 Jan 2021]