

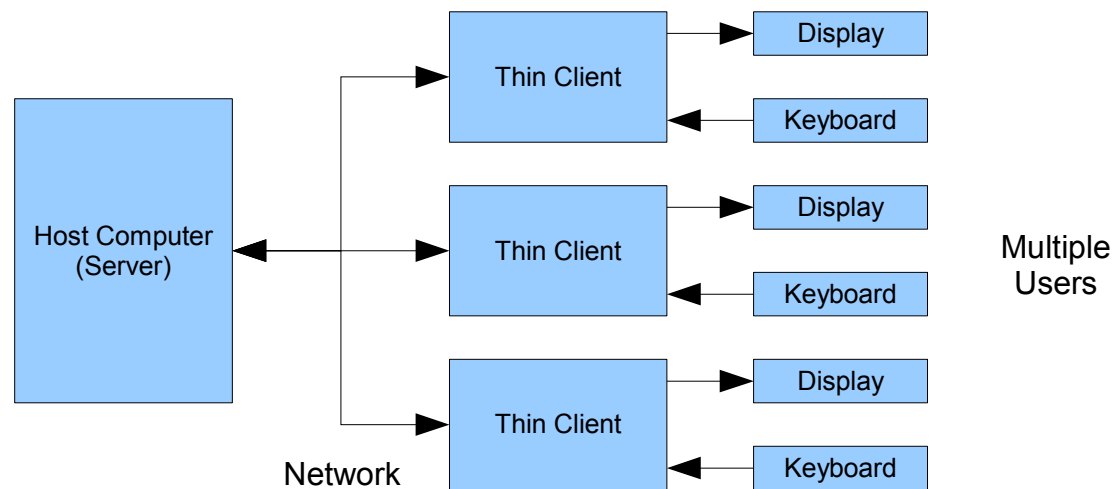
# Design and Implementation of an FPGA Based Thin Client

By Liam Davey (13648671)

Supervised by Dr Cesar Ortega-Sanchez

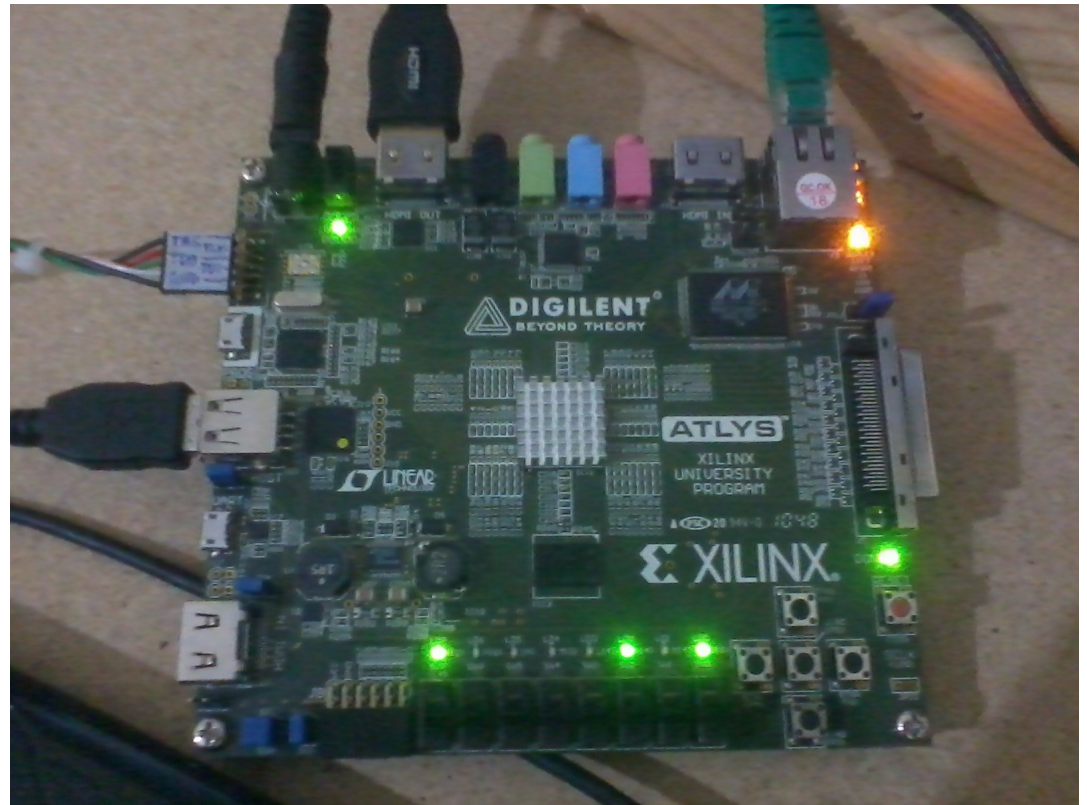
# Introduction

- Thin Clients allow server resources to be shared among multiple users
- Traditional Thin Clients compress video data for transmission over low bandwidth networks meaning high processing overhead
- Proposed solution: FPGA Thin Client decoding uncompressed video transmitted over high bandwidth network



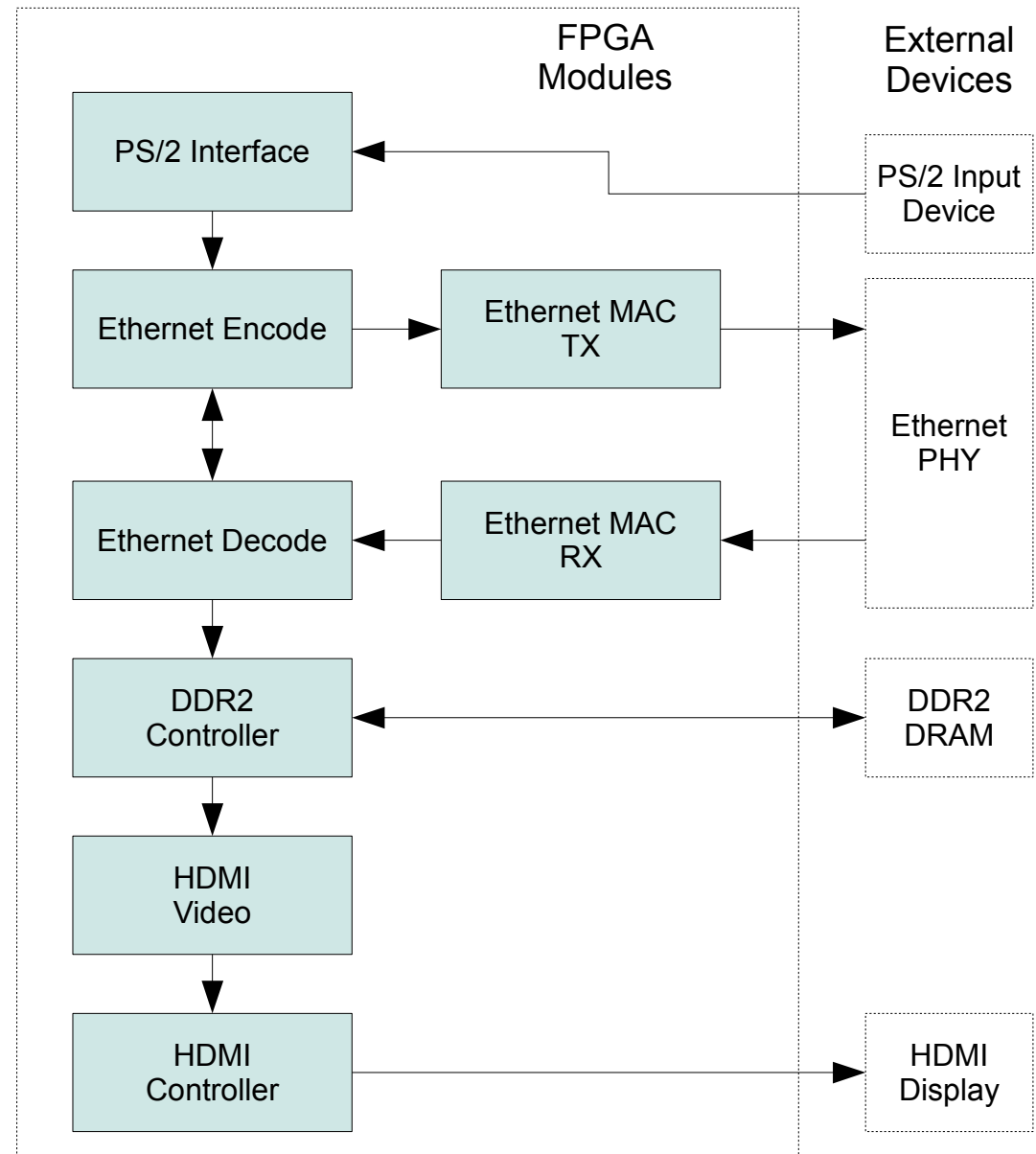
# Problem Solution

- Gigabit Ethernet network interface
- HDMI digital display interface
- PS2 for input devices
- FPGAs good for high throughput applications
- Verilog HDL used
- Digilent Atlys development board as prototype hardware platform



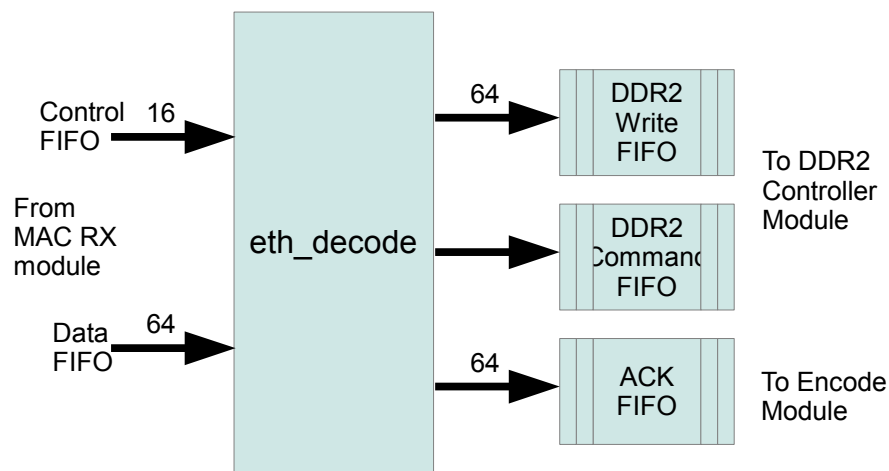
# Top Level Design

- Ethernet subsystem and HDMI subsystem communicating via dual port DDR2 controller
- Framebuffer held in external DDR2 DRAM

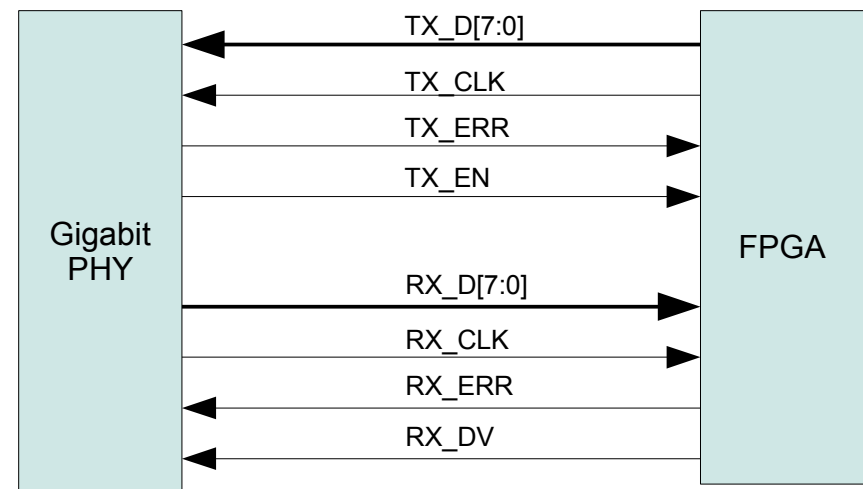


# Ethernet Subsystem Design

- Receive side decodes frames, writes RGB data to framebuffer, writes acknowledgement if necessary
- Transmit side encodes outgoing acknowledgement frames or keyboard data frames
- Gigabit Media Independent Interface (GMII) between Ethernet PHY and FPGA



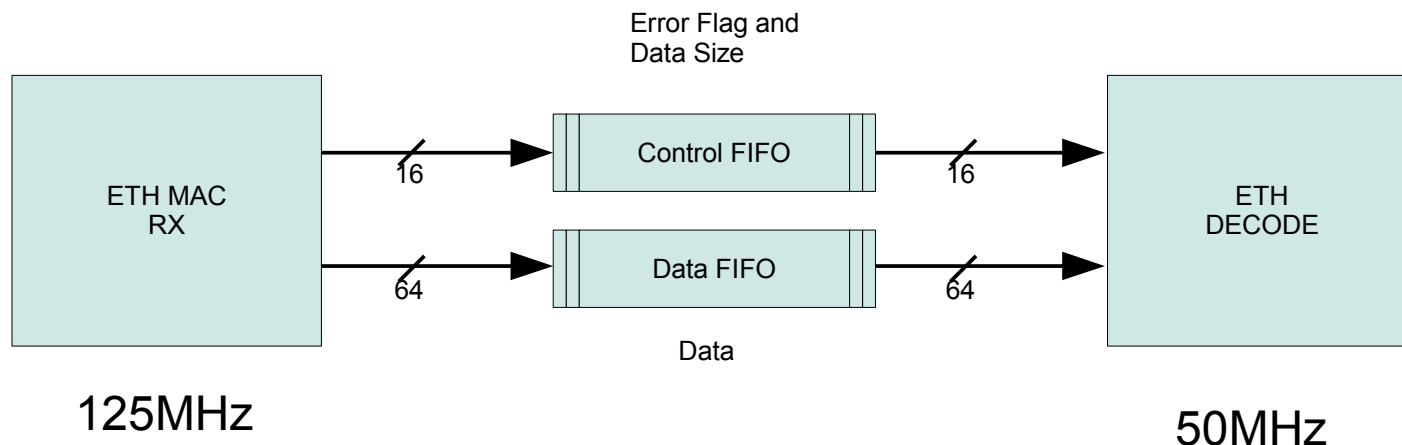
Ethernet Decode Module



GMII

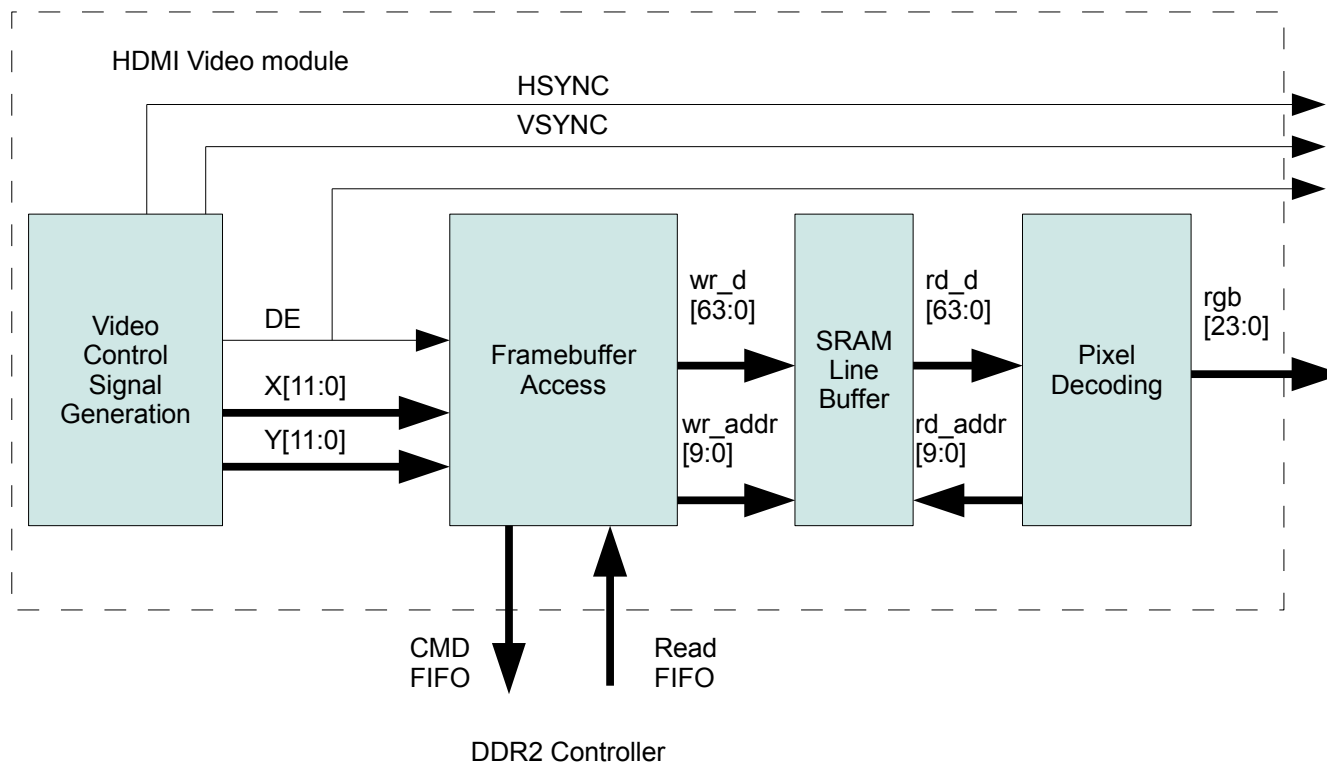
# Frame Buffering and Checksum

- Interface between MAC and decode module problematic
- Cross clock domain (125MHz → 50MHz)
- Frames cannot be checked for validity until checksum is received at end of each frame
- Decode module must wait until frame checksum validated before decoding
- Solution: Control and Data Asynchronous FIFOs



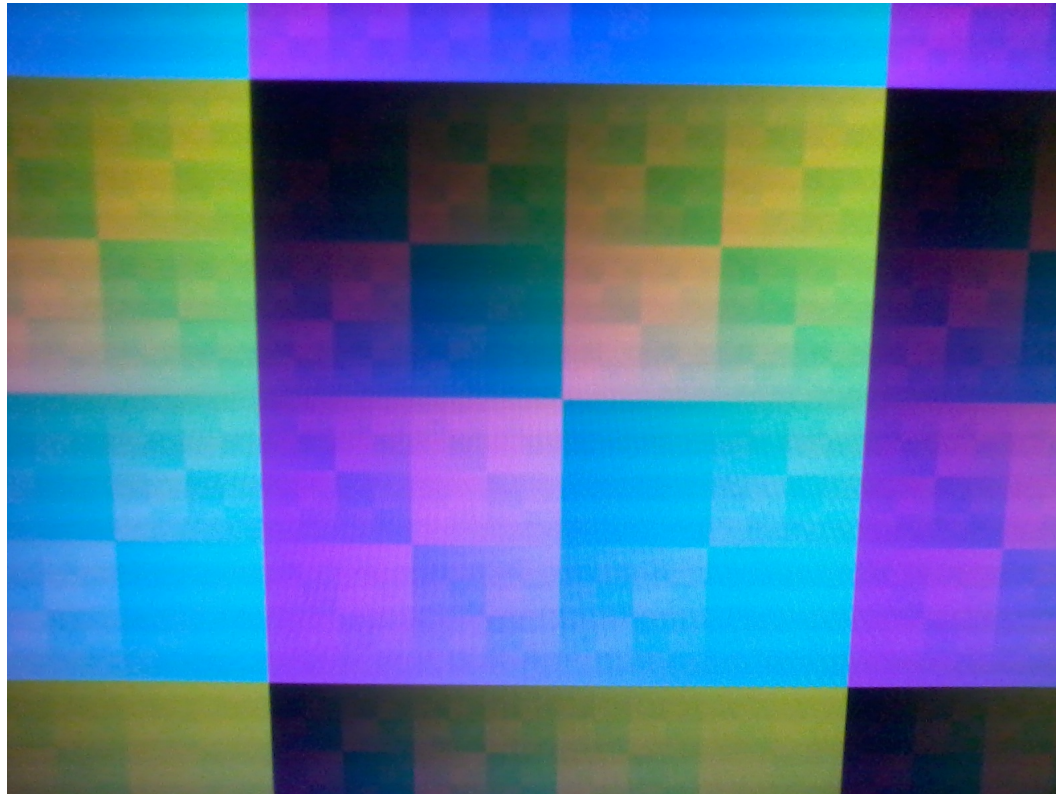
# HDMI Video Design

- Generates VSYNC, HSYNC, DE video signals
- Reads RGB data from DDR2 framebuffer
- Line buffer to prevent underflow and overflow problems



# HDMI Video Test Mode

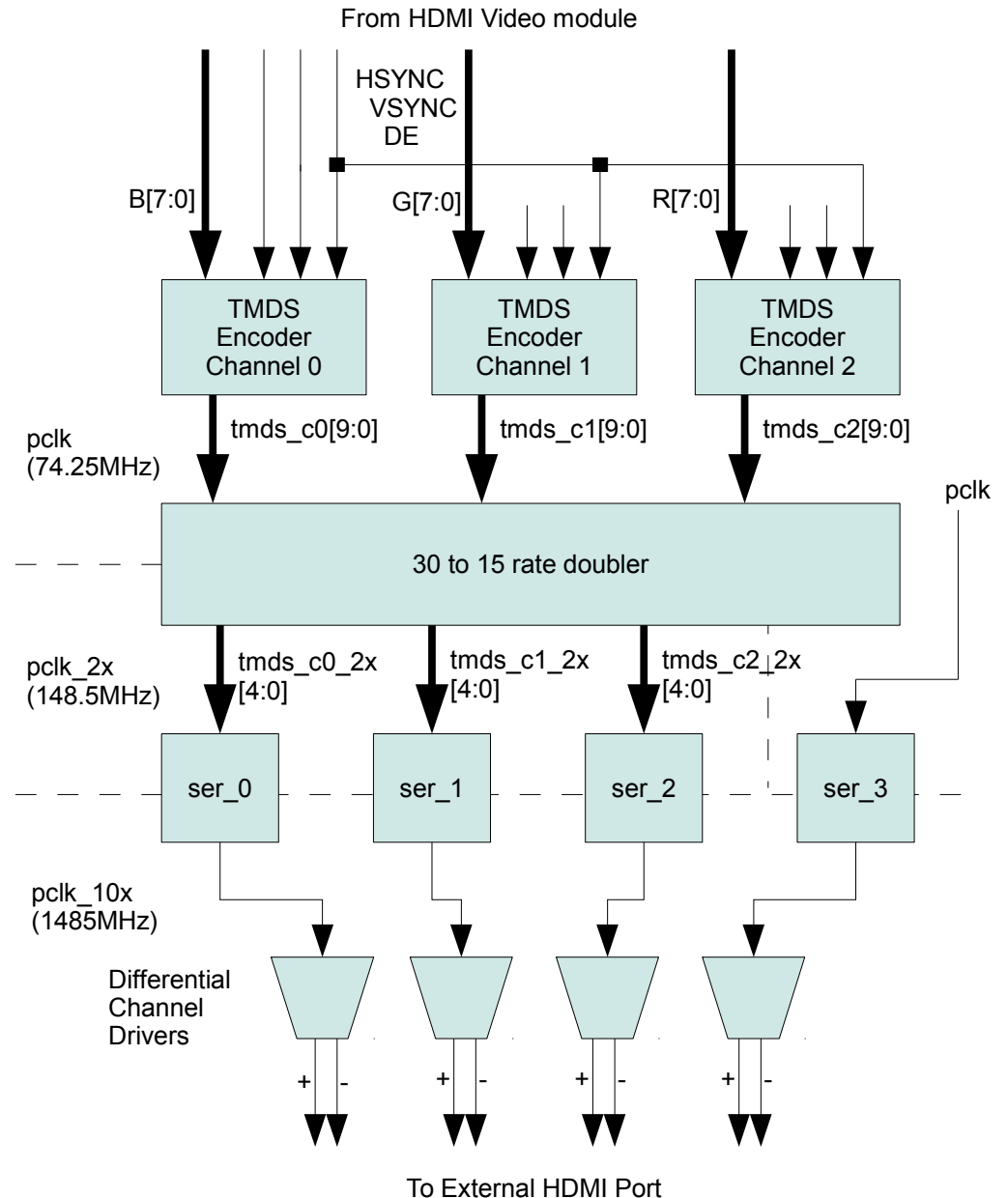
- Test mode generates test pattern and ignores data read from framebuffer
- Used to test video signal generation and HDMI controller separately





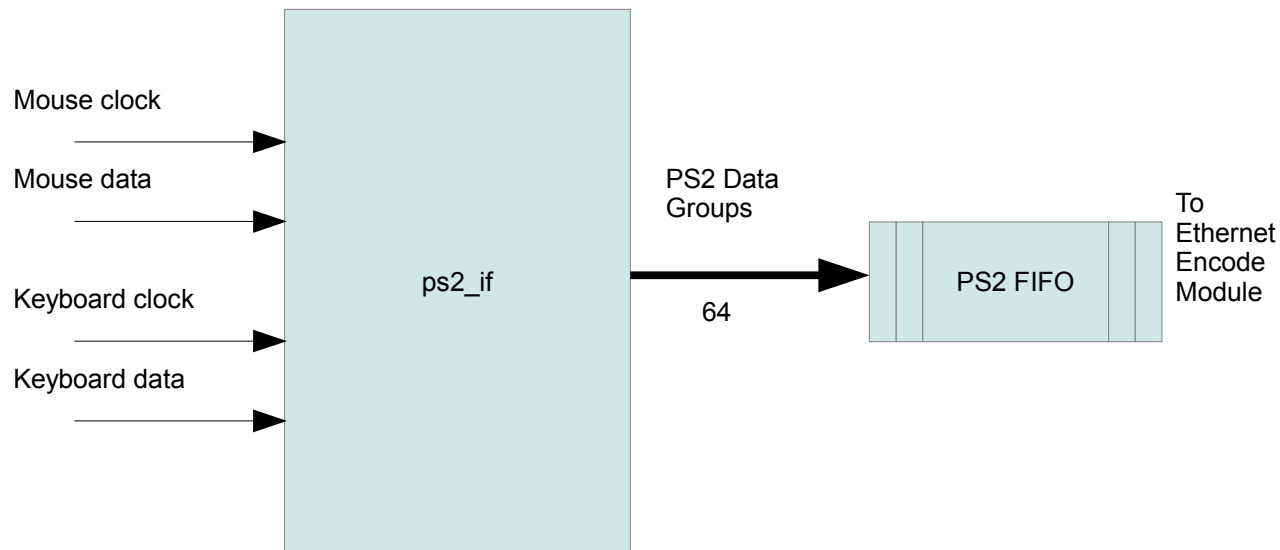
# HDMI Controller

- Receives RGB data and video signals from HDMI video module
- Encodes using TMDS algorithm from DVI Specification
- Serialized using FPGA hardware SERializer-DESerializers (SERDES)
- Transmitted on four differential channels for HDMI



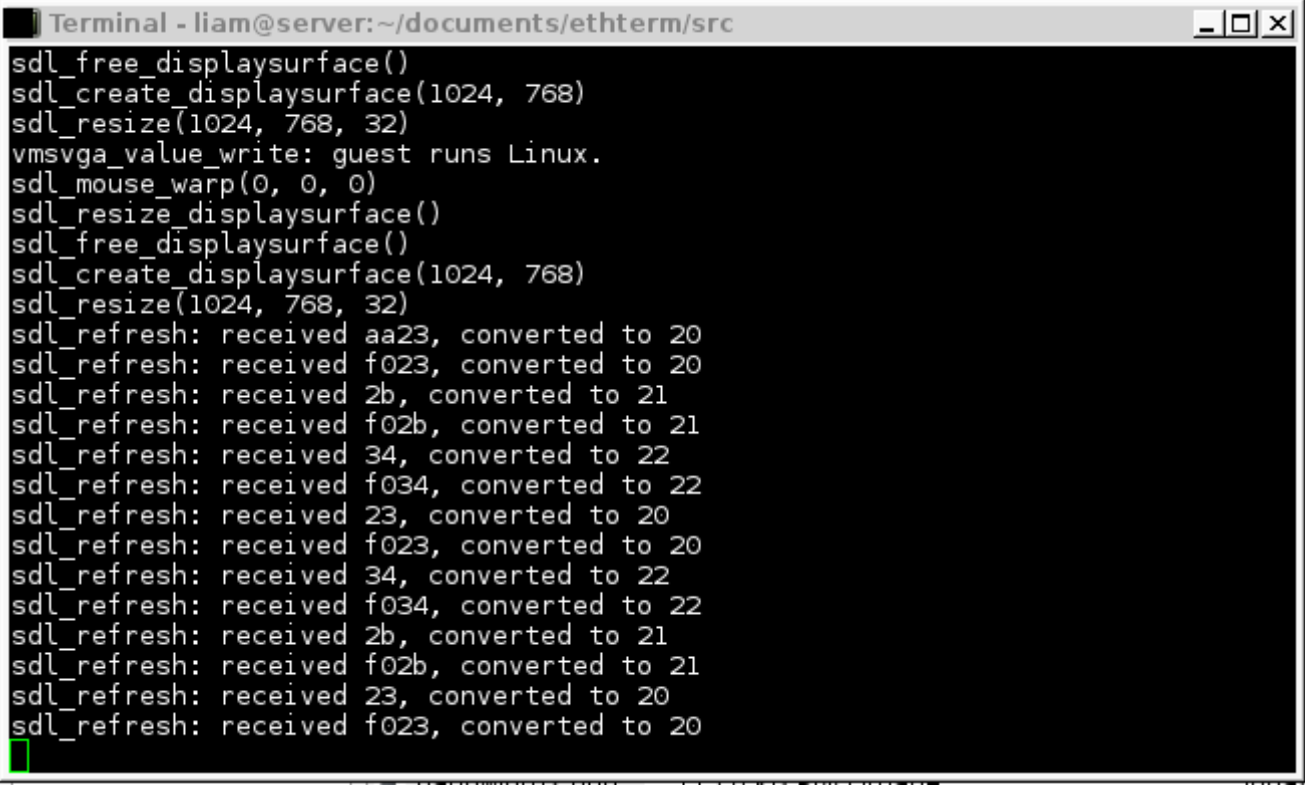
# PS2 Controller

- Reads scan codes from keyboard
- Sends scan codes to Ethernet Encode module



# Software

- QEMU, an open source virtualization solution
- Modified default video output driver to send Raw Ethernet frames
- Listened for incoming frames for keyboard data

A terminal window titled "Terminal - liam@server:~/documents/ethterm/src" with standard window controls. The terminal displays a series of SDL function calls and QEMU console output. The first part shows SDL initialization and window creation. Then, a message "vmsvga\_value\_write: guest runs Linux." appears. This is followed by a series of "SDL refresh" messages, each showing a received hex value and its converted decimal value. The values received are: aa23 (20), f023 (20), 2b (21), f02b (21), 34 (22), f034 (22), 23 (20), f023 (20), 34 (22), f034 (22), 2b (21), f02b (21), 23 (20), and f023 (20).

```
Terminal - liam@server:~/documents/ethterm/src
SDL_free_displaySurface()
SDL_create_displaySurface(1024, 768)
SDL_resize(1024, 768, 32)
vmsvga_value_write: guest runs Linux.
SDL_mouse_warp(0, 0, 0)
SDL_resize_displaySurface()
SDL_free_displaySurface()
SDL_create_displaySurface(1024, 768)
SDL_resize(1024, 768, 32)
SDL_refresh: received aa23, converted to 20
SDL_refresh: received f023, converted to 20
SDL_refresh: received 2b, converted to 21
SDL_refresh: received f02b, converted to 21
SDL_refresh: received 34, converted to 22
SDL_refresh: received f034, converted to 22
SDL_refresh: received 23, converted to 20
SDL_refresh: received f023, converted to 20
SDL_refresh: received 34, converted to 22
SDL_refresh: received f034, converted to 22
SDL_refresh: received 2b, converted to 21
SDL_refresh: received f02b, converted to 21
SDL_refresh: received 23, converted to 20
SDL_refresh: received f023, converted to 20
```

Modified QEMU console output showing received keyboard data

# Ethernet Verification

- Simulated with Verilog test benches then tested on FPGA
- Raw frames dumped and analyzed using Wireshark
- Problems with bit ordering and frame check sum

Capturing from eth0 [Wireshark 1.6.2]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length
13404	5.063997	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13405	5.064001	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13406	5.064004	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13407	5.064007	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13408	5.064010	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13409	5.064013	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13410	5.064016	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13411	5.064019	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13412	5.064022	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13413	5.064025	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13414	5.064028	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13415	5.064031	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13416	5.064033	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632
13417	5.064036	CompalIn_54:e8:5a	Vmware_ab:cd:ef	0x88b5	632

▶ Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits)  
▶ Ethernet II, Src: Vmware\_ab:cd:ef (00:50:56:ab:cd:ef), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
▶ Data (50 bytes)

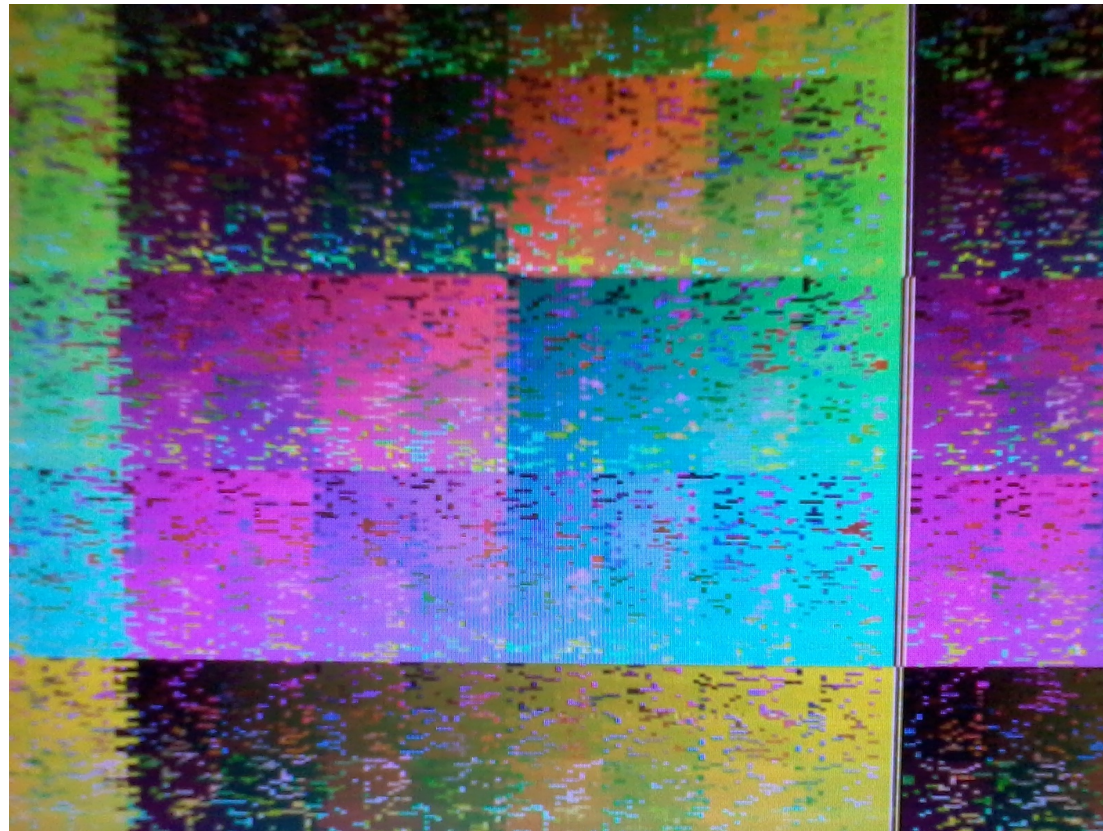
0000	ff ff ff ff ff ff 00 50 56 ab cd ef 88 b5 43 21	.....P V.....C!
0010	00 00 00 00 00 00 00 00 2b 00 00 00 00 00 00 00	.....+ .....
0020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

eth0: <live capture in progress> File: Packets: 13417 Displayed: 13417 Marked: 0

Example frame capture in Wireshark

# HDMI Verification

- Test mode used to discover problem with incorrect clock
- DDR2 controller underflow and overflows caused image distortion
- Distortion fixed by adding line buffer



Distorted test pattern

# System Testing

- Testing with High Definition video
- Program written to send uncompressed video data to Thin Client
- 20 – 30 Frames per second displayed (1280 x 720 resolution)

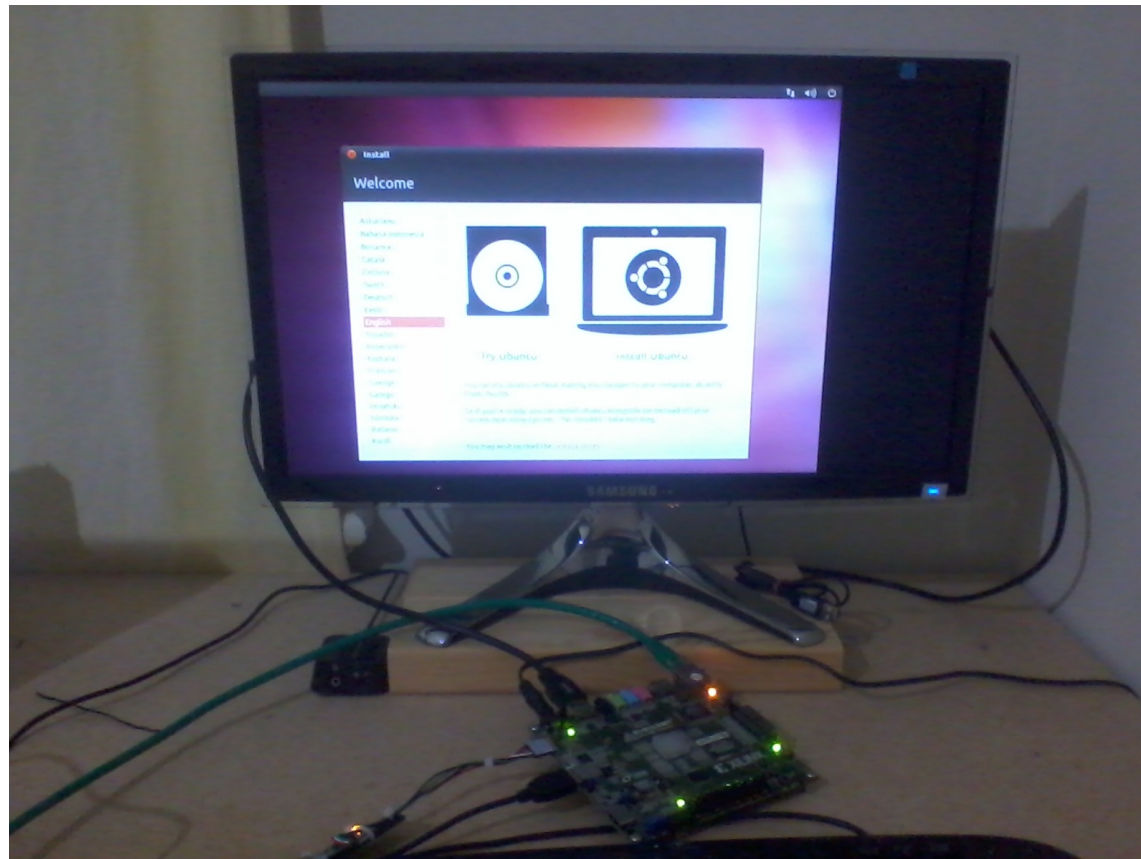


A HD video being displayed on the prototype Thin Client



# System Testing

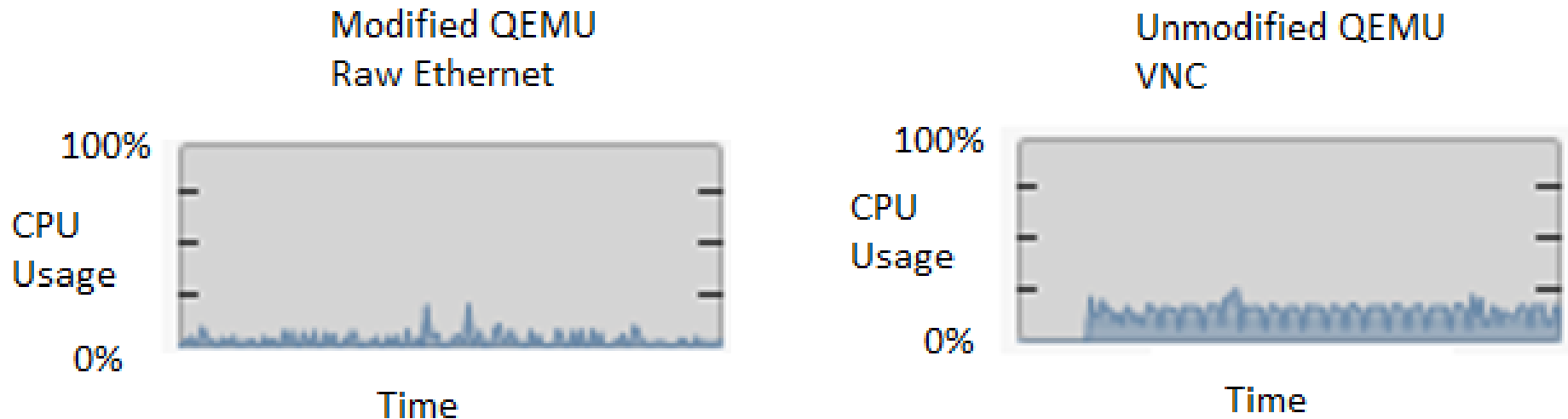
- Testing with modified QEMU
- Live CD used as virtualized guest under QEMU



Prototype Thin Client being used to access virtualized guest on server

# Test Results

- 57MB/s throughput using integrated Realtek NIC on Host PC
- 98MB/s throughput using Intel NIC on Host PC
- Test performed comparing VNC and modified QEMU CPU usage



Note that 25% CPU usage in these graphs represents 100% usage of a single core of the quad core processor used in the test.



# Conclusions

- Prototype FPGA based Thin Client successfully designed and implemented
- Server CPU overhead reduced compared to when using video compression
- Future Improvements: USB support, Audio support, more HDMI resolutions, Ethernet reliability improvement

End