ECE 385

Fall 2024

Nintendo DS Capture Card
Final Project

Aaron Zawislak, Guyan Wang
HG, 12/16/2024
TA: RJ

1. Idea and Overview (Introduction)

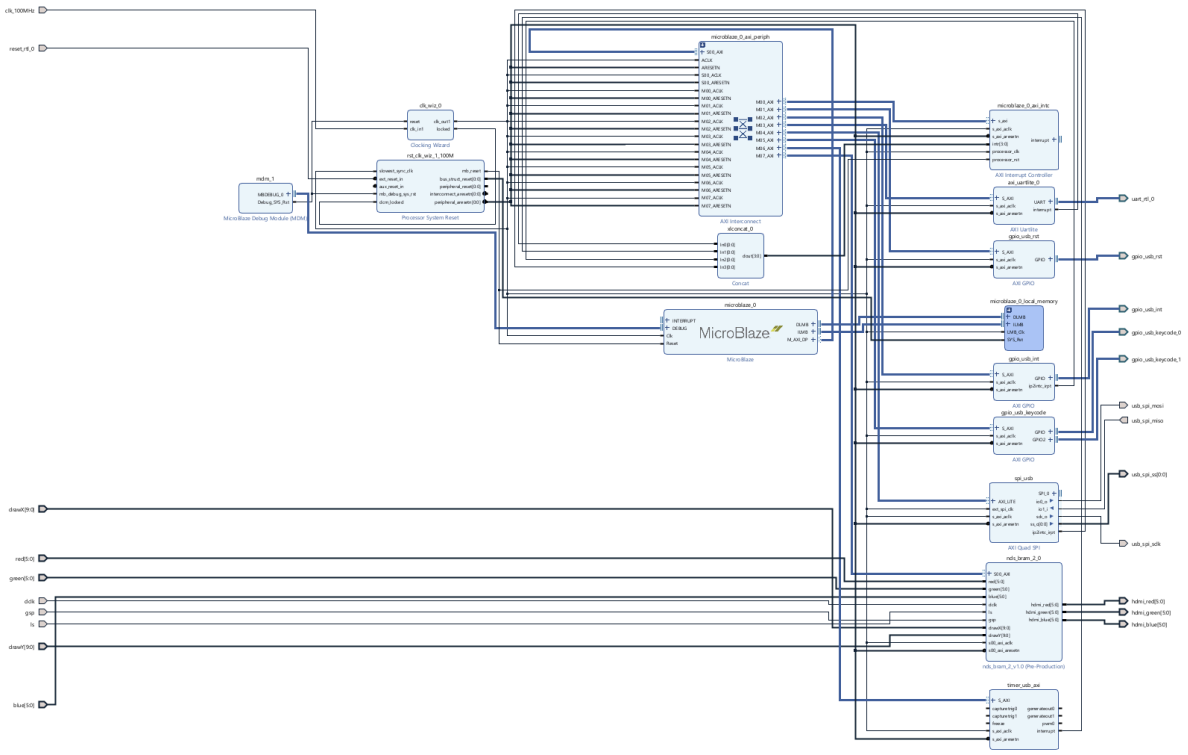
For our Final Project of ECE 385, we have made a capture device for the Nintendo DS Lite. Like many other portable game consoles, the board does not natively come with HDMI output capabilities, nor the ability to record output game data.

2. Written Description

The SystemVerliog code that we have implemented works entirely in hardware for the HDMI passthrough of the Nintendo DS top screen. First, we solder onto the board 30 gauge hookup wire that allows us to read the signals off the board. After confirming the pins are of the IO standard LVCMOS 33 by reading off an oscilloscope, we solder wires from our wire kit onto the wires off our board, in order to connect the DS to the FPGA. After setting the pin constraints in the Vivado Project and from referencing the Urbana Board manual, we are ready to run our code. During every frame of the Nintendo DS, at every display clock rising AND falling edge, we read the color data that comes into the GPIO pins, set the address to store in memory (a simple pixel counter worked for us, but a more rigorous implementation of address setting can work as well to ensure proper data alignment). Once GSP (essentially the Nintendo DS display vsync) goes low, we reset the counter to start writing the next frame. This ensures that we always have a full frame to render in bram at any given moment in time. At the same time as this is happening, we are reading the current value of DrawX and DrawY and calculating the appropriate address to read from memory. Using bram in a dual port configuration allows us to simultaneously write and read data. We then assure that the data outputted is stable for a single frame, and then pass the outputted data to the VGA to HDMI converter. If we are outside the bounds of the middle of the screen (we define the bounds where we draw in as drawX > 170 && drawX <= 470 && drawY > 144 && drawY <= 336, which equates to a 300x192 display region, 256 horizontal pixels with a 44 pixel blanking area per line), we set the hdmi rgb to 18’b0 and pass that to the converter instead.

3. Block Diagram

Since our solution is implemented entirely in hardware, we do not have a relevant block diagram, other than the provided Lab 6.2 which we built off of. However, in order to send data over USB/Ethernet/Serial/etc in the future (urbana board limitations make a ~30Mbit/second data stream near impossible, due to the USB driver not including a peripheral mode, pmod being taken by NDS signals, and serial being too slow), in my own time (so not included in the submitted project zip) i instantiated the nds bram ip and made the appropriate modifications to index bram from the microblaze processor, and moved our input signals into the block diagram. Below is the implementation of that module (again this is not included in the submitted project zip since it has no use in our final implementation)



4. Module Descriptions

Module: `mb_usb_hdmi_top.sv`

Inputs: `Clk, reset_rtl_0, [0:0] gpio_usb_int_tri_i, usb_spi_miso, uart_rtl_0_rxd, T_R0, T_R1, T_R2, T_R3, T_R4, T_R5, T_G0, T_G1, T_G2, T_G3, T_G4, T_G5, T_B0, T_B1, T_B2, T_B3, T_B4, T_B5, DCLK, GSP, LS,`

Outputs: `gpio_usb_rst_tri_o, usb_spi_mosi, usb_spi_sclk, usb_spi_ss, uart_rtl_0_txd, hdmi_tmds_clk_n, hdmi_tmds_clk_p, [2:0]hdmi_tmds_data_n, [2:0]hdmi_tmds_data_p, [7:0] hex_segA, [3:0] hex_gridA, [7:0] hex_segB, [3:0] hex_gridB, LED0, LED1, LED2, LED3`

Description: This module is the comprehensive top-level for our NDS capture card driver. It integrates several smaller modules like `mb_usb` for USB handling, `hex_driver` for display outputs, and `clk_wiz_0` to manage clock requirements, especially for HDMI output. The `vga_controller` and `hdmi_tx_0` are critical for generating video output, taking in VGA sync signals, and converting them into HDMI-compliant output. The NDS signal processing is handled by `nds_bram` which stores Nintendo DS display pin inputs into BRAM, and grab from BRAM based off DrawX and DrawY for input into the HDMI converter. Debugging and additional functionalities are managed through the `pin_xor` module and status LEDs which indicate various operational statuses or errors, though the module is mainly included to ensure that the GPIO signals are not optimized out for Debug Cores logic analyzer.

Purpose: The NDS signal processing is handled by `nds_bram` which stores Nintendo DS display pin inputs into BRAM, and grab from BRAM based off DrawX and DrawY for input into the HDMI converter.

Module: `hex_driver.sv`

Inputs: `clk, reset, [3:0] in[4]`

Outputs: `[7:0] hex_seg, [3:0] hex_grid`

Description: This module is a typical Hex display we used in our final project and normal labs throughout the semester. Clk is the signal from the clock. Reset is used to clear the value on the HEX display. It is the 4-bit value we want to display through the HEX light. The output hex_seg and hex_grid are the segments and sections of the board that can be visualized through the FPGA board.

Purpose: This module will light up the HEX display on our FPGA board so that we can display the value inside both registers, which is very important and helpful for debugging and demoing.

Module: `VGA_controller.sv`

Inputs: `pixel_clk, reset,`

Outputs: `hs, vs, active_nblank, sync, drawX, drawY`

Description: This module is designed to generate timing signals for a standard 640x480 VGA display using a slightly adjusted pixel clock of 25 MHz instead of the typical 25.175 MHz. The module uses counters (`hc` for horizontal and `vc` for vertical) to keep track of the current pixel and line being processed. The horizontal pixels are indexed from 0 to 799, and the vertical lines from 0 to 524. Generates the horizontal and vertical sync pulses. The horizontal sync pulse occurs for 96 pixels during the transition from pixel 656 to 752, and the vertical sync pulse spans 2 lines, specifically at lines 490 and 491. Determines the display window to output the active video signal, restricted to horizontal coordinates 0-639 and vertical coordinates 0-479 to fit a 640x480 VGA format. The controller includes logic to turn off the display output (blanking) outside the defined active video window, ensuring no video signal is output during the horizontal and vertical retrace intervals.

Purpose: This module is a fundamental building block for generating the correct timing and control signals necessary for interfacing with VGA-compatible displays to output the content generated by the signals from nds.

Module: `pins_xor.sv`

Inputs: `R0, R1, R2, R3, R4, R5, G0, G1, G2, G3, G4, G5, B0, B1, B2, B3, B4, B5,DCLK, GSP, LS, CLK, logic reset_n.`

Outputs: `LED0, LED1, LED2, LED3`

Description: This module is designed for debugging purposes, specifically for logic analysis of signal behavior of color and clock signals on the NDS motherboard. It primarily functions to compute the XOR across multiple sets of input signals related to RGB color data and additional control signals, outputting the result to LEDs for quick visual feedback on the signal processing. An additional XOR operation combines the results of the RGB XORs with the synchronization signals GSP and LS, as well as an overall XOR of these results, which is then outputted to LED0. LED1, LED2, and LED3 are unused in our current design and will be reserved for future development purposes. Upon reset (`reset_n` asserted low), all internal flip-flops holding the XOR results are cleared to `0`, ensuring a clean state for accurate debugging post-reset.

Purpose: This module is useful for debugging the capture card circuit where understanding the interaction between multiple signal groups is crucial. It can help in tracing issues in color data processing or synchronization mechanisms within a larger system by providing a real-time, visual representation of the logical relationship between various signal groups.

Module: `nds_bram.sv`

Inputs: `[5:0] red, [5:0] green, [5:0] blue, dclk, ls, gsp, reset, clk, [9:0] drawX, drawY`

Outputs: `[5:0] hdmi_red, [5:0]hdmi_green, hdmi_blue`

Description: This module is created to interface between an NDS-like display and an HDMI encoder, using a Block RAM (BRAM) as an intermediary to store and manipulate the pixel data. **red, green, blue** (6 bits each) are RGB color inputs from an NDS-like display system. **dclk** (Display Clock) is the clock signal for synchronization of display data inputs. **ls** (Line Sync) is the horizontal synchronization signal, similar to HSYNC. **gsp** (Global Sync Pulse) is the vertical synchronization signal, similar to VSYNC. **reset** is the active-high reset signal for initialization or reinitialization of the module. **clk** is the system clock signal for internal operations. **drawX, drawY** (10 bits each) are coordinates used for indexing data within the BRAM for output generation. **hdmi_red, hdmi_green, hdmi_blue** (6 bits each) are color outputs to an HDMI encoder, representing the processed pixel data. Firstly, this module packs the incoming RGB data into a single word (`dina`) and stores it in a BRAM. The BRAM is configured for dual-port operation, with one port dedicated to writing data and the other to reading data. Secondly, it calculates the address for pixel data storage and retrieval based on the sync signals and the display clock, managing how data is written to and read from the BRAM. Thirdly, it Implements logic to synchronize the display clock to the system clock and detect rising and falling edges of the display clock for accurate timing of data storage. Fourthly, it manages the write operations based on the state of the `gsp` signal and pixel count, ensuring that data is stored only during valid display times and within the correct memory bounds. Finally, it retrieves and aligns the data from the BRAM to the HDMI output format, adjusting the data based on the calculated screen position (`drawX`, `drawY`) to center the display content correctly.

Purpose: This module is very useful since it can adapt lower resolution or differently formatted display outputs to standard HDMI, which is very crucial for retro gaming consoles like NDS. The module is also very useful for all the projects requiring the conversion and

scaling of display signals from a native display format to a format suitable for HDMI output. The detailed internal operations, such as handling the edge cases in synchronization, addressing in BRAM, and output alignment, make this module a robust solution for handling complex video data transformations.

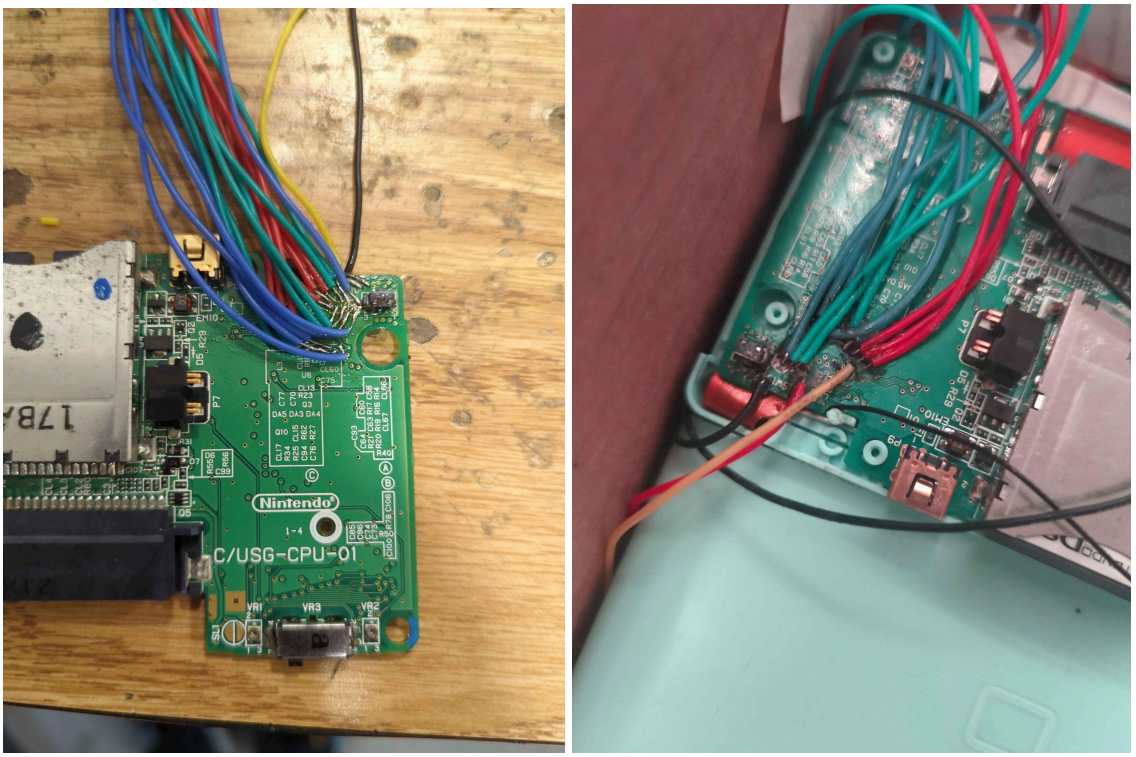
5. List of Features

HDMI Passthrough of Nintendo DS Lite Top Screen
Frame Data storage in BRAM

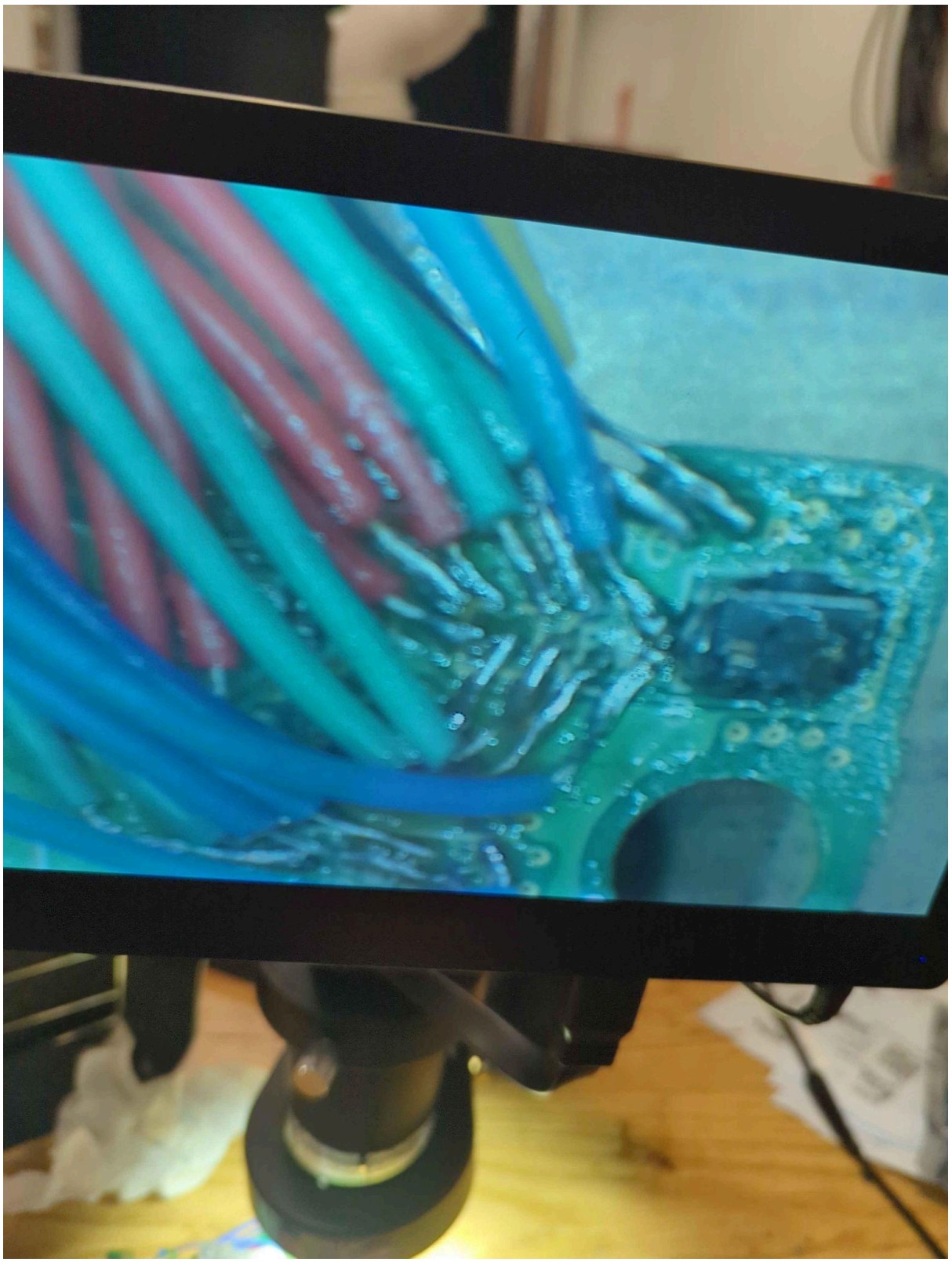
Other proposed features like a software renderer, controller passthrough, audio passthrough, touchscreen passthrough, etc, needed to be scrapped due to hardware limitations of the Urbana Board. Specifically, the Software Renderer needed to be scrapped due to the USB port being configured as a host, and not a device, which we'd need to be able to request pixel data from every address 30 times a second from the host PC. UART is too slow for this task as well, with maximum baud rate being 115200, meaning a frame would be rendered once every 10 seconds. Post demo thoughts of a USB to Ethernet setup may work, however continuing this project on different and more versatile hardware may be the optimal solution. Controller passthrough, audio passthrough, and touchscreen passthrough were all also scrapped, due to there not being enough PMOD pins left over from the display data input (there was only 1 left over!)

6. Simulation Waveforms and Images

(Logic analysis of NDS input signals in the second stage using Debug Cores)



(Soldering signals on board, left is a test run on a practice board, right is final revision, minus pins that fall off due to movement in transport, looking at it wrong, etc.)



(Our first working attempt at soldering onto a practice board, microscope view)

(Final display output)

7. Document the Design Resources and Statistics from the lab manual.

LUT	3010
DSP	5
BRAM	36.5
FF	3111
Latches	0
Frequency	100
Static Power	0.077W
Dynamic Power	0.383W
Total Power	0.460W

8. Expected Difficulty

Our project was rated 9-9.5 out of 10 difficultly points, which is reasonably hard. Fortunately, we implemented the crucial feature of the top-screen synchronous display of the Nintendo DS top screen.

9. Proposed Timeline

During the first week of the project, we mainly focused on gathering useful information about the signals on pins that are used in the NDS such as dclk and gsp. In the second week, we soldered the wires on the NDS motherboard to make all the signals available to input to the FPGA, and we made sure that the FPGA could correctly recieve the signal on board. In the third week, we start developing the driver of our capture card in Vivado using the IP from lab 7.2. In the final week, we turned the base of our project from lab 7.2 to lab 6.2 since it is unnecessary to use an IP editor for the software driver which will make our project much more complex, and implement the top-screen display feature successfully in the end.

10. Conclusion

Due to the high difficulty of our project, we did not have enough time to implement all the functionality required by a typical Nintendo DS capture card. Fortunately, we implemented the crucial feature of the top-screen synchronous display, which is very astonishing and groundbreaking in our project. However, there are still things like the bottom screen synchronous display we need to work on in the future. We can achieve this by implementing a USB pass-through to the Urbana board. Overall, we think our project is very successful and applied among gamers.