**Digital Circuit Lab**

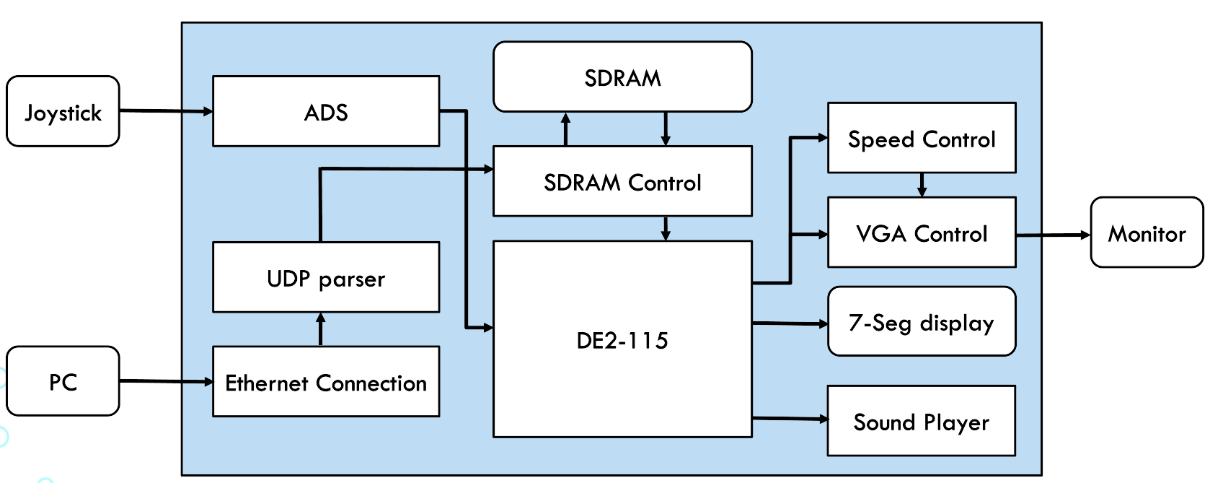
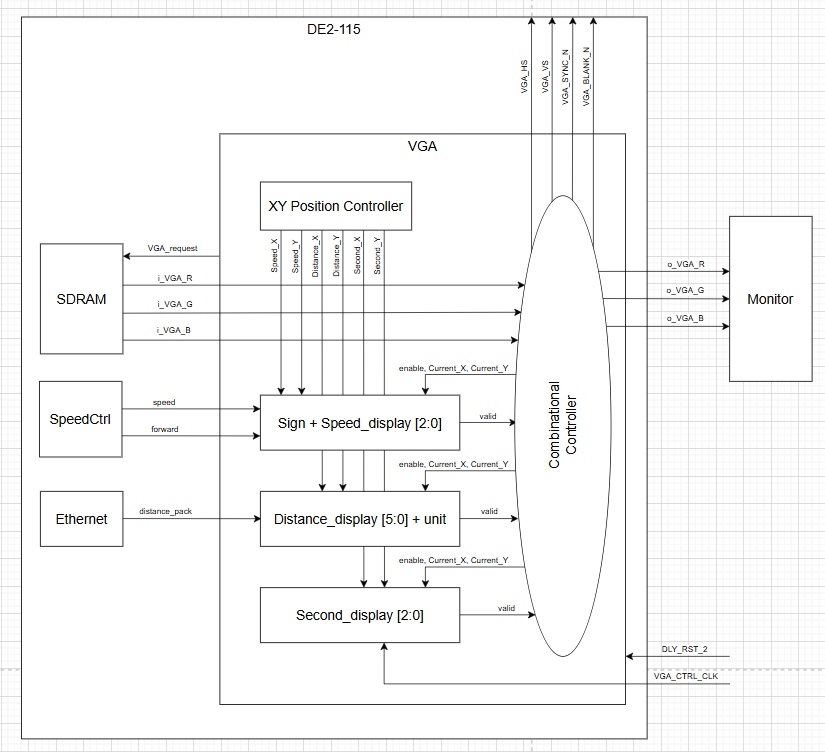
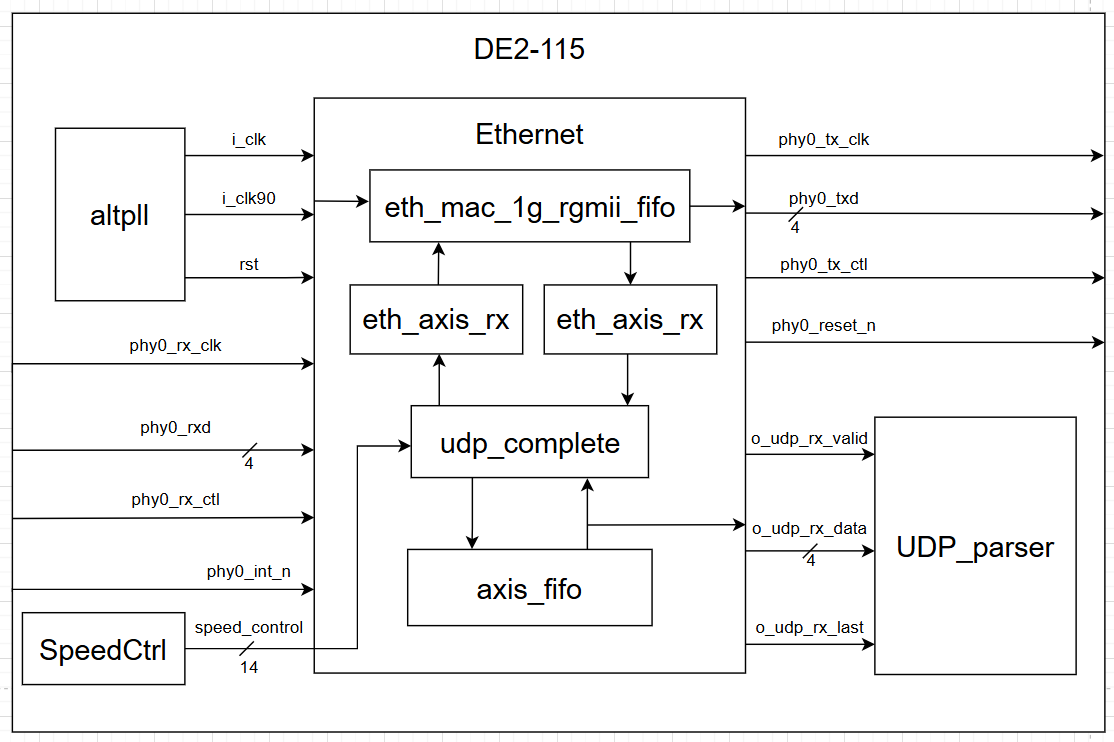
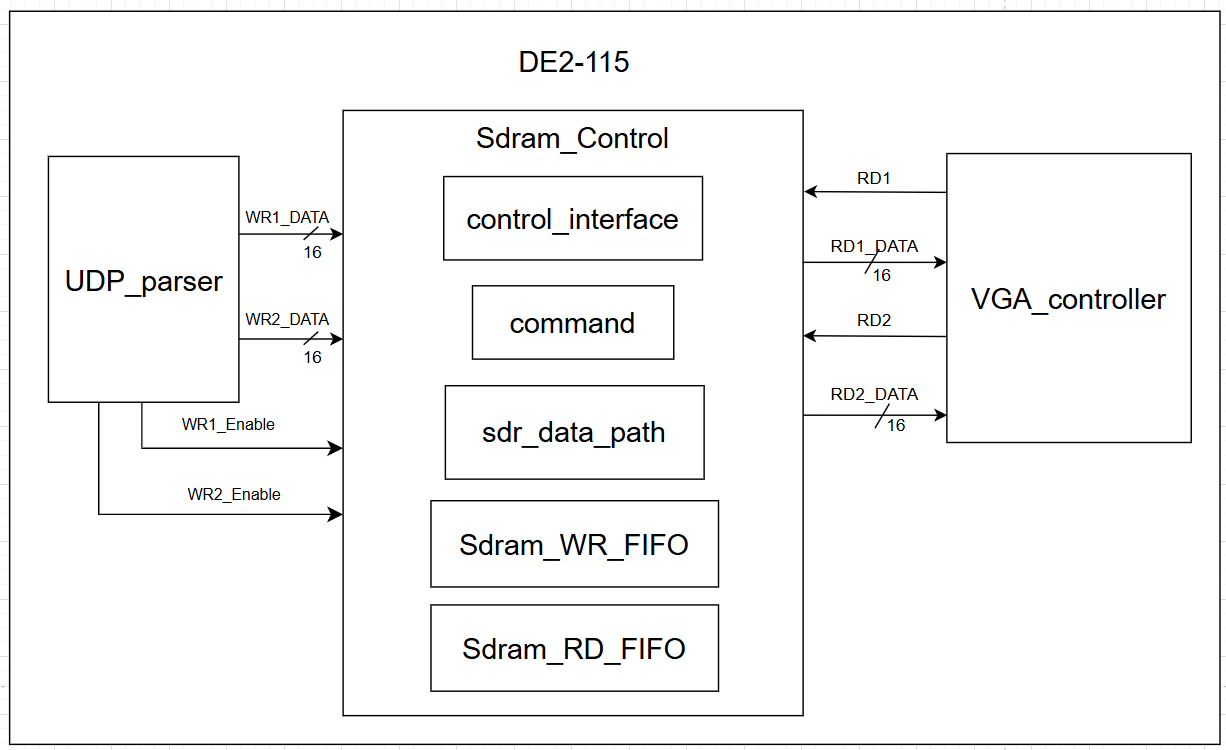
Team 09 / Lab 3 Report

B11901022 丁睿濂

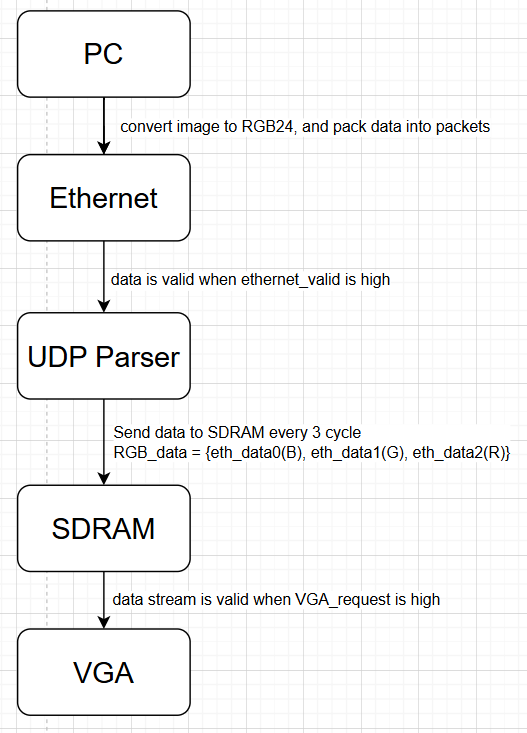
B11901023 林靖軒

B11901030 劉代恩

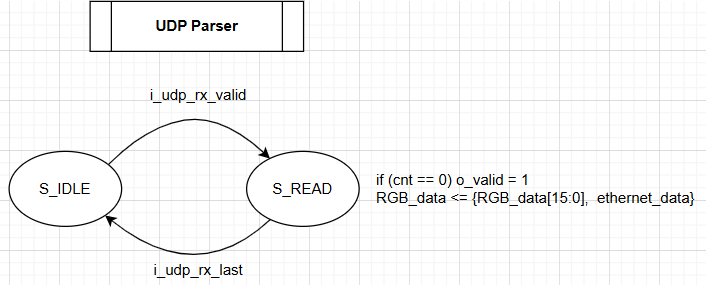
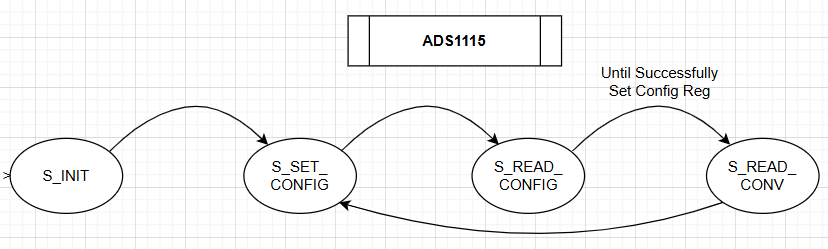
1. File Structure  
     
   /team09\_final  
   ├ team09\_final\_report.pdf  
   ├ team09\_final\_presentation.pdf  
   └ /src  
    ├ /ADS  
    │ └ ads1115\_controller.v  
    ├ /DE2-115  
    │ ├ DE2\_115.sv  
    │ ├ Debounce.sv  
    │ └ hex\_display.v  
    ├ /Ethernet  
    │ ├ arp.v  
    │ ├ arp\_cache.v  
    │ ├ arp\_eth\_rx.v  
    │ ├ arp\_eth\_tx.v  
    │ ├ axis\_gmii\_rx.v  
    │ ├ axis\_gmii\_tx.v  
    │ ├ eth\_arb\_mux.v  
    │ ├ eth\_axis\_rx.v  
    │ ├ eth\_axis\_tx.v  
    │ ├ eth\_mac\_1g.v  
    │ ├ eth\_mac\_1g\_rgmii.v  
    │ ├ eth\_mac\_1g\_rgmii\_fifo.v  
    │ ├ eth\_mux.v  
    │ ├ Ethernet.v  
    │ ├ iddr.v  
    │ ├ ip.v  
    │ ├ ip\_arc\_mux.v  
    │ ├ ip\_complete.v  
    │ ├ ip\_eth\_rx.v  
    │ ├ ip\_eth\_tx.v  
    │ ├ ip\_mux.v  
    │ ├ lfsr.v  
    │ ├ oddr.v  
    │ ├ rgmii\_phy\_if.v  
    │ ├ ssio\_ddr\_in.v  
    │ ├ ssio\_ddr\_out.v  
    │ ├ udp.v  
    │ ├ udp\_checksum\_gen.v  
    │ ├ udp\_complete.v  
    │ ├ udp\_ip\_rx.v  
    │ ├ udp\_ip\_tx.v  
    │ ├ UDP\_parser.v  
    │ └ /lib  
    │ ├ arbiter.v  
    │ ├ axis\_async\_fifo.v  
    │ ├ axis\_async\_fifo\_adapter.v  
    │ ├ axis\_fifo.v  
    │ ├ priority\_encoder.v  
    │ └ sync\_reset.v  
    ├ /python  
    │ └ send\_video.py  
    ├ /Sdram  
    │ ├ command.v  
    │ ├ control\_interface.v  
    │ ├ sdr\_data\_path.v  
    │ ├ Sdram\_Control.v  
    │ ├ Sdram\_Params.h  
    │ ├ sdram\_pll.v  
    │ ├ Sdram\_RD\_FIFO.v  
    │ └ Sdram\_WR\_FIFO.v  
    ├ /SpeedCtrl  
    │ └ SpeedCtrl.sv  
    └ /VGA   
    ├ Distance\_display.v  
    ├ Letter\_cm\_display.v  
    ├ Minus\_display.v  
    ├ Reset\_Delay.v  
    ├ Speed\_display.h  
    ├ Time\_display.v  
    ├ VGA\_Controller.h  
    └ VGA\_Param.h
2. System Architecture
3. Block Diagram

  
Whole System  
  
  
VGA  
  
Ethernet  
  
  
SDRAM

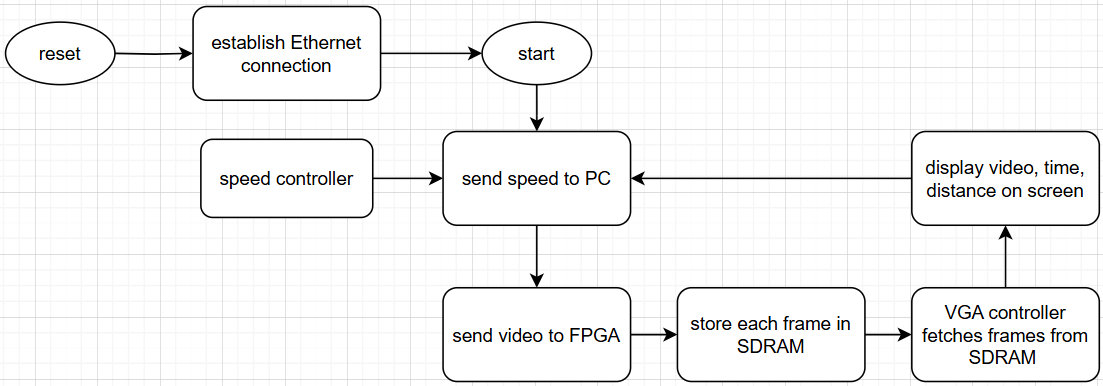
1. Data Path

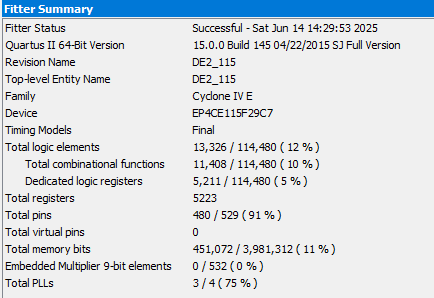


1. Hardware Scheduling  
   1. Finite State Machine

  
UDP Parser  
  
  
ADS1115

* 1. Flow Chart



1. Fitter Summary  
   
2. Timing Analyzer

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. Reflection and Feedback  
   1. Reflection  
        
       At the beginning of our final project, we assumed that implementing the basic functions would be relatively straightforward. However, we encountered a number of unexpected challenges as we began working through them.  
        
      We initially had a USB controller that could connect to the Nintendo Switch, but it did not adhere to any known protocol, making it difficult to interpret its communication patterns. As a result, we decided to build our own controller from scratch. We considered using a rotary encoder to detect input signals, but since it only measures relative motion rather than absolute position, we ultimately opted for a potentiometer. To process its output, we used the ADS1115 analog-to-digital converter, which communicates with the FPGA via the I²C protocol. Unfortunately, the ADS1115 datasheet lacked a clear explanation of the I²C communication details, so we relied on an Arduino library and a logic analyzer to decode the messages successfully.  
        
       Next, we tackled the Ethernet connection between the FPGA board and our PC. We followed instructions from a GitHub repository outlining how to implement Ethernet communication on an FPGA. However, our unfamiliarity with computer networking made it difficult to configure the settings and set up the necessary clock signals. After extensive trial and error, we managed to establish a connection. One limitation we encountered was our inability to control the timing of outbound messages from the FPGA. The original setup only allowed the FPGA to send a response immediately after receiving data from the PC. Although we could modify the content of the response, we couldn't decouple it from the timing of the incoming message. Our workaround was to have the PC transmit data at a higher frequency, enabling the FPGA to maintain near real-time communication.  
        
       Lastly, we worked on utilizing the SDRAM on the DE2-115 development board. Although we started with the board’s sample code, we struggled to control the SDRAM's read and write addresses. As a result, we were only able to use it as a simple buffer. Our original plan was to regulate the output frame rate by selectively reading from specific addresses, but due to these limitations, we had to rely on a Python script to control the frame rate externally by adjusting the speed at which frames were sent.  
        
       Although we technically achieved all the intended functionalities, we had to employ several workarounds to overcome challenges we didn’t fully understand. In the future, we hope to deepen our understanding of Ethernet communication and SDRAM operations so that we can leverage their full potential and design more robust systems.
   2. Feedback  
        
       Over the course of this semester, we became more familiar with FPGA development and Verilog programming. However, we spent a significant amount of time trying to understand—and at times merely manage to implement—the Ethernet and SDRAM functionalities. To help future students overcome similar challenges more efficiently, we recommend that the TAs offer additional lectures or curated learning resources focused on these topics. This would greatly enhance our ability to implement complex features more effectively and with greater confidence.