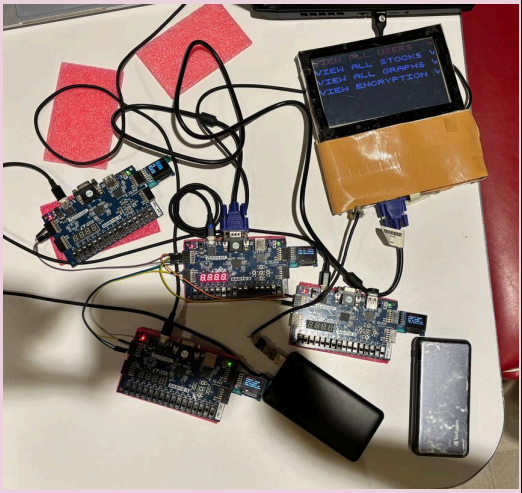
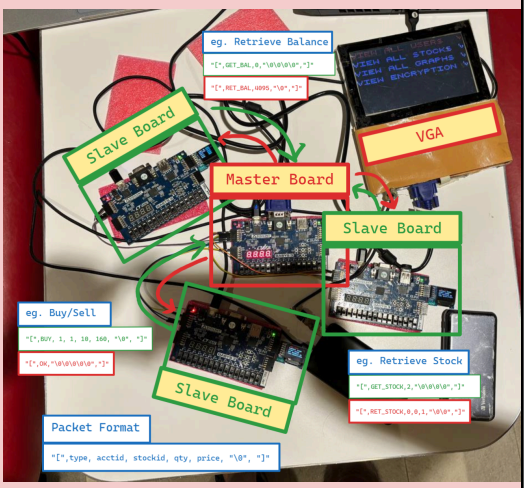
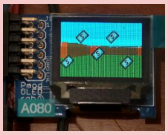
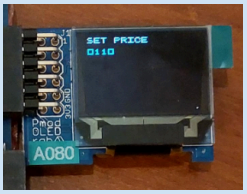
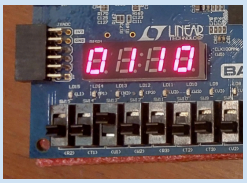
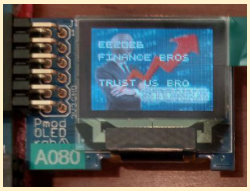
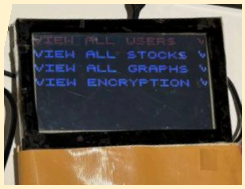
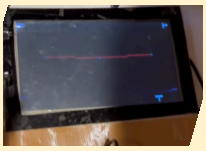
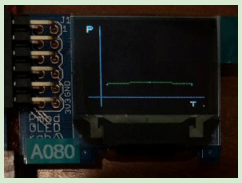
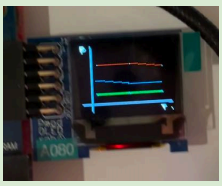
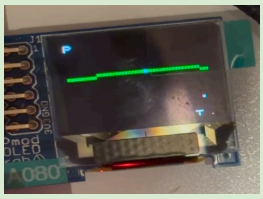
| PERSONAL AND TEAM IMPROVEMENTS | | |
|---|---|---|
| Student and Improvement Name | Improvement Description | Images/Photos |
| Team<br>“S1-04”<br>Monday Lab<br><br>EE2026<br>Finance Bros | Our team implemented a basic trading simulation with price adjustments and visualisations. There are 4 boards, 1 master board and 3 trader/ slave boards. At the start, the boards would be configured with **switches**. The boards can then access the various functionalities through **menus**.<br><br><u>The master board</u> would display tables and graphs about the stock prices, user balances & stock quantities. A **mouse** would be plugged in to control the graphs. <u>The trader/ slave boards</u> should first have their account IDs set. They would be able to **buy and sell** stock. A simplified form of **price adjustment** would be applied. With the right strategy, traders would be able to earn profits (in the simulation). All the communications can be **encrypted**, and the visualisations can be shown on the **VGA monitor** connected to the master board.<br><br>**sw[0]**: Master Mode,<br>**sw[1]**: Trader/ Slave Mode<br>**sw[2]**: Animation Mode |  |
| Student A:<br>Chan Zun Mun Terence<br><br>UART Communications, Market Movement Logic & Trading System | <u>UART Communications</u><br>Integrated the **UART** Module to send and receive 8-byte packets (64-bit) using a shift register. Designed the packet format, which allows the traders to buy, sell, get balance and current stock holdings over UART **with a suitable response** from the master (OK, FAIL, RETURN_BALANCE, RETURN_STOCK)<br><br>Pinouts: **JA1, JA2** (rx0, tx0), **JA3, JA4** (rx1, tx1), **JA7,JA8**(rx2, tx2) rx0 and tx0 of the slave boards would connect to the tx<num> and rx<num> on the master board respectively. The boards would be grounded<br><br><u>Market Movement & Verification</u><br>For every 2 units of stock bought, the price increases by $1. For every 2 units of a stock sold, the price decreases by $1. Traders would be unable to buy stocks that they do not have the funds for, and would be unable to sell stocks they do not have. A green screen is shown if the trade went through.<br><br><u>Animation Mode</u><br>Implemented a raycast view of a room to move around in. Has an animation of money bills falling in<br>**btnL, btnR**: Rotate left and right<br>**btnU, btnD**: Move forward and back<br>**btnC**: Hide money bills (why would you do it?) |  <br>  <br>  <br>  |
| Student B:<br>Low Tjun Lym<br><br>Menu Layout, Encryption | <u>Menus</u><br>Handled the menu layout, UI and FSM, Configuration of the various parameters (eg. account id, price, qty, operation type) and selection of the various menu options is possible through buttons.OLED and/or **7 segment display** is used to display number input for ease of visibility<br>1. **btnU, btnD, btnL, btnR**: Change selection option/ value<br>1. **btnC**: Select item/ Confirm/ Go back to main Menu<br>2. **sw[15], sw[14], sw[13]**: Set increments to be in terms of 1000, 100, and 10 respectively (Selection of numbers)<br><br><u>Encryption</u> |  <br>  |

| | | |
|---|---|---|
| | Based on the **Data Encryption Standard (DES**). I separated the plaintext message into two halves, and through a series of **XOR** and **bitshift** operations performed encryption on the packets sent via UART. They are then decrypted on the other board. A state is created on both slave and master boards to view the encryption on rx0 of the master. On the slave board (left), there are the Unencrypted, Encrypted and Decrypted packets to prove that the encryption is two way, and the master board (right) has the encrypted packet received and the decrypted packet to show that the encrypted packet is sent through.<br><br>On the slave board, **sw[8]** enables encryption of packets to send and **sw[7]** enables decryption of received packets. On the master board, **sw[10:8]** enables the encryption of packets to be sent to the 3 slave boards, and **sw[7:5]** enables the decryption of the received packets from the slave for slaves 2, 1 and 0 respectively. |  <br>  <br>  <br>  |
| Student C:<br>Sparsh<br><br>Text Generation, VGA graphics | <u>Dynamic Text</u><br>Tables on the OLED **update dynamically** depending on the current price of the stock/ balance/ stock quantity held by the users. Up to 7 lines of text 15 characters long can be displayed and changed on runtime while minimising LUT utilisation.<br>**btnL/btnR**: Switch between tables (for users only)<br>**btnC**: return to main menu<br><br><u>Images</u><br>Prepared a Python Script to generate an image mem file, which can then allow us to display an image on the OLED/ VGA.<br><br><u>VGA</u><br>VGA display can be used to mirror the OLED or for displaying graphs while the OLED is used for other visualisations. This is done with the help of an intermediary buffer. Conversion of the colours as well as scaling was done from the OLED to the VGA.<br>**sw[3]**: Switch between mirroring OLED & graphs on the VGA |  <br>  <br>  <br>  <br>  |
| Student D:<br>Alam Ahmed Shaiyan<br><br>Graphs | <u>Table lines:</u><br>Drawing table lines based on x and y coordinates to use in graphs and menu for clarity<br><br><u>Graphs</u><br>The past (3) prices of the stock(s) will be displayed on a graph. Lines are plotted on the OLED with the help of the formula **y=mx+c**. I also used Sparsh's text module to label the Price and Time axis on the graphs<br>**btnL/btnR**: Switch between graphs (single line, or multiple lines with flickering)<br>**btnC**: return to the main menu<br><br><u>Zoom</u><br>**Moving** the mouse cursor and **Right clicking** on any section in the graph would allow the display to zoom in. The zoom factor would follow the position of the mouse cursor.<br>**Left Click** would exit zoom<br>The zoomed view on the graph updates dynamically without the need to zoom out first. The frame adjusts instantly to the cursor's new position. |  <br>  <br>  <br>  |

References:
https://github.com/FPGADude/Digital-Design/blob/main/FPGA%20Projects/UART/baud_rate_generator.v
https://github.com/FPGADude/Digital-Design/blob/main/FPGA%20Projects/UART/uart_transmitter.v
https://github.com/FPGADude/Digital-Design/blob/main/FPGA%20Projects/UART/uart_receiver.v
https://github.com/FPGADude/Digital-Design/blob/main/FPGA%20Projects/UART/uart_top.v (Modified significantly)
https://github.com/FPGADude/Digital-Design/blob/main/FPGA%20Projects/UART/fifo.v (Modified significantly)
https://github.com/FPGADude/Digital-Design/blob/main/FPGA%20Projects/VGA%20Projects/VGA%20Controller/vga_controller.v