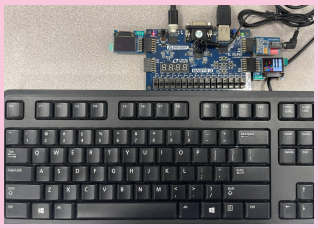
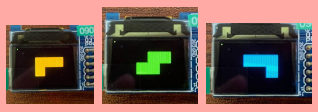
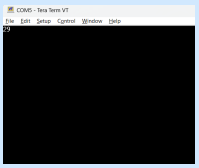
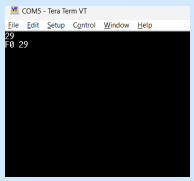
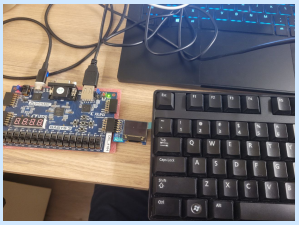
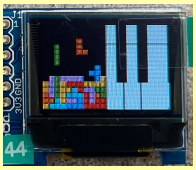
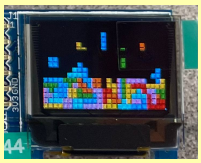
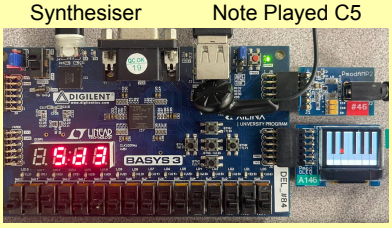
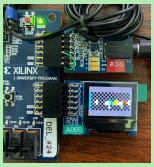
Team S1-06 Chow Cheng En (A0252551L) Davidson Nga (A0252053R)
Vimalapugazhan Purushothaman(A0259193U) Pan Yongjing (A0257283X)

PERSONAL AND TEAM IMPROVEMENTS

Student and Improvement Name	Improvement Description	Images / Photos
Team S1-06 "Entertainment System"	The group theme is a two in one entertainment system consisting of a Tetris game and a monophonic Synchoriser. The system comprises 2 OLEDs, 1 Audio output, 1 Keyboard and the Basys 3 Board. By utilising the buttons, the users will be able to toggle between the main menu, Tetris game or Monophonic Synthesiser and confirm the selections by pressing btnC. The game will allow the user to play Tetris on the OLEDs and keyboard while the Synchoriser plays the note chosen by the user and displays the note.	
Student A: VIMALAPUGAZHAN PURUSHOTHAMAN "Tetrimino Generation & Next Block Display"	While the Tetris game is in play, the Tetrimino spawns randomly from the middle columns of the grid and the next block will be shown on the OLED connected to **JA**. Each tetrimino consists of 4 blocks and one of the 4 is defined as the centre block of the Tetrimino. This centre block would be set to the **initial position** of **t_col = 8** and **t_row = 2** such that it spawns at the **top** of the grid. The remaining blocks are referenced off the central block, thus ensuring that the block remains in the same orientation when **t_col** is updated due to the left / right shifting inputs and **t_row** is updated due to the dropping feature implemented. This is done for all **6 Tetriminos** with a total of 25 states due to the **4 states of rotation** on most Tetriminos - **I**(4), **O**(1), **L**(4), **J**(4), **Z**(4), **T**(4), **S**(4). The Pseudo Random Tetrimino Generator uses a more basic form of **Linear Congruential Generator** where the Tetrimino is from the **last digit** of the seed value (seed % 10) and the seed is updated (seed / 10) and the seed value is updated when it drops below 10. The next Tetrimino is determined in a similar fashion. The second OLED is split into a **4 x 8 grid** and displays the 4 blocks of the next Tetrimino in a similar fashion as how the Tetrimino is represented in the game.	 Random Block generation and its corresponding next block display below.  Examples of Next Block Display
Student B: DAVIDSON NGA "Keyboard interfacing, tetris controls and game logic."	Interfacing with keyboard, able to read and interpret keyboard input data and translate the key inputs to be able to perform appropriate actions by acting as a "switch". The keyboard sends a PS/2 (Personal System 2) scan code. The scan codes transmitted are known as **make codes**. When a key is held down, the make code will be sent continuously, at a rate predefined. When said key is released, an additional scan code is sent over to indicate that action, otherwise known as **break code**. Majority are 8 bits, with exceptions of extended keys. They have a value of **8'hE0** before their make codes.Break codes have the prefix of **8'hF0**. Upon starting the tetris game, the user can: • Press the "**A**" key to **shift left** aka shift the position of the current tetris block to the left. • Press the "**D**" key to **shift right** aka shift the position of the current tetris block to the right. • Press "**Spacebar**" to reset the game at any time. • Press the "**W**" key to **rotate** the block (clockwise rotation). Collision (Dead) block detection (Together with Yongjing) -Check if any part of the current piece has landed By reaching the bottom (b1_row > 22 or b2_row > 22 or b3_row > 22 or b4_row > 22) **Or** The block below any tetromino block is dead (i.e. occupied[b1_row+1][b1_col] or occupied[b2_row+1][b2_col] etc.) If it is true, the current piece is locked in place in the corresponding 2D array (occupied[b1_row][b1_col] <= 1, etc.) Clear row detection (Together with Yongjing) -Check for each row if every column is occupied; occupied[i] == 16'b0001_1111_1111_1000 -Have a counter to add for every column that is fully occupied. - If that counter is not 0, shift down the rows, make the top row empty.	 Key is pressed (LED lights up) Using a serial monitor to test keyboard inputs.  When key is held  When key is released  Connection

Student C: CHOW CHENG EN “Main Menu UI & Sound System”	For the main menu UI, **btnL and btnR allows for the user to toggle between selecting** the Tetris game or Monophonic Synthesiser UI respectively and displaying either the game or piano. **Pressing btnC will confirm the selection** and start the desired system to use. **btnD resets and exits the system** and returns back to the main menu. If Tetris is selected, the **theme song for tetris** will begin to play until btnD is pressed and the game is exited. If Piano (Monophonic Synthesizer) is selected, **sw[13:15] determines the octave** of the notes from octave 3 to 5 respectively. Only after the octave, sw[13:15], is chosen will **sw[0:11] determine the note** being played which are: **C, C-Sharp, D, D-sharp, E, F, F-Sharp, G, G-Sharp, A, A-Sharp, and B** respectively. In order to prevent conflict of the switches, **priority of the octave and note chosen will be given to the lowest sw value**. i.e if sw[13] and sw[14] is on, octave 3, sw[13], is chosen. If sw[0] and sw[1] is on, note C, sw[0] will be chosen. When the note and octave is selected, the **7-segment display shows the octave and note** played. **AN[0]** shows the octave. **AN[1]** shows the note played. **AN[2]** shows if the note is # (Sharp). While the **OLED** displays a **piano indicating the corresponding note** is being played with a red circle played on the corresponding piano key.	 Main Menu  Tetris with audio  Synthesiser  Note Played C5  Note Played #D3
Student D: PAN YONGJING “Tetris controls and game logic”	When the user confirms selection of Tetris by pressing btnC, the Tetris game can be played using **keyboard controls**. Tetris is played on a **10x20 grid**, where **each block is 4 pixels (= 1 grid pixel)** on the OLED screen. Grid pixels of the same row and column parity are the same colour, creating an **alternating grey and black** pattern, for ease of distinguishing them (if **g_row % 2 == g_col % 2**, display grey, else display black) By using a **fall_counter** and **shift_counter** in always@(posedge basys_clock), - The **tetrimino falls by 1 grid pixel every 1/3 second** (= rate of 3Hz). - Pressing and holding **A** causes the tetrimino to **shift left** at 1 grid pixel per 1/3 second. - Pressing and holding **D** causes the tetrimino to **shift right** at 1 grid pixel per 1/3 second. <u>Collision detection (together with Davidson):</u> A **2D array occupied[23:0][15:0]** keeps track of whether a grid pixel is occupied (=dead). The bottom row 23 is initialised to be occupied. When a block becomes dead, its grid pixels become occupied. Thus when a tetromino **lands on the bottom row** or **on top of other “dead blocks"**, it becomes **“dead”** and can no longer be controlled. The next tetrimino will then spawn. <u>Row clearing (together with Davidson):</u> When a row is full of blocks, it will **clear** and **all rows above will shift down.** If **x rows are cleared at once**, all rows above will **move down by x.** When the **grid is filled** such that a **new tetrimino cannot spawn** as it overlaps with “dead” blocks (occupied[b1_row][b1_col], etc.), the **game is over** and the **screen displays the background colour,** blue. Pressing the **spacebar** at any point **restarts** the game. The grid is emptied and the first tetrimino will fall.	 Tetrimino falling, shift left,  Land on bottom row, shift right, Land on dead blocks  Row clear when full  Game Over, restart  Connection to Basys board (JC ports)

References:

- **Convert images to OLED:** https://github.com/nvbinh15/FPGA-Project-EE2026/blob/main/Python_helpers/ee2026_converter.ipynb
- **Star Wars Song:** https://github.com/FPGADude/Digital-Design/tree/main/FPGA%20Projects/Star%20Wars%20Imperial%20March%20Song
- **Block Generation**: https://github.com/nus-wira/EE2026-FPGA-Project/blob/master/sources_1/new/calc_cur_blk.v
- **Polyphonic Synthesiser:** https://github.com/ComicAddict/Polyphonic_FPGA_Synthesizer/blob/master/Polyphonic_Synthesizer.srcs/constrs_1/new/constraints.xdc
- **Keyboard demonstration:** https://github.com/Digilent/Basys-3-Keyboard/tree/v2018.2-3