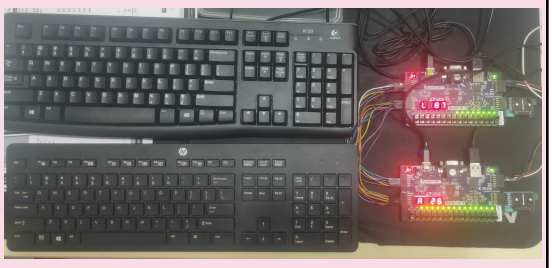
Group ID: S1_12

Members: Sim Jun Hong A0252616H, Foo Shi Xiang A0252250U, Daryl Tay Chin Kian A0252257E, Eugene Chan Jiajun A0258443Y

PERSONAL AND TEAM IMPROVEMENTS

Student and Improvement Name	Improvement Description	Images/Photos

Team — 2-Player Serial Communication Gun Game



Our game is a 2D shooter game that is similar to dodgeball. Each player will start off with 16 health and the game will only end when either player loses all their health.

There will be a line in the middle of the screen to separate the 2 players in the game. Each player can shoot a bullet to the opposing player and try to damage the opponent.

Players will be able to control their sprites using the keyboard.

- WASD: For sprite movement along x-y axis
- Enter: To shoot a bullet

There will be various weapons and pickups in the game that players can collect to enhance their shooting ability or gain special effects. The player's health is displayed on the LEDs of the Basys3 board. For example, 16 LEDs on = Full Health. The current gun ammo/reload status is displayed on the 7-segment display.

There will be a ready indicator on both screens before the game starts. Once both players are ready and both player switches are turned on, either player can click on btnC to start the game.

The Basys boards are connected through serial communications to send and receive data such as keyboard inputs from either of the players.

Student A: Sim Jun Hong — Weapons (ammo, cooldown, reload, damage, player health), pickups, hitboxes, collisions and pickup data transfer protocol

Both players start with a pistol and 16 health. Pickups with a **random type and position** will spawn along the middle line. If a pickup is currently active, no other pickups will spawn. Else, a 10s counter will start and the next pickup will spawn. When a player **presses enter on his keyboard**, a bullet (from any non-laser weapon) will fly towards the opposite end of the screen. If the bullet collides with the opposing player, he will lose a **specified amount of health depending on the weapon that fired the bullet**. If the opposing player's health is reduced to 0, the game ends. If the bullet collides with a pickup, the player will get a **new weapon or effect depending on the pickup type**. If the pickup type is the same weapon that the player is currently using, the player's ammo will be fully restored. The pickup icon will **despawn on collision**. Bullets will always **despawn on collision**, else they will fly to the edge of the screen. The laser beam will **travel to the nearest target available** (opposing player/pickup) in a straight line, else it will go to the edge of the screen. When a player runs out of ammo on his current weapon, there will be a **1.5s reload time** and he will **automatically switch back to the pistol with full ammo**.

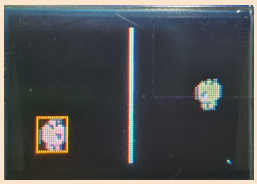
Pickup	Icon	Functionality
Rifle/ Sniper	Chest	Shoots projectile that can be used to damage enemy player or collect pickups
Laser		Continuous beam that can be used to damage enemy player or collect pickups
Shield	Yellow Box	Gives player a shield effect that **reduces incoming damage by half and prevents knockback**. Lasts for 8s. Yellow square around hitbox of player while shield is active
Health	Green Plus	Restores 4 health to player

Pistol: Damage: 2 Ammo: 7, Cooldown between bullets: 0.5s, Projectile speed: 1 pixel at 200Hz
Rifle: Damage: 2, Ammo: 30, Cooldown between bullets: 0.125s, Projectile speed: 1 pixel at 200Hz
Sniper: Damage: 16, Ammo: 5, Cooldown between bullets: 1s, Projectile speed: 1 pixel at 300Hz
Laser: Damage: 1 damage every 0.25s of **continuous collision** (0.5s with shield), 0.5s of **continuous collision** to get pickup, Ammo: 100, Cooldown between bullets: 1 ammo depleted every 0.05s, Projectile Speed: Instantaneous

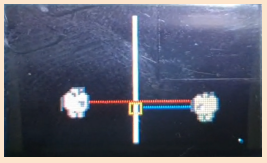
The 9 bits of pickup data from the **LFSR random number generator** needs to be transferred from one Basys board to the other since they are not in sync. During the 10s where the pickup is not active, a transfer wire is set to each bit of the pickup data **every 1s, at t=0,1,2…** On the receiving end, the receive wire is read **every 1s, at t=0.5, 1.5, 2.5…** This continues until all 9 bits are read.

Left player firing pistol with chest icon active

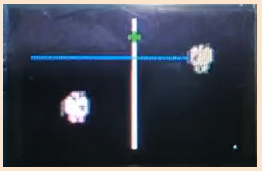
Both players firing rifle



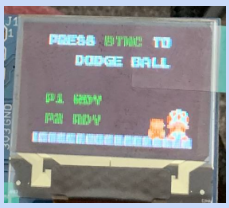
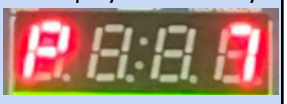
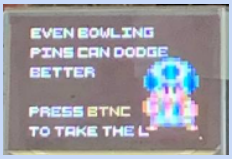
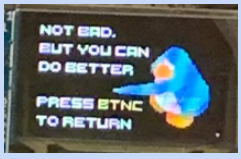
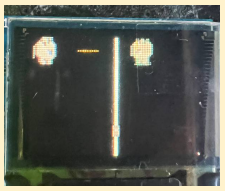
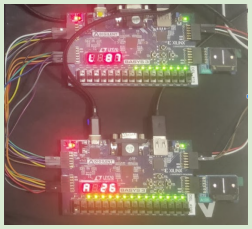
Left player with shield



Left player firing laser at right player, right player firing laser at shield



Right player firing laser to the end with health pickup active

Student B: Foo Shi Xiang Simple Keyboard Interface Pre-game screen Post-game screen Player HUD (with 7 seg and leds)	**Keyboard Interface**: Keyboard inputs (WASD + Enter) are mapped to a 5 bit value by reading keyscan codes. This module made use of the keyboard demo PS2Receiver code (Kappenman, 2015) to shift in keycodes. **Pre-game state**: Player 1: led[15] (most left led) , Player 2: led[0] (most right led) Depending on the player ready state, the leds will **fade in and fade out** in a continuous manner to indicate the respective player is not ready / the game is not ready to be start (Refer to student's D player state setting). The leds **will stay bright** if the respective player state is confirmed to be ready to start. A startup screen on the oled, **depicting mario and toad from SMB1 and some text** will be shown to visually indicate the player states. If the game is ready to be start (Both players rdy), the text will direct players to "Press btnC" to start the game. **In game - Player HUD:** A double dabble algorithm FSM code (Macfie, 2020) is used to convert required binary values to BCD for easier implementation. The 7 Segments **display the current weapon type** (Pistol (P), Rifle (A) etc) and **the ammo remaining for the player**. Once ammo runs out, **the words RE and LOAD flashes** to indicate the player is reloading. If the game round is determined with a winner, the words "ggEZ" and "LOSE" will show on the respective winner and loser sides on the basys board. At the same time, the player health (from 0 to 16) is represented by the amount of leds ON on the respective basys board. **Post-game screen**: A postgame oled screen with **texts and some character animations** will be shown to the winner and loser. The player can then move to the pregame state by pressing 'btnC' **References:** Kappenman, T. (2015) Nexys4DDR Keyboard Demo. https://github.com/Digilent/Basys-3-Keyboard/blob/v2018.2-3/src/hdl/PS2Receiver.v Macfie, J. (2020) VerilogDoubleDabble. https://github.com/josh-macfie/VerilogDoubleDabble/blob/master/BCDConvert.v	 Both players are ready  Pistol with 7 ammo   7 Segments display  Player lost screen  Player won screen
Student C: Daryl Tay Chin Kian Player movements, Player animation, Player collision physics	**Player movements**: When the player presses the keys, w, a, s ,d on the keyboard, the player sprite will move in the direction of the key pressed. For example if the player presses the w key, the player sprite will move upwards. The players are able to move freely in all directions. When the game restarts, the player sprites will move back to the initial positions and the game will then start again. **Player animations**: When the player sprites moves, there is an animation for the player sprite when it moves. The player character is represented by two bunny sprites, one in pink and the other in green, with corresponding animations. **Player collision physics**: There is a collision between the player and the boundaries of the screen to prevent the player from moving out of the OLED screen. In the game, a white line will prevent the players from crossing over to the opponent's sides. There is also a knockback effect when the bullet from one player collides with the opposing player. **References for sprites**: https://totuslotus.itch.io/characterpack	  Knockback effect
Student D: Eugene Chan Jiajun Data transfer, synchronisation of clock	Both Basys 3 boards will be **connected via the PMOD** ports to transfer input data between both players to both the boards to enable **real time updates** of the same game. Both OLED screens will be **connected on the same PMOD port** to output the same game but for different players and the screen will update the opponent behaviour based on the inputs received. Players can either switch on **sw[0] or sw[15] but not both** to indicate which side of the game they will be at (**Switching both on would indicate both players are not ready**). The ready state will update accordingly on both boards (Eg: when player 1 is ready on board 1, both board 1 and board 2 will show that player 1 is ready vice versa). Once both players are ready, then the game can be started. Based on the active switch, players would control their respective sprites tied to the switch (Refer to Student B for HUD). Depending on whichever switch is turned on, (Eg sw[0] is on) that board would transmit its own state and input data to the opponent board and receive state and input data of the opponent. For **clock synchronisation**, a player with sw[0] set as active is the **main driver board** which will send the clock signal over to the board with sw[15] set active. A **check has been set** such that whichever board is player 1 or 2, they will take in the appropriate clock signal from either the transmit or receive ports.	 Connected Basys boards