Project Proposal: Memory Game

Overview (not many changes made from initial proposal, I put this here just for reference)

The Memory Game is a logic-based game where the system generates a random sequence of LED flashes that the player must replicate. With each round, the sequence grows longer and the player must match it by pressing corresponding buttons. If the player makes a mistake, the game ends, and their final score, which is based on the round number, is displayed.

Core Functionality

1. **Pattern Generation**:
    - A random sequence of LED flashes will be generated at the start of each round.
    - Each round will add one new LED flash to the sequence, increasing the length with each level.
2. **Player Input**:
    - The user will interact with the game by pressing buttons corresponding to the LEDs. The buttons will allow the player to replicate the LED sequence generated by the system.
3. **Game Logic**:
    - The system will compare the player's input against the generated sequence to check for correctness. If the player's input matches the sequence, the game progresses to the next round. If there is an error, the game ends, and the final score will be displayed.
4. **Scoring**:
    - The score will be tracked by the round number.
5. **Display**:
    - The LEDs will display the pattern generated by the system.
    - A seven-segment display will show the current score or a start message.

Rough Design Descriptions

1. **Finite State Machine (FSM)**:
    - The FSM will manage the game flow, transitioning through various states:
        - **Initial state**: Wait for the start button to begin the game.
        - **Pattern generation**: Create and display a random LED sequence.
        - **Player input**: Wait for the user to press buttons and validate the input.
        - **Error checking**: Check if the player's input matches the sequence.
        - **Round increment**: If correct, increment the score and generate a new pattern.
2. **Pattern Memory**:
    - The sequence of LED flashes will be stored in a shift register to maintain the pattern through each round.

Game Modes

1. Classic Mode

- Objective: Replicate the growing LED sequence correctly with each round
- Gameplay: The game generates a random LED sequence, and the player must replicate it using button presses. Each round, the sequence gets longer. The game ends if the player makes a mistake, and their score is displayed.

2. Time Challenge Mode

- Objective: Replicate as many sequences as possible within a limited time.
- Gameplay: The player has a set amount of time (I'm thinking within this I can have 3 options like 20s, 40s, 1min but for now start with 1min) to complete as many rounds as possible.

3. Speed Memory Challenge Mode

- Objective: Replicate the sequence of LED flashes, where the pattern gets longer and the flashes become faster with each round.
- Gameplay: The game generates a random LED sequence, and the player must replicate it using button presses. Each round, the sequence gets longer and the flashes get faster.

4. Reverse Mode

- Objective: Replicate the sequence in reverse order.
- Gameplay: After the system generates the LED sequence, the player must replicate the pattern, but in reverse order.

I/O Description

1. **Inputs**:
    - clk: clock
    - rst_n: reset
    - player_input[15:0]: 16-bit signal to capture the player's button press
    - start: signal indicating when the game should begin
    - Select[3:0]: to select a gamemode
2. **Outputs**:
    - leds[15:0]: 16 LEDs used to display the generated pattern
    - display[7:0]: 7-segment display output for the current score or game status.

Hardware Design Sketches & Thought Process

FSM

- **Initialization State**: The game will begin with a reset, initializing the necessary components (LEDs, buttons, and 7-segment display).
- **Mode Selection State**: After initialization, the FSM will wait for the player to select a game mode via the Select input, to choose between:
    - Classic Mode
    - Time Challenge Mode
    - Speed Memory Challenge Mode
    - Reverse Mode
- **Pattern Generation State**: The game will generate a random LED sequence. This state will be responsible for adding one more LED to the sequence in Classic Mode and adjusting the flashing speed in Speed Memory Challenge Mode.
- **Player Input State**: This state will wait for the player to input their guesses via the player_input buttons. For Time Challenge Mode, the timer will continue to count down.
- **Error Checking State**: After the player has entered their input, the FSM will compare the player's input to the generated sequence. If the input is correct, the game transitions to the next round; if incorrect, the game ends and the final score is displayed.
- **Round Increment State**: In Classic Mode, the sequence grows by one LED each round. In Speed Memory Challenge Mode, both the sequence length increases, and the flashing speed increases. The score (round number) will be incremented, and the display will reflect the current score.

Components needed

1. Register to store pattern
2. Counter for time challenge mode
3. Counter to increment round number
4. Component to adjust the speed of the flashes
5. Random number generator to generate the LED sequence

Hardware Peripherals (to purchase)

- **Display**: To display the current round number or a start message
- **input buttons**: Used for player input to replicate the LED pattern (I want to use the ones already on the FPGA to control game mode selection/start and reset)
    - Adafruit NeoTrellis RGB Driver PCB for 4x4 Keypad
    - Silicone Elastomer 4x4 Button Keypad - for 3mm LEDs
- Speaker: it would be nice to play some sounds/song while the game is playing but I'm not sure I will get to this