Boston University
College of Engineering

ENG EC 311 – Introduction to Logic Design
Section A1

Whack-a-Mole on FPGA

An LED Reaction Game on the Nexys4 DDR

Final Project Report

Student:	Parth Kheni
Team Members:	Hongming Du
	Cankun Wang
	Joe Nguyen
Professor:	Prof. Ajay Joshi
Course:	ENG EC 311 – Fall 2025
Group:	19
Date:	December 15, 2025

Source code repository: https://github.com/parthnkheni/whack-a-mole-fpga

1 Introduction

Field-Programmable Gate Arrays (FPGAs) provide a flexible platform for implementing digital systems directly in hardware. For this project we designed and implemented a Whack-a-Mole style reaction game on the Digilent Nexys4 DDR FPGA board (Artix-7 xc7a100t). Instead of simulating the design only in software, our goal was to build a complete hardware system that interacts with real buttons, switches, LEDs, and seven-segment displays on the board.

In the final game, five on-board LEDs represent mole positions, the player selects a position using the corresponding slide switch, and a push-button acts as a "hammer" to hit the mole. The system handles a pre-game countdown, a fixed-length game window, difficulty selection, scoring, and output on the seven-segment display. Designing this system required us to apply most of the major topics from ENG EC 311: finite state machines (FSMs), counters and timing, debounced inputs, modular Verilog design, and top-level integration on an FPGA.

This report describes the behaviour of the game, the hardware and timing specifications, the architecture and module-level design, and our verification strategy. We also summarize the main challenges and lessons learned while bringing the design from block diagrams to a working hardware implementation.

2 Project Overview

The Whack-a-Mole game implements a simple reaction-time task on the Nexys4 DDR board:

- Five LEDs form a horizontal "mole row." At any instant, exactly one LED is lit to indicate an active mole.
- Beneath each LED there is a dedicated slide switch. To hit the current mole, the player flips the matching switch and presses a hammer button.
- A seven-segment display shows either a five-second pre-game countdown or the player's score.
- The game lasts for a fixed 30-second window. During that time, moles appear one at a time in pseudo-random positions, and the player tries to hit as many as possible.
- A difficulty setting controls how long each mole remains visible; higher difficulty corresponds to shorter visibility windows.

From reset to game over, the system progresses through three main phases:

1. **Countdown:** After the player presses start, a five-second countdown (5, 4, 3, 2, 1, 0) is shown on the seven-segment display.
2. **Playing:** Once the countdown reaches zero, the game enters a 30-second play phase. Moles appear one at a time. When the player selects the correct switch and presses the hammer button before the mole times out, the score is incremented and a new mole appears.
3. **Game Over:** After 30 seconds the game ends. The final score remains on the display until the player starts a new round.

From a design standpoint, the project brings together many subsystems: input processing and debouncing, clock division and counters, a central game-control FSM, mole control and pseudo-random selection, scoring, and seven-segment display driving. The top-level Verilog module instantiates these blocks, connects them through well-defined interfaces, and maps them to the physical pins of the board via an `.xdc` constraint file.

3 Requirements and Specifications

3.1 Gameplay Requirements

The functional requirements can be summarized as follows:

- A single 100 MHz on-board clock drives all synchronous logic.
- The game supports a five-second pre-game countdown and a 30-second playing interval.
- Exactly one mole LED in a set of five is active at any time during gameplay.
- The player hits the mole by flipping the matching switch and pressing the hammer button while the mole is still visible.
- The system tracks the number of successful hits (score) and displays it on the seven-segment display.
- A difficulty button cycles through multiple difficulty levels; each level corresponds to a different mole visibility duration.
- A clear/score button resets the score without requiring a power cycle.
- A reset button returns the entire system to the idle state.

3.2 Timing Specifications

All timing is derived from the 100 MHz system clock using counters and enable pulses. Key timing parameters are:

- **System clock:** 100 MHz input clock from the Nexys4 DDR oscillator. Every sequential module (flip-flop based) is synchronous to this clock.
- **Countdown:** A one-second enable tick, derived from the system clock, drives a counter from 0 to 5. The value shown on the seven-segment display is $5 - \text{count}$, so the user sees 5, 4, 3, 2, 1, 0.
- **Game duration:** A second counter tracks elapsed game time from 0 to 30 seconds. When the counter reaches 30, the FSM transitions to the game-over state.
- **Mole visibility:** A difficulty timer, driven directly by the 100 MHz clock, determines how long each mole stays lit. Different thresholds are used for easy, medium, and hard modes to make moles appear slower or faster.
- **Hit registration:** All button inputs are debounced. Hits are detected by sampling the switch vector on a clean, one-cycle hammer pulse and comparing it to the current mole index.

3.3 Top-Level I/O Specification

The logical signals of the design map to the physical interfaces of the Nexys4 DDR board as follows:

- **Clock and reset**
  - `CLK100MHZ`: 100 MHz input clock.

  - `BTNC`: active-high reset button; returns the game to the idle state and clears all counters.

- **Control buttons**
  - `BTNU`: start button; initiates the five-second countdown or restarts the game.
  - `BTNR`: difficulty button; cycles through difficulty levels (easy/medium/hard).
  - `BTNL`: clear-score button; clears the score and game timer.
  - `BTND`: hammer button; used to attempt a hit on the current mole.

- **Switches and LEDs**
  - `SW[4:0]`: five position switches, one associated with each mole LED.
  - `LED[4:0]`: five mole LEDs; exactly one is high when a mole is active.

- **Seven-segment display**
  - `seg[6:0]`: active-low segment lines for segments a–g.
  - `an[3:0]`: active-low digit-enable lines for the four digits.

These I/O definitions are reflected directly in the constraint file used by Vivado to bind top-level ports to the appropriate FPGA pins.

4 System Architecture

4.1 Subsystems

At a high level, the design is composed of the following subsystems:

1. **Button and difficulty interface** (`button_io`, `debounce_one_pulse`): synchronizes and debounces mechanical push-buttons, produces one-clock pulses, and maintains the selected difficulty level.
2. **Clock division and timing** (`clock_divider`, `sec_counter`): generates a 1 Hz enable tick for human-scale timing and a faster scan clock for the seven-segment display; implements the countdown and game-time counters.
3. **Game control FSM** (`game_control_fsm`): coordinates the overall game flow across IDLE, COUNTDOWN, PLAYING, and GAME_OVER states.
4. **Mole logic and random selection** (`random`, `mole_led_ctrl`, `mole_led_and_random`, `difficulty_timer`): chooses mole positions pseudo-randomly, enforces difficulty-based visibility windows, and generates hit and timeout events.
5. **Score and display subsystem** (`score_counter`, `seven_seg_decoder`, `two_digit_7seg`): tracks the player's score and multiplexes the seven-segment display to show score or countdown values.

These blocks are instantiated and connected in the `top` module, which also interfaces to the external pins defined in the constraint file.

4.2 Data and Control Flow

Raw button presses and switch positions enter the button interface and are converted into stable, one-clock pulses. The difficulty register maintained in this block encodes the current difficulty level (e.g., 2-bit code for easy/medium/hard).

The clock divider produces a 1 Hz tick, which drives two instances of the second counter: one implements the 5 s countdown, and the other implements the 30 s game timer. Both counters expose enable and clear signals that are driven by the game control FSM.

The game control FSM observes the debounced start and clear pulses, the countdown and game timer values, and the current score. Based on this information, it decides which state the game is in and accordingly:

- Enables or clears the countdown counter, game timer, score counter, and mole controller.
- Chooses whether the seven-segment display should show countdown time or score.
- Passes the selected difficulty level to the mole subsystem.

In parallel, the mole subsystem operates as follows: the random generator produces a pseudo-random index, which is mapped to one of five LED positions. A difficulty timer counts system clock cycles to determine the current mole's visibility window. If the hammer pulse occurs while the switch vector matches the active mole index, the subsystem generates a hit pulse; otherwise, when the timer expires, it generates a timeout pulse. Either event causes the mole controller to clear the current LED and spawn a new mole at a different index.

The score counter increments on each hit pulse when scoring is enabled by the FSM, and can be cleared at the beginning of a new game or when the clear-score button is pressed. The current score, or derived countdown value, is sent to the display subsystem, which converts it to segment patterns and time-multiplexes the physical digits using the scan clock.

5 Module Design

5.1 Button Interface and Debouncing

Mechanical push-buttons exhibit contact bounce, which can produce multiple unintended transitions for a single press. To avoid this, every button used in the design is passed through the `debounce_one_pulse` module. This block synchronizes the input to the system clock, waits for the signal to remain stable for a fixed number of cycles, and then updates an internal level register. A single-cycle pulse is generated on the rising edge of this debounced level, providing a clean event signal to downstream logic.

The `button_io` module instantiates debouncers for the start, difficulty, and clear-score buttons and maintains a small difficulty register. Each time the difficulty pulse occurs in the idle or game-over states, the register cycles through the available difficulty codes. The selected difficulty drives the mole visibility timing used by the difficulty timer.

5.2 Clock Divider and Second Counters

The `clock_divider` module divides the 100 MHz clock down to two lower-frequency signals. One is a 1 Hz enable tick used by the `sec_counter` blocks; the other is a kilohertz-range scan clock used for seven-segment multiplexing. Deriving human-scale timing from a single master clock avoids multi-clock-domain complications.

The generic `sec_counter` module counts seconds whenever its enable input is asserted and resets to zero when clear is asserted. One instance implements the five-second countdown, and another tracks elapsed game time up to 30 seconds. Both counters are driven by the game control FSM.

5.3 Game Control Finite State Machine

The `game_control_fsm` is the central controller of the system. It operates with four states:

IDLE: All counters and the mole subsystem are disabled and cleared. The score display reads zero, and difficulty can be adjusted. A start pulse transitions the FSM into COUNTDOWN.

COUNTDOWN: The countdown counter increments once per second. The seven-segment display shows the remaining countdown time. When the counter reaches five seconds, the FSM enters PLAYING. Restarting the game from this state is possible by pressing start again.

PLAYING: The game timer counts from 0 to 30 seconds. The mole logic and score counter are enabled, and the seven-segment display shows the current score. When the timer reaches 30 seconds, the FSM transitions to GAME_OVER. Pressing start early aborts the current game and restarts the countdown.

GAME_OVER: The mole subsystem and timers are disabled; the final score remains visible. Difficulty may be changed. Pressing start initiates a new countdown and game.

This FSM cleanly separates high-level game flow from low-level timing and mole behaviour, making the design easier to reason about and test.

5.4 Mole Logic and Random Selection

The mole subsystem is implemented by the combination of `random`, `mole_led_ctrl`, and `difficulty_timer`. An LFSR-based random generator produces a pseudo-random 3-bit value. This value is mapped into the range 0–4 and interpreted as the next mole index. The `mole_led_ctrl` block decodes this index into a one-hot LED vector so that exactly one of the five LEDs is lit at a time.

The `difficulty_timer` counts raw system clock cycles while a mole is active. Its threshold depends on the difficulty code; for example, easy mode may use a longer threshold (more cycles) than hard mode. When the timer reaches the threshold, it asserts a timeout pulse and clears the active mole. If a hammer pulse occurs while the player's switch selection matches the active index, a hit pulse is generated instead, and the score counter is incremented.

By confining random selection and timing to this subsystem, we can tune mole behaviour without modifying the rest of the design.

5.5 Score Counter and Display Subsystem

The `score_counter` is an 8-bit up-counter that increments on each hit pulse while enabled and resets on clear or at the start of a new game. To keep the display simple, we restrict scores to the range 0–99, which fits on two decimal digits.

The display subsystem splits the 8-bit value into tens and ones, decodes each digit with `seven_seg_decoder`, and uses `two_digit_7seg` to multiplex the two digits across four physical positions on the Nexys board. A scan counter driven by the scan clock cycles through digit enables fast enough that the human eye perceives a steady two-digit number. During the countdown phase, the FSM provides the countdown value instead of the score to the display subsystem.

6 Implementation and Constraints

The design targets the Artix-7 xc7a100tcsg324-1 device on the Nexys4 DDR board. Vivado is used for synthesis, implementation, and bitstream generation. A custom constraint file binds top-level signals (`CLK100MHZ`, buttons, switches, LEDs, and seven-segment lines) to their corresponding FPGA pins and sets the appropriate I/O standards.

All logic runs on the single 100 MHz clock domain. Human-scale timing is produced using counters and enable pulses instead of additional clock sources, which simplifies timing closure. Implementation reports show that the system meets timing with comfortable slack, and design rule checks pass once all pins are properly constrained.

7 Verification and Testing

Verification took place at two levels: simulation and hardware testing.

7.1 Simulation

Module-level testbenches were written for key blocks such as the second counter, score counter, game control FSM, and mole controller. To keep simulations short, timing parameters (such as countdown and game duration) were scaled down. Testbenches checked correct counting, proper state transitions, enabling/clearing behaviour, and hit detection based on synthetic button and switch inputs.

At the top level, a behavioural simulation verified that the game sequence (IDLE $\rightarrow$ COUNTDOWN $\rightarrow$ PLAYING $\rightarrow$ GAME_OVER) matched expectations and that moles, scoring, and display updates reacted properly to stimuli.

7.2 Hardware Testing

After simulation, the bitstream was loaded onto the Nexys4 DDR board. We performed interactive tests:

- Resetting the system and starting a new game from each state.
- Confirming that the countdown lasted approximately five seconds and the game phase approximately 30 seconds.
- Verifying that exactly one mole LED was lit at any time and that difficulty changes affected mole speed.
- Checking that the score incremented only on correct hits when both the matching switch and hammer button were used.
- Ensuring that the final score remained stable after the timer expired and that the clear-score button worked as intended.

Observed behaviour on hardware matched the expected design, providing confidence that the Verilog implementation, constraints, and timing were all correct.

8 Design Challenges and Lessons Learned

Several practical issues arose during development:

- **Button bounce and spurious events:** Early tests showed multiple hits or game restarts from a single button press. Introducing systematic debouncing and edge detection through `debounce_one_pulse` eliminated these glitches.
- **Seven-segment polarity and multiplexing:** The active-low nature of the segment and digit lines initially caused confusing behaviour (only some digits lit or inverted patterns). Once we accounted for polarity and implemented a proper scan counter, the display became stable.
- **Timing selection for gameplay:** Choosing the countdown, game duration, and mole visibility windows required experimentation to balance playability with demonstration value. Very short mole windows made the game frustrating; overly long windows made it trivial.
- **Pin constraints and board mapping:** Small mistakes in the constraint file (e.g., swapped pins or incorrect names) led to unresponsive inputs. Carefully cross-checking against the Nexys4 DDR reference manual resolved these issues.

From this project we learned the importance of designing block diagrams and FSMs before diving into code, keeping modules small with clear interfaces, and relying on both simulation and hardware testing to debug and validate the system. We also experienced the value of version control and team communication when multiple people are working on different parts of the same FPGA design.

9 Conclusion

We successfully implemented a fully functional Whack-a-Mole reaction game on the Nexys4 DDR FPGA board using Verilog. The final system includes a five-second pre-game countdown, a 30-second gameplay window, difficulty-dependent mole visibility, robust hit detection using switches and a hammer button, score tracking, and real-time seven-segment display output.

By structuring the design into reusable modules and coordinating them through a central FSM, we were able to reason about, simulate, and debug the system effectively. The project demonstrates how concepts from ENG EC 311—including FSM design, counters, debouncing, clock division, and modular hardware design—can be combined to create an interactive, real-time digital system that runs entirely in hardware.