Alan Dautov
Patrick Peat
Thomas Bowler
Vian Ambar Agustono
Professor Joshi
Section A1

# Logic Design Final Project: Whack-a-Mole

Alan Dautov, Patrick Peat, Thomas Bowler, Vian Ambar Agustono

Professor Joshi

EC311

10 December 2021

**Project Overview/Design Functionality**

The goal of this project was to implement a simplified variant of the arcade game Whack-a-Mole onto the Xilinx Nexys 4 Artix-7 FPGA board, where a randomized set of LEDs act as the moles and the buttons act as the hammers. Our baseline list of features for project completion was to have the game starting according to a five second timer and ending according to a thirty second timer, a single randomly activated LED blinking for an entire clock cycle acting as a mole, the LEDs deactivating if button input is received while it is on, and for an incrementing counter to display the score on the onboard seven-segment display whenever an LED is successfully turned off.

From the beginning we intended for our project to be centralized entirely onto the onboard mechanics of this specific FPGA board, disregarding the use of external connectivity features like the VGA or ethernet port. While, from a commercial level, the intentionally crude realization of the game seems outwardly too simple, the project served less as a construction of a product and more as a learning experience for the entire team in the process of embedded system design from conceptualization all the way to the final implementation. We are in strong faith that the single-minded focus on utilizing only the FPGA served us much more as undergraduate students than if we had instead worked from the mindset that the project was a profession.

## Simplified Logic

### Pre-Game[1]

From a pseudo-start state, initialization of the game by turning on the FPGA or toggling the reset switch begins a counter that counts down from five and displays the passing seconds on the segmented display. Once the timer ends, control is passed to the game state.

### Game

When the game state is reached, a second counter separate from the earlier five second counter begins, now counting down from thirty and retaining the value within the FPGA rather than displaying it on the segmented display.

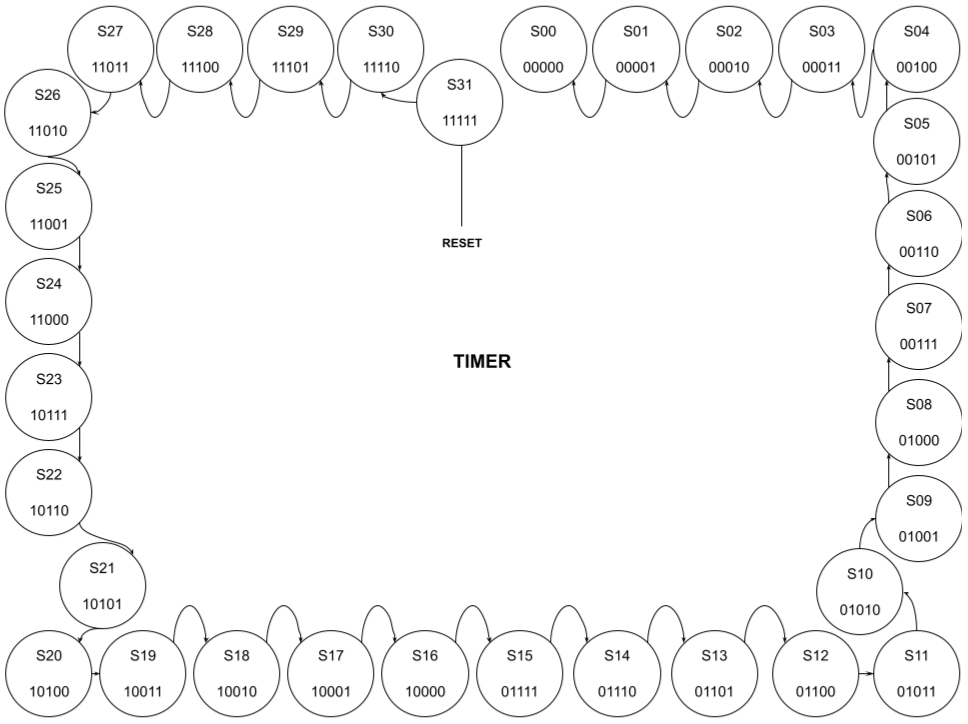
One of the first five LEDs is selected to turn on per second and at random via a linear-feedback shift register. LEDs are on HIGH for one second before turning off as another LED is chosen. Each of the five face buttons on the FPGA is bound to a single LED. Should the LED's state and its corresponding button's state be HIGH at the same time, a signal is sent to a score module that increments the user's score by one and displays this value onto the right side of the segmented display.

When the thirty second counter ends, the game is concluded and the FPGA stops receiving input from the buttons. From here the user can either turn the board off or use the reset switch to return control to the pre-game state.

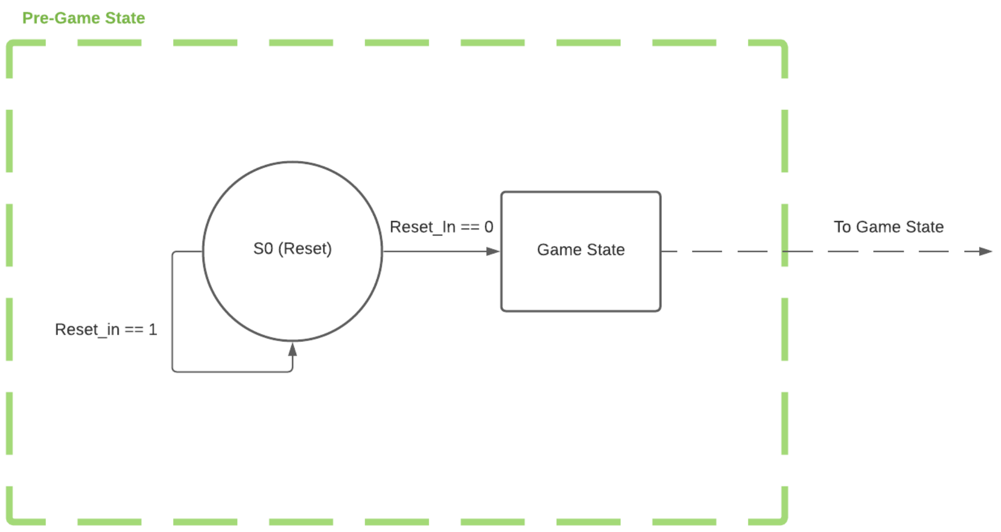
[1] The distinction between a Pre-Game and traditional start state is that the purpose of the pre-game state is to act as a transition between the player requesting to play and the actual game loading. While we were unable to implement a proper start state (Would await for player input to begin pre-game state), this does not stop the pre-game state from being no more than a transition; the transition being between the game ending and the player using the reset switch.

**Finite State Diagrams**

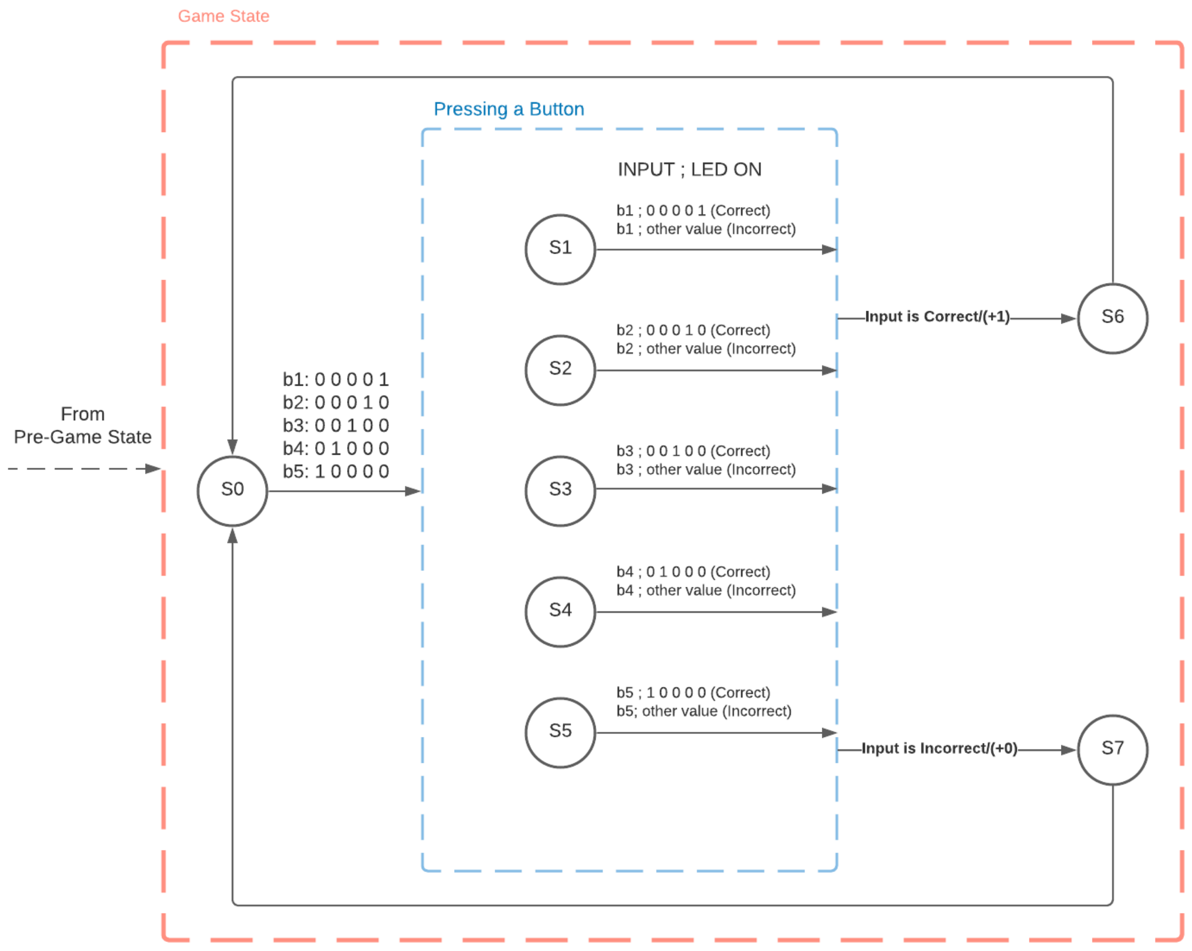
Timer Diagram



Pre-Game Diagram



Game Diagram



For visual clarity, the FSM diagram for the entire project has been divided into the two main states (pre-game and game) and expected/redundant to mention state transitions (e.g. from S0 to each button on state and vice-versa & every button's interaction with S6 and S7) have been curtailed.

As the main game's states are controlled by the user's choice of button in conjunction with the currently alight LED, the project's game state predictably formed a Mealy machine with each individual state of the main game being a possible button input (or no input at all) within a clock cycle. The button and LED data use a five bit one-hot encoded value for the ease at which it took to implement the buttons and bind them to control the LEDs. The initial state of the game state machine and the state it returns to after every clock cycle is when the FPGA is receiving no user input (0 0 0 0 0). With each button input the game checks this input with the LED's encoded value and either awards a point or returns nothing when it routes back to the S0. After thirty seconds, as ordained by the separate timer, the game state ends but as this ending is not interpreted by the actual game's logic and is more akin to killing the process after thirty cycles, it is not reflected in the game state machine.

**Explanation of Design**

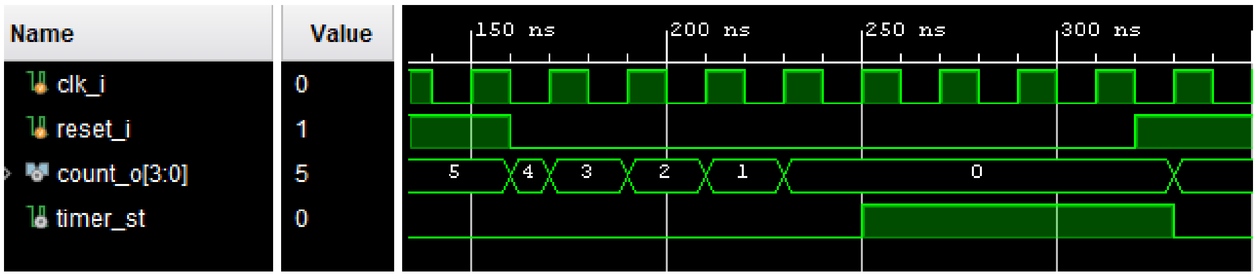
The limitation of only using onboard FPGA functions, rather than hamper the design process, happened to bolster our approach to controlling input and output functions. With little in terms of feature-creep induced distraction and Verilog already being low-level in its approach to logic, nothing stood in the way of directly programming the intricacies of our gameplay loop, seeing where it fails, and strengthening it with each revision of the project. Early onward we noted the similarities between the finalized block diagram of the project and the at the time recently completed Lab 2 and sought to merge these similarities to make implementing later features easier. Doing so allowed us to reuse many of the module instances that we created in our individual Lab 2 assignments and retool them to better fit their specialized role in the Whack-a-Mole game. For instance, the debouncer modules were borrowed over from Lab 2 and applied to the "whack" button inputs to ensure that the input signals are accurately interpreted by the FPGA without rebounding into the next clock cycle and denying the user a chance at play.

There was a steep learning curve when approaching the project, as a result the realization of many initial concepts had to be dropped or heavily subdued in the final implementation due to either time-constraints, unfamiliarity with the hardware, or a conflict between a planned feature and other parts of game logic. For instance, initial drafts for randomized modules used Verilog's preloaded {random} function which, while creating acceptable outcomes in testbenches, does not actually implement anything onto the physical board. Consequently, our team created a pseudo-random number generator: a linear feedback shift register. Likewise, early iterations of our reset function would only work if the button we had assigned to it was held down for several seconds; this setback in particular led to the change to a reset switch rather than a button.

**Module Simulations**

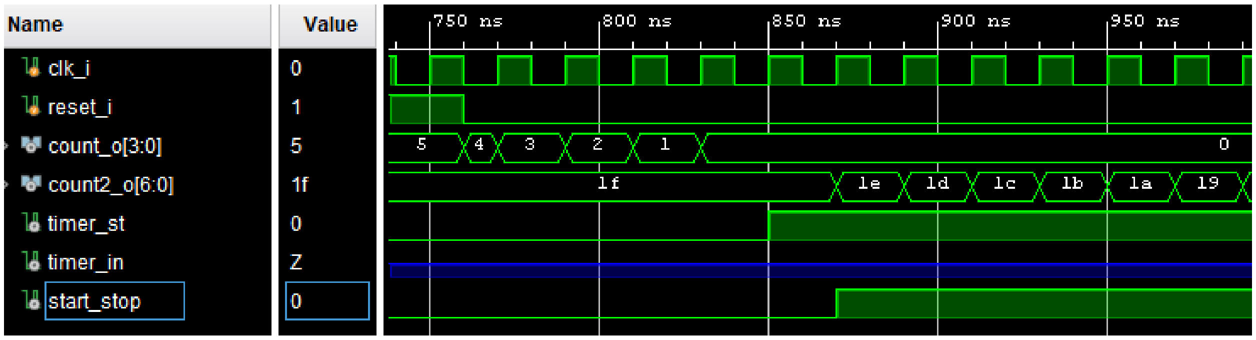
Note: Simulations of modules from Lab 2 will not be shown.

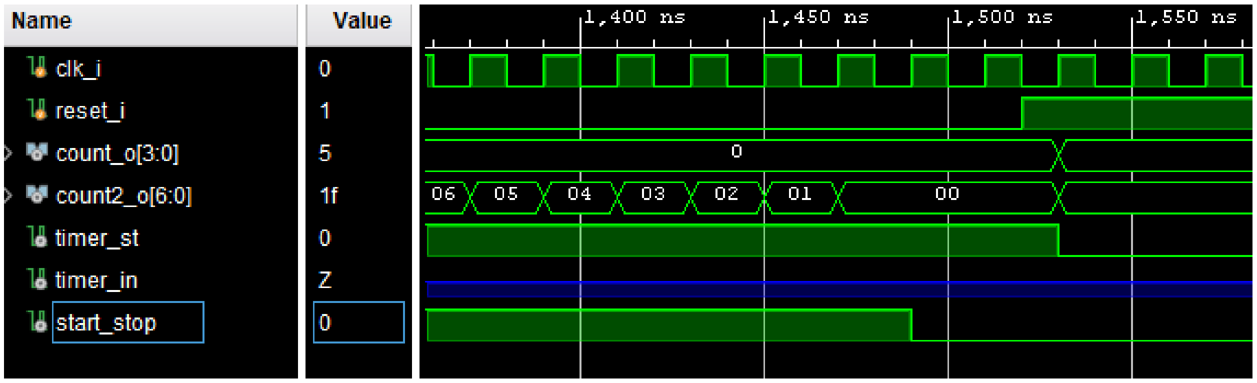
*Five Second Countdown:*



As soon as the reset signal ends, the countdown begins. While it appears as though '4' is not displayed for as long, this is merely a fault of the simulation and in reality there is no such error. If the reset signal is reactivated, the countdown is set to 5 again until the signal is canceled.

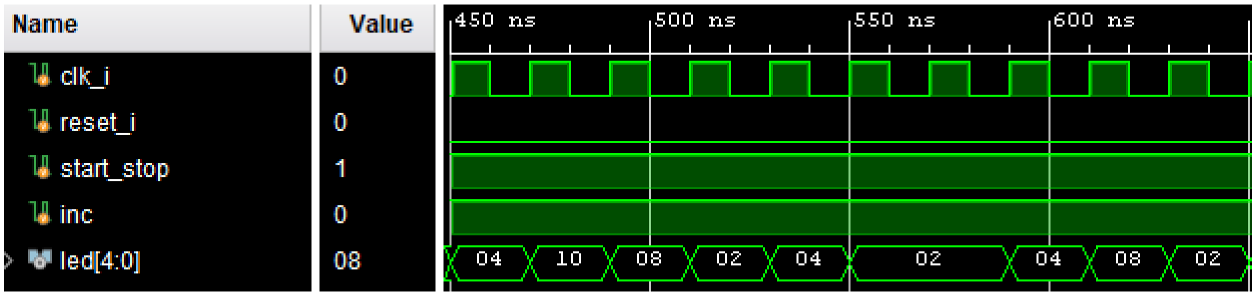
*30 Second Countdown:*





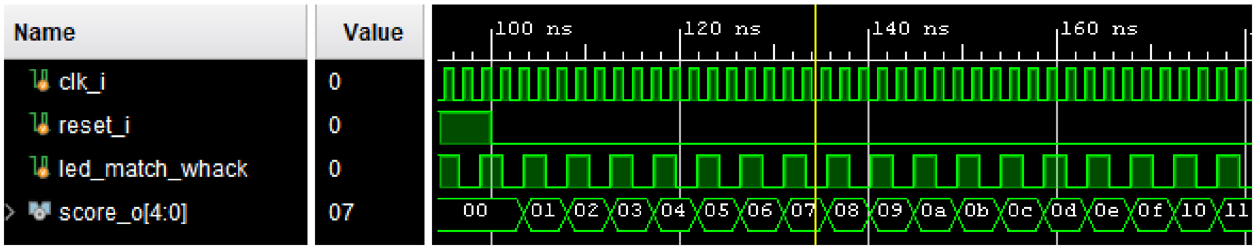
Calling both countdowns helps solidify the functionality of the 30 second countdown. It is clear that once the first 5 seconds pass, the 30 second timer begins. Once the 30 seconds passes, the start_stop signal terminates which is essential in ensuring that the LED’s stop appearing each second. The ‘timer_in’ signal is an internal wire for testing, so it is not important that it displays a high impedance signal.

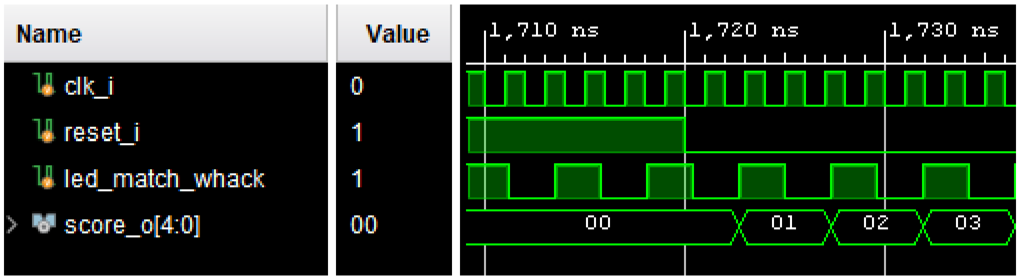
*Random Number Generator:*



There is no discernible pattern of numbers generated. While not truly ‘random,’ there is enough unpredictability that the player could never figure out the algorithm while focused on the gameplay. These values also pertain to only one active high in the 5 bit bus at a time, so that only one LED is lit each clock cycle.

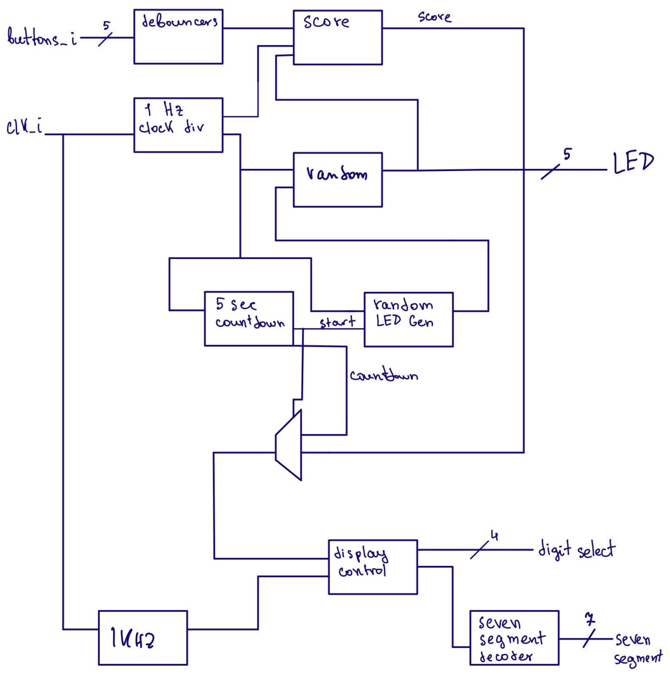
*Score:*





Until reset is done, the score remains at 0. After that, anytime the condition of the whack corresponding to the correct LED is met, score increments. If the reset signal is sent once more, the score is set back to 0.

## Block Diagram



## Schematics