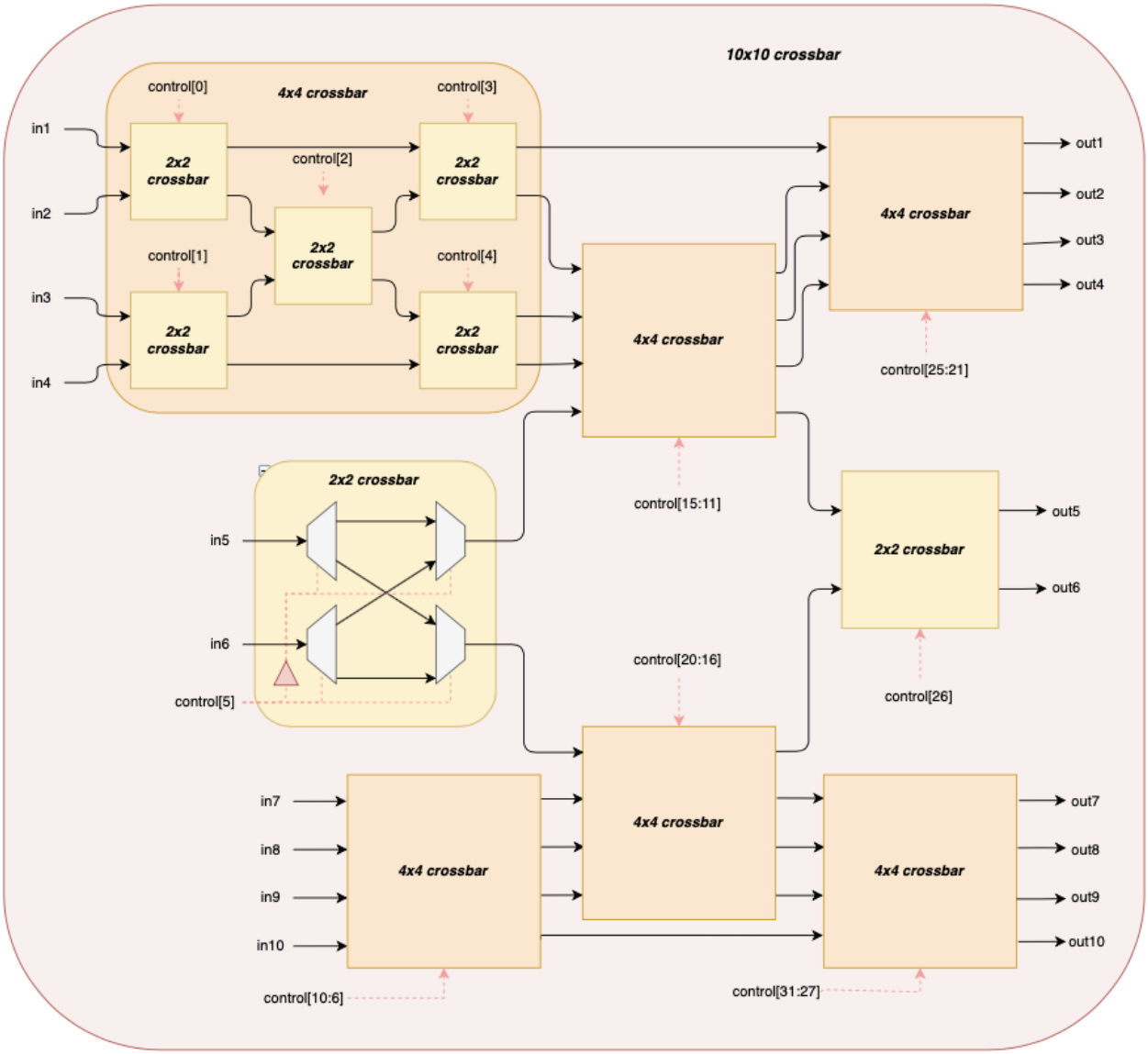
LAB 4 REPORT

(FPGA) The 1A2b Game

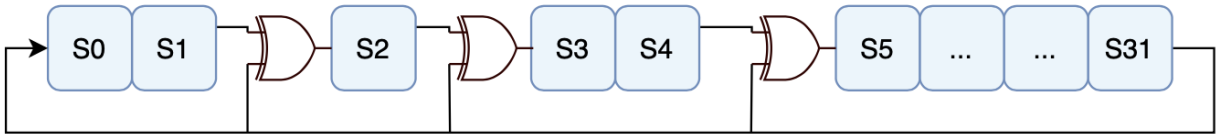
Team 37:

1. 徐美妮 Mary Madeline Nicole 109006205
2. 林之耀 Kevin Richardson Halim 109006277

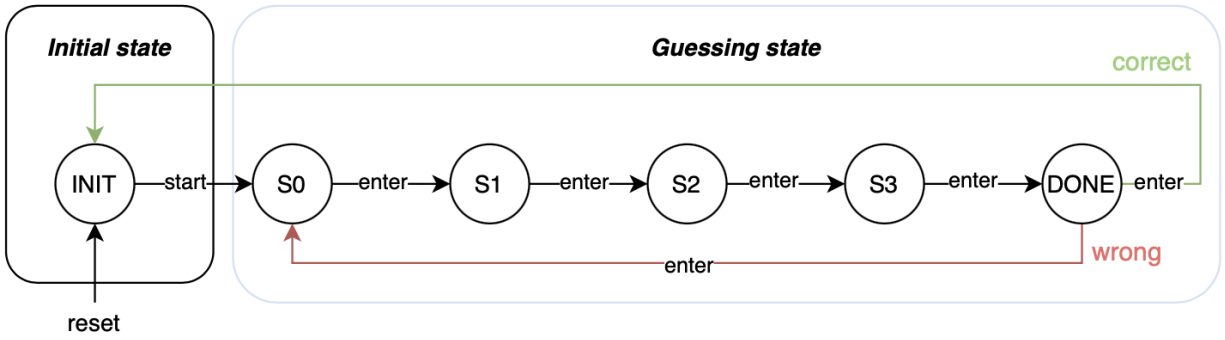
I. Block diagrams and state transition diagrams



1.1. 10x10 crossbar from four 4x4 crossbars and two 2x2 crossbars



1.2. 32-bit LFSR for random number



1.3. State transition diagram of 1A2b

II. Design Process

The 1A2b Game design we start by drawing the state of the gameplay. The initial phase which is the main menu and then the gameplay phase. The initial phase only has 1 state named *INIT* and the gameplay phase consists of 5 states named *S0, S1, S2, S3 and DONE*. The initial phase only acts as a menu and the gameplay phase the player will be guessing 4 random non-repeated numbers. The answer will be shown instantaneously whenever the gameplay phase is entered and displayed as a binary number on the LED. The I/O for this design, displaying each phase, and how the game works in detail will be explained in the following.

This design input consists of 4 switches for guessing the number and 3 buttons as start, reset, and enter button. The *start* button functions as a start button when the player is in the initial phase and wants to enter the gameplay phase, the *reset* button to return to the initial phase, and the *enter* button to register the number the player wants to guess. All of the buttons will pass the debouncer and one pulser signal and the switches will function as numbers guessed which are represented as 4-bits binary.

The output consists of four 7-segment displays to show '1A2b' at the initial phase or number that has been guessed by the player at gameplay phase or showing the result of the guessed number. The output of each state are different, so we used a *flow* to control whats output shall pass to the 7-segment display as its updating every $1/2^{17}$second. The game module has A, b, and phase output to signal the FPGA_UI module which output to select. The answer for every random number will be shown as 4 4-bits binary representation in the

LED. To achieve beeping on the most right 7-segment display, we use a conditional shorthand if and wired to the clock cycle.

The 1A2b gameplay is divided into two phases, the initial phase and the guessing phase. As the game will start only when the start button is pressed, it will change from the initial phase to the guessing phase. Then, the guessing game will start. Guessing phase starts with the first state, S0, then the user will input the number sequence, and press enter. Then, the state will change to the next state, S1. This process repeats until four numbers are input. After all the numbers are received, the checking process will start at the DONE state. If the user enters the correct number sequence, the guessing phase ends and will return to the initial phase. Else, It will go back to guessing another random number sequence.

Second, creating a randomizer for the number sequence. Our idea to create the number randomizer is by using a 32-bit LFSR. We utilized a 10x10 crossbar to generate the random numbers that have no duplicates and have no value more than 9, where the 32 control bits are randomly generated by the LFSR. From the 10 bit output of the crossbar, we only picked the two most significant output and the two least significant output as the output, so it will output a 4 random number.

We used four comparator arrays to check which address contains the value, while the priority encoder received 4-bit input from each comparator array and will output the biggest address that contains the value.

At the guessing phase, at every state (S0, S1, S2, S3), when the user pressed enter, we shift the state and receive the number input to the current state. We also check if the current number input at the nth state and the random number sequence at the nth index matches or not. If it matches, we add the count of the A, if it only has a matching number but not at the n-th index, we add the count of the b.

Last, we connect the game module with our FPGA. We created 2 types of clock. One for flickering and the other for display of the seven segment display. The debouncer connected to the faster clock and onepulse connected to slower clock (1 second every clock cycle) each button to eliminate inputs that are more than a clock cycle and to eliminate any glitches.

III. Design Test

We tested the success of our program by looking at the waveform result of the randomizer. This way we know if a problem occurs at which modules.. Then, we check our program from there. When we test the correct sequence, the display should be 4A0b, and when it is wrong, it should change accordingly. After testing out by changing the number inputs from the FPGA switch, we proved that our program works well. We also tested the reset, enter, and start button, as well as the constraints of no number more than 9 to make sure that our program covers all the requirements.

IV. What we have learned from Lab 4 FPGA

- ➔ Design a finite state machine to have a specific behavior on each state.
- ➔ Generating random numbers with LFSR, then connecting it to a 10x10 crossbar to make it more random and non-repeated number.
- ➔ Using a mealy-moore machine design to generate output or signal to trigger other modules.