CS M152A: Lab 4
Locker

Mark Tai
Michelle Dai

March 9, 2016

Introduction

We created a locker that takes in a 4-digit input as a password, remembers it, and locks itself. When another 4-digit key is entered, but does not match the memorized key, the locker does not unlock. All the I/O will be on the Nexys3 FPGA board, with additional input boards attached (such as a number pad and speaker).

Functionality

Features

- **CLEAR (C on number pad)**: Pressing this button clears whatever digits were previously entered.
- **NUMBER PAD**: inputting a sequence of four of the numbers 0-9 on the attached number pad becomes the passcode.
- **LOCK / UNLOCK (B on number pad)**: after a 4-digit passcode is entered, pressing the lock button will commit the passcode to memory, and the locker will "lock" itself. If there is already a memorized passcode, the locker will unlock if the 4-digit input matches the passcode.
- **RESET (BTNR, the rightmost button on the bottom edge of the board)**: because of Murphy's Law, this feature must be included as a salvation from a multitude of problems.
- **BREAK (SW0, rightmost switch on the bottom edge of board)**: To simulate real-world functionality, flipping this switch will break the locker. In this mode, every 5th input is dropped, and no sound will be played for the button press.
- **LED**: When a passcode is set, an LED light will turn on and remain on until the locker is unlocked.
- **SPEAKER**: When any button is pressed, a speaker plays a sound. The tones ascend chromatically with the button presses.

Schematic Diagram

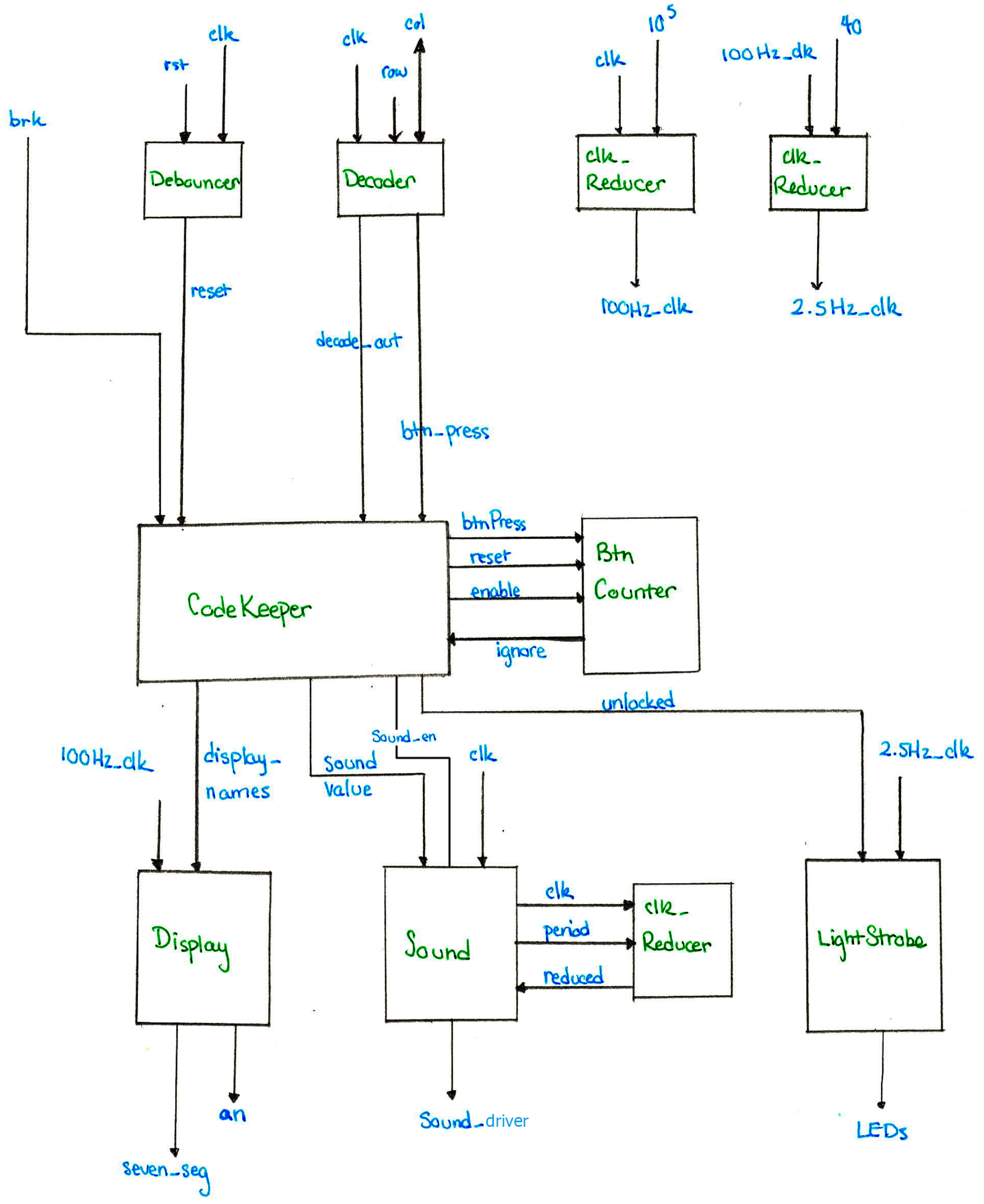

Figure 1: **Schematic Diagram of the 4-digit Passcode Locker.** This shows the high level organization of the Locker.

Modules

Debouncer

This module was not modified from Lab 3.
A substantial amount of noise is introduced in the act of pushing buttons. To counteract this, we extend the high of a signal. We implemented this by creating a module debouncer, which extends the input signals. If the signal goes high, it outputs high for the next 8 clock cycles. We wrote up several test cases to ensure the debouncer functioned properly. Test cases are shown in the waveform below.

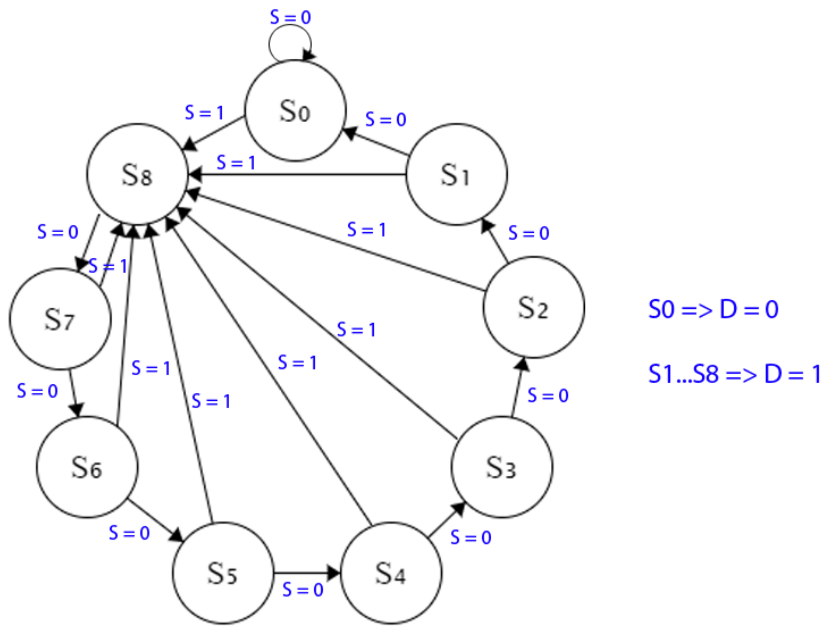

Figure 2: **State diagram or the Debouncer.**

ClockReducer

This module was not modified from Lab 3.
Four different clocks were used in this lab, derived from the 100 MHz master clock. We used 1 Hz, 2 Hz, 4Hz, and 10 kHz. The 1 Hz, 2 Hz and 4 Hz clocks are used for normal ticks and blinking in ADJ mode, and the fast 10 kHz clock used for cycling through the 4 digits on the 7-segment display so that to the human eye, the display appears to output 4 digits simultaneously.
To implement the clocks, we created a module that uses an internal 32-bit counter. When the counter is greater than or equal to the 32-bit input ratio, the output clock is set to high.

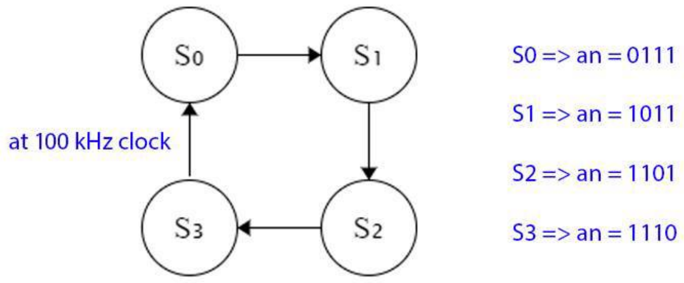

Figure 3: **State diagram or the ClockReducer.**

Decoder

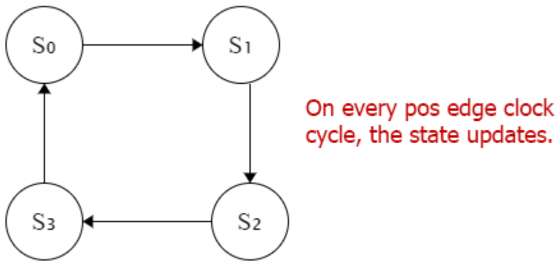

Figure 4: **State diagram of Decoder module.** This shows the display cycling through each of the 4 digits at a 10kHz clock cycle.

The decoder is the module that decodes the input from the numberpad and returns a parsed signal. It cycles between the four columns, which must be driven as high before accessing the specific row in that column that is being pressed.
The decoder was a module downloaded from Diligent, but was modified by us. We added the features of having two major outputs: one that signifies when a data signal is high, and one that is a 4 bit representation of the button press, where all 1's represents no button press. We disabled A, D, E, and F keys on the keypad, so there was room for a null keypress encoding.
Due to the interaction of the hardware and the software, we decided to test this module on the device, as we did not want to assume how the numberpad worked after a few failed attempts.

CodeKeeper

CodeKeeper is the main module that handles the logic of the locker. The CodeKeeper module handles parsing the input numbers from the decoder and reset, and determines what sounds to play, what to display, and whether the locker is unlocked or not. On a high level, it uses two sets of four 4-bit registers: one contains the reference code used to lock, and the other contains the attempt code used to unlock. It ignores a button press whenever the internal BtnCounter's output goes high, but it still sends the btnPress to BtnCounter.
CodeKeeper has a synchronous reset to both unlock and to set both codes to blank.

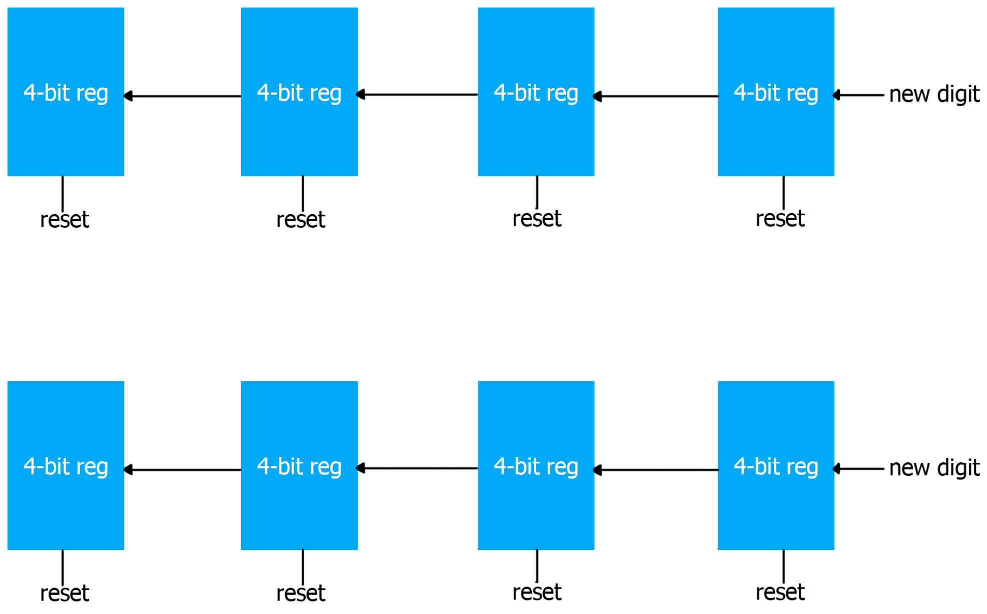

Figure 5: **Circuit diagram of CodeKeeper module.** The top four 4-bit registers hold the locked passcode, and the bottom four registers hold the passcode attempts. The locked passcode only changes when the module is unlocked and the passcode attempts only change when the module is locked.

BtnCounter

This module is important in implementing the Break mode for our Locker. It is hard coded to ignore every fifth user input. On every positive edge due to a button press is detected, the state is incremented. If the current state is the 4th button press, the next button press sets it back to the original initial state, and the module outputs a high signal. If enable is low, or the reset value is high, the state is returned to the initial state.

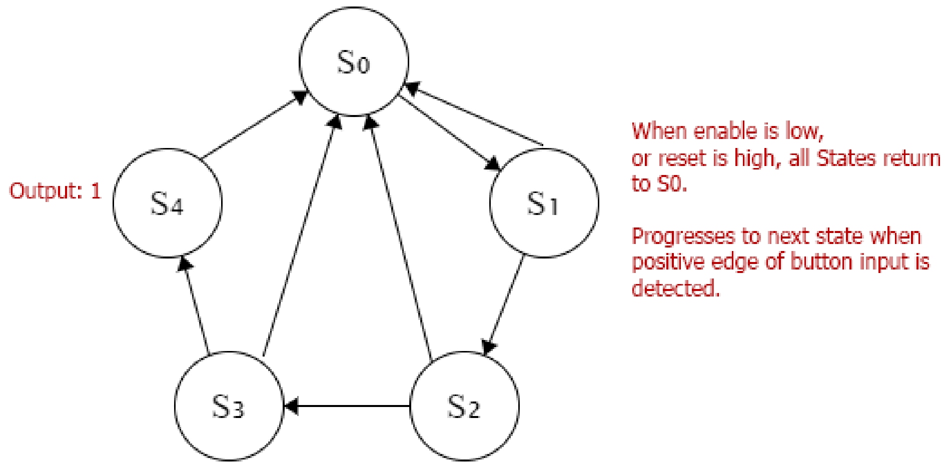
S0
S1
S2
S3
S4
Output: 1
When enable is low, or reset is high, all States return to S0.
Progresses to next state when positive edge of button input is detected.

Figure 6: **State diagram of BtnCounter module.** Only when the state is S_4 the output is 1. Otherwise, the output is 0.

Display

This module takes in 8 inputs: the clock, blinking clock, SEL, ADJ, and four 4-bit numbers for the 7-segment display. After researching how to use the 7-segment display, we coded the digits in correspondence to 7-bit words. We kept a register to keep track of whether the 7-segment display should show blanks or not, creating a ”blinking” effect. If it should be blanked, i.e. if it’s adjusted and selected, then the digits in the corresponding selection are blanked. This module takes in 4 digits, and uses a switch statement to figure out what to display. Every cycle of the 10kHz clock, it cycles through the 4 digits, but because this is faster than the human eye can detect, we see a constant display of digits.
In this lab, we did not use the blink function at all, and modified it such that sending a 15 to a display would make it display blank.

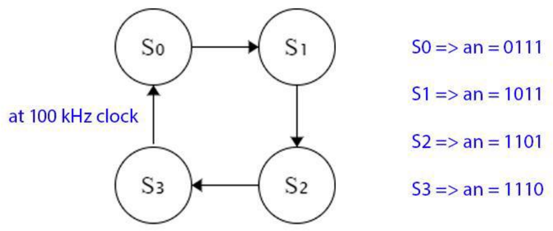

Figure 7: **State diagram of Display module.** This shows the display cycling through each of the 4 digits at a 100Hz clock cycle. *Due to a sharelatex.com bug, we could not change the 100kHz. The correct frequency for this lab is 100Hz.

Sound

This module drives the sound effects of the locker. It takes in a 4-bit value and an enable value. When the enable value is high, the module will use a switch statement to parse the sound_value 4-bit value to determine which period it should pass to its internal clk_reducer. We hardcoded 16 notes in a chromatic scale for the input of the Sound module, and we only use 13 in the actual locker. The sound_driver output is a signal with a frequency of the reduced output from the internal clk_reducer. The frequency and period values exactly correspond so that the root value has a frequency of 440 Hz, the sound of A4.

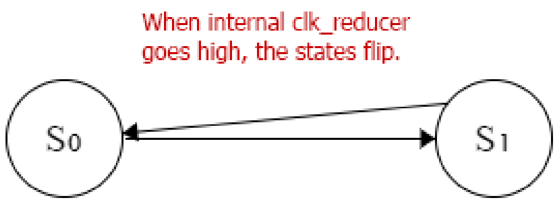

Figure 8: **State diagram of Sound module.** The state of the output flips on a specified frequency to match the frequency of the chromatic scale.

LightStrobe

The LightStrobe module controls whether the lights are strobing or all on. It cycles between 14 states, achieving the strobe effect. These states are updated on the positive edge of the 2.5Hz clock fed into the LightStrobe module. When the LightStrobe module is not unlocked, it resets the state to the initial one, so the strobe always starts on the right side and moves left initially.

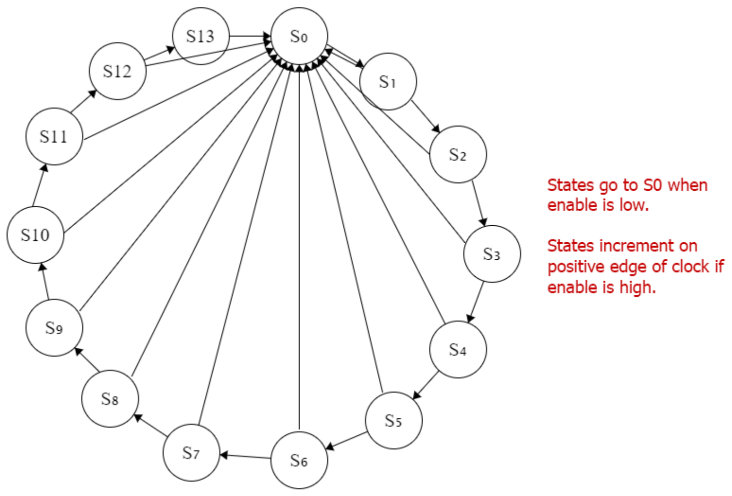
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
States go to S0 when enable is low.
States increment on positive edge of clock if enable is high.

Figure 9: **State diagram of LightStrobe module.** The states go in a circle, unless the enable is low, when they all reset back to S_0.

Waveforms

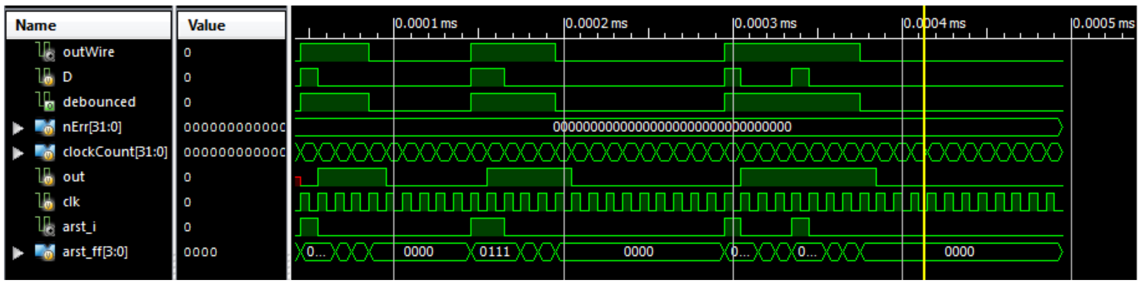

Figure 10: **Waveform of the debouncer.** This extends a signal to produce a better input. "debounced" shows our test cases: a pulse that lasts 1 clock cycle, one that lasts 2 clock cycles, and one that is intermittent. For the test case 2 clock cycles long, the output also lasted one more clock cycle in length.

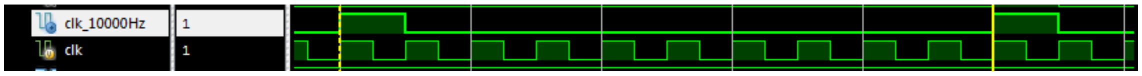

Figure 11: **Waveform of the ClockReducer.** When clk cycles 10 times, clk_10000Hz cycles once. There are no real edge cases for this module; if it works for 10, it should work on any number.

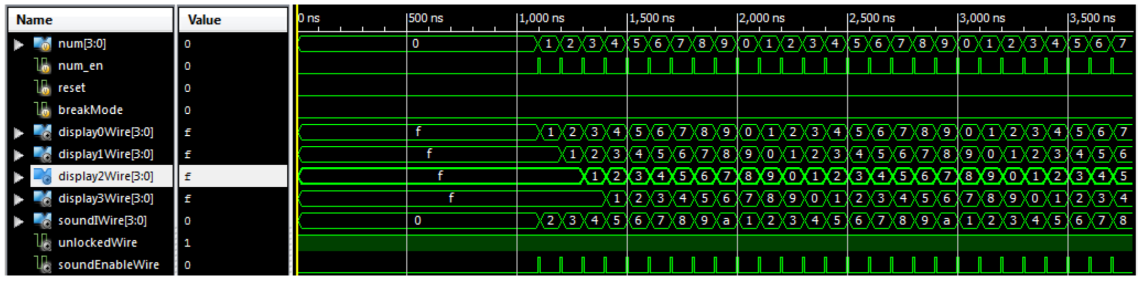

Figure 12: **Waveform of the CodeKeeper.** This waveform shows that the 4 number outputs are correctly cascading from left to right with basic input. f represents a blank.

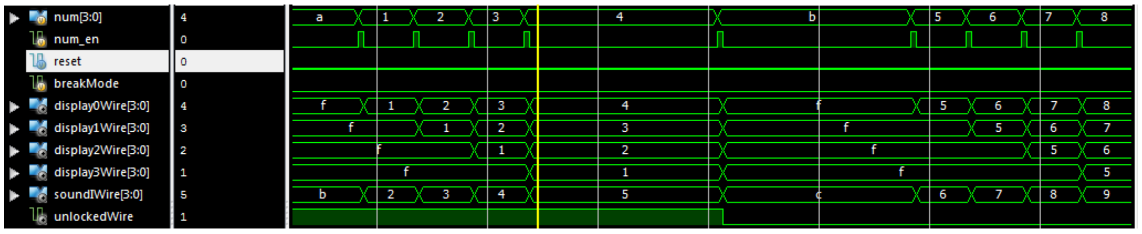

Figure 13: **Waveform of the CodeKeeper.** This waveform shows that the module correctly locks when the lock key, b, is pressed. The desired behavior is that all the output numbers reset to blank and the unlocked output goes low.

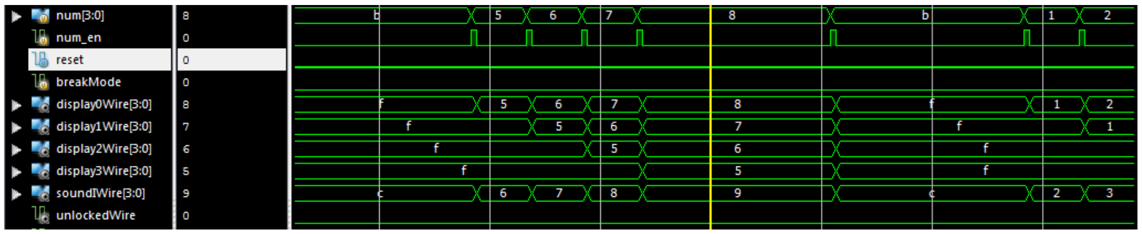

Figure 14: **Waveform of the CodeKeeper.** This waveform shows that the module does not unlock when the correct password (1234) is not used before the unlock key is pressed (b).

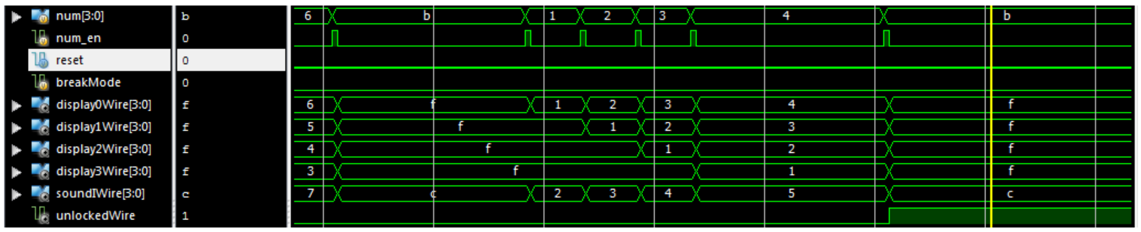

Figure 15: **Waveform of the CodeKeeper.** This waveform shows that the module unlocks when the correct password (1234) is used before the unlock key is pressed (b).

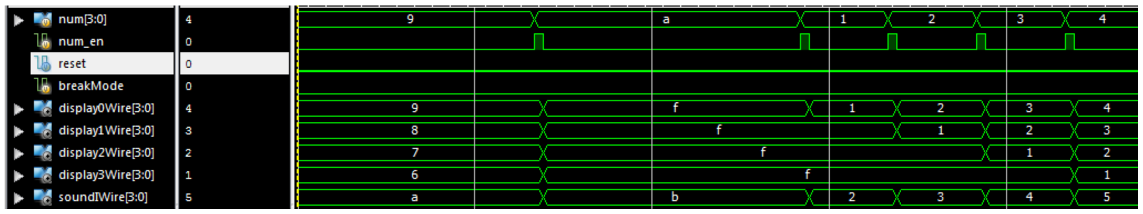

Figure 16: **Waveform of the CodeKeeper.** This waveform shows that the module clears the current code when the clear key, a, is pressed.

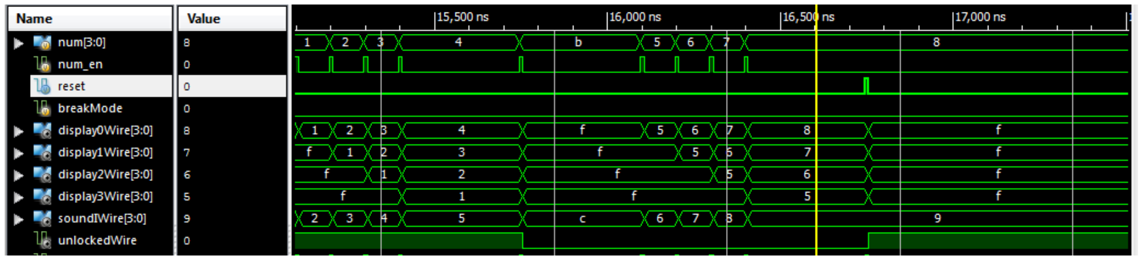

Figure 17: **Waveform of the CodeKeeper.** This waveform shows that the module unlocks and clears all codes when the reset button is pressed.

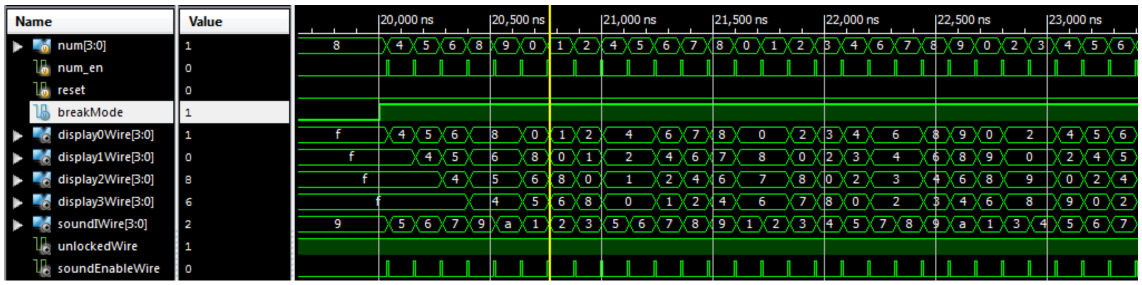

Figure 18: **Waveform of the CodeKeeper.** This waveform shows that the module ignores every fifth input when breakMode is turned on. Desired features are that the sound outputs (shown by the changing soundWire and soundEnableWire), but the display wires do not change on the fifth input.

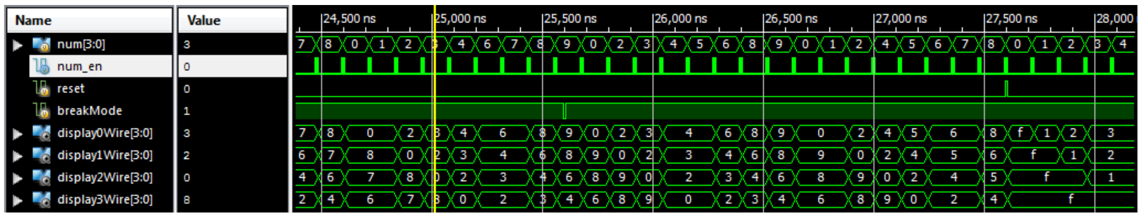

Figure 19: **Waveform of the CodeKeeper.** This waveform shows that the module resets the breakMode internal counter when breakMode is toggled.

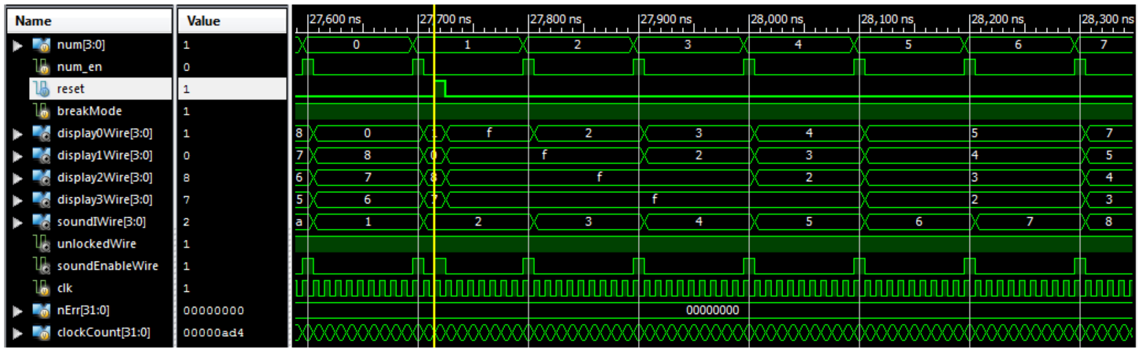

Figure 20: **Waveform of the CodeKeeper.** This waveform shows that the module resets the breakMode internal counter when the reset button is pressed.

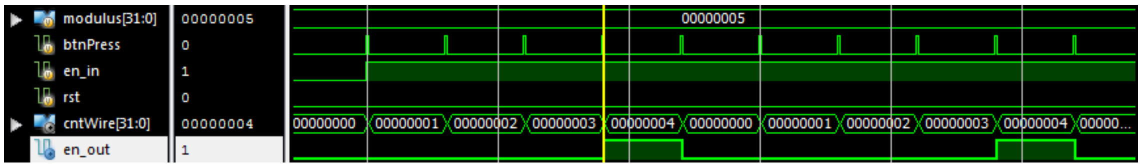

Figure 21: **Waveform of the BtnCounter.** This waveform shows that the module is a modulo 5 counter and outputs high when the count is modulus-1.

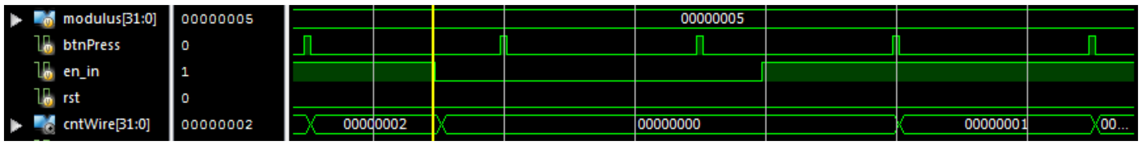

Figure 22: **Waveform of the BtnCounter.** This waveform shows that the module resets its internal count when the en_in is low.

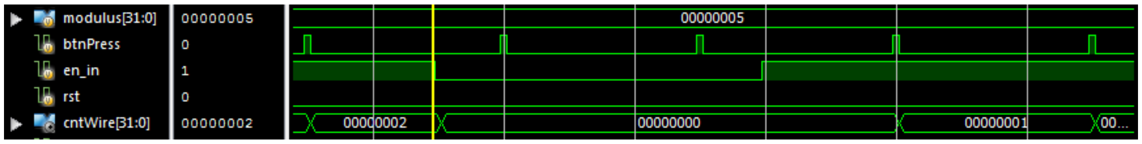

Figure 23: **Waveform of the BtnCounter.** This waveform shows that the module resets its internal count when the reset input goes high.

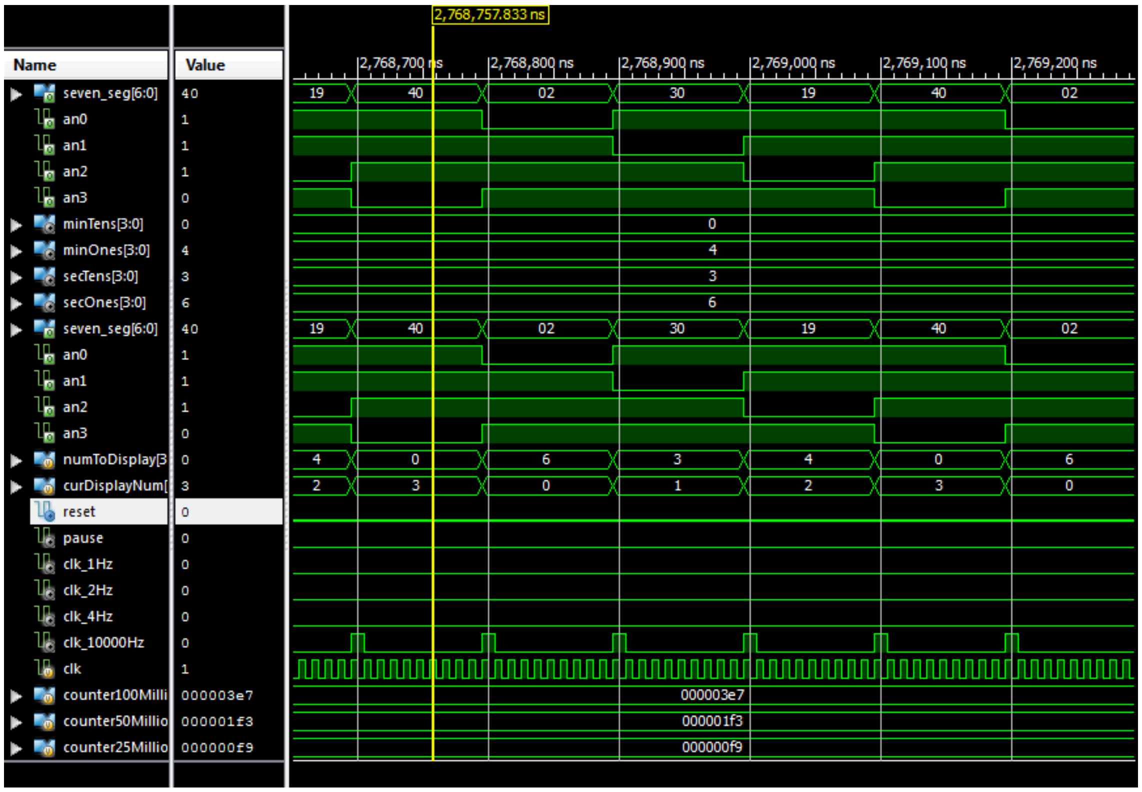

Figure 24: **Waveform of the display.** The output to the 7-segment display is shown here. This tests that only one of the an-outputs are set to low at a time. numToDisplay is cycling through secOnes, secTens, minOnes, and minTens one at a time.

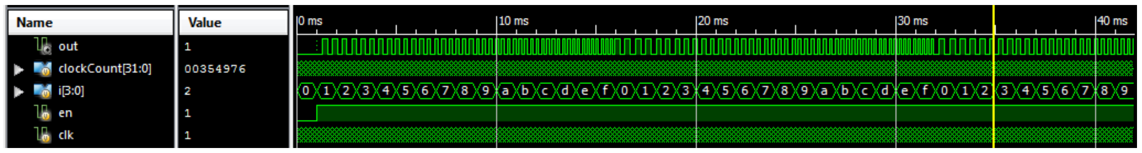

Figure 25: **Waveform of the sound module.** This waveform shows that the output, out, of the sound module is of when the en input is low, and goes increasingly high in frequency as i increases. We tested that it was the correct frequencies on the board.

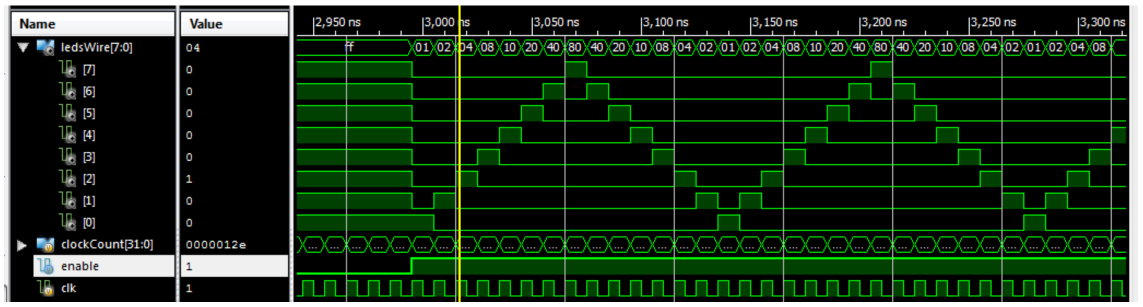

Figure 26: **Waveform of the LightStrobe module.** This waveform shows that when enable is low, ledsWire is all high, but when enable is high, ledsWire strobes between 0 to 7 and starts at 0.