**Final Project:**

Task: Game Nexys

From: Oliver Jarvis

Date: 11, December, 2023

The GitHub for this report is https://github.com/OliverJarvis1/FINAL\_PROJECT and a video of the working system can be found at https://www.youtube.com/watch?v=1aRDoCxtWCQ.

**Design Method**

The design for this project stems heavily from Pong Chu’s given Vivado and Vitis code. I set up Chu’s timer, UART, LED GPO, RGB LED GPO (although I ended up not using this), 7sseg, Switch GPI, Button GPI, PS2, and accelerometer SPI cores with the mmio\_sys\_vanilla and mcs\_top\_vanilla hardware. The UART was used to print to putty with the stages and any text information from the system. The LED GPO was used for various games, as well as an output to represent which game was selected. The 7-segment display core was used for the button game to show which button to press next. The button GPI core was used to confirm user choices (which game to play, when they wanted to play, and if they wanted to go back to the menu). The SPI accelerometer core was used to take the x, y and z data of the nexys board and use that as a random seed for each game. Lastly, the PS2 core was used for the Wordle game for the user’s input.

**Implementation**

**Overview:** Many of the files were imported from Chu’s library and were not altered. The exception to this is the main\_vanilla\_test, which had a different main loop that consisted of calling the game\_select() function, followed by the respective game (chosen by if statements based on the output of the game\_select() function). All of my code is included in the files game\_commands.h and game\_commands.cpp. game\_commands.h is simply the header file that includes the definitions of the various game functions.

**Game Select:** The first function I created was the game\_select() function. This function used the uart, led, sseg, sw and btn cores. The purpose of this function is to select and output the game chosen by the user. This is done in a while loop, which is controlled by the centre button. Until a game is selected and the centre button is pressed, the code does not exit this function. The switches and buttons are read at the start of the function, and the switches 5:0 select which game is to be played. If multiple switches are up, the leftmost game selected is picked. No other switches alter the game chosen. If the top button is pressed with these switches on, then the game is confirmed and its respective led lights up (and the name is outputted through UART), but the game is not started yet. The game can still be changed at this point. If the centre button is pressed after a game is confirmed, then the loop is exited, leds turn off, and the game is outputted.

**Game Over:** After each game, there is essentially the same code that uses UART to ask the user to press the centre button to play again, or the top button to go to game select. If the user hits the centre button, different values depending on the game are reset to their starting values and the game loop is not broken. If the user hits the upper button, the loop is broken and depending on the game, leds or sseg or nothing are turned off.

**Buttons Game:** This is the first game I created, and it is selected in the game select using switch 2. This function uses the uart, sseg, spi and btn cores. Initially, the spi accelerometer is set up and records the x, y and z values and uses that for the random seed. I did this because the accelerometer I used had a bit of noise, and this gave me random games even after restarting the nexys device. I also wanted to use the accelerometer for something but did not want have the user constantly tilting the board and possibly harming the wiring if they touch the back. After this initial setup, a while loop is entered which is only exited if the user hits the centre button after the game ends. For the game, a for loop is called and runs ten times. Using the random value, one of the five buttons is selected as the requested button. The same button cannot be selected twice in a row, which is determined by n\_btn and p\_btn not being equal in order to go to the next steps. Based on this, the 7-sseg will display a “C”, “L”, “d”, “r”, or “U”, corresponding with centre, left, down, right, upper buttons. 200ms later, an input is recorded. If the correct button is pressed, the user’s score will not go down as much. If the incorrect button is held, points will be rapidly deducted. Every 1ms, a small amount of points are deducted. The score is printed through UART after each game.

**Switches Game:** This is the second game I created, and it is selected in the game select using switch 3. This function uses the led, spi, sw, and btn cores. Initially, the spi accelerometer is set up and records the x, y and z values and uses that for the random seed. After this, the game loop is entered. A random integer is taken from 0 to 15, corresponding to an led that will be turned on or off. This value is then put into the pow command (from math.h) which takes 2 to the power of the random number (this essentially writes the single bit value since the LED expects binary inputs) and outputs it to add\_sub. This add\_sub value is either added, or subtracted from the led\_on value, which determines which leds are on. This is done by taking the modulus of led\_on with the pow(2, random number +1). If this is greater than or equal to add\_sub, then the LED is already on and must be turned off. If it is not greater or equal to, then the LED is off and must be turned on. Led\_on is then written to the led core, and after 1s, the sw inputs are recorded. If the sw input does not equal led\_on, then the switches did not match the LEDs and the user fails. Else, the game continues. UART is used to print the final score at the end (which increments by 1 every loop).

**Switches and Buttons Game:** This is the third game I created, and it is selected in the game select using switch 1. This essentially combines the switches and buttons game, so the concepts are the same as above, and the cores will be the combination of both. One major difference, is that a random number is generated at the start of each loop. If this value is 0, the switches/leds code will run. If it is 1, the buttons code will run. Additionally, the buttons code does not run 10 times, but rather once per loop (if called by the random number). After outputting to the 7-sseg, 1 second is passed and the user must be holding the button to pass. The user does not need to continue to hold the button after the prompt. UART is once again used to print the final score at the end, which increments by 1 every loop and resets every game/restart.

**Memory:** This is the fourth game I created, and it is selected in the game select using switch 0. This function uses the led, spi, sw, and btn cores. Once again, the spi accelerometer is used for the random seed. After this, the game loop is entered. Led\_on is a random value modulated with 65525, which is 2^15. This gives a random value that turns on the 15 LEDs randomly when led\_on is written to the led core. After the leds are written to, UART is used to prompt the user to turn off all switches and press the centre button when ready. When the centre button is pressed, if switches are on the game auto loses to prevent cheating. Else, the LEDs will turn off and the user must toggle the switches to be in the same order as the LEDs they saw. To confirm their answer, the user will press the centre button again. The switch core is then read, and if the output equals the led\_on value, the user wins and gets a congratulation message through UART. If they lose, UART outputs a losing message.

**Wordle:** This is the fifth game I created and was my main inspiration behind the Game Nexys. This game is selected through switch 4, and uses the LED, SPI, PS2, UART, and BTN cores. Once again, the SPI accelerometer is used for the random seed. For the game loop, the random number picks one of 4 pre-determined words. The user is then prompted (through UART) to input their word through the keyboard. The characters are inputted into the guess character array, and each time a key is pressed (and is not space), the array position is moved up one. If space is pressed, the array position is moved back one (essentially acting as a backspace. I was planning on reverting this to simply using backspace for backspace, but was not certain how backspace got detected since my keyboard never worked with PS2). If the array position is at 5, which indicates that 5 characters are inputted, no more inputs are taken unless space is entered. At this point, the user can press the centre button to confirm their answer. If the guessed word equals the wordle word, the user wins. This is checked by checking each array position equals each other. If a guessed letter equals the letter of the wordle word, then 2 to the power of the array position is added to a “correct” value. If they are not equal, then the code checks if the letter is included in the word somewhere. This is checked by an if loop, which checks if the guessed letter equals any letter in the wordle word. If it does, it takes 2 to the power of the array position and adds the value to a “close” value. If the correct value equals 31, then the guessed word is correct and the user wins. Else, I increments by 1 (subtracting 1 guess from the user). The close and correct values are then recorded in led\_flash and led\_on respectively. Led\_on then equals the addition of itself and led\_flash and written to the LED core. Every time the loop runs (which is at least 50ms apart from the last run due to sleep\_ms commands), led\_flash is subtracted or added to led\_on. This makes the correct values get represented by solid leds, while letters that are included in the word (but are in a different position) will be represented by LEDs that flash on/off.

**Note:** It should be noted that the Wordle game is not functioning right, and so it was commented out in the main loop. I believe the issue is with Vivado, since the PS2 keyboard is never getting read.

**Testing**

To test the system, I built the code and programmed the Nexys 4 board. To troubleshoot issues, I made sure to build small pieces of the project (game select first, then games). Each game I built one at a time and used UART and the LEDs/7-sseg to confirm that the code was working as intended. I also walked through my code by looking at it in Vitis and imagined special scenarios that could break the system, then I would test those scenarios to make sure they would not break the system.