Jeffrey Ross
CSC 314
Program 2

CSC314 Programming Assignment 2

The goal of this project was to create a functioning Simon game using a Raspberry Pi 2 with a breadboard, 4 LEDs, and 4 push buttons. Most this project was written in ARM assembly language. The final implementation includes not only the Simon game, but two other additional game modes, Mastermind and Whack-A-Mole. The program was written across 5 different files, 4 written in assembly, and 1 written in C. All files will be explained in detail except for IO.S which was provided in a previous lab. All output statements from omxplayer are immediately removed right after using system(clear).

MainSimon.S

This file is where the program starts, it serves as both main as well as functions to support the Simon game. The program starts out by calling the function provided in the IO.S file which maps the devices in memory to be used by the program. It then calls functions gpio_dir_input and gpio_dir_output which will take the pins attached to the buttons and LEDs and set them to receive input or send output respectively. Once completed it outputs the main menu to the screen using printf.

Next it moves on to a loop that starts by checking the bits in GPLEV0 to see if any buttons are pressed. The loop will remain idle until it detects an input from any of the buttons. When detected it will link to one of four menu functions which displays the corresponding LED along with playing a menu selection sound. This is done by activating the LED, calling system using a string to play an .mov file with omxplayer, and then turning off the LED. These functions are global and are called throughout the program. Once returned from the corresponding menu function it branches to the corresponding game mode main function.

When branched to the Simon game mode, it starts off by playing 4 display functions which are used throughout the Simon program. These display functions display a certain color of LED and play a corresponding piano note using the same protocol as the menu button functions. Once each LED has been displayed, a makeArray function is called. This functions purpose is to take an array of length 50 and fill it with random values from 0 – 3. This is accomplished by calling the time function included in the STL which returns an ever-changing value, and using that changing value as a seed in the srand function. Once the seed is set, the rand function is called to get a pseudo random number. This number is modded by 3 before being put into the array. Rand is called 50 times to fill the array. Once the array has been created, it initializes the counter for how many rounds it’s been through and branches over to point A.

Jeffrey Ross
CSC 314
Program 2

POINT A (simonloop): This is the start point of the outer loop. This point initializes the counter for the inner loop back to zero. Following this, it takes the counter keeping track of the current length of the sequence. This value is then passed into a play sequence function which takes the array containing the random sequence, and plays the sequence with a length of the value passed to it. Each light is displayed using the display functions previously mentioned. Once returned from this function it sets a value to indicate a button has not been pressed and goes over to point B.

POINT B (readloop): This starts out by reading the states of the GPLEV0 registers and stepping through and seeing where things need to be set. It checks the state of every button, if a button is being pushed, it sets the state so that the corresponding LED should be turned on, if a button is not being pushed, it sets the state so that the corresponding LED should be turned off. Following this, if it is marked to play audio, it steps through each LED to check if they are active, if active, it plays the corresponding sound for each color by calling the omxplayer through system(). After this it returns back to the top of point B, marking that audio should not be played again. If the audio was already marked to not be played it skips over to check whether or not this was the correct button press. The i*4th value of the array is compared to the value passed from checking the LEDs, if it is correct, it increments the counter and returns to point B. If the current portion of the sequence is equal to the current sequence length, it is linked back to point A. If it is incorrect, it is linked back to the main menu.

Mastermind.S

When linked over to the mastermind function, it begins with a loop of reading the GPLEV0 register and then checking the bits for every button. Once a button is pressed, the corresponding menu function will be called and an array will be created to store a sequence of random numbers corresponding to the number selected. (See user manual) The function used to generate the array uses the same algorithm as used in the Simon game to generate a random sequence. Following this it will enter another menu to get the number of attempts for the player to use. The value chosen will be saved for future use, and it will branch to the start of the game.

The game will start by calling a function which displays the green LED by setting the bit, then using omxplayer to play the success sound and then clearing the bit. This is then repeated again for the red LED and the failure sound. After this it leads to the main body of the game, where it has a very similar structure to Point A and point B previously described. The outer loop increments attempts that the player has used to try and find the sequence. For every time the player has attempted the full length of the sequence this will check whether it is marked as correct. If it is marked as correct, it will branch to a success function which plays success music and exits back to the main menu. If it is marked as incorrect it will increment the sequence counter, if it is greater than the total number of attempts then it goes to the failure loop where it outputs a failure sound and goes back to the main menu. If it is not greater than the total number of attempts it goes into the inner loop.

The inner loop first checks to see if it has reached the end of the sequence, if it has it goes to the outer loop. If it has not it checks the states of the buttons until activity is found. Once a button has been pressed the corresponding LED will light up and the current spot in the sequence will be checked. If the value for the color and the value for the spot in the array are the same, then the indicator for a correct sequence will go unchanged. However if they are not equal, the indicator will be changed to indicate the sequence was not complete. It then returns back to the top of the inner loop.

Time.c

Time.c contains 3 short functions which are used by the Whack-A-Mole game to measure the amount of time taken. The first function, timer_start(), uses the clock_gettime function to get the current time from a monotonic clock. This function then returns a timespec structure from the time.h library. The next two functions are called to get the ending time and get the time difference between the structure obtained from the original function. One function returns a time in seconds, the other returns a time in nanoseconds. Both are passed the structure as an argument.

WhackAMole.S

The whack function is called from the main menu and starts with a menu exactly like the ones used in Mastermind. These menus are used to determine how many “moles” the player would like to “whack”. Once selected omxplayer is again used to play an audio cue before calling the timer_start function and going into an outer loop. The outer loop goes through each LED and sets the pins to turn them all off. It then checks if the number of moles has exceeded the amount selected, if not it takes the previous LED value (initialized to 4) and sends it to a randVal function. This function is similar to the other random generators, but adds the extra condition that it cannot be the same as the last one and only generates one value instead of an array.

The function returns the random value and it is held in a register. After this it sets the bit for the corresponding LED for the number generated. It then moves on to the inner loop. All the inner loop does is check the state of the buttons, if a button is pressed it checks to see if it matches the randomly generated number, if it does not, it continues the loop checking the buttons. Once a matching button has been found it increments the mole counter and goes to the beginning of the outer loop.

Once the program exits out from the outer loop it will call the two ending functions from time.c to get the time values in seconds and nanoseconds. These values are held on to while omxplayer is called and a short completion sound is played. Once the sound is finished the seconds value and the nanoseconds value is output to the command prompt using printf. It then prints another statement prompting the user to press any button to return to the main menu and goes into a loop that waits for the user to press any button. Once the button is pressed the program is returned to the main menu.

The project's completion was a success, as it added three game modes to the program, rather than just one. Each game mode has a different aspect to it to keep them from feeling the same, where Simon is a memory game, Mastermind is a puzzle game, and Whack-A-Mole is a speed based game. The success of the project will allow it to see usage outside of the class, for a variety of entertaining purposes.