

**American University of Beirut**

**EECE 425 Project**

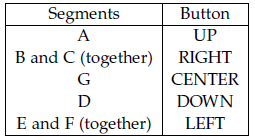
Simon Game

Jad Wahab

Gaelle El Feghali

1. **Game Description: Functional perspective and user interaction**

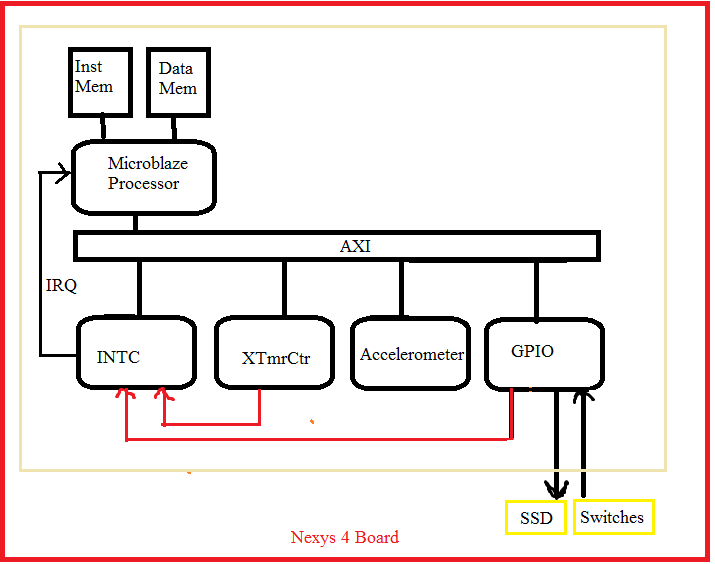
Simon is a memory game where the computer flashes a random pattern on one seven segment display and the player reproduces the pattern using the five push buttons. The game starts easy, flashing only one random segment, but it gets harder, flashing an additional segment after every correct answer. Thus, in level 1, the first display is flashed and in level 8 all 8 displays are flashed one after the other for the user to replicate. The speed of the game is adjusted to make it more challenging. If difficulty one is chosen, the interval between consecutive displays is 1 second. If difficulty two is chosen, the interval between consecutive displays is 0.5 seconds. If difficulty three is chosen, the interval between consecutive displays is 0.25 second. Player loses the game if he fails to reproduce a pattern correctly, or if he takes longer than 5 seconds to reproduce a pattern. The current game level and a score that is a function of the current level and, the speed with which the player reproduces the pattern is displayed on available seven-segment displays. A winning/losing message or animation is also displayed on the seven segment display along with the final score at the end of the game. Segments and buttons are related as follows:

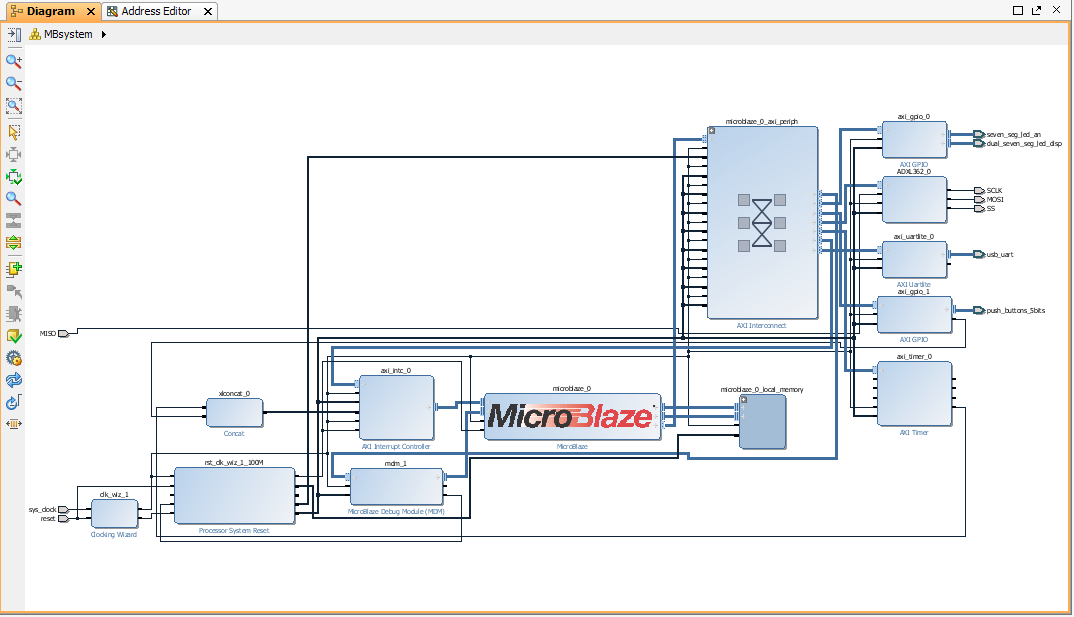


**Bonus:** The accelerometer is used to reproduce the patterns. In this case segment G (corresponding to the center button) is not used, and the accelerometer is only be used to detect up/down/right/left movements that correspond to segments A, D, B+C, and E+F, respectively.

1. **System Hardware Organization**

* Block Diagram:





* System's hardware components:
* **Microblaze processor**: controls all hardware components
* **AXI Interocnnect**: connects the GPIO, Interrupt Controller, accelerometer, timer… all peripheral devices to the microblaze processor.
* **GPIO**: The seven segment display is used to display the winning/losing messages, the game level, along with the sequences to reproduce. The buttons are used as inputs. The players had to reproduces the sequences using the appropriated buttons.
* **Interrupt Controller**: Manage interrupt from the gpio buttons and the timer that rolls back whenever it counts up or down from a reset value.
* **Timer**: The timer is used many times in the game. The players has to reproduces the pattern displayed within less than 5 seconds. The timer is also used to control the flashing of the sequences on the seven segment display.
* **Accelerometer:** Used to reproduce the patterns and Used to detect up/down/right/left movements that correspond to segments A, D, B+C, and E+F, respectively

1. **System Software Organization**

* **Main()**: Initilizations of variables, interrupts, timer, gpio. The SIMON message is displayed. In an infinite while loop, firs the Difficulty panel is displayed. After the player chooses his difficulty, level 1 id displayed, followed by the sequence to be repeated. Note that the button interrupt is disabled until the goscreen. Then, the go screen is displayed and the counter starts decreasing from 5. During that time, the program waits for the user to input the correct sequence (using the buttons or the accelerometer). If the correct code is inputted, the PASS message is flashed and the next level is displayed along with the sequence to be repeated. The process is repeated until the user wins or loses. In case of a loss, the message LOSE is displayed along with the score. If the winner finishes all 8 levels, the final score is displayed.
* **buttonsHandler(void \*CallbackRef):** Interrupt Handler for whenever a button is pressed. It is the interrupt service routine that is executed whenever a button on the GPIO is pressed. The CallbackRef parameter must be declared, but is not actually used in the handler. In this function, the user chooses the difficulty (1, 2, 3) on the menu display using the left, center and right button.
* **TimerCounterHandler(void \*CallBackRef, u8 t)**: Interrupt service routine for whenever the timer rolls back, generating an interrupt. The CallbackRef & 8 bit unsigned t parameters must be declared, but are not actually used in the handler. If the user wins, the timer is set to 5 seconds in order to exit the go scereen.
* **displayPattern(u32 display, u32 patIndx):** Displays a seven-segment pattern on a specific display. We read the anode activation and display patterns. We then use the patterns to drive the SSD
* **displayPatternSegment**(u32 display, u32 patIndx): displays the current game segment on the seven segment display. We read the anode activation and display patterns. We then use the patterns to drive the SSD. This function displays the buttons pressed (left displays the left bar, right displays the right bar etc)
* **displaySIMON()**: Displays on the seven segment display “SIMON”
* **displayLevel():** Displays on the seven segment display the level of the game.
* **displayGame(u32 lvl)**; Depending on the value of the input level, the function displays the sequence to be repeated by the user on the seven segment displays. The higher the level, the fastest and longer the sequence displayed.
* **displayLose();** Displays on the seven segment display “LOSE” along with the player’s final score.
* **displayPass():** Displays on the seven segment display “PASS”. The score is updated with the function: gameScore= gameScore + gameLevel\*diff where diff represents the game difficulty.
* **displayGoScreen():** Displays on the seven segment display “GO” with the counter counting down from 5 to 1 seconds. During that time the user should try and repeat the sequence displayed on the seven segment display.
* **Normal game**: When the user tries to replicate the sequence, the button pressed is compared to the value in the sequence. If the correct button is pressed, the user proceeds to the next level.
* **BONUS game:** for the bonus question, the accelerometer direction is checked during that time. With trial and error, the board directions were compared in respect to their x and y values (z values doesn’t affect in this case the direction). The thresholds were +/- 100 for x and y. If (x<-100) the board is moving up, if (x>100) the board is moving down, if (y>100) the board is moving left, and finally if (y<-100) the board is moving right. A variable accVar is used to move to the next number in the sequence when the board is put back in its original position.
* **checkAccel():** this function checks the actual value of the accelerometer against the value to be repeated in the sequence. If the accelerometer is titlted in the incorrect position, the player loses the game.
* **displayDiff()** displays on the seven segment display diff followed by 1 2 3. The player can choose the difficulty of the game by pressing the left button for difficulty one, center button for difficulty 2 and right button for difficutly 3.
* **gameDiffChosen():** Depending on the difficulty chosen, the time interval is set to 1 second for difficulty 1, ½ a second for difficulty 2 and 0.25 s for difficulty 3. Then a random sequence is generated for the game using rand() modulo 5 since buttons cans only be from 0 to 4. In the case of the accelerometer, the center value corresponding to the number 2 is not used in the sequence.
* **waitTime():** sets the interval between segment displays

1. **Conclusions and Future Work**

* Lessons:
* Use the accelerometer and its threshold values
* Use the accelerometer to generate random numbers as a seed.
* Add one function at a time
* Test the parts before the whole (we implemented the game without the accelerometer and then added it)
* Manage our time despite the fact we both had different finals and different schedules.
* Debug the program whenever it failed to output any result.
* Project Expansion:
* Add a beep from the buzzer every time the player loses or wins.
* Make the game a competition between two players