Project Documentation
Group 7 - Team BR

January 13th of 2023

1 Team members

- Luiz Henrique Rezende Ramos - 4220012
- Rafael Dal Moro - 4220013
- Victor Orfeu Merlo - 4220011
- Vinícius Carvalho Gomes - 4220009

2 Concept/Idea description

The idea of the project is to implement a simplified version of the video game known as "Guitar Hero" in the FPGA. In summary, the game allows users to play along to famous songs by hitting the notes in sync with the melody. Its core mechanic consists of a set of 5 tracks controlled by a button each. The melody‘s notes slide through the tracks until they reach the hit spot. To score points, the player must press the corresponding button at the right time.

Also, the melody is played when the player hit the note correctly as if an instrument played it. When the song finishes, performance stats such as points, and maximum notes hit in a streak show on the screen. Figure 1 shows a screenshot of the gameplay from Guitar Hero III.

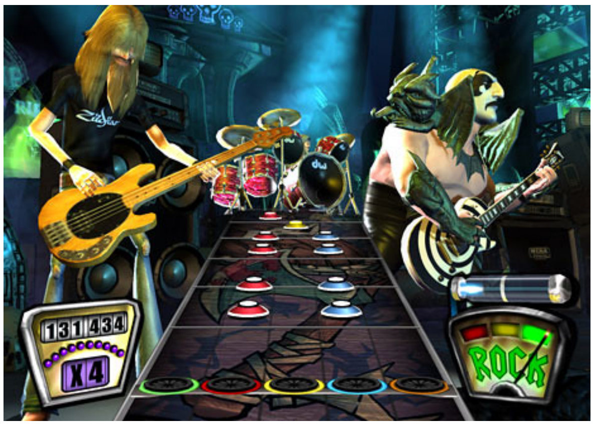
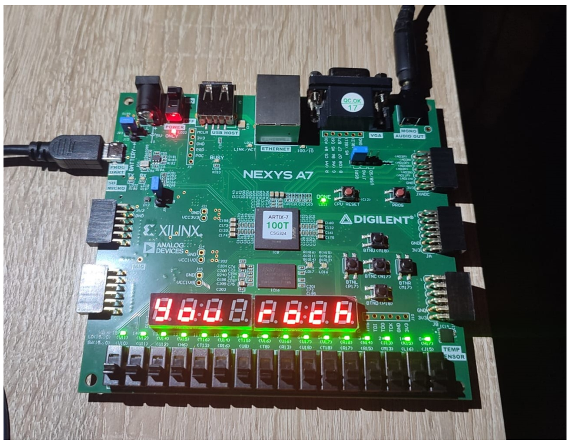


Figure 1: Gameplay screenshot from Guitar Hero III.

The simplified version implemented in the FPGA uses the 7-segment displays of the development board as the screen, with segments representing the notes moving through the tracks. The board‘s LEDs indicate the current streak of the player. The


(a) 7-segment Driver.

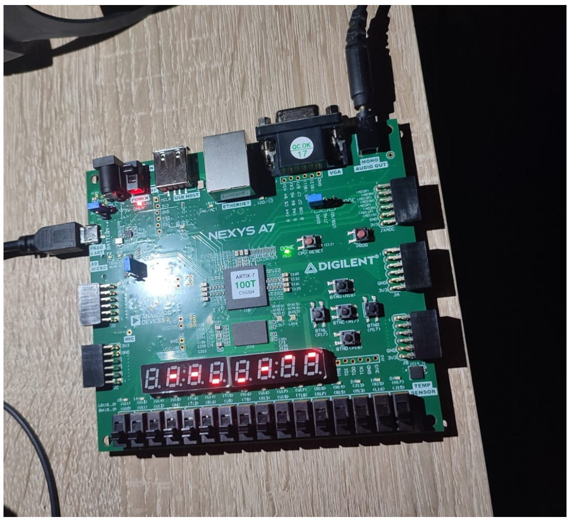

(b) GPIO and UART Drivers.

Figure 2: Project overview.

push buttons represent the player‘s controller. The sound of each note is played through the Mono Audio jack on the board.

The game has only one song, Asa Branca from Luiz Gonzaga. Figure 2a shows the displays when the music ends and figure 2b shows the displays while the music is being played. In both figures, the headphone connector can be seen on top, plugged into the board’s audio jack.

3 Project/Team management

The team leveraged its scale and synergy to simplify management processes. Most of the work was made collaboratively, in collective working sprints. First, all members met to create a backlog of tasks and requisites for the project development. Then, each member spontaneously picked a topic to explore existing answers on the internet.

In numerous moments, tasks were executed in doubles. This strategy incorporated the perks of peer programming into development, reducing the risks of late error identification. In later stages of production, external counseling was crucial and part of the team schedule an appointment with the professor.

Finally, testing was an essential part of the development process. Each feature was verified before advancing to the next task. This approach minimized the need for regression tests and promoted a healthy growth of production. The project’s backlog ended up as follows:

- Victor developed VHDL code and helped with MicroBlaze code.
- Luiz developed VHDL and MicroBlaze code.
- Vinícius developed MicroBlaze code and documentation
- Rafael developed MicroBlaze code and documentation

4 Technologies

The technological approaches used to implement the project were:

- VHDL IDE: Vivado
- FPGA board: Nexys A7-100T
- Board I/O: LEDs, 7-Segment Displays, Push-Buttons, and Audio Jack.
- Soft microprocessor: Microblaze
- Other VHDL components: 7-Segment Displays Driver and PWM Sound Driver

5 Hardware Implementation

The project was made using the Block Diagram tools in Vivado, so that we could use IP cores made by Xilinx and by the group members. The final block diagram, show in Figure 3, is divided into four main sections: 7-segment driver, GPIO driver, sound driver, and Microblaze with peripherals. Each of these can be seen in more details in Figure 4.

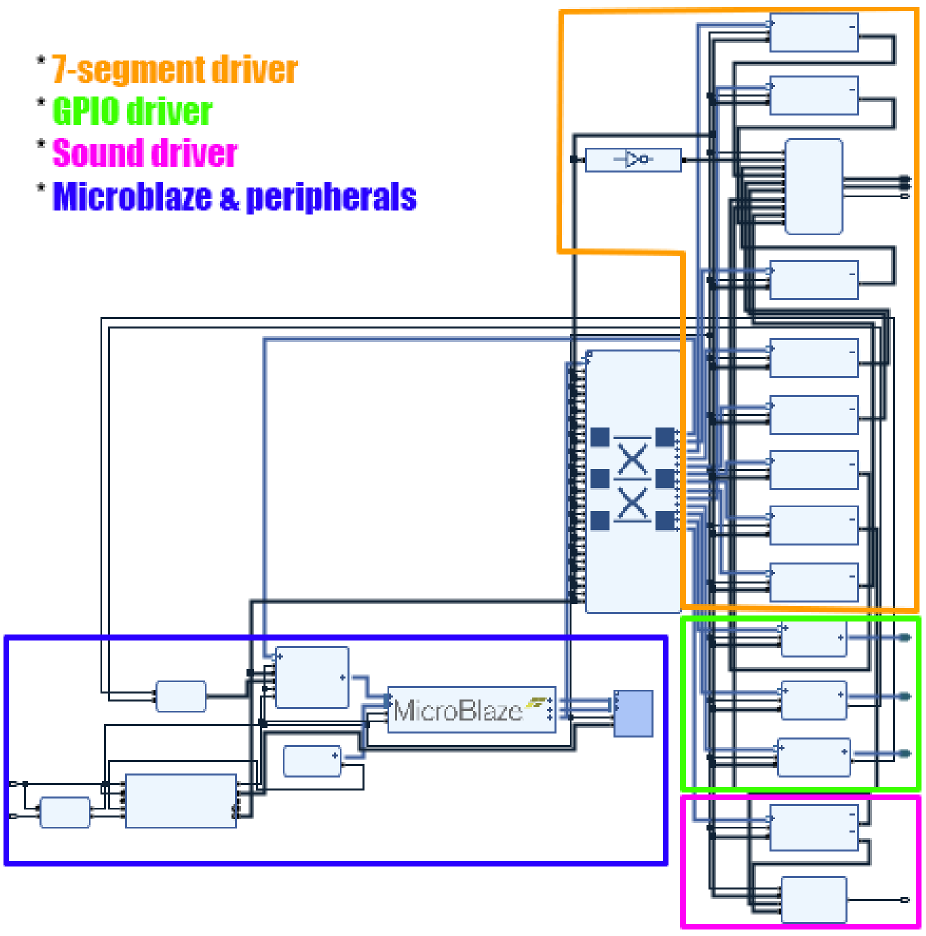
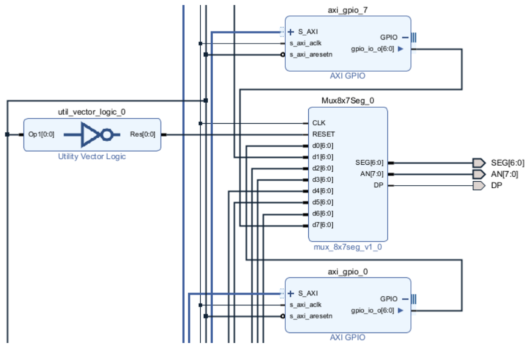
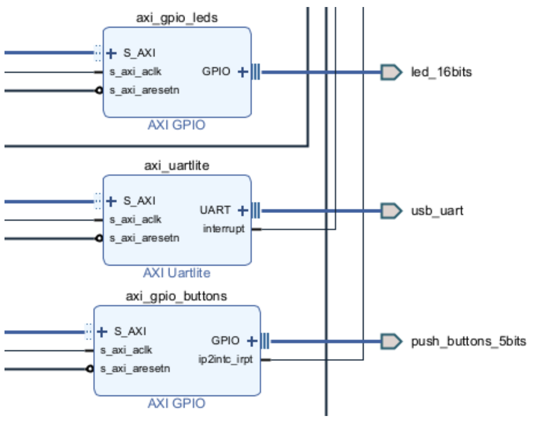


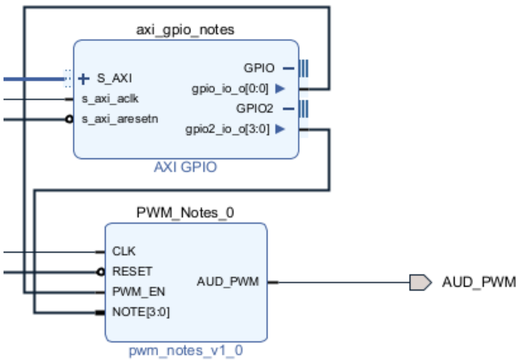
Figure 3: Complete Block Diagram of the hardware generated by Vivado



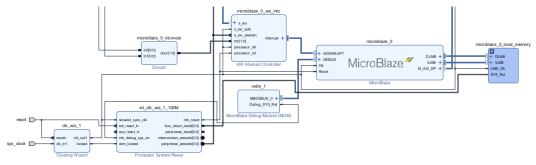
(a) 7-segment Driver.



(b) GPIO and UART Drivers.



(c) Sound Driver.



(d) Microblaze with it's auto-generated peripherals.

Figure 4: Detailed view of each main section of the Hardware Block Diagram.

The 7-segment driver consists of an IP block made by the group members and several AXI GPIO blocks. It connects to Microblaze through the GPIOs, and to the board through the `SEG`, `AN` and `DP` pins. The block receives as inputs what should be shown in each 7-segment display of the board. It then time-multiplexes the inputs to the correct displays, since the board only has one bus for the 7-segments and one bus to select which segments are turned on.

The sound driver also consists of an IP block made by the group members and a AXI GPIO block. the IP block receives as input a 4-bit vector that selects a note to play. It then outputs a PWM signal that goes to audio jack through the `AUD_PWM` pin. The audio jack has a Sallen-Key Butterworth filter that converts the PWM wave to an analog voltage as a function of the duty cycle. So changing the duty cycle periodically changes the voltage, creating an audible sound wave.

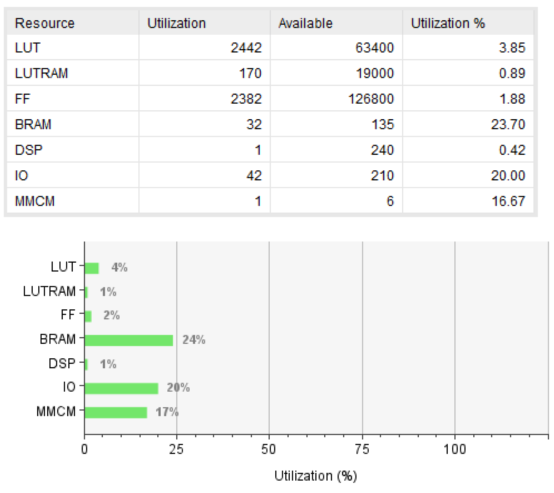
The GPIO drivers consist of 2 AXI GPIO blocks that interface with the LEDs and the push-buttons. The LEDs are controlled by Microblaze code that tells them when to turn on/off. The push-buttons have interrupt enabled so that the code can process any button press without requiring polling.

The UART driver is a IP block from Xilinx that provides connection between Microblaze and a serial terminal through the USB cable that also powers the board. Vitis own terminal was used for this serial interface.

The Microblaze is a soft microprocessor made by Xilinx. It's peripherals were automatically instantiated and connected by Vivado and include: Memory, Clock controller, Interrupts controller and AXI controller.

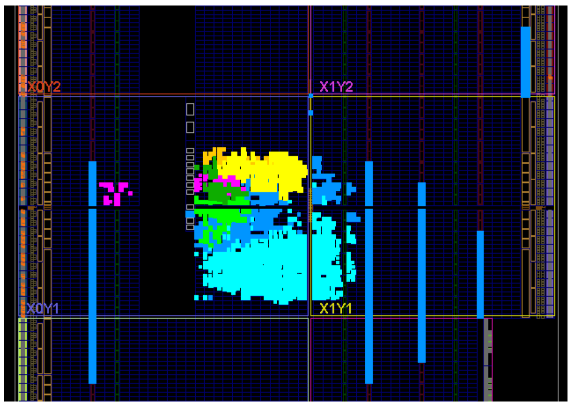
About utilization, Figure 5a shows a summary of resources utilization, created by Vivado. Figure 5b, also created by Vivado, shows how these resources are distributed across the FPGA device. The components, similarly to Figure 3, where highlighted using the colours as follows:

- Light blue: Microblaze
- Blue: Microblaze's peripherals (Memory and clock, interrupts and AXI controllers)
- Orange: 7-segment driver
- Yellow: GPIO blocks connected to the 7-segment driver
- Dark green: UART driver block
- Green: GPIO blocks connected to LEDs and push-buttons
- Pink: Sound driver



Resource	Utilization	Available	Utilization %
LUT	2442	63400	3.85
LUTRAM	170	19000	0.89
FF	2382	126800	1.88
BRAM	32	135	23.70
DSP	1	240	0.42
IO	42	210	20.00
MMCM	1	6	16.67

(a) Summary of resource utilization.



(b) View of device utilization.

Figure 5: Project implementation results.

6 Software Implementation

A block diagram of the software implementation, written in C, is shown in Figure 6.

Each 7-segment display is represented by a position in an 8 positions vector (named `sequence`). In each iteration of the loop the vector is updated and projected in the 7-segment display. Two 147 position vectors are previously defined, the first defining the sequence of positions to be displayed in the 7-segment display (named `song`), and the second defining the notes to be played at each correct pressed button (named `melody`).

An interruption was implemented to update a variable representing which button the player has pressed. When a note reaches the last position of the sequence vector (index 7) the program will check if the pressed button corresponds to the note on the display. If so, the player gets a point and the streak score is updated (if this is the highest streak score, this information is also updated). If not, the streak score becomes 0.

Once all notes have been read from the `song` vector into the `sequence` vector, the game ends. When the game ends the 7-segment will display "you rock" and the highest streak, percentage of notes hit and total score will be displayed in the serial monitor.

A reset button was also implemented that, when pressed, will exit the loop and stop the game.

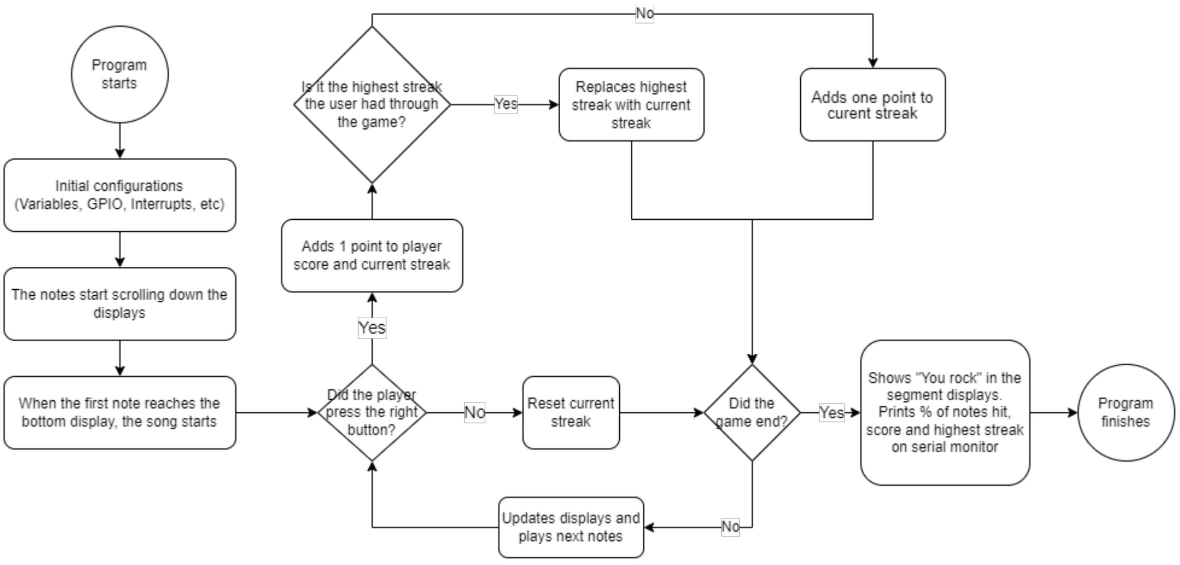

Program starts
Initial configurations (Variables, GPIO, Interrupts, etc)
The notes start scrolling down the displays
When the first note reaches the bottom display, the song starts
Did the player press the right button?
Yes
Adds 1 point to player score and current streak
Is it the highest streak the user had through the game?
Yes
Replaces highest streak with current streak
No
Adds one point to curent streak
No
Reset current streak
Did the game end?
Yes
Shows "You rock" in the segment displays. Prints % of notes hit, score and highest streak on serial monitor
Program finishes
No
Updates displays and plays next notes

Figure 6: Block Diagram of the software implementation

7 Sources/References

This project's interruption logic was inspired by the example codes from Xilinx, available inside Vitis. The PWM generator inside the Sound driver was taken from `https://forum.digikey.com/t/pwm-generator-vhdl/12652`. Finally, the song code was built from `https://github.com/robsoncouto/arduino-songs`, accessed: 09/01/2023.