EEL 4930 - System-on-Chip Design

Final Project Report

Submitted by,
Harshit Raj
Nitesh Bakhati
Benjamin Jacob
Alexis D. Villegas

Task 1

In this initial phase, 8-bit audio tables were created to represent different sound effects and melodies triggered by specific game events. The following scenarios were addressed:

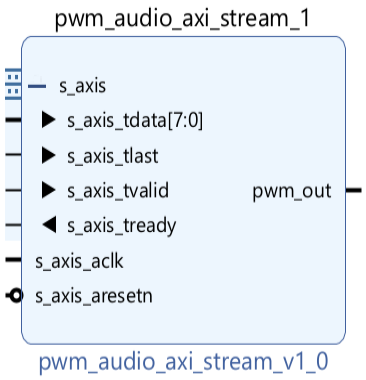
- **Win/Loss Event:** When a player wins or loses, the system plays the melody "Mary Had a Little Lamb," following the provided reference.
- **Ball Hits Paddle:** A distinct 0.25-second tone is played each time the ball contacts a paddle.
- **Point Scored:** A separate 0.25-second tone, different from the paddle tone, is played whenever a point is scored.

To achieve this, tables of digital values corresponding to the musical notes and tones were generated. These tables serve as the basis for audio playback in subsequent tasks.

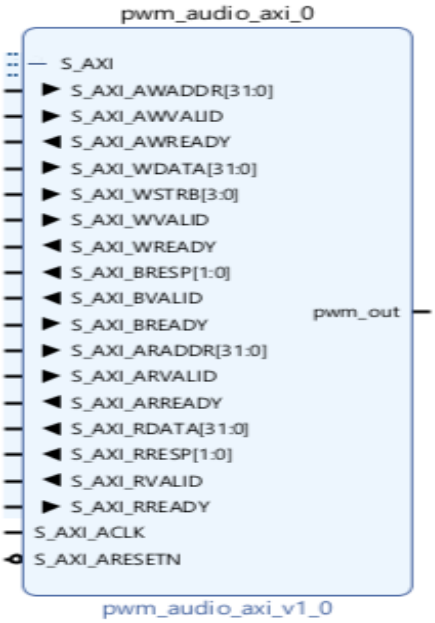
Task 2

A Pulse Width Modulation (PWM) driver was developed to output audio signals through the Urbana Board's audio port. The PWM driver is responsible for converting the digital audio tables into analog signals, enabling the playback of tones and melodies via a connected speaker. We have created two of the PWM custom IP one of them is AXI_Stream and the other one is AXI_MMSlave but have used AXI_Stream during our implementation.

Below is the block diagram of the AXI_Stream IP that we have used in our implementation.



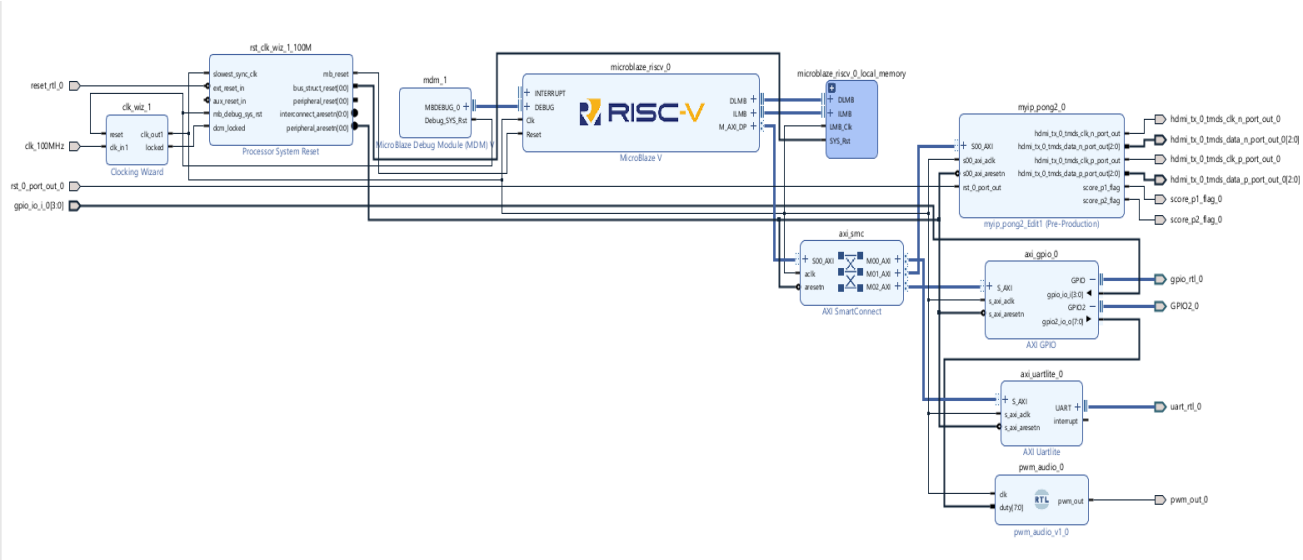
Below is the block diagram of the AXI_MM (Memory-Map) Slave IP that we have used in our implementation.



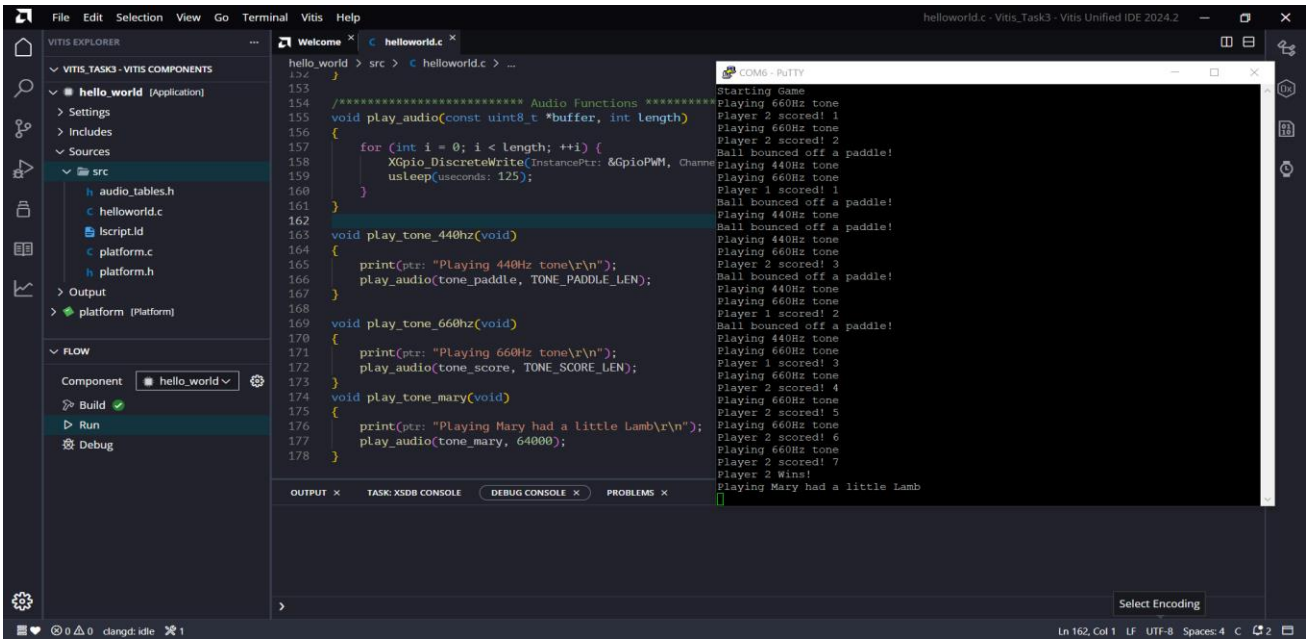
Task 3

In this stage, the system-on-chip (SoC) was constructed without using Direct Memory Access (DMA). The implementation included integrating the Pong game logic (Player vs. Environment) from the previous lab and adding hardware to support PWM audio output. Developing software to trigger and play the audio forms generated in Task 1, based on in-game events. The result was a fully functional Pong game where the specified audio cues are played directly from the board in real time.

Below is the block diagram of the SoC without using DMA.



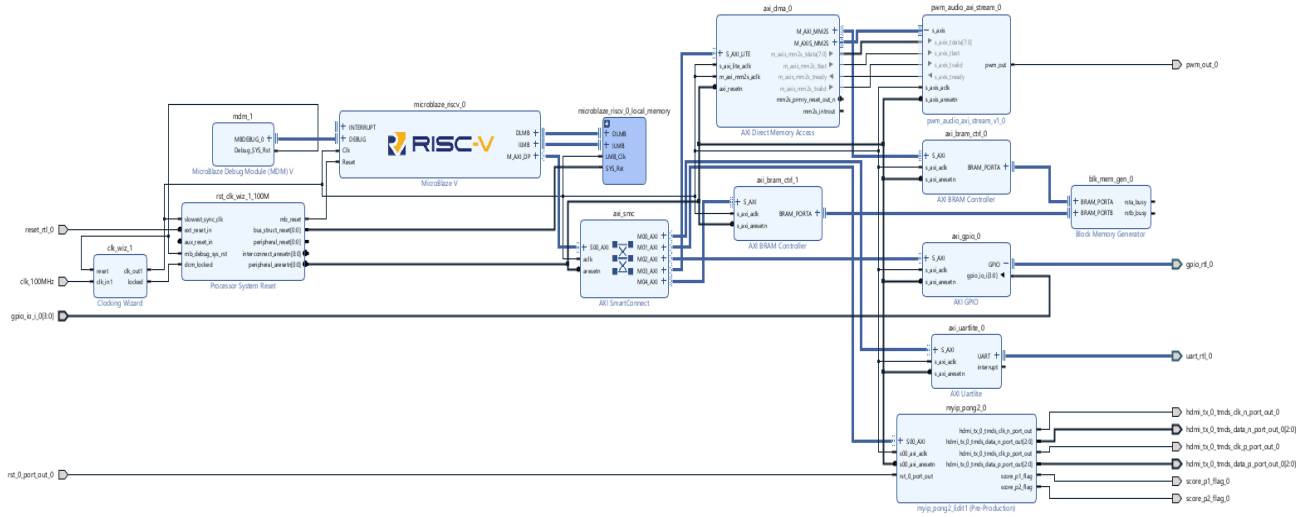
The screenshot of the output terminal for task 3 is attached below.



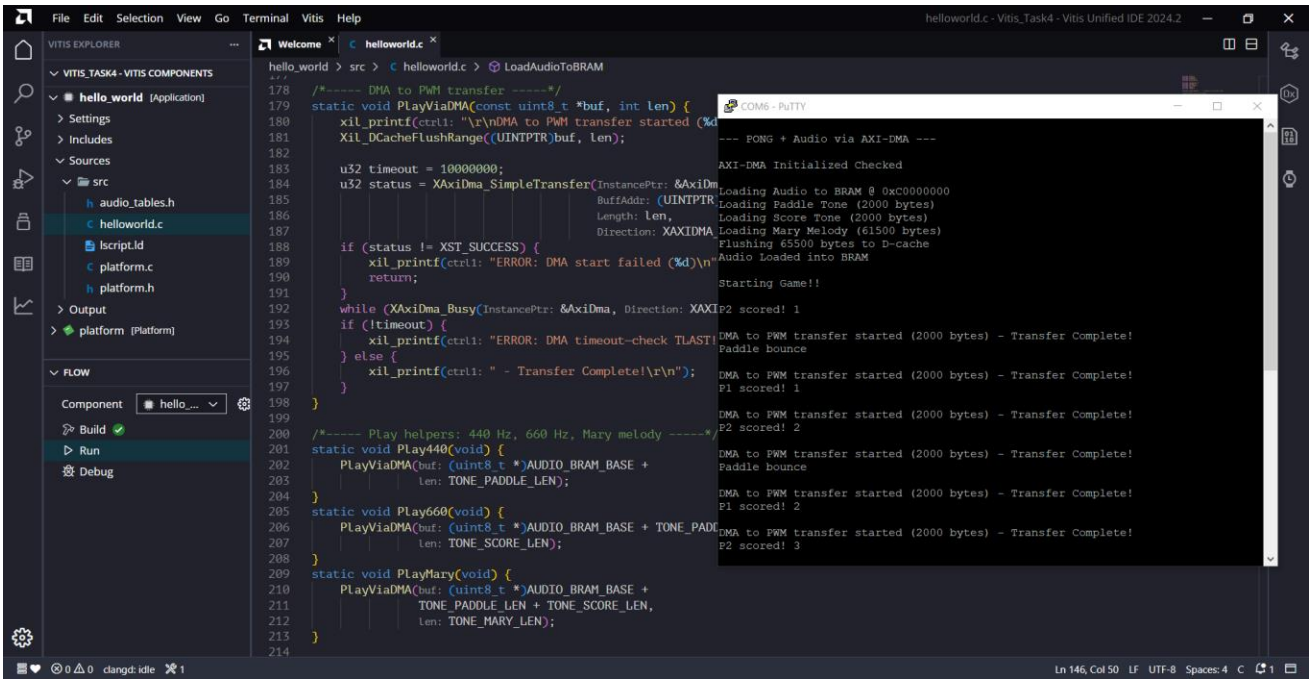
Task 4

In this task, we extended the previous implementation by incorporating the AXI Direct Memory Access (CDMA) controller. This allowed efficient transfer of audio note data to the PWM driver, reducing CPU overhead and improving overall system performance. The audio output behavior remained the same as Task 2, but with enhanced data handling using DMA.

Below is the block diagram of the SoC using DMA



The screenshot of the output terminal for task 4 is attached below.



Problems Faced & Troubleshooting

While Tasks 1 through 3 were implemented without significant obstacles, Task 4 presented considerable technical challenges. The integration of Direct Memory Access (DMA) into our design required multiple iterations and architecture modifications before achieving success.

Our initial approach involved wrapping our Pulse Width Modulation (PWM) module into an AXI_Lite IP component for integration with the AXI Central DMA (CDMA). However, we subsequently discovered that AXI_Lite interfaces cannot be directly connected to CDMA controllers as they are unable to handle burst signals from CDMA due to protocol incompatibilities (which can be solved by adding AXI interconnect between CDMA and AXI_Lite interface).

After evaluating our options, we determined that AXI_Stream would be more suitable than AXI_Lite for audio signal transfer applications, as it offers better support for continuous data streaming required in audio processing. Despite this change, we encountered additional difficulties when attempting to access the Local Memory of the Microblaze processor through the AXI DMA.

Our final solution involved implementing a Block RAM (BRAM) unit as an intermediary buffer between the Microblaze processor and the DMA controller. This architecture allowed the Microblaze to write audio note data to BRAM, from which the DMA controller could then efficiently transfer it to the PWM module. This approach effectively resolved our integration challenges while maintaining the intended functionality of our audio system.