EE/CSE 371
Nov 7, 2025
Lab 3 Report

Lab 3

Design Procedure

This lab explores digital signal processing on the FPGA using the onboard audio CODEC to capture and output sound. In Task 1, the focus is on passing an input audio signal through the FPGA to the speakers. The design reads data from the input and writes it to the output when both buffers are ready, preventing dropped samples. Task 2 generates a tone from a memory file. Waveform samples are generated via a python script and stored in a ROM MIF file. It is played back through the CODEC, producing a steady note sound and does not rely on live audio input like task 1. This task showcases that sound can be digitally recreated from stored data. Task 3 introduces an FIR filter to filter noise from audio. A FIFO buffer and accumulator track the last N samples and compute a moving average that reduces fluctuations in the signals. We are able to switch between filtered and unfiltered output through a switch so the effect can be compared and recorded on both the audio from Task 1 and Task 2. All three tasks are combined in a top level module. Switches toggle between the audio recording, the tone from memory tone, and their respective filtered signals which are sent to the speakers. This lab demonstrates digital signal processing, filtering, synchronization, buffering, and modular design. It also serves as a good example of how audio is processed in real time.

Overall System Descriptions, Charts and Block Diagrams

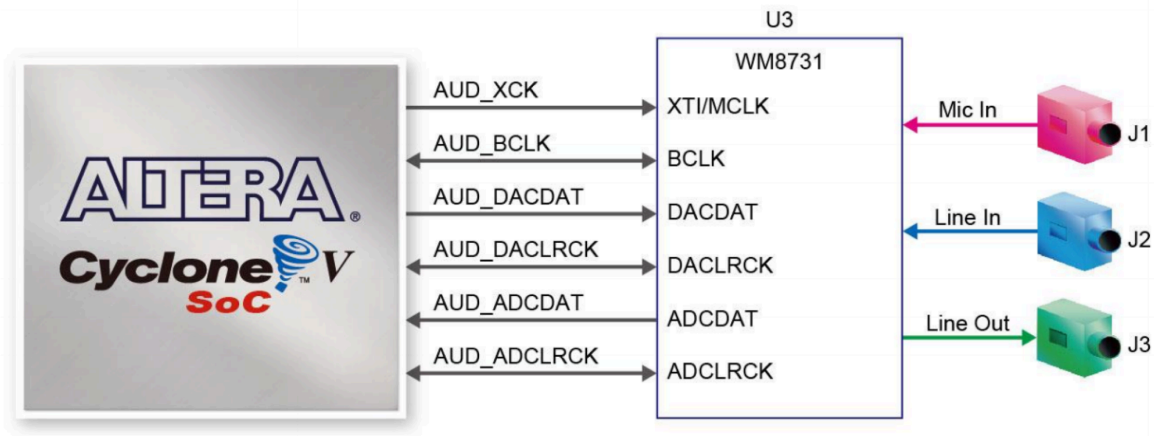


Figure 3: Connections between the FPGA and the Audio CODEC.

Figures 1: CODEC and Circuit Interface for Task 1 circuit

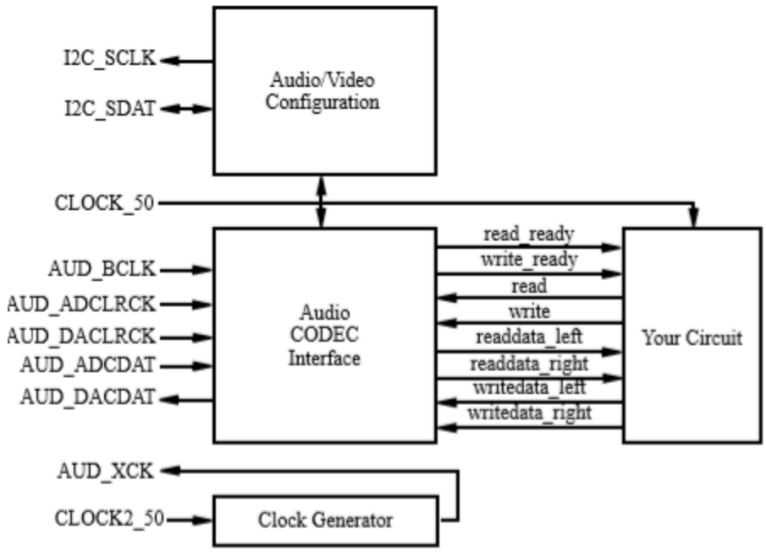


Figure 2: Audio System in Lab 3

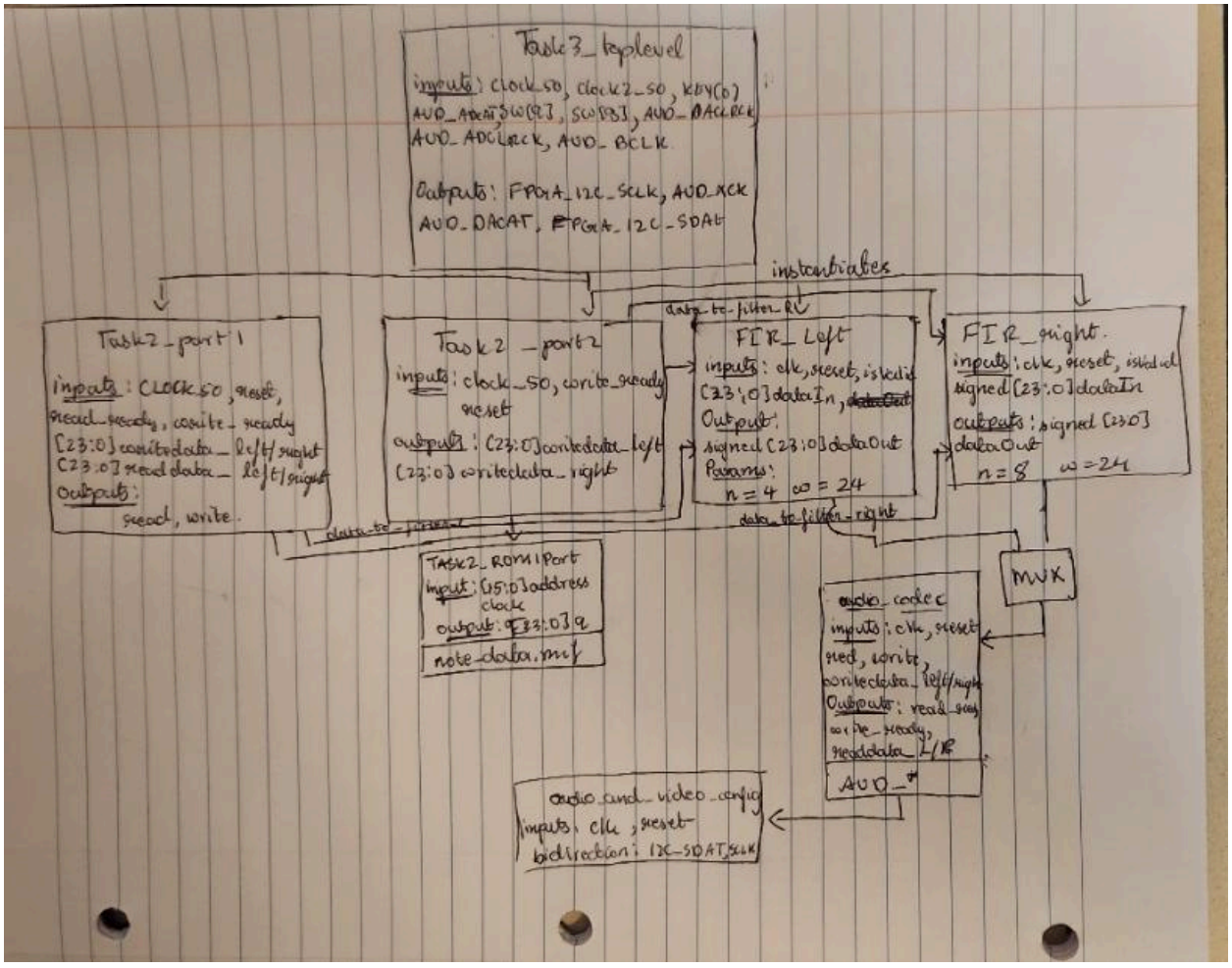


Figure 3: Module interactions

Task 1

Task1_part1 is a top-level interface for an audio system on an FPG. It connects an audio codec to manage audio input and output. It takes two clock inputs (CLOCK_50 and CLOCK2_50) and a

reset button (KEY[0]), and provides connections to an I2C interface (FPGA_I2C_SCLK/SDAT) for configuring the audiocodec and audio signals (AUD_*) for transmitting and receiving audio data in the circuit. Wires handle reading and writing audio data for left and right channels and pass audio from the input microphone to the output speaker. It works by assigning read data directly to write data. Reads and writes can only occur when the codec is ready meaning read_ready and write_ready signals are high. It acts as a wire for sound data looping the audio input back to the output creating a playback That means the codec is ready to both provide new input samples and accept new output samples.Next, three submodules are instantiated, clock generator creates the clock for codec, audio_and_video_config sets up codec over I2C, and audio_codec handles reading/writing to the interface and managing ready signals and the left and right audio data.

Task 2

Task2_part1 is similar to Task1 part 1 with a few changes made for integration. In Task 1, the module served as the top level design and directly connected to the Codec and handled audio read/write. It controlled the entire Codec pipeline. In Task 2, it plugs into Task2_TopLevel and only provides read/write signals and audio data which get passed up to the top module. It serves as a microphone passthrough function. Codec and I2C hardware interfaces, AUD_XCK, AUD_BCLK, AUD_DACLRCK, AUD_ADCLRCK, AUD_ADCDAT, AUD_DACDAT, are removed. By putting clock_generator, audio_and_video_config, audio_codec instantiation in the top level, the Task 1 functionality is kept modular and reusable. Task 1 still generates read and write signals but those signals only take effect when they are selected by SW[9] in the top level.

Task2_part2 is a tone generator. Unlike Task 1 which records audio from the mic, part 2 plays a stored waveform from memory. A counter ROM_address traverses through the address indexes of the ROM file. It starts at zero and each clock cycle when write_ready is high, it increments this address to read the next 24 bit audio data. On the clock edge, if the audio codec is enabled to accept new data, a chunk of data is read Tone_data and sent to speaker channels. If the counter reaches end index address of the ROM, 47999, it wraps back to zero. The ROM is instantiated using Task2_ROM1Port and is a memory lookup table with each address in ROM corresponding to a part of a waveform that forms the tone. This allows the tone to sound smooth and continuous when listening. A reset signal KEY[0] clears the counter and starts play accessing the audio data at the beginning of the ROM. There are no read signals in this module because there is no live input. The read data from the ROM is sent to both writedata_left and writedata_right, so the same tone plays on both speaker channels. It just generates and outputs this audio automatically when the write_ready signal is high.

Task2_TopLevel decides whether Task, live microphone playback, or Task 2, generated tone, is active. SW[9] allows the user to switch between the two. It connects a 50 MHz clock, reset KEY[0], and audio codec interface just like Task 1, adding selection logic to choose which subtask audio data is sent to the codec. An always block inside monitors the reset signal and switch input. When the system resets, the outputs and control signals get set to zero. When SW[9] is 0, Task 1 is active audio data readdata_left/right is immediately written to the output

writedata_left/right. Read and write are controlled by the codec's ready signals. When SW[9] is 1, Task 2 provides tone data from the ROM into the output. This allows the user to switch between live audio passthrough and pre-recorded tone playback on the FPGA.

Task 3

In Task 3, we integrate an N sample moving average FIR filter into the audio system to remove noise from both live microphone input (Task 1) and pre-recorded tones (Task 2). The top-level module adds filtering capability controlled by SW[8], which allows users to toggle between filtered (SW[8] = 1) and unfiltered (SW[8] = 0) audio output while maintaining the existing Task 1/ Task 2 selection via SW[9].

The top-level module for Task 3 instantiates 2 FIR_Filter instances, one for the left audio channel and one for the right channel, each with parameter n = 8, resulting in a 256-sample window. Filters receive their clock. From clock_50, Reset from KEY[0], and use write_ready from the audio codec as the is_valid signal. To ensure filtering occurs only when valid audio is available, always block routes the appropriate audio data based on SW[9] into data_to_filter_left and data_to_filter_right signals that feed the filters continuously. Based on SW[8], the module then selects either the filtered outputs or original unfiltered signals to send to master_output_left/right, which connect to the audio codec. The filter uses write_ready as its isValid signal to ensure synchronization with the codec.

The FIR_Filter module implements a moving average using a FIFO buffer and an accumulator. It operates in 2 phases: prefill and sliding window. During prefill, the first 2^n samples are accumulated while a counter tracks from 0 to 2^n. Each input is divided by 2^n via arithmetic right shift before entering the FIFO and no samples are being removed yet because the filter does not yet have enough samples for a full window. Once prefill completes, sliding window mode activates, where both FIFO read and write happen simultaneously. FIFO read removes old samples, FIFO write allows new samples to enter. The accumulator maintains a running sum by adding the newest sample and subtracting the oldest. Without the prefill functionality, the output data would be incorrect at the beginning because the average would be based on fewer samples than necessary. The output extracts bits [23+n:n] from the accumulator. [23+n:n] slices out the middle of the accumulator so output stays in the same scale as input. The accumulator grows wider because it keeps adding divided samples, so its top bits hold the actual average while lower n bits only store fractional detail. This shifts the value right by n bits, dividing by 2^n,, which brings the large summed number back to the original 24 bit range of data. dataIn, dataOut, divided, out, and accumulator are declared as signed thus the filter preserves correct polarity of the signal so the filtered output accurately reflects the waveform instead of treating negative values as large positive numbers which distorts sound.

Results and Testing

Figure 3: LabsLand interface in audio mode showing playing and recording to device

Task 1

No test bench was required for Task 1. Given that only 4 lines of code were added, we were able to test functionality on the board after confirming with TA that our logic was correct.

Task 2

This testbench tested Task2_part2 functionality and that the ROM in Task2_part2 outputs the correct audio values for given addresses. It is simulating a portion of the MIF file. It begins by declaring a write_ready signal, a reset, a 16-bit ROM_address, and a 24-bit wire writedata_left which will carry the ROM output. It instantiates a simple ROM module and stores the first eleven expected sample values in a small memory array. An array of expected values at those addresses is preloaded for the first eleven addresses. First reset is asserted for a few clock cycles to initialize address and other registers then deasserts to start normal operation. A for loop

iterates over the first eleven addresses assigning ROM_address and waiting a clock cycle for the ROM output q to propagate. The ROM itself is a straightforward model containing 48,000 word memory array, initializes the first eleven entries to match the expected audio samples, and on a positive clock edge of clock outputs the value at the current address to q. This allows the testbench to confirm ROM addressing and data output behave correctly.

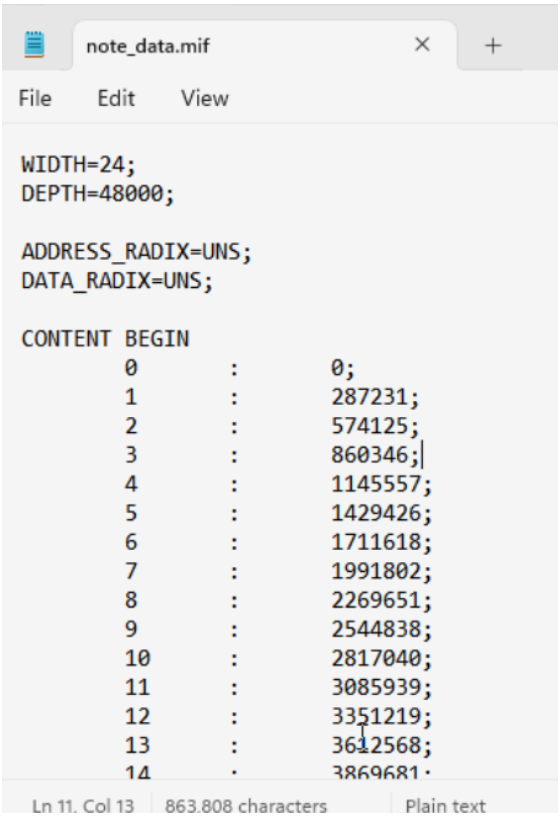


Figure 4: note_data.mif showing signal values for Task2 Part 2 signal generation.

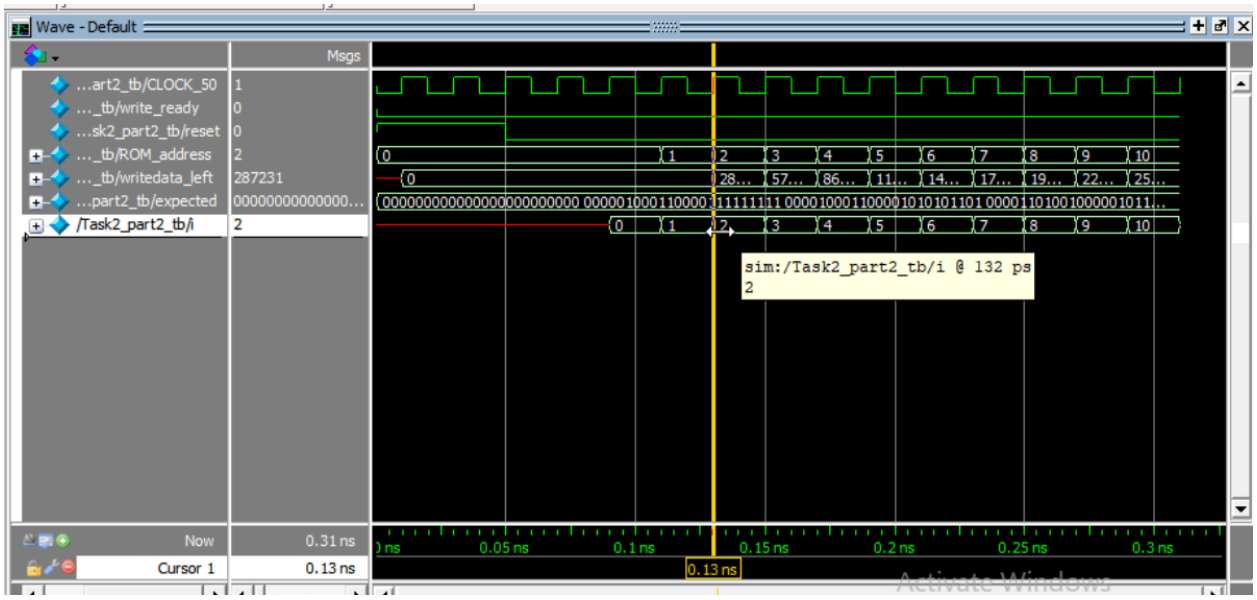


Figure 5: ModelSim Waveforms for Task2_Part2 showing the signal value 287231 at ROM address 2 - performing as expected.

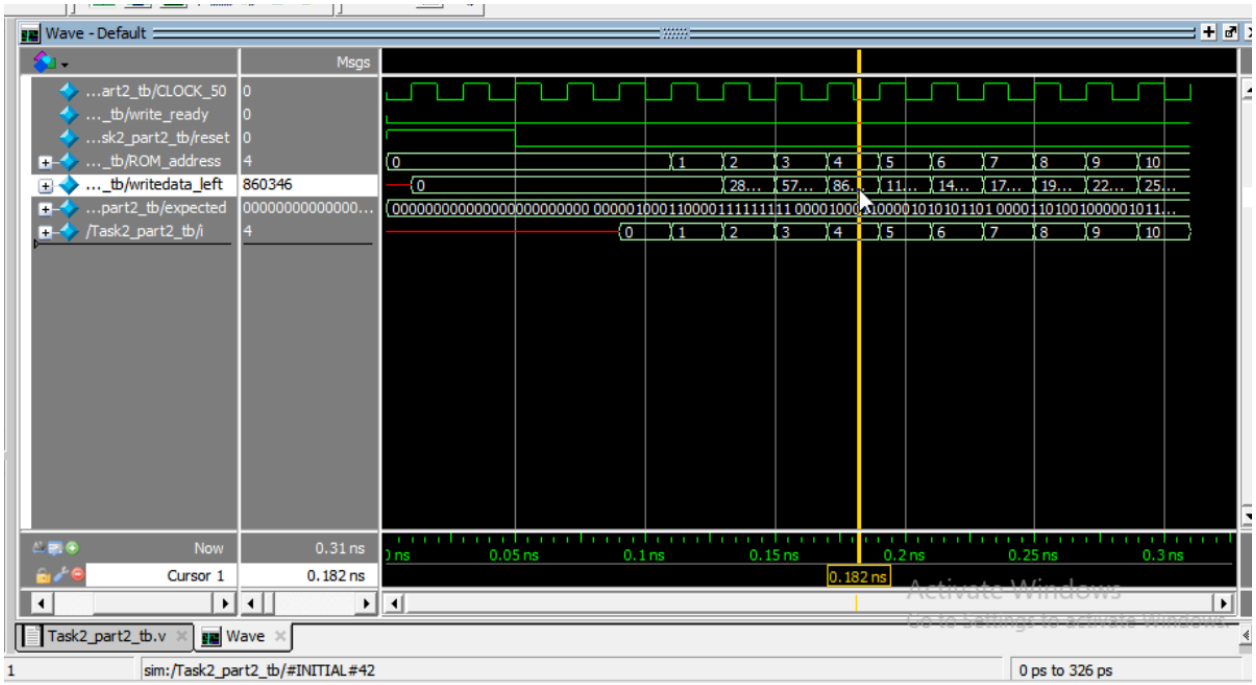


Figure 6: ModelSim Waveforms for Task2_Part2 showing the signal value 860346 at ROM address 4 - performing as expected.

Task 3

The test bench we created tests the FIR_Filter's moving average functionality, and the sliding FIFO buffer operation (Sliding window). It instantiates the FIR_Filter with parameters n = 4 (16 sample window) and w = 24 (24 bit data width).

The test sequence demonstrates the filter's 2 operating phases. It first sends 18 samples of 160 to complete the prefill phase (first 16 samples) and reach a steady state, where data out stabilizes at 10 ((160/16)/16). This is shown in the screenshot below, where dataOut is stabilized at 10 between 195000 ps and 245000 ps:

Figure 7: ModelSim screenshot showing stabilized dataout = 10 over multiple inputs

Next, the testbench introduces 3 samples each of progressively higher values (320, 480, and 640) to demonstrate the sliding window functionality. As new values enter the 16-sample window, the dataOut value transitions gradually after every increment (note that there is a delay between the data in and data out changes due to a 1-cycle latency in our design)

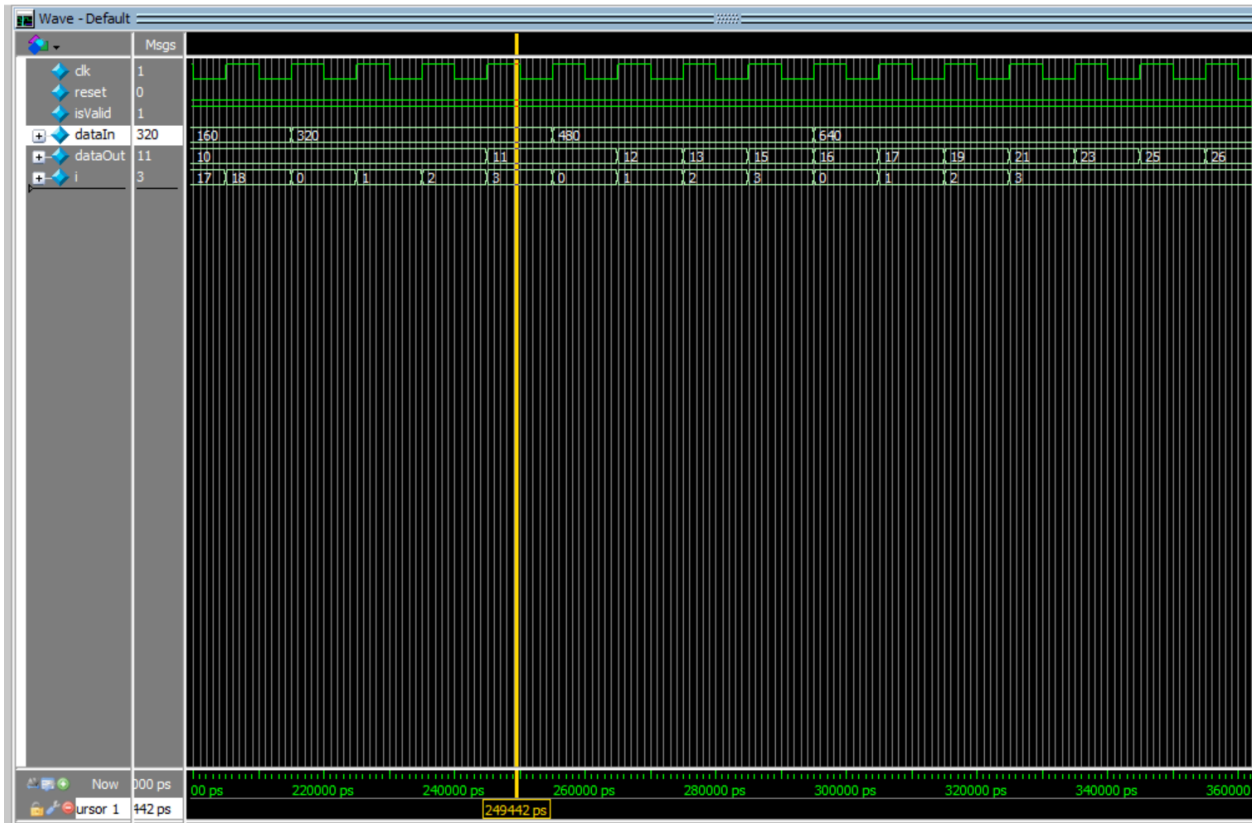
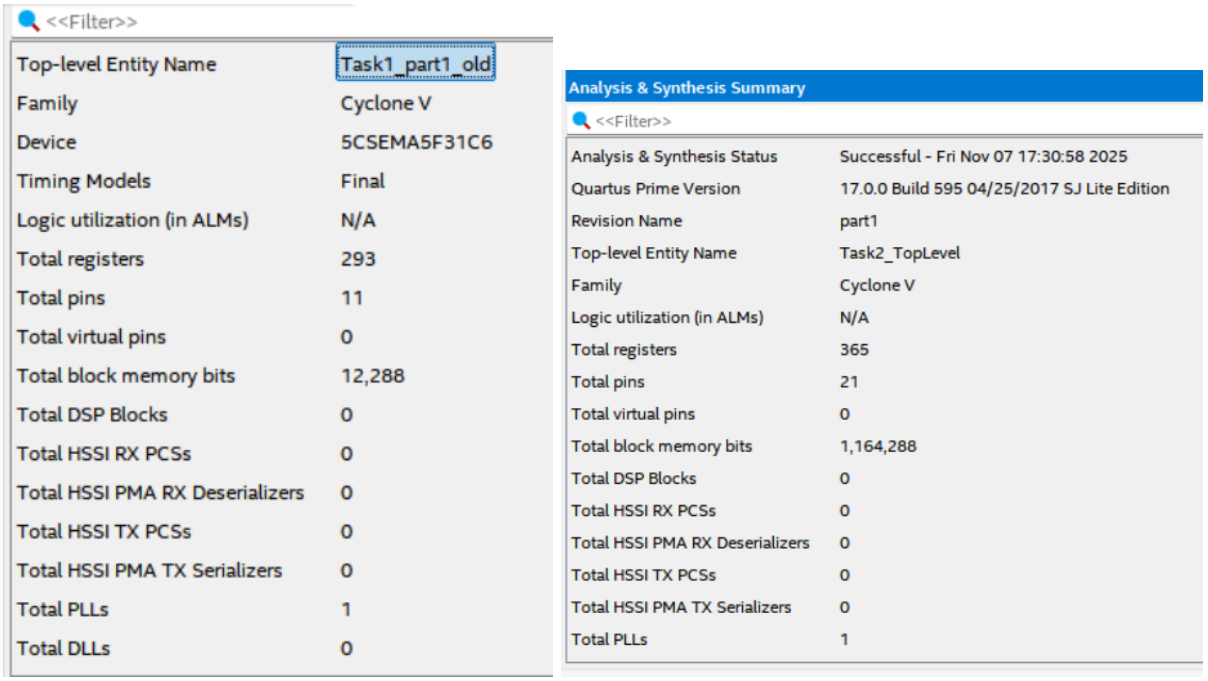


Figure 8: ModelSim screenshot showing a gradual increment of the dataOut value with new value addition

As shown above, the waveforms clearly show the gradual transitions in dataOut as the window slides, demonstrating that both the prefill accumulation and the sliding window mechanism work.

Flow Summary



Top-level Entity Name	Task1_part1_old
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	293
Total pins	11
Total virtual pins	0
Total block memory bits	12,288
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	1
Total DLLs	0

Analysis & Synthesis Summary

Analysis & Synthesis Status	Successful - Fri Nov 07 17:30:58 2025
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	part1
Top-level Entity Name	Task2_TopLevel
Family	Cyclone V
Logic utilization (in ALMs)	N/A
Total registers	365
Total pins	21
Total virtual pins	0
Total block memory bits	1,164,288
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	1

Flow Summaries for Task 1 & 2

Top-level Entity Name	Task3_TopLevel
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	1079
Total pins	21
Total virtual pins	0
Total block memory bits	1,176,576
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	1
Total DLLs	0

Flow Summary for Task 3