Introduction

The objective of this lab is to implement a Digital Finite Impulse Response (FIR) filter on an FPGA, using MATLAB HDL Coder and Vivado for rapid prototyping and hardware co-simulation.

FIR filters are widely used in digital signal processing (DSP) due to their inherent stability and linear phase characteristics.

In this task, we first verify the functionality of a given FIR filter using a MATLAB software implementation, followed by an FPGA implementation.

We then optimize the design using pipelining, retiming, and data broadcasting techniques to improve performance in terms of critical path delay and overall timing, at the cost of increased area complexity.

This lab provides an opportunity to explore model-based design techniques for FPGA development and the use of HDL Coder for efficient DSP implementation.

Design Description

The following digital filter was implemented:

$$y(n) = \sum_{i-0}^{16} a_i x(n-i)$$

Un-optimised design

The un-optimised FIR filter design show in figure 1, follows a simple direct-form structure, where the input signal is processed through a series of Delay, Gain, and Add blocks. Each sample of the input signal is multiplied by a given coefficient, and the results are accumulated using the Add blocks. The design includes delays at each stage. The Gain blocks apply these coefficients, and the Add blocks accumulate the results to produce the output signal.

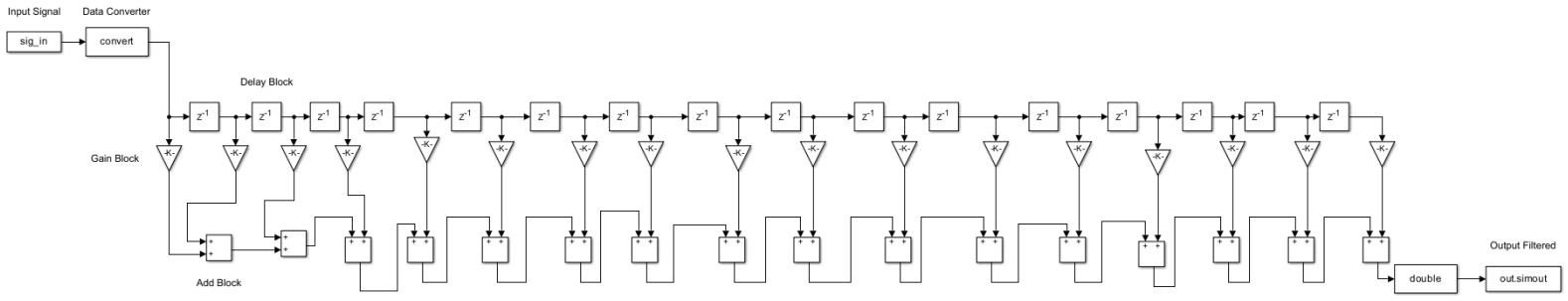

Figure 1 Un-optimized FIR filter

Optimised design

The optimised FIR filter design improves upon the un-optimised design by incorporating techniques such as pipelining, retiming and using a data broadcasting structure as seen in figure 2.

Retiming was used to move the existing delays before the gain to before the addition block optimising the placement of these registers to minimize the delay, having a data broadcast structure, then pipelining inserts additional registers to separate operations.

Although, this increased the latency and amount of resources, it reduced the critical path

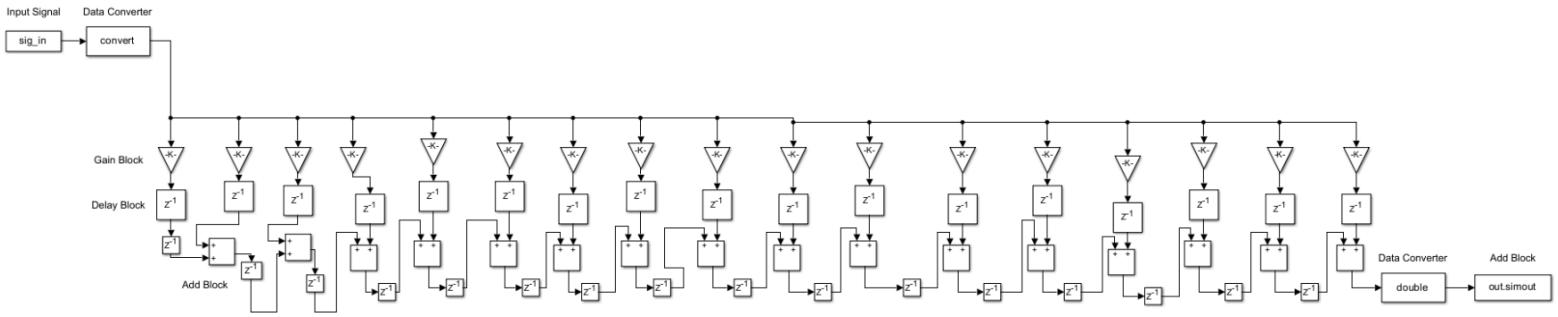

Figure 2 Optimized FIR filter

Word-length selection

The selection of the fixed-point word length for the FIR filter design was primarily based on trial and error, with the goal of achieving a good balance between precision and hardware efficiency. The process involved adjusting the word lengths and analysing the output graphs to minimize errors while ensuring that FPGA resources were utilized effectively.

For the **Convert block**, which mimics the Analog-to-Digital Converter (ADC), the word length was selected as **W = 24** and **D = 12**. This choice was made after testing various options and analysing the output graph, particularly focusing on the attenuation of high frequencies. This configuration provided the required precision without introducing unnecessary resource overhead.

Similarly, the **Adder** and **Gain blocks** were also configured with a word length of **W = 24** and a precision of **D = 12**, maintaining consistency across the components and ensuring sufficient accuracy for signal processing.

For the **Gain blocks' parameter precision**, the fractional bit length was adjusted based on trial and error. Through testing, it was determined that **W = 24** and **D = 10** provided the best results. This configuration allowed for better attenuation of the high frequencies compared to the software implementation, where increasing or decreasing the fractional bits beyond **D = 10** resulted in performance degradation, further deviating from the desired software output.

Functional Simulation Results

The functional simulation results for both the un-optimised figure 3 and optimised designs figure 4 were compared to the software implementation to verify the correctness and performance of the hardware implementation.

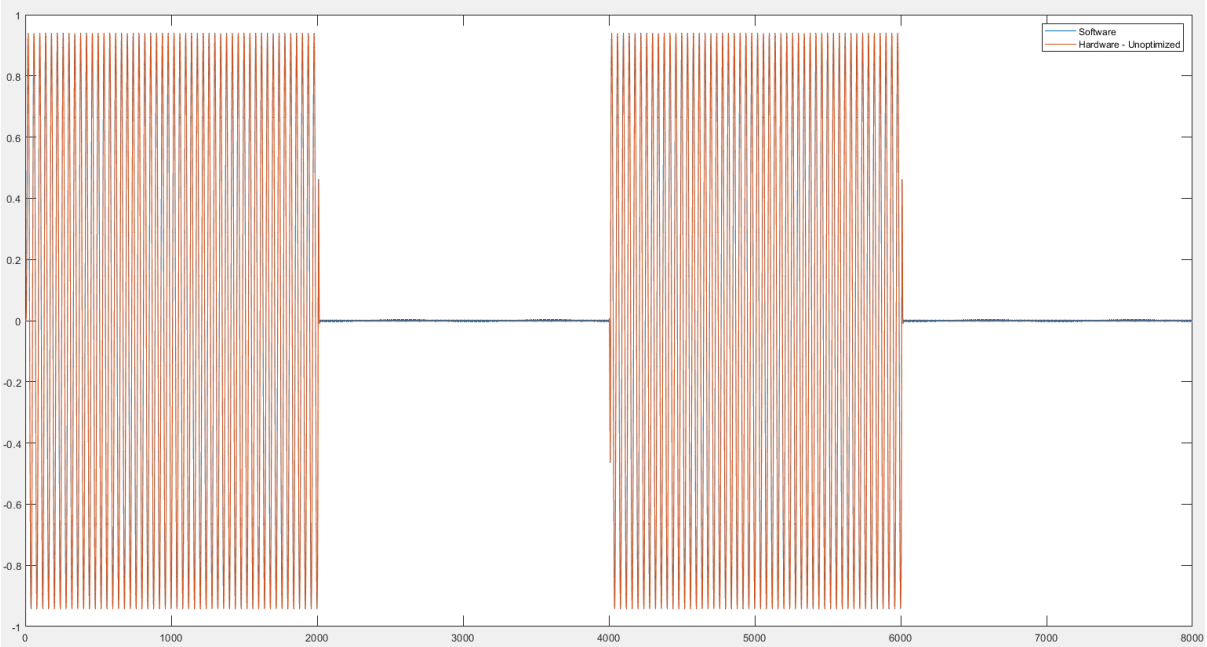

Figure 3 Un-optimised filter output

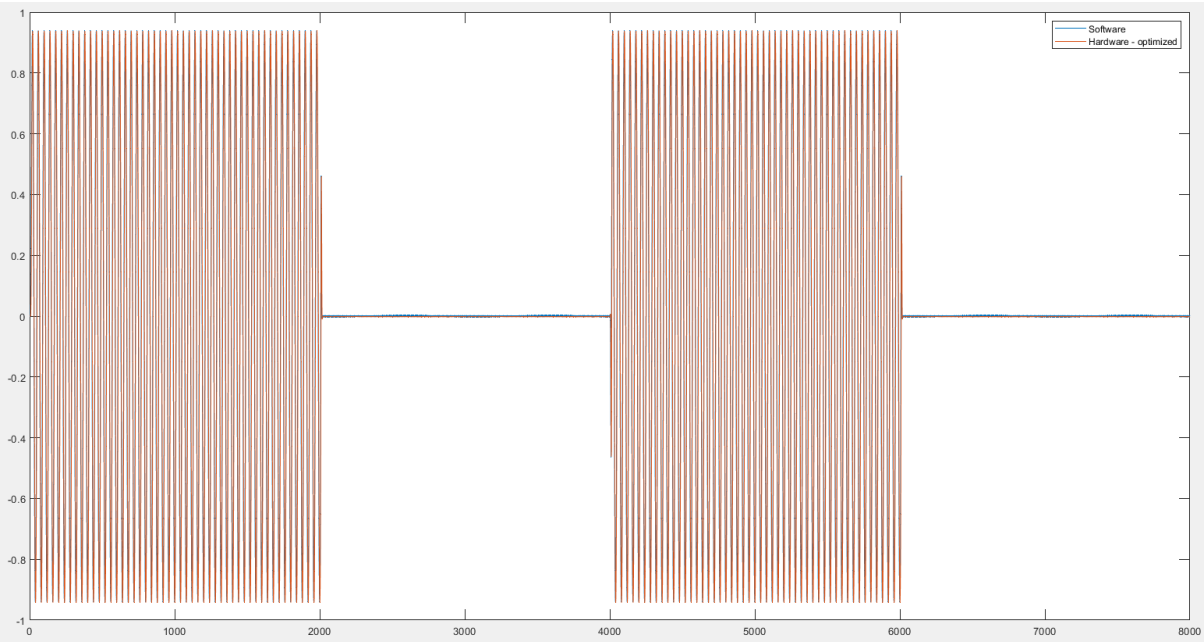

Figure 4 Optimised filter output

For both the **un-optimised** and **optimised** designs, the hardware output closely matched the software output, confirming the functionality of the filter in the hardware domain.

In terms of frequency attenuation, both the un-optimised and optimised hardware designs were able to replicate the software output, with a slight improvement in high-frequency attenuation in the hardware implementations. This indicates that the hardware was able to achieve better performance in terms of filtering.

The similarity in results between the un-optimised and optimised designs further highlights the effectiveness of the pipelining and retiming techniques, which helped improve the performance without introducing significant trade-offs in accuracy.

FPGA resources, timing parameters and Comparison

The **un-optimised** FIR filter design uses the following FPGA resources from figure 5:

- **Multipliers**: 13
- **Adders/Subtractors**: 18
- **Registers**: 432

Generic Resource Report for unoptimised

Summary

Multipliers	13
Adders/Subtractors	18
Registers	18
Total 1-Bit Registers	432
RAMs	0
Multiplexers	0
I/O Bits	52
Static Shift operators	0
Dynamic Shift operators	0

Figure 5 Un-optimised Resource Report

From the **resource summary** report, we see the utilisation of FPGA resources for the un-optimised design:

- **Slice LUTs**: 1195
- **Slice Registers**: 432

Timing Parameters

- **Data Path Delay**: 11.154 ns
- **Slack**: -1.162ns
- **Clock Frequency**: 89.59 MHz

The negative slack confirms the design does not meet the timing constraint (target period tighter than 11.154 ns, e.g., 10 ns for 100 MHz). In other words, at the requested clock the combinational path is too long; the achievable frequency is about 89.6 MHz.

Resource summary

Resource	Usage	Available	Utilization (%)
Slice LUTs	1195	63400	1.88
Slice Registers	432	126800	0.34
DSPs	0	240	0.00
Block RAM Tile	0	135	0.00
URAM	0	0	

Parsed timing report file: timing_post_map.rpt.

Timing summary

	Value
Requirement	10 ns (100 MHz)
Data Path Delay	11.154 ns
Slack	-1.162 ns
Clock Frequency	89.59 MHz

WARNING: Timing constraints not met.

```
Generated Post Map Timing Report hdl_prj\vivado_prj\timing_post_map.rpt.
Task "Run Synthesis" successful.
Generated logfile: hdl_prj\hdlsrc\unoptimised\workflow_task_RunSynthesis.log
```

Figure 6 Un-optimised Resource and Timing Summary

Critical Path Analysis

- **Critical Path Delay**: 42.396 ns

As we can see from figure 8 the critical path delay is $T_m + 16T_a$

Critical Path Report for unoptimised/Subsystem

Summary Section

Critical Path Delay : 42.396 ns
Critical Path Begin : Delay
Critical Path End : Add15
Highlight Critical Path: hdl_prj\hdlsrc\unoptimised\criticalPathEstimated.m

Figure 7 Un-optimised Critical path delay

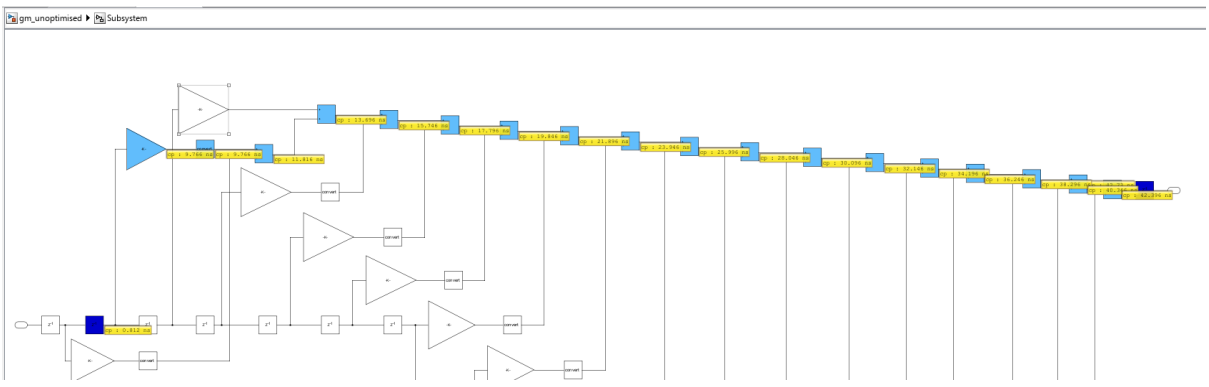

Figure 8 Un-optimised Critical Path delay

The **Optimised** FIR filter design uses the following FPGA resources from figure 9:

- **Multipliers**: 13
- **Adders/Subtractors**: 18
- **Registers**: 792

Generic Resource Report for optimised

Summary

Multipliers	13
Adders/Subtractors	18
Registers	33
Total 1-Bit Registers	792
RAMs	0
Multiplexers	0
I/O Bits	52
Static Shift operators	0
Dynamic Shift operators	0

Figure 9: Optimised Resource Report

From the **resource summary** report, we see the utilisation of FPGA resources for the un-optimised design:

- **Slice LUTs**: 874
- **Slice Registers**: 554

Timing Parameters

- **Data Path Delay**: 2.599 ns
- **Slack**: 7.393 ns
- **Clock Frequency**: 383.58 MHz

The optimised design meets timing with comfortable slack. Although its achieved clock is lower than the earlier (over-optimistic) un-optimised result, the applied pipelining/retiming shortens the longest combinational path and yields a robust implementation at 383.6 MHz.

Parsed resource report file: optim_utilization_synth.rpt.

Resource summary

Resource	Usage	Available	Utilization (%)
Slice LUTs	874	63400	1.38
Slice Registers	554	126800	0.44
DSPs	0	240	0.00
Block RAM Tile	0	135	0.00
URAM	0	0	

Parsed timing report file: timing_post_map.rpt.

Timing summary

	Value
Requirement	10 ns (100 MHz)
Data Path Delay	2.599 ns
Slack	7.393 ns
Clock Frequency	383.58 MHz

```
Generated Post Map Timing Report hdl_prj\vivado_prj\timing_post_map.rpt.
Task "Run Synthesis" successful.
Generated logfile: hdl_prj\hdlsrc\optimised\workflow_task_RunSynthesis.log
```

Figure 10: Optimised Resource and Timing Summary

Critical Path Analysis

- **Critical Path Delay**: 9.288 ns

As we can see from figure 12 the critical path delay is T_m, after pipelining and retiming was applied.

Critical Path Report for optimised/optim

Summary Section

Critical Path Delay : 9.288 ns
Critical Path Begin : *Gain1*
Critical Path End : *Delay3*
Highlight Critical Path: hdl_prj\hdlsrc\optimised\criticalPathEstimated.m

Figure 11: Optimised Critical path delay

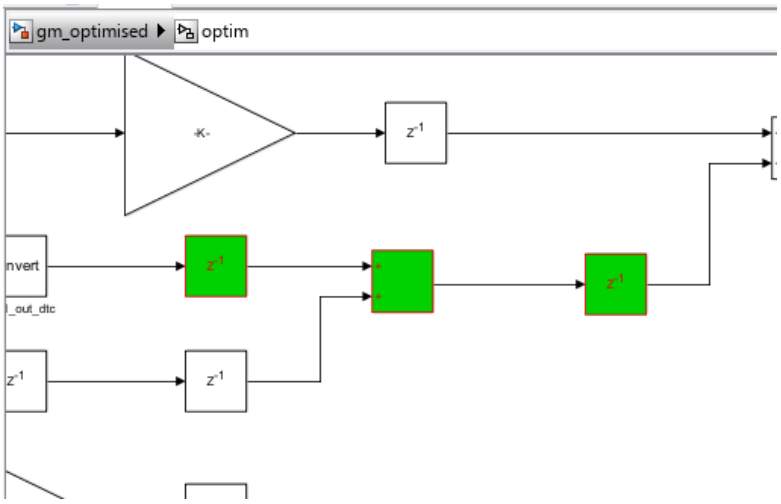

Figure 12: Optimised Critical Path delay

The optimised design uses more registers (792 vs 432) compared to the un-optimised design. This increase is expected, as additional delay blocks were introduced to implement pipelining and retiming, improving the design's overall timing performance. However, the number of multipliers and adders remains the same in both designs, showing that the optimisations focused on enhancing timing and data flow rather than adding extra computational units. Interestingly, the optimised design uses fewer slice LUTs (874 vs 1195), indicating that the applied optimisations improved logic efficiency and reduced redundant combinational paths.

The critical path delay was significantly reduced from 42.396 ns in the un-optimised design to 9.288 ns in the optimised version, reflecting a clear improvement in timing performance. This reduction directly translates to a higher sampling frequency, increasing from 23.587 MHz to 107.67 MHz. Because the sampling frequency is inversely proportional to the critical path delay, this demonstrates that the optimised design processes data much faster and more efficiently.

Since no parallel processing was used in either design, the throughput is equivalent to the sampling frequency. The un-optimised design achieves a higher nominal clock frequency of 23.587 MHz, while the optimised design operates at 107.67 MHz. Despite this lower achievable frequency compared to some earlier results, the optimised design remains far more efficient due to its shorter critical path, enabling faster effective data sampling and better real-time performance.

Regarding data path delay, the optimised design shows a larger value (2.599 ns compared to 11.154 ns for the un-optimised design). This increase is expected, as more pipeline stages and delay elements were introduced to balance and distribute the computation more evenly. Overall, these modifications significantly improved timing and efficiency, resulting in a more robust and high-performing FIR filter implementation on the FPGA.

Hardware Co-simulation Results

The hardware co-simulation of the optimised FIR filter design was performed on the Nexys 4 FPGA board. The Build Summary screenshot (Figure 13) shows the successful compilation of the design, while the Bitstream loaded onto the FPGA board screenshot (Figure 14) confirms that the bitstream was successfully loaded and the design was implemented on the FPGA. Figure 15 shows the output scope of the filter.

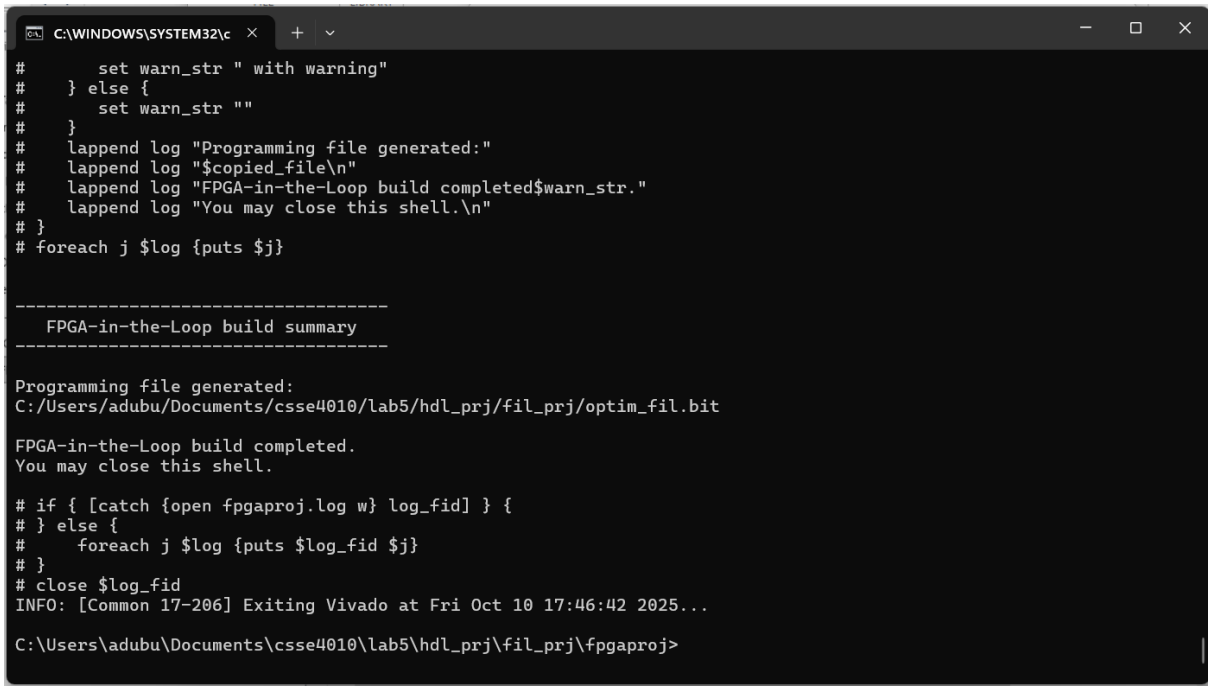

Figure 13: Build summary

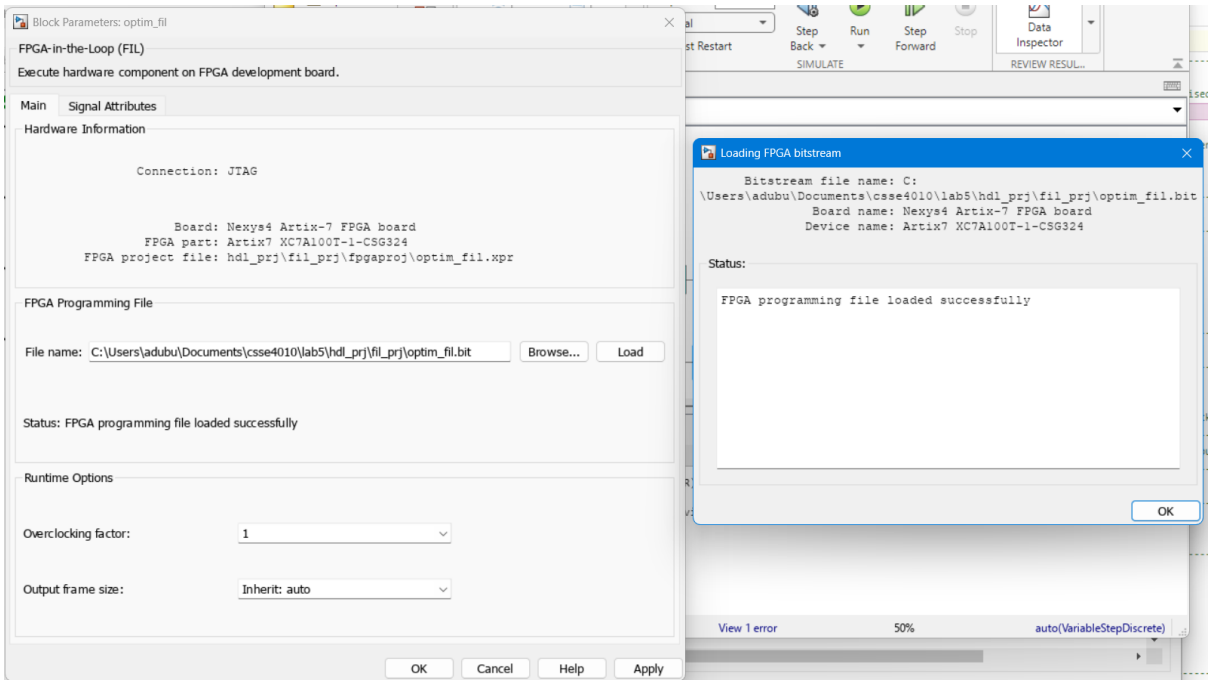

Figure 14: Bit stream loaded into FPG

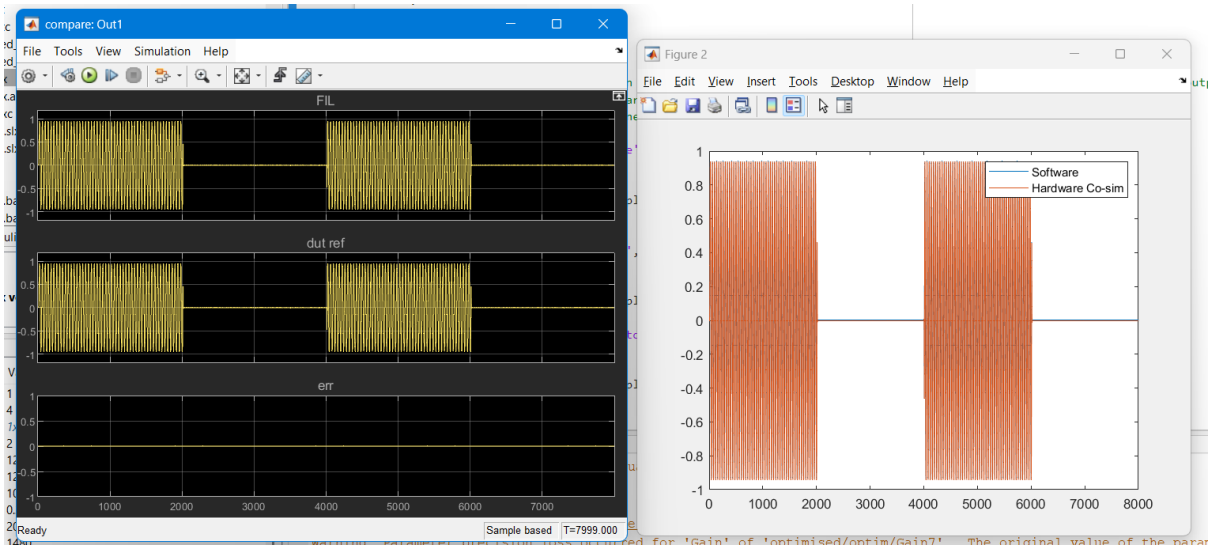

Figure 15: Scope of Co-sim out

Discussion and Conclusion

In this lab, we successfully implemented a Digital FIR filter on an FPGA using MATLAB HDL Coder and Vivado. Through the design process, we compared an un-optimised and optimised version of the FIR filter. The optimised design demonstrated a significant reduction in critical path delay and improved timing performance, although it operated at a lower clock frequency compared to the un-optimised design. Despite this, the optimised design-maintained efficiency and achieved a higher sampling frequency and better attenuation of high frequencies.