NC State University

Department of Electrical and Computer Engineering

ECE 720 (Prof. William Rhett Davis)

Project #2: Finite Impulse Response (FIR) Filter Accelerator

By

Gagana Sindhu Sabbavarapu

200497115

CONTENTS

1. Introduction

Finite Impulse Response (FIR) filters play a crucial role in digital signal processing applications, offering an effective means of filtering and processing signals. As systems-on-chip (SoCs) continue to integrate diverse functionalities, there is an increasing need for optimized hardware accelerators to enhance the performance of FIR filters. This project aims to design a hybrid FIR filter accelerator, leveraging both hardware and software components to achieve a balance between computational efficiency and flexibility.

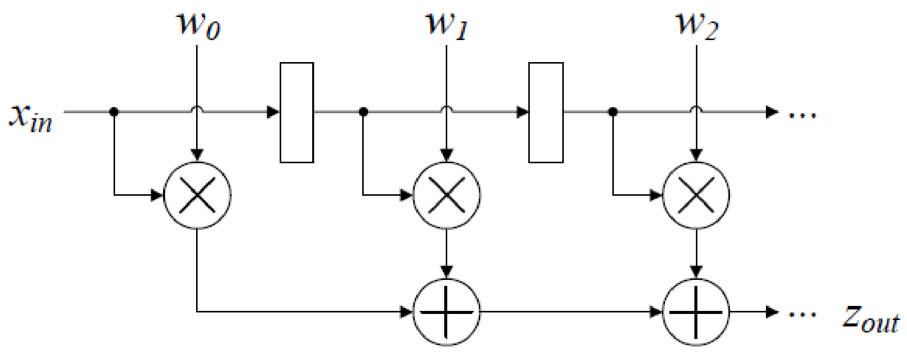


Figure: FIR Filter

The primary objectives of this project include partitioning the FIR filter computation between dedicated hardware and software, evaluating the resulting speedup and area overhead compared to a software-only implementation, and demonstrating the synthesizability of the FIR unit using High-Level Synthesis (HLS) tools.

By distributing the workload between hardware and software, we aim to harness the strengths of each component, optimizing the overall performance of the FIR filter. The hardware accelerator will be responsible for the computationally intensive convolution operation, while the software portion manages control, data transfer, and other tasks. This partitioning strategy is expected to yield improvements in terms of execution speed and resource utilization.

Furthermore, an analysis of the speedup and area overhead relative to a software-only FIR filter implementation will provide insights into the efficiency gains achieved through hardware acceleration. This assessment is critical for understanding the trade-offs between computational performance and the additional resources required for hardware implementation.

To demonstrate the practicality and ease of integration, we will employ HLS tools to synthesize the FIR unit. HLS allows for a high-level description of the FIR filter algorithm, automatically generating hardware descriptions from the software-like input. The ability to synthesize the accelerator using HLS will showcase its viability for integration into SoCs and its potential for rapid development.

2. Technical Overview

The simulation of the System-on-Chip (SoC) with an FIR Unit revolves around processing 16 time-steps of an FIR filter concurrently, optimizing computational efficiency. The key components of the simulation involve a Central Processing Unit (CPU), a Direct Memory Access (DMA) unit, and the FIR Unit.

1. Input and Weight Initialization:

- Prior to the simulation, the inputs (X) and weights (W) are assumed to be stored in Dynamic Random-Access Memory (DRAM).
- The C-program executed on the CPU is responsible for initializing the FIR filter by specifying the locations of the input and weight arrays in DRAM.

2. DMA Unit Configuration:

- The C-program communicates with the DMA unit, indicating the memory locations of the input and output buffers.
- Burst transactions are utilized to efficiently transfer data between DRAM and the FIR Unit, streamlining the movement of input values from memory to the processing unit and storing the results back in memory.

3. FIR Filter Processing:

- The FIR Unit is designed to process 16 time-steps concurrently, optimizing the throughput of the filtering operation.
- The DMA unit orchestrates the movement of input values in bursts, facilitating the efficient execution of the FIR filter computation.

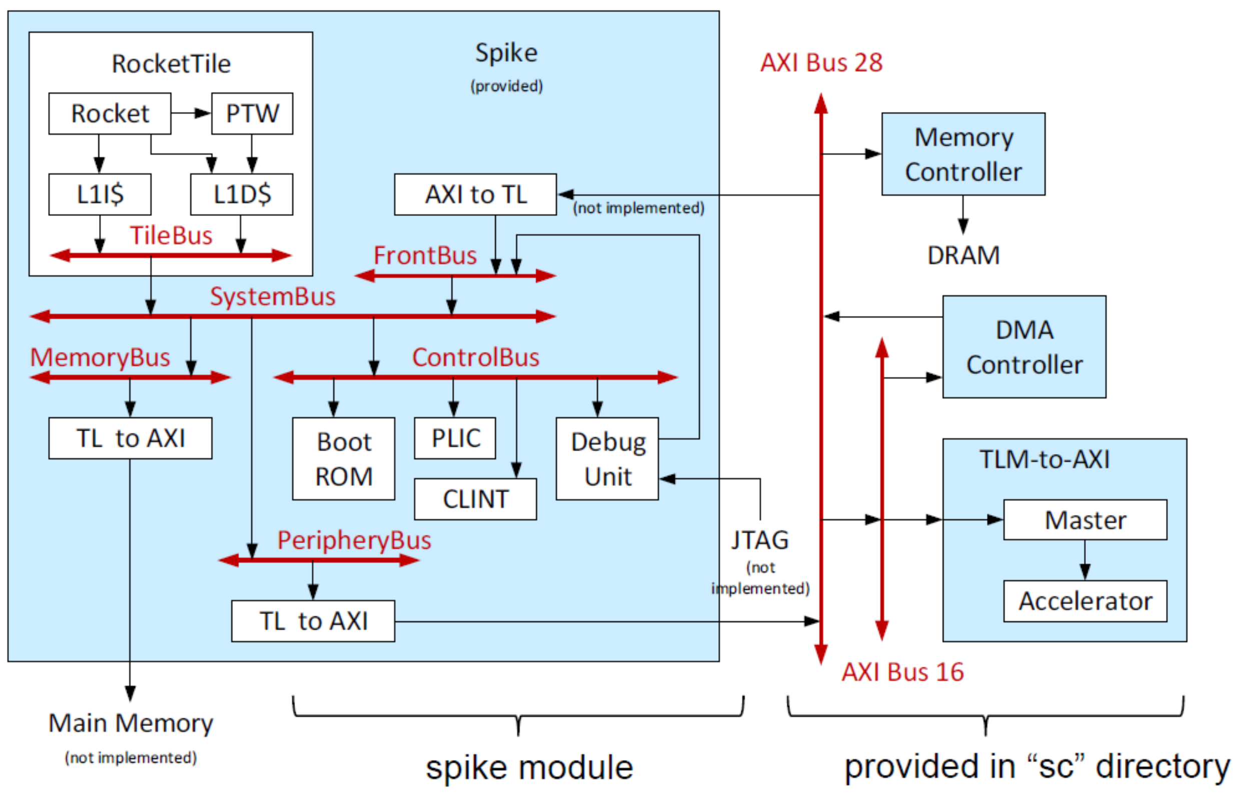
4. Error Checking:

- Once the FIR filter computation is complete, the C-program on the CPU is programmed to check the error of the final values.
- This step ensures the accuracy of the FIR filter output, comparing the simulated results against the expected values.

5. Coordinated Hardware-Software Interaction:

- The simulation emphasizes the collaboration between hardware (FIR Unit) and software (C-program on CPU) components.
- The CPU controls the DMA unit to initiate data transfers, while the FIR Unit focuses on the parallel processing of 16 time-steps.

Project System:



3. Implementation

The technical implementation of the SoC simulation with an FIR Unit involves a step-by-step process to efficiently process 16 time-steps of an FIR filter. The detailed steps are as follows:

1. Initialization of Coefficients:

- Coefficient values are fetched from the initialized location in DRAM using DMA.
- These coefficients are then written into the Accelerator Coefficient register.
- The Accelerator Control register is triggered with a specific value to instruct the accelerator to store the coefficients in an array for computational ease.

2. Register Configuration for Input Processing:

- DMA is employed to fetch the first 16 inputs from the DRAM, writing them into the Accelerator Input registers.
- The Accelerator Control register is utilized to initiate the processing of the computation.

3. Computation Execution:

- The accelerator processes the inputs, generating outputs in groups of four consecutive inputs.
- An empty input array is used to read individual input elements from the accelerator registers, and the corresponding outputs are computed.

4. Output Storage and Memory Transfer:

- After processing four consecutive inputs, the combined 32 bytes of four outputs are written to the Accelerator Output registers.
- The resulting outputs are transferred to the memory using DMA from the C-program.
- This process continues iteratively for subsequent sets of 16 inputs until all outputs are generated.

5. Error Checking:

- Once all outputs are written to memory, error checking is performed.
- The obtained outputs from the accelerator are compared against the expected outputs, ensuring the accuracy of the computation.

6. Resource Optimization:

- Different hardware implementations are explored to assess resource utilization.
- One implementation involves a loop for the convolution, while the other uses unrolled loop.
- Resource consumption is compared against a software-only implementation to evaluate the efficiency gains.

The iterative processing of inputs, efficient use of DMA for data transfer between DRAM and the accelerator, and the orchestrated control through accelerator registers demonstrate the effective integration of hardware acceleration and software control. The resource-conscious approach considers varying implementations to find the optimal balance between hardware utilization and computational efficiency.

4. Resource Comparison

The below table summarizes and compares the resources used in FIR Unit to the Software only implementation.

Metric	Software only design	Hardware Design without Unroll	Hardware Design with Unroll
Simulation time	62564 ns	19447 ns	19447 ns
Latency	3	126	63
Throughput	1	124	60
Critical Path	4.917849 ns	3.742561 ns	4.852933 ns
Area_mux	5930.9 μm^2	6873.7 μm^2	6535.2 μm^2
Area_func	996.8 μm^2	2177 μm^2	12431.5 μm^2
Area_logic	850.4 μm^2	1402.6 μm^2	1557.3 μm^2
Area_buffer	0 μm^2	0 μm^2	0 μm^2
Area_mem	0 μm^2	0 μm^2	0 μm^2
Area_rom	0 μm^2	0 μm^2	0 μm^2
Area_reg	13071.2 μm^2	17210.2 μm^2	17832.6 μm^2
Area_fsm_reg	10 μm^2	10 μm^2	10 μm^2
Area_fsm_comb	10 μm^2	10 μm^2	10 μm^2
Total execution time	**437948 ns**	**136129 ns**	**136129 ns**

From the provided data, it is evident that the simulation time has been significantly reduced, demonstrating a noteworthy **3x** speedup. The simulation time has decreased from **62,564 ns** to **19,447 ns**, showcasing the enhanced efficiency achieved through the hardware acceleration of the FIR filter.

Additionally, the post-assignment area for the hardware implementation is measured at **35,515.7 μm^2**. Notably, this is considerably smaller when compared to the Rocket tile area, which is recorded at **1,443,717 μm^2.**

Upon comparing various hardware implementations, it is evident that unrolling the loop leads to improvements in both latency and throughput. However, this enhancement comes at the cost of increased area utilization and critical path delay, primarily due to the utilization of a greater number of multipliers in the unrolled configuration. Consequently, the design employing the loop is inferred to be a superior implementation, striking a balance between computational efficiency and resource utilization.

(Note: While calculating the simulation time the error calculation and print was removed from the fir.c and readded for submission. So, the simulation time might vary from results above if run with those included)

5. Conclusion

In conclusion, the exploration of hardware acceleration for Finite Impulse Response (FIR) filter processing within a System-on-Chip (SoC) environment has yielded valuable insights. Through meticulous simulation and comparative analysis of different hardware implementations, a substantial 3x speedup in simulation time, reducing it from 62,564 ns to 19,447 ns was observed. This improvement underscores the efficiency gains achievable through the integration of specialized accelerators.

Furthermore, the assessment of alternative hardware configurations revealed a trade-off scenario. While unrolling the loop demonstrated enhancements in latency and throughput, it concurrently led to an increase in both area utilization and critical path delay, primarily due to the expanded use of multipliers. The comparison highlights the significance of striking a balance between computational efficiency and resource utilization.

The utilization of Direct Memory Access (DMA) for efficient data transfer, orchestration through accelerator registers, and the judicious control of hardware-software interactions exemplify the successful integration of hardware acceleration and software coordination in FIR filter processing. The comprehensive exploration also considered the impact on area utilization, revealing that the post-assignment area for the hardware implementation was substantially smaller than the Rocket tile area, reinforcing the space efficiency gained through hardware acceleration.

In making design decisions, the preference for a loop-based implementation over an unrolled loop configuration was justified by the observed trade-offs. The loop-based design showcased a more favorable balance, aligning with the project's goals of optimizing computational efficiency while managing resource utilization effectively.