CSE 237C Assignment 1
Shrivatsan Rajagopalan

Introduction

This report describes the design and optimization of an FIR filter using a HLS tool. As a part of the first task, an 8 tap FIR filter was designed without any particular consideration for optimizing the code. The second task involved designing a 128 tap FIR filter and experimenting with multiple design choices to analyze the usage of resources, computation time and consider the tradeoffs.

Experimentation and Results

This section details the experiments carried out, the results produced and a discussion on each result.

Baseline Implementation

The baseline implementation only pays attention to functionality. The implementation has one for loop with 2 separate segments, a MAC segment and a TDL segment. There is also an if..else.. clause present in inside the for loop which hinders performance improvements. The measurements taken from this setup will be used as the main comparison metric to evaluate future designs

Optimization	Throughput (MHz)	#FFs	#LUTs	#DSP48Es
None	0.1833	161	211	3

Optimization 1 - Bit Width optimization

This step involves using an optimized number of memory depending on the type of data. This was achieved in C by replacing integer variables which occupy 32 bits with data types which reserve smaller memory. The counter and coefficients variables are examples where bit width optimization was carried out. For this particular dataset, the input samples could be occupied using a signed 8 bit variable.

Optimization	Throughput (MHz)	#FFs	#LUTs	#DSP48Es
Bit Width	0.2994	59	147	1

Since both the coefficient variable and input were reduced to 8 bit variables, the number of DSP48Es which does the multiply operation was also reduced. The number of flip flops reduced because multiple variables with data occupying less than 8 bits were stored in 8 bit registers instead of 32 bit registers.

Optimization 2 - Loop Pipelining

The next design choice is to employ pipelining. This means that throughput can be increased at the cost of an increase in hardware resources.The number of LUTs (to carry out out more simultaneous operations) should see an increase. This was confirmed from the results obtained.

Optimization	Throughput (MHz)	#FFs	#LUTs	#DSP48Es
Loop Pipelining	0.4537	131	270	3

Bit Width optimization was not carried out here. A further decrease in number of LUTs could have been obtained if bit width optimization had been carried out in conjunction with loop pipelining.

Optimization 3 and 4 - Removing Conditional Statements (and pipelining)

The conditional statement inside the for loop causes additional latency while also preventing further exploitation of pipelining. The conditional statement was removed and the results were observed with and without loop pipelining.

Optimization	Throughput (MHz)	#FFs	#LUTs	#DSP48Es
No Conditional no Pipelining	0.2308	132	246	3
No Conditional with Pipelining	0.89701	199	288	3

The removal of conditional statements with the addition of pipelining has drastically increased the throughput. This is because the presence of an if else statement hampered the efficacy of pipelining. Without the conditional statement the code became fully sequential and streamlined which was exploited by loop pipelining.

Optimization 5 and 6 - Loop Partitioning with and without pipelining

This step involves partitioning the 2 segments inside the for loop (the TDL and MAC sections). This was done by setting the factor value in the pragma to 2. This allows for parallel execution of for loops. This was done using the unroll pragma.

Optimization	Throughput (MHz)	#FFs	#LUTs	#DSP48Es
Loop Unravelling	0.2617	266	289	6
With pipelining	0.8902	275	363	6

Even though loop unravelling on its own did not produce huge changes in throughput, it being used along with pipelining increased throughput by a very good amount. This was because pipelining was done on both for loops simultaneously.

Optimization 7 and 8 - Memory Partitioning
The register used to store past values of x needs to be traversed each time to access an element which is not in the zeroth index. This memory can be partitioned to allow for faster access and hence higher throughput at the cost of more memory resources.

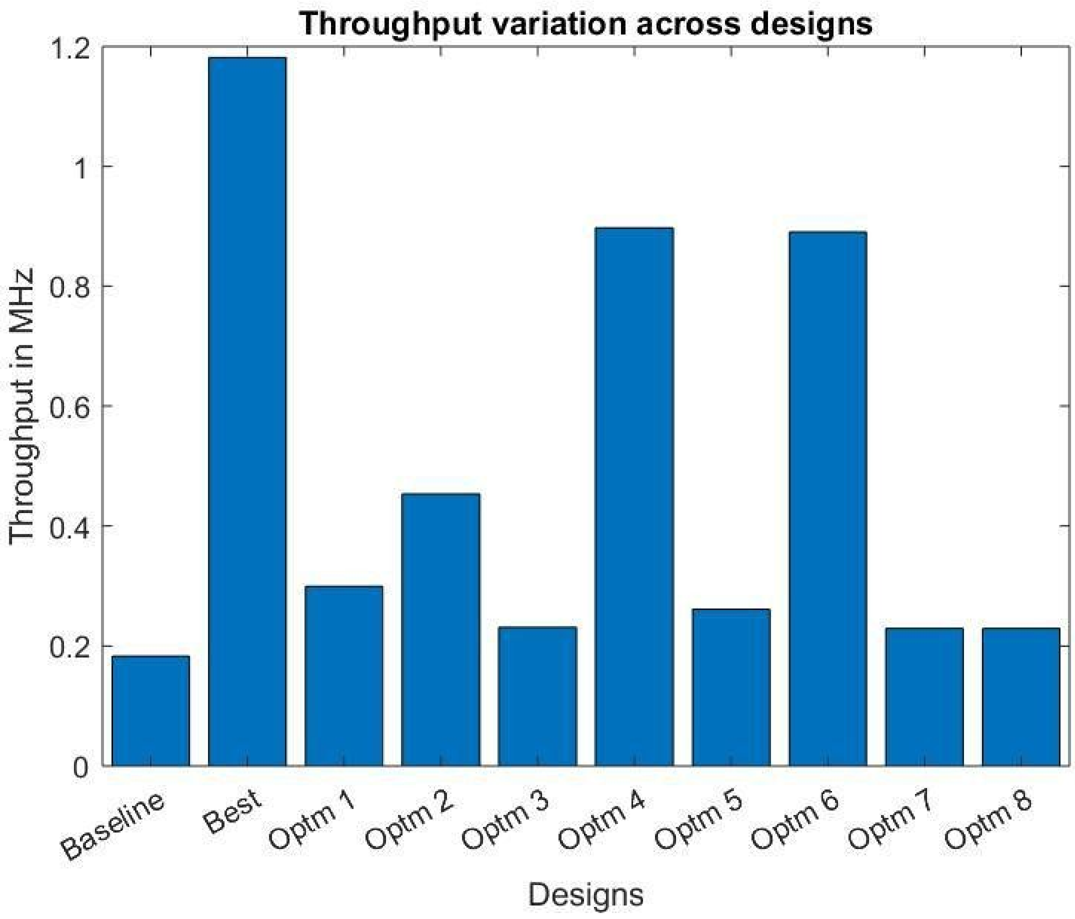
Optimization	Throughput (MHz)	#FFs	#LUTs	#DSP48Es
Complete partition	0.229	4191	1716	3
Block 64	0.229	4223	784	3

From the above results it can be seen that complete partition is overkill for this scenario and 64 blocks provided similar performance to 128 partitions. Experimentation can lead to saturation value for memory partitioning.

Optimization 9 - Best
By looking at the individual cases bit width optimization it can be seen that provides almost no ill effects and only gains which makes it a sensible choice for a final design. Pipelining provided massive gains to throughput with minimal increase in resources when used properly. Removal of the conditional statement further increased the efficiency of pipelining. These three techniques were used to design my best method. This provides a reasonable good throughput (compared to baseline) with almost no increases in resources.

Optimization	Throughput (MHz)	#FFs	#LUTs	#DSP48Es
Loop Pipelining	1.1816	66	171	1

Conclusion



The optimizations with highest throughouts involved pipelining in conjunction with another design technique.