CSE237C Lab4 Report

Hanyang Xu A92068025 hax032

In this report, I introduced one architecture that implement FFT using HLS. The implementation optimized for throughput and fit on the limited resources on FPGA. I achieved latency per 1024-point FFT of 938 + some pipeline overhead and frequency of 9.489ns which is around 110kHz while meeting the resources usage limite.

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.489	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
7770	7770	938	938	dataflow

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	362	-
FIFO	-	-	-	-	-
Instance	14	200	36126	48342	-
Memory	144	-	0	0	0
Multiplexer	-	-	-	720	-
Register	-	-	80	-	-
Total	158	200	36206	49424	0
Available	280	220	106400	53200	0
Utilization (%)	56	90	34	92	0

At top level, the design is partitioned into 11 stages with the first stage being Bit Reverse and the rest of the stage corresponding to each of the FFT stages with stage 1 being 2-point FFT and stage 10 being 1024-point FFT. This partition allows me to achieve task level pipelining using #pragma HLS dataflow. This architecture allows each stage to overlap with each other and could greatly increase throughput when performance multiple different 1024-point FFT consecutively.

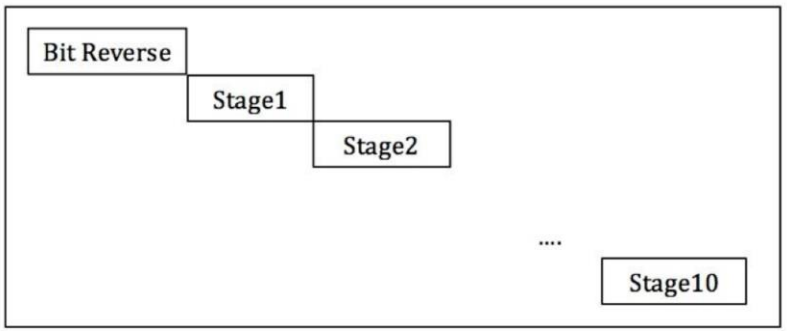

Figure 1 Task Pipeline Stages

At stage level, the architecture reduces the latency as much as possible while still fit into the resource's requirement of the FPGA hardware. Since the task pipelining interval depends on the longest latency of the stages, reducing the latency of each stage is important. The optimization that gives the best performance would be using #pragma HLS pipeline and #pragma HLS unroll together which essentially give you multiple pipeline with 1/multiple stage in each pipeline. However, this costs large resources usage, so I only use this on bit reverse.

For stage1 and stage 6 to 10, I pipelined the entire outer loop since the inner loop is small and easy to flatten. For stage 2 to 5, since it is too computational heavy to flatten the large inner loop completely. I unroll the small outer loop completely and pipeline the large inner loop. This gives worse performance than only pipelining, but it saves much more resources as flattening the inner loop cost much more than pipelining it and is synthesized fast.

The architecture has 10 duplicated memories to store intermedium results for task pipelining. Memory is partitioned as cyclic since loop pipelining and loop unrolling requires access to the same array at the same time with a cyclic pattern. The partition factor is determined experimentally, which is the minimal factor that synthesis will not complain about delayed memory access due to port limit.