HLS Self-paced Learning Project
[pp4fpga] Discrete Fourier Transform

r08943154 謝承翰

1. Discrete Fourier Transform

   Fourier transform maps a signal in the time domain to the frequency domain. This is widely used in digital signal processing. We can analyze the frequency information of the signal and impose some processing algorithm on it to generate cool effects. Discrete Fourier transform (DFT) converts a finite number of samples to a finite number of complex sinusoids. The DFT can be implemented by multiplying signals to the cos and sin phase and summation.

```
// Calculate each frequency domain sample iteratively
for (i = 0; i < N; i += 1) {
    temp_real[i] = 0;
    temp_imag[i] = 0;

    // (2 * pi * i)/N
    w = (2.0 * 3.141592653589  / N) * (TEMP_TYPE)i;

    // Calculate the jth frequency sample sequentially
    for (j = 0; j < N; j += 1) {
        // Utilize HLS tool to calculate sine and cosine values
        c = cos(j * w);
        s = -sin(j * w);

        // Multiply the current phasor with the appropriate input sample and keep
        // running sum
        temp_real[i] += (sample_real[j] * c - sample_imag[j] * s);
        temp_imag[i] += (sample_real[j] * s + sample_imag[j] * c);
    }
}
```

2. HLS

   (1) Data type

   The given code use double as the data type. The synthesis result shows that it is out of DSP resources when we use double.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	151	-
FIFO	-	-	-	-	-
Instance	16	226	12340	18705	-
Memory	4	-	0	0	0
Multiplexer	-	-	-	478	-
Register	-	-	1023	-	-
Total	20	226	13363	19334	0
Available	280	220	106400	53200	0
Utilization (%)	7	102	12	36	0

So I change the datatype to float and run the synthesis again. The resource limit meets but the latency looks large. Then, I'll focus on the latency optimization.

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
4788994	5313282	47.890 ms	53.133 ms	4788994	5313282	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	151	-
FIFO	-	-	-	-	-
Instance	16	203	11765	17338	-
Memory	2	-	0	0	0
Multiplexer	-	-	-	514	-
Register	-	-	706	-	-
Total	18	203	12471	18003	0
Available	280	220	106400	53200	0
Utilization (%)	6	92	11	33	0

(2) Latency Optimization

First, I add the pipeline pragma into second loop. The result shows that the latency is improved about 15x but the II is still 5.

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
346882	346882	3.469 ms	3.469 ms	346882	346882	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1	346368	346368	1353	-	-	256	no
+ L2	1335	1335	61	5	1	256	yes
- Loop 2	512	512	2	-	-	256	no

To my surprise, the resource usage decrease with the pipeline pragma. It only uses half of the DSP, less LUT and less FF.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	151	-
FIFO	-	-	-	-	-
Instance	16	104	7072	9025	-
Memory	2	-	0	0	0
Multiplexer	-	-	-	481	-
Register	0	-	1047	64	-
Total	18	104	8119	9721	0
Available	280	220	106400	53200	0
Utilization (%)	6	47	7	18	0

To see the difference, we can dig into the detail report of the resource usage.

Instance

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
dft_dmul_64ns_64ntde_U44	dft_dmul_64ns_64ntde	0	11	317	578	0
dft_fadd_32ns_32nocq_U35	dft_fadd_32ns_32nocq	0	2	205	390	0
dft_faddfsub_32nsncg_U34	dft_faddfsub_32nsncg	0	2	205	390	0
dft_fmul_32ns_32npcA_U36	dft_fmul_32ns_32npcA	0	3	143	321	0
dft_fmul_32ns_32npcA_U37	dft_fmul_32ns_32npcA	0	3	143	321	0
dft_fmul_32ns_32npcA_U38	dft_fmul_32ns_32npcA	0	3	143	321	0
dft_fmul_32ns_32npcA_U39	dft_fmul_32ns_32npcA	0	3	143	321	0
dft_fpext_32ns_64sc4_U43	dft_fpext_32ns_64sc4	0	0	100	138	0
dft_fptrunc_64ns_rcU_U41	dft_fptrunc_64ns_rcU	0	0	128	277	0
dft_fptrunc_64ns_rcU_U42	dft_fptrunc_64ns_rcU	0	0	128	277	0
dft_sitofp_32ns_3qcK_U40	dft_sitofp_32ns_3qcK	0	0	340	554	0
grp_sin_or_cos_double_s_fu_199	sin_or_cos_double_s	8	88	4885	6725	0
grp_sin_or_cos_double_s_fu_218	sin_or_cos_double_s	8	88	4885	6725	0
Total		13	16	203	11765	17338	0

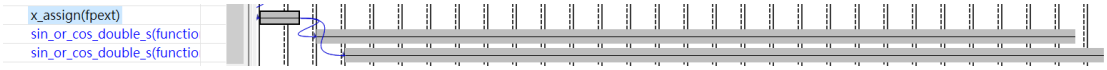
Instance

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
dft_dmul_64ns_64nsc4_U34	dft_dmul_64ns_64nsc4	0	11	317	578	0
dft_faddfsub_32nsncg_U29	dft_faddfsub_32nsncg	0	2	205	390	0
dft_fmul_32ns_32nocq_U30	dft_fmul_32ns_32nocq	0	3	143	321	0
dft_fpext_32ns_64rcU_U33	dft_fpext_32ns_64rcU	0	0	100	138	0
dft_fptrunc_64ns_qcK_U32	dft_fptrunc_64ns_qcK	0	0	128	277	0
dft_sitofp_32ns_3pcA_U31	dft_sitofp_32ns_3pcA	0	0	340	554	0
grp_sin_or_cos_double_s_fu_219	sin_or_cos_double_s	16	88	5839	6767	0
Total		7	16	104	7072	9025	0

From the report, we can see that the HLS tool choose different implementations for pipeline and no pipeline. With the pipeline pragma, the HLS tool makes sin_cos function be able to pipeline and only use one module. Not only the synthesis report shows this, the schedule viewer can also show the timeline. The following figure shows that data is assigned to

the module simultaneously.

The following figure is the pipelined version and shows that the data is assigned to the module in a pipeline manner. However, the pipelined version needs a longer latency. This is the reason why we can only achieve II=5.



(3) Precomputed Table

We can further optimize the design by a precomputed sin and cos table. In this case, the design does not need to calculate sin and cos every time.

```
c = cos_table[i * j % N];
s = sin_table[i * j % N];
```

The synthesis result shows that this can save a little bit latency while the II is still 5.

⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
332547	332547	3.325 ms	3.325 ms	332547	332547	none

⊟ Detail

⊟ Instance

N/A

⊟ Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1	332288	332288	1298	-	-	256	no
+ L2	1295	1295	21	5	1	256	yes
- Loop 2	256	256	2	1	1	256	yes

I cannot achieve II=1 with N=256. But if we change N=32, the pipeline can achieves II=1 by separating accumulation and multiplication.

```
            TEMP_TYPE rj = sample_real[j];
            TEMP_TYPE imj = sample_imag[j];
            tmp_real[j] = (rj * c - imj * s);
            tmp_img[j] = (rj * s + imj * c);
        }
        TEMP_TYPE t1=0, t2=0;
        L3:for (j = 0; j < N; j += 1) {
//#pragma HLS pipeline
#pragma HLS unroll
            t1 += tmp_real[j];
            t2 += tmp_img[j];
        }
        temp_real[i] = t1;
        temp_imag[i] = t2;
    }
```

⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
1230	1230	12.300 us	12.300 us	1230	1230	none

⊟ Detail

⊞ Instance

⊟ Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- L1_L2	1194	1194	172	1	1	1024	yes
- Loop 2	32	32	2	1	1	32	yes

3. Conclusion
   (1) If resource is not enough, changing a lighter datatype is necessary.
   (2) Pipeline structure might save some hardware resources by pipelining the operation in the HLS tool.
   (3) Schedule viewer is a good tool for us to analyze the reason of stall and the dataflow.
   (4) II=1 is hard to achieve when the design is too large.
   (5) Codes and report are available on my github https://github.com/qmo1222/MSOC_HLS

4. Reference
   (1) Textbook and codes in https://github.com/KastnerRG/pp4fpgas