HLS LabB Report

D10943018 戴妤珊

Baseline

```
// Write your code here
static
 data_t_8 shift_reg[N];
acc_t16 acc;
int i;

acc = 0;
Shift_Accum_Loop:
for (i=N-1;i>=0;i--){
    if(i==0){
        acc += x*c[0];
        shift_reg[0] = x;
    }
    else{
        shift_reg[i]=shift_reg[i-1];
        acc += shift_reg[i]*c[i];
    }
}
*y = acc;
```

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.510 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
257	513	2.570 us	5.130 us	257	513	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	2	0	124	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	120	-
Register	-	-	210	-	-
Total	1	2	215	254	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

Throughput (Mhz) = 1000/(Clock Period(ns) * #Clock Cycles)
= 1000/(10*513) = 0.195

Question 1 - Variable Bitwidths

It is possible to specify a very precise data type for each variable in your design. The number of different data types is extensive: floating point, integer, fixed point, all with varying bitwidths and options. The data type provides a tradeoff between accuracy, resource usage, and performance.

Change the bitwidth of the variables inside the function body (do not change the bitwidth of the parameters). How does the bitwidth affect the performance? How does it affect the resource usage? What is the minimum data size that you can use without losing accuracy (i.e., your results still match the golden output)?

```
15 typedef ap_int<5> coef_t_5;
16 typedef ap_int<8> data_t_8;
17 typedef ap_int<16>  acc_t16;
```

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.016 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
257	385	2.570 us	3.850 us	257	385	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	113	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	114	-
Register	-	-	113	-	-
Total	1	0	118	237	0
Available	280	220	106400	53200	0
Utilization (%)	~0	0	~0	~0	0

Reducing the bitwidth can accelerate the computation, therefore, the maximum interval is reduced from 513 to 385. Also, the numbers of FF and LUT decrease from 215 and 254 to 118 and 237, which significantly saves the resource requirements. The minimum size of coef_t, data_t, acc_t are 5, 8, 16, respectively.

Throughput (Mhz) = 1000/(Clock Period(ns) * #Clock Cycles)
= 1000/(10*385) = 0.260

Question 2 - Pipelining

Pipelining increases the throughput typically at the cost of additional resources. The initiation interval (II) is an important design parameter that changes the performance and resource usage.

Explicitly set the loop initiation interval (II) starting at 1 and increasing in increments of 1 cycle. How does increasing the II affect the loop latency? What are the trends? At some point setting the II to a larger value does not make sense. What is that value in this example? How would you calculate that value for a general for loop?

```
Shift_Accum_Loop:
for (i=N-1;i>=0;i--){
#pragma HLS PIPELINE II=2
    if(i==0){
        acc += x*c[0];
        shift_reg[0] = x;
    }
    else{
        shift_reg[i]=shift_reg[i-1];
        acc += shift_reg[i]*c[i];
    }
}
*y = acc;
```

II=1 results in II violation

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Shift_Accum_Loop	256	256	3	2	1	128	yes

II=2

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.016 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
258	258	2.580 us	2.580 us	258	258	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Shift_Accum_Loop	256	256	3	2	2	128	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	0	0	121	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	132	-
Register	-	-	100	-	-
Total	1	0	105	263	0
Available	280	220	106400	53200	0
Utilization (%)	~0	0	~0	~0	0

II=3

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.016 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
386	386	3.860 us	3.860 us	386	386	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Shift_Accum_Loop	384	384	3	3	3	128	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	0	0	113	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	120	-
Register	-	-	91	-	-
Total	1	0	96	243	0
Available	280	220	106400	53200	0
Utilization (%)	~0	0	~0	~0	0

II=4

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.016 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
386	386	3.860 us	3.860 us	386	386	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Shift_Accum_Loop	384	384	3	3	4	128	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	0	0	113	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	120	-
Register	-	-	91	-	-
Total	1	0	96	243	0
Available	280	220	106400	53200	0
Utilization (%)	~0	0	~0	~0	0

Larger values of II imply longer latency and less area, since they lead to less levels of pipeline. In my case, II=2 results in latency of 256 (128*2) cycles, and II=3 leads to latency of 384 (128*3) cycles. However, when I set II larger then 3, the achieved II no longer increases, which keeps in 3 instead. I think the reason is the latency and area trade-off of II larger than 3 doesn't improve compared to II=3. Since the optimized design goal is to achieve II=1, II=2 is the closest setting and thus the most suitable ones in the design. However, II is still larger than 1, there is some room for improvement.

Throughput (Mhz) (II=2) = 1000/(Clock Period(ns) * #Clock Cycles)
= 1000/(10*258) = 0.388

Question 3 - Removing Conditional Statements

If/else statements and other conditionals can limit the possible parallelism and often

require additional resources. If the code can be rewritten to remove them, it can make the resulting design more efficient. This is known as code hoisting.

Rewrite the code to remove any conditional statements. Compare the designs with and without if/else condition. Is there a difference in performance and/or resource utilization? Does the presence of the conditional branch have any effect when the design is pipelined? If so, how and why?

```
Shift_Accum_Loop:
for (i=N-1;i>0;i--){
#pragma HLS PIPELINE II=2
    shift_reg[i]=shift_reg[i-1];
    acc += shift_reg[i]*c[i];
}
acc += x*c[0];
shift_reg[0] = x;
*y = acc;
```

Since the if/else condition for i==0 only occurs on the last iteration, we can execute the statements after the loop ends. Therefore, the if/else control flow is removed out of the loop, and the loop bound also changes from i >= 0 to i > 0.

With if/else condition

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.016 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
258	258	2.580 us	2.580 us	258	258	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Shift_Accum_Loop	256	256	3	2	2	128	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	0	0	121	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	132	-
Register	-	-	100	-	-
Total	1	0	105	263	0
Available	280	220	106400	53200	0
Utilization (%)	~0	0	~0	~0	0

Without if/else condition

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	6.508 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
256	256	2.560 us	2.560 us	256	256	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Shift_Accum_Loop	254	254	3	2	2	127	yes

Utilization Estimates

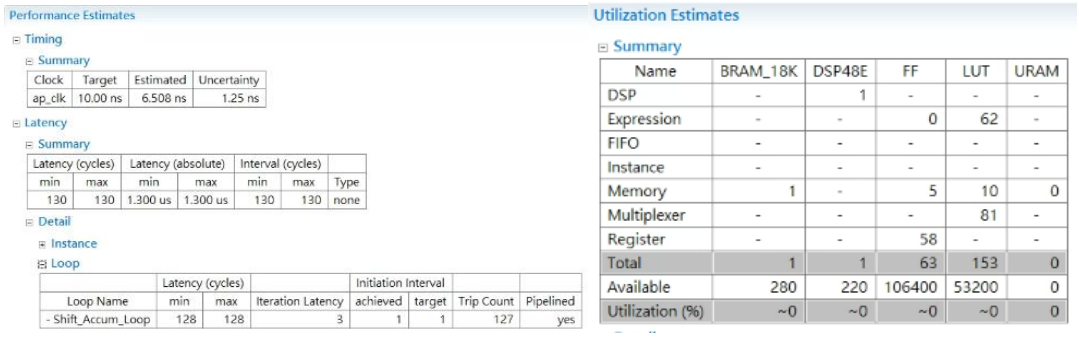
Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	60	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	99	-
Register	-	-	57	-	-
Total	1	1	62	169	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

Comparing the designs with and without if/else condition, we can observe the latency after removing the if/else condition reduces by 2 cycles than the counterpart. Moreover, the required FF and LUT reduce from 105 and 263 to 62 and 169. Since the

if/else condition needs additional logic gates, removing the condition from the loop enhances the efficiency of my design.

II=1



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	6.508 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
130	130	1.300 us	1.300 us	130	130	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Shift_Accum_Loop	128	128	3	1	1	127	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	62	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	81	-
Register	-	-	58	-	-
Total	1	1	63	153	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

After removing the if/else condition, II can be reduced to 1 due to the removal of checking branch process.

The schedule of original design with if/else condition (II=2):

cycle	1	2	3	4	5	6
Iter1	If/else (i==0)	Read c[i]/shift_reg[i]	MAC	Store acc		
Iter2			If/else (i==0)	Read c[i]/shift_reg[i]	MAC	Store acc

The schedule of optimized design without if/else condition (II=1):

cycle	1	2	3	4
Iter1	Read c[i]/shift_reg[i]	MAC	Store acc	
Iter2		Read c[i]/shift_reg[i]	MAC	Store acc

Throughput (Mhz) (II=1) = 1000/(Clock Period(ns) * #Clock Cycles)
= 1000/(10*130) = 0.769

Question 4 - Loop Partitioning

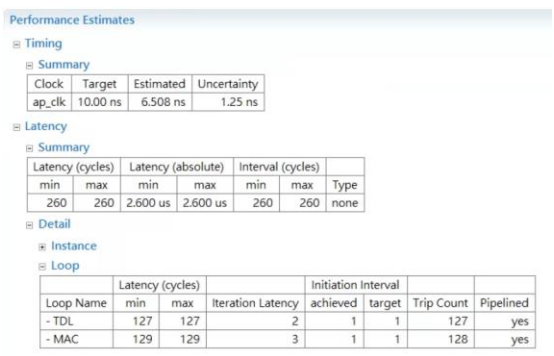
Dividing the loop into two or more separate loops may allow for each of those loops to be executed in parallel (via unrolling), enable loop level pipelining, or provide other benefits. This may increase the performance and the resource usage.

Is there an opportunity for loop partitioning in FIR filters? Compare your hardware designs before and after loop partitioning. What is the difference in performance? How do the number of resources change? Why?

```
TDL:
for (i=N-1;i>0;i--){
#pragma HLS PIPELINE II=1
    shift_reg[i]=shift_reg[i-1];
}
shift_reg[0]=x;

acc = 0;
MAC:
for (i=N-1;i>=0;i--){
#pragma HLS PIPELINE II=1
    acc +=  shift_reg[i]*c[i];
}
*y = acc;
```

Since there are two fundamental operations within the for loop, shifting data in shift_reg array and multiplication and accumulation for output. Therefore, I separate original Shift_Accum_Loop into TDL (Tapped delay line) and MAC loops.



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	6.508 ns	1.25 ns

Latency

Summary

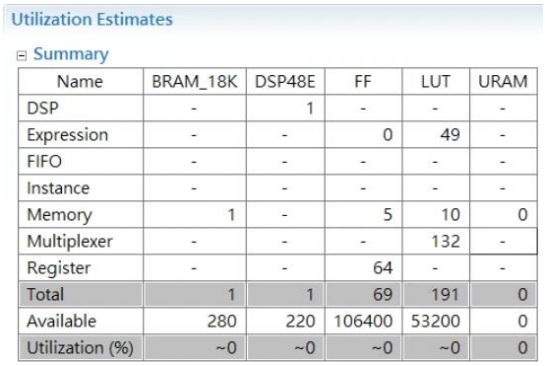
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
260	260	2.600 us	2.600 us	260	260	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- TDL	127	127	2	1	1	127	yes
- MAC	129	129	3	1	1	128	yes



Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	49	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	132	-
Register	-	-	64	-	-
Total	1	1	69	191	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

Since Vivado HLS tool synthesizes for loops in a sequential manner, loop partitioning increases the latency to approximately two times of the original one. The resource requirement also slightly increases due to the elimination of loops co-optimization.

Throughput (Mhz) = 1000/(Clock Period(ns) * #Clock Cycles)
= 1000/(10*260) = 0.385

Question 5 - **Memory Partitioning**

The storage of the arrays in memory plays an important role in area and performance. On one hand, you could put an array entirely in one memory (e.g., BRAM). But this limits the number of read and write accesses per cycle. Or you can divide the array into two or more memories to increase the number of ports. Or you could instantiate each of the variables as its own register, which allows simultaneous access to all of the variables at every clock cycle.

Compare the memory partitioning parameters: block, cyclic, and complete. What is the difference in performance and resource usage (particularly with respect to BRAMs and

FFs)? Which one gives the best performance? Why?

We discuss the two loop functions, TDL and MAC, separately.

For TDL, if we just use loop unrolling with factor 2, the design ends up with II violation. Assume that we store the shift_reg array in one BRAM, and that BRAM has two read ports and one write port. Thus, we can perform two read operations in one cycle but must sequentialize the write operations across two consecutive cycles.

```
TDL:
for (i=N-1;i>0;i--){
#pragma HLS PIPELINE II=1
#pragma HLS unroll factor=2
    shift_reg[i]=shift_reg[i-1];
}
shift_reg[0]=x;
```

Unroll with factor = 2

Performance Estimates

- Timing
  - Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	6.508 ns	1.25 ns

- Latency
  - Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
262	262	2.620 us	2.620 us	262	262	none

  - Detail
    - Instance
    - Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- TDL	129	129	2	2	1	64	yes
- MAC	129	129	3	1	1	128	yes

Utilization Estimates

- Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	71	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	140	-
Register	-	-	73	-	-
Total	1	1	78	221	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

Unroll with factor = 4

Performance Estimates

- Timing
  - Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	6.380 ns	1.25 ns

- Latency
  - Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
264	264	2.640 us	2.640 us	264	264	none

  - Detail
    - Instance
    - Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- TDL	131	131	4	4	1	32	yes
- MAC	129	129	3	1	1	128	yes

Utilization Estimates

- Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	123	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	179	-
Register	-	-	99	-	-
Total	1	1	104	312	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

After adding array reshape with complete, it is possible to read and write to each individual register on every cycle.

Unroll with factor 2 + array_reshape complete

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	22.840 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
199	199	4.545 us	4.545 us	199	199	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- TDL	66	66	3	1	1	64	yes
- MAC	129	129	3	1	1	128	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	45010	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	0	-	5	10	-
Multiplexer	-	-	-	123	-
Register	-	-	6312	-	-
Total	0	1	6317	45143	0
Available	280	220	106400	53200	0
Utilization (%)	0	~0	5	84	0

Unroll with factor = 2 + array_reshape block factor = 2

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.683 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
265	265	2.650 us	2.650 us	265	265	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- TDL	131	131	4	2	1	64	yes
- MAC	130	130	4	1	1	128	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	1039	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	212	-
Register	0	-	276	32	-
Total	1	1	281	1293	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	2	0

Unroll with factor = 2 + array_reshape cyclic factor = 2

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	14.636 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
328	328	4.800 us	4.800 us	328	328	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- TDL	194	194	5	3	1	64	yes
- MAC	130	130	4	1	1	128	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	1005	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	191	-
Register	0	-	254	32	-
Total	1	1	259	1238	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	2	0

As for the case applying array partition with block factor = 2 but without unrolling, the resulting design receive the same latency but larger number of resources then the original. For FF and LUT, the numbers increase from 69 and 191 to 94 and 253.

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	6.380 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
260	260	2.600 us	2.600 us	260	260	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- TDL	127	127	2	1	1	127	yes
- MAC	129	129	3	1	1	128	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	88	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	21	18	0
Multiplexer	-	-	-	147	-
Register	-	-	73	-	-
Total	1	1	94	253	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

As for the MAC loop, it multiplies a value from the array c with a value from the array

shift array. In each iteration, it accesses the i-th value from both arrays. Then, it adds the result of that multiplication into the acc variable. Therefore, we use unroll with factor 4 to remove the dependency of read-after-write (RAW) hazard.

Unroll with factor 4 + array_reshape complete

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	22.840 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
104	104	2.375 us	2.375 us	104	104	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- TDL	66	66	3	1	1	64	yes
- MAC	34	34	4	1	1	32	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	0	0	73385	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	132	-
Register	0	-	8605	64	-
Total	0	2	8605	73581	0
Available	280	220	106400	53200	0
Utilization (%)	0	~0	8	138	0

Unroll with factor 4 + array_reshape block factor = 2

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	10.269 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
201	201	2.064 us	2.064 us	201	201	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- TDL	131	131	4	2	1	64	yes
- MAC	66	66	5	2	1	32	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	0	0	1984	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	20	10	0
Multiplexer	-	-	-	236	-
Register	-	-	367	-	-
Total	1	2	387	2230	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	4	0

Unroll with factor 4 + array_reshape cyclic factor = 2

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	14.636 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
264	264	3.864 us	3.864 us	264	264	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- TDL	194	194	5	3	1	64	yes
- MAC	66	66	5	2	1	32	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	0	0	1695	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	20	10	0
Multiplexer	-	-	-	275	-
Register	-	-	321	-	-
Total	1	2	341	1980	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	3	0

From the above results, we can observe array partition complete results in best latency. However, the resulted resource requirements are intolerable for practical usage. Array partition with block and cyclic lead to similar performance. Since our design in Question3 is already achieved II=1, further partition just increases the number of resources. Therefore, I think this design is not need extra partitioning.

Throughput (Mhz) (array partition complete) = 1000/(Clock Period(ns) * #Clock

Cycles)
= 1000/(10*104) = 0.962

Question 6 - Best Design

Combine any number of optimizations to get your best architecture. A design with high throughput will likely take a lot of resources. A design that has small resource usage likely will have lower performance, but that could still be the best depending the application goals.

In what way is it the best? What optimizations did you use to obtain this result? It is possible to create a design that outputs a result every cycle, i.e., get one sample per cycle, so a throughput of 100 MHz (assuming a 10 ns clock).

```
static
 data_t_8 shift_reg[N];

acc_t16 acc;
int i;
#pragma HLS array_reshape variable=shift_reg cyclic factor=3
#pragma HLS array_reshape variable=c cyclic factor=3
acc = 0;
Shift_Accum_Loop:
for (i=N-1;i>0;i--){
#pragma HLS UNROLL
#pragma HLS PIPELINE II=1
    shift_reg[i]=shift_reg[i-1];
    acc += shift_reg[i]*c[i];
}
shift_reg[0] = x;
acc += shift_reg[0]*c[0];

*y = acc;
```

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	9.871 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
2	2	20.000 ns	20.000 ns	2	2	none

Detail

Instance

Loop

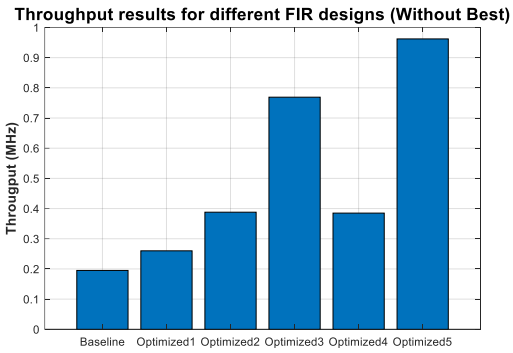
Utilization Estimates

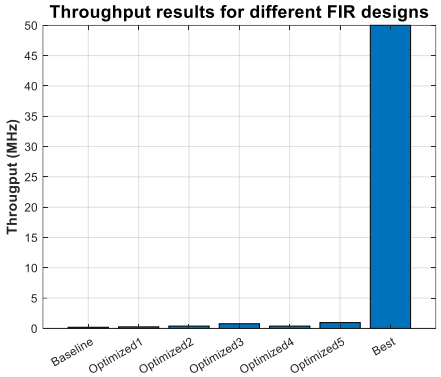
Summary

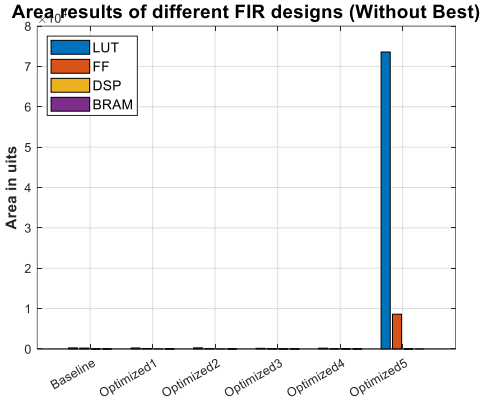
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	24	-	-	-
Expression	-	-	0	3887	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	21	-
Register	-	-	1483	-	-
Total	0	24	1483	3908	0
Available	280	220	106400	53200	0
Utilization (%)	0	10	1	7	0

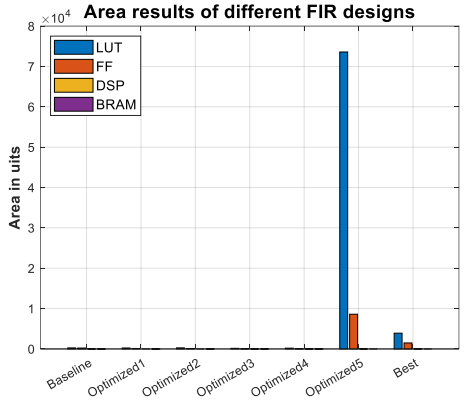
The performance comparison results are shown in the following figures. In my best version, I use **unroll, pipeline and cyclic array partition**. Since the throughput of FIR system is the most important issue in the real-world scenario, my design aims to achieve low latency. In fact, compared to other optimized methods, the resource requirements are far less than the complete array partition version (Optimized5). My design can achieve II=1, i.e., get one sample per cycle.

Throughput (Mhz) = 1000/(Clock Period(ns) * #Clock Cycles)
= 1000/(10*2) = 50









Github link: https://github.com/Shan-handsome/2022_AAHLS_FIR