Rajan Verma

WES MAS – WES 237 C

HW 1 : FIR

- **Question 1 - Variable Bitwidths:** You can specify a precise data type for each variable in your design. There many different data types including 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 minimum data size can you use without losing accuracy (i.e., your results still match the golden output)?

  Answer:

  Performance with dafault variable types:

Target	Estimated	Uncertainty
10.00 ns	5.876 ns	2.70 ns

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
▼ fir				-	132	1.320E3	-	133	-	no	3	1	120	241	0
▶ fir_Pipeline_VITIS_LOOP_29_1				-	130	1.300E3	-	130	-	no	3	1	85	182	0

With Variable bitwidths :
Code in : fir128_optimized1

Following are the minimum bit width that can be used.

ap_int<5> c[N]

static **ap_int<8>** shift_reg[N];

ap_int<16> acc;

ap_int<8>I;

Performance and resources with optimized bitwidths :

I see that in general the FF and LUT unitilization has decreased which is inline with the theory that when in program we use the bitwidth efficiently it reflects in less hardware required.

Estimated performance also increased.

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
fir				-	132	1.320E3	-	133	-	no	1	0	85	239	0
fir_Pipeline_VITIS_LOOP_31_1				-	130	1.300E3	-	130	-	no	1	0	58	196	0

Target	Estimated	Uncertainty
10.00 ns	4.351 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

n Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
		-	132	1.320E3	-	133	-	no	1	0	85	239	0
		-	130	1.300E3	-	130	-	no	1	0	58	196	0

- **Question 2 - Pipelining:** Pipelining increases throughput, typically at the cost of additional resources. The initiation interval (II) is an important design parameter that affects 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?

  with used pragma HLS unroll factor=4 with no comparative if else

Target	Estimated	Uncertainty
10.00 ns	2.198 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
				-	134	1.340E3	-	135	-	no	1	1	113	230	0
1				-	132	1.320E3	-	132	-	no	0	1	109	122	0

with

```
#pragma HLS unroll factor=4

#pragma HLS pipeline II=1

    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];

      }

    }
```

#pragma HLS pipeline II=1 Throughput : 193.24MHz

Target	Estimated	Uncertainty
10.00 ns	5.175 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
			-	3	30.000	-	1	-	yes	0	23	2894	5155	0

#pragma HLS pipeline II=2 , Throughput = 171.48 Mhz.

Target	Estimated	Uncertainty
10.00 ns	5.832 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP
fir				-	4	40.000	-	2	-	yes	0	23

#pragma HLS pipeline II=3 Throughput = 171.24 Mhz.

Target	Estimated	Uncertainty
10.00 ns	5.840 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
			-	5	50.000	-	3	-	yes	0	23	1272	4544	0

#pragma HLS pipeline II=4 Throughput = 171.24 Mhz.

Target	Estimated	Uncertainty
10.00 ns	5.840 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
			-	6	60.000	-	4	-	yes	0	23	1279	4559	0

#pragma HLS pipeline II=1 gives the best throughput and afterwards the throughput is at 171Mhz for other IIs values.

We will set II value depending upon

- **Available hardware resources (DSPs, LUTs, etc.)**
- **Number of operations per iteration**
- **Resource sharing requirements**
- **Memory port limitations**

- **Question 3 - Removing Conditional Statements:** If/else statements and other conditionals can limit the possible parallelism and often require additional resources. Rewriting the code to remove them 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?

  Resources:

  FF reduced from -> 120 -> 113

  LUTS reduced from 241->230

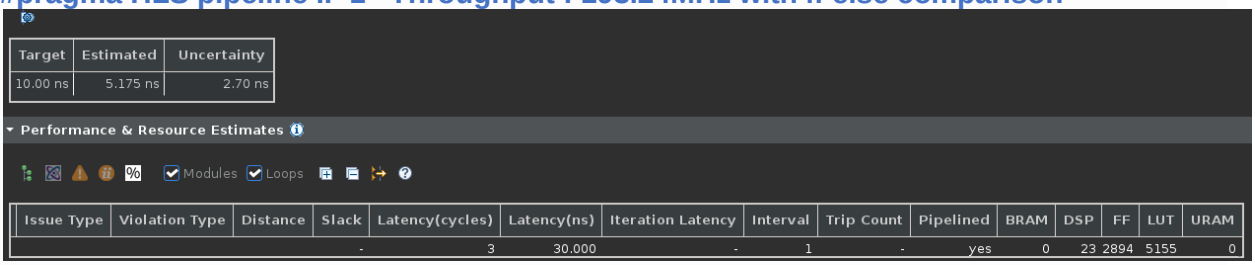
  Conditional statement leads to design always checking for the condition to be true false for every loop, this adds extra cycles which can be reduced.

  Code:

```
        for(i =N - 1; i > 0; i--) {

          shift_reg[i] = shift_reg[i - 1];

          acc += shift_reg[i] * c[i];

      }

     acc += x*c[0];
```

```
shift_reg[0] = x;

*y = acc;
```

#pragma HLS pipeline II=1 Throughput : 193.24MHz with if else comparison



Target	Estimated	Uncertainty
10.00 ns	5.175 ns	2.70 ns

Performance & Resource Estimates

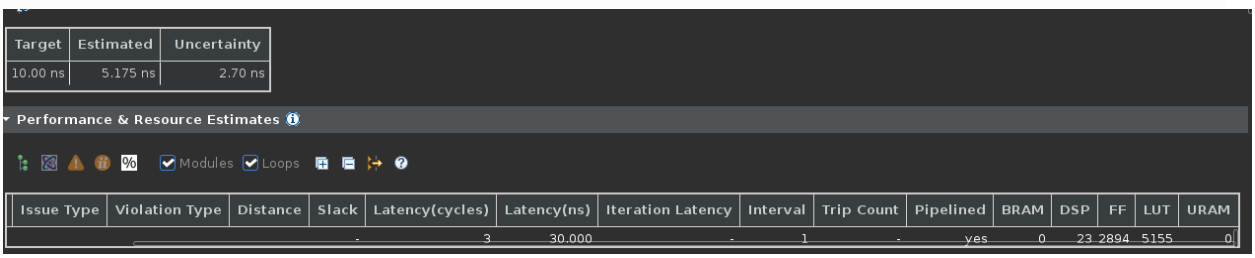
Modules Loops

Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
			-	3	30.000	-	1	-	yes	0	23	2894	5155	0

Vs

#pragma HLS pipeline II=1 and pragma unroll Throughput : 193.24MHz without if else comparison statement . Its coming out to be same.

Designs are different though they both are using same resources. I am surprised at this.



Target	Estimated	Uncertainty
10.00 ns	5.175 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
			-	3	30.000	-	1	-	yes	0	23	2894	5155	0

- **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 performance and 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 does the number of resources change? Why?

Yes shift registers and accumulator can be separated out.
This will lead to more performance but also lead to more resources being used.

But somehow I see same resources! Designes are also pretty similar. Is the tool optimizing it already?

Target	Estimated	Uncertainty
10.00 ns	5.175 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

y(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
3	30.000	-	1	-	yes	0	23	2894	5155	0

```
#pragma HLS unroll factor=4

#pragma HLS pipeline II=1

#pragma HLS ARRAY_PARTITION variable=shift_reg cyclic factor=4

#pragma HLS ARRAY_PARTITION variable=c cyclic factor=4

for(int i =N - 1; i >= 0; i--) {   shift_reg[i] = shift_reg[i - 1]; }

shift_reg[0] = x;

for(int i =N - 1; i >= 0; i--) {

    if(i==0) {acc += x*c[0] ; }

    else {   acc += shift_reg[i] * c[i]; }

}


*y = acc;
```

- **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 variable as a register allowing simultaneous access to all 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?

Cyclic:

Target	Estimated	Uncertainty
10.00 ns	5.175 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

y(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
3	30.000	-	1	-	yes	0	23	2894	5155	0

Block:

Target	Estimated	Uncertainty
10.00 ns	5.175 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

y(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
3	30.000	-	1	-	yes	0	23	2894	5155	0

Complete is also same! I don't know why no difference…

- **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 with 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).

It is possible that some optimizations have little (or no effect). Some optimizations may only work when used in combination with others. This is what makes the design space exploration process difficult.

Target	Estimated	Uncertainty
10.00 ns	5.175 ns	2.70 ns

Performance & Resource Estimates

Modules Loops

y(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
3	30.000	-	1	-	yes	0	23	2894	5155	0

I used following :

```
#pragma HLS unroll factor=
#pragma HLS pipeline II=1
#pragma HLS ARRAY_PARTITION variable=shift_reg complete
#pragma HLS ARRAY_PARTITION variable=c block complete
```