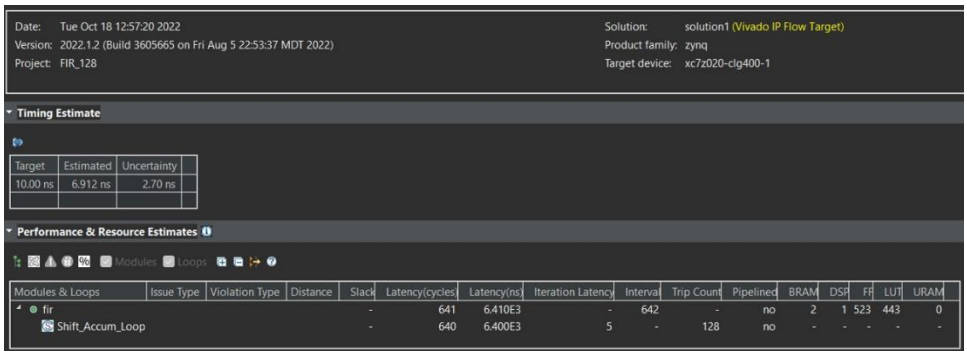
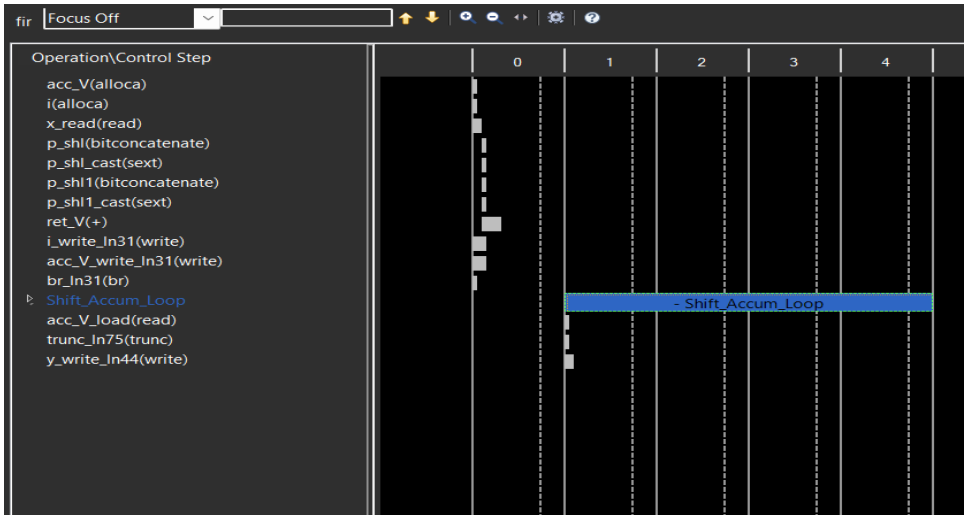
WES 237C Project 1 Report

Hassan Ahmad: A59014371

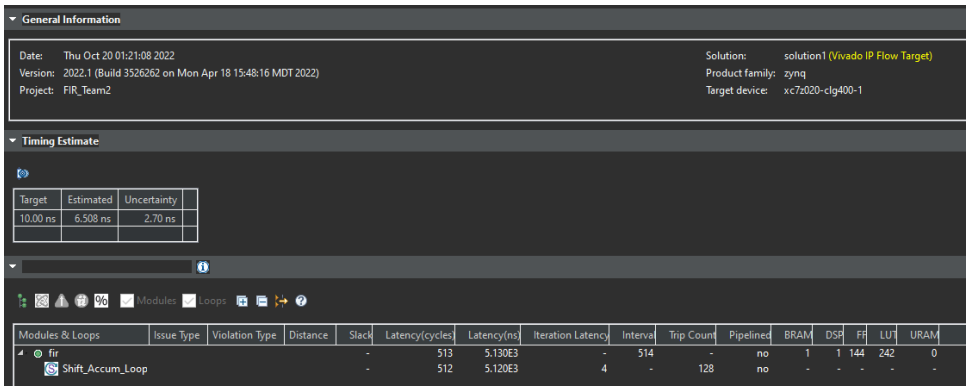
Sang Ryul Pae (Eric): A59014357

** Baseline: 6.41us*





1. Variable Bit Width: 5.12us

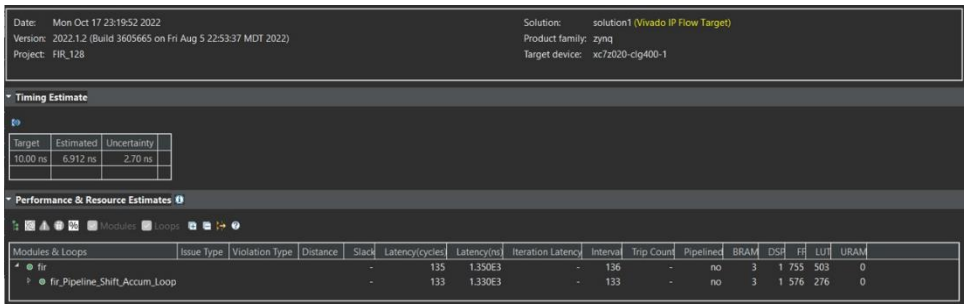


Using variable bit width did not show much improvement in performance. We set these following arbitrary precision data types initially:

```
#include "ap_int.h"
typedef ap_int<5>       coef_t;
typedef ap_int<17>      data_t;
typedef ap_int<22>      acc_t;
```

When we decreased the size of the largest data type, acc_t, from a bit width of 22 down to 20, we did see a slight improvement in the total number of cycles compared to the baseline performance.

2. Pipelining: 1.35us



With no II set, we got the above latency of 135 cycles. When we set II to 1, we got a low latency of

116 cycles, but a slightly longer estimated clock cycle at 7.186 ns. So, II doesn't help performance in this example as II doesn’t give any benefit to other tasks . With II set to 2, we got the same performance as when II = 1, showing that adding excessive initiation interval did not improve the execution of the loop iterations.

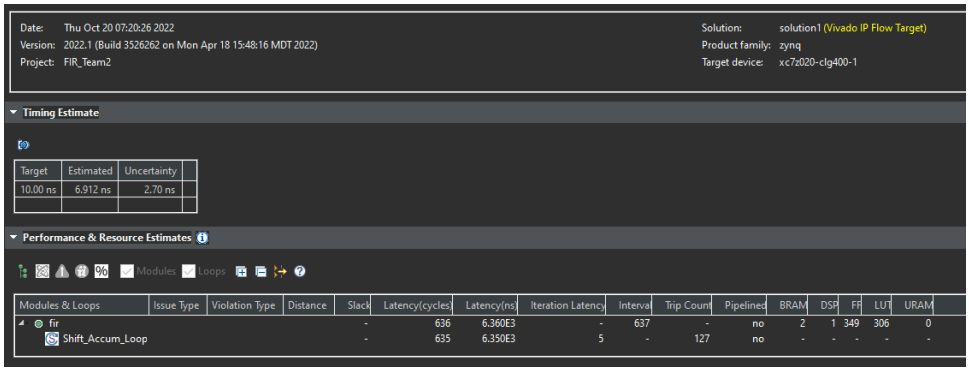
====== pipeline/unroll/partitioning in fir.cpp ==================

(<-- partitioning + unrolling gave big improvements then pipeline didn`t give any further improvement whth them.)

```
void fir (  data_t *y,  data_t x  )
{
#pragma HLS PIPELINE
#pragma hls interface s_axilite port=return
coef_t c[N] = {...};
static data_t shift_reg[N];
acc_t acc;
int i;
acc = 0;
```

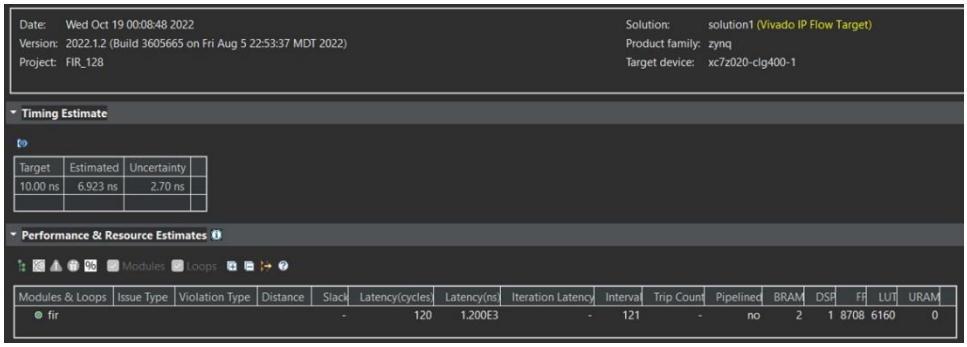
```
#pragma HLS ARRAY_PARTITION variable=c type=complete dim=1
Shift_Accum_Loop:
for(i=N-1;i>=0;i--){

#pragma HLS unroll
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 ARRAY_PARTITION variable= shift_reg type=complete dim=1
}
}
*y = acc;
}
```

3. Removing Conditional Statements: 6.35us



When the conditional if-statement was removed from the for-loop in our code, we did see a slight improvement in the latency. When Pipelining is included, there is not much change as the if statement only was relevant during one iteration of the loop.

4. Loop unrolling : 1.20us



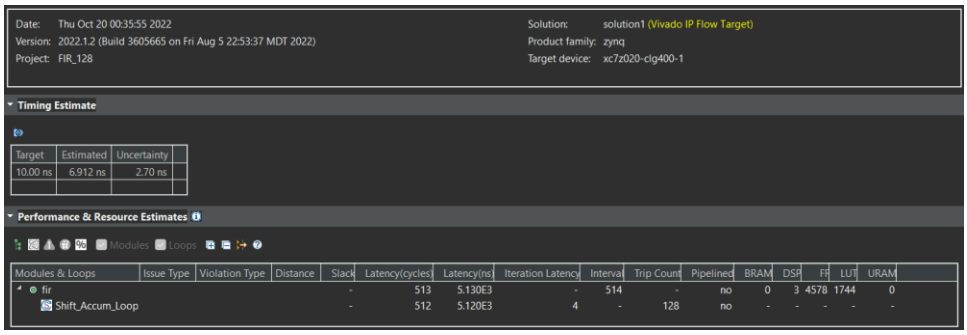
There is opportunity for loop partitioning in FIR filters, as the algorithm in the filter is identical for the entire calculation. The loops can be separated into individual sections and unrolled to improve performance.

Also, we can see the resource change like ‘FF’ from 144 to 8708 and ‘LUT’ from 443 to 6160.

Definitely it’s for parallelism via loop unrolling.

5. Memory Partitioning: 5.14us

: c [] / shift_reg[] partitioning(complete)



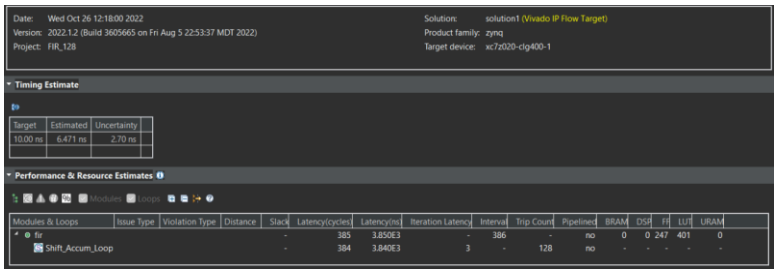
In our implementation we only tested the complete memory partitioning, as that allowed us to access all the variables simultaneously to give best parallelism. This would be the most effective method to use along with loop unrolling.

<partitioning with block/cycle >

*It didn’t work without bit width setting with AP_int<>

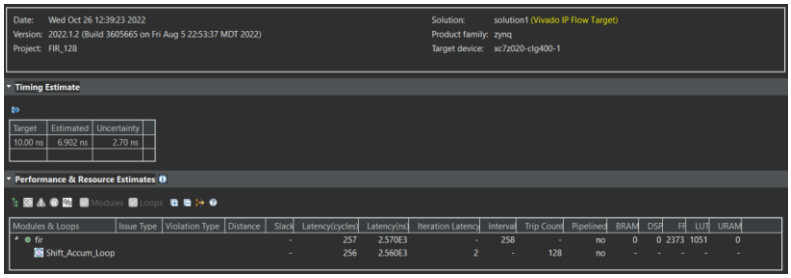
Block(factor:4) & cycle(factor:4) both improved performance from 5.12us to 3.85us but it's worse than complete partitioning in the same conditions(2.85us)

BRAM/URAM were not used in all the cases tested but FF count is less in block/cycle cases and as it's because it's handling in block while complete handles each memory separately.



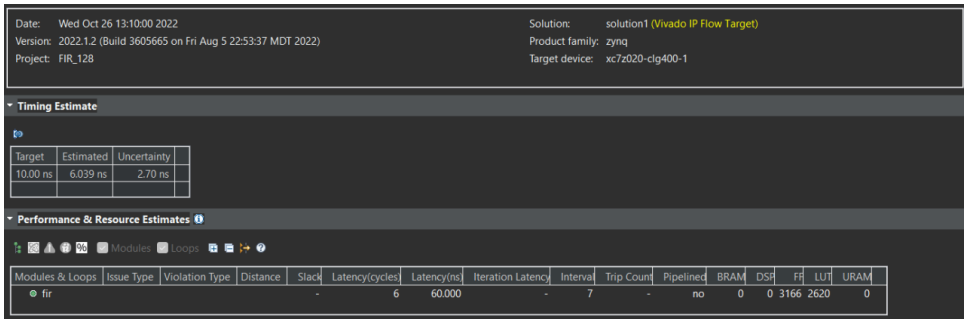
Block(factor:32) & cycle(factor:32) both improved performance from 5.12us to 2.57us and it was better than complete partitioning in the same conditions(2.85us)

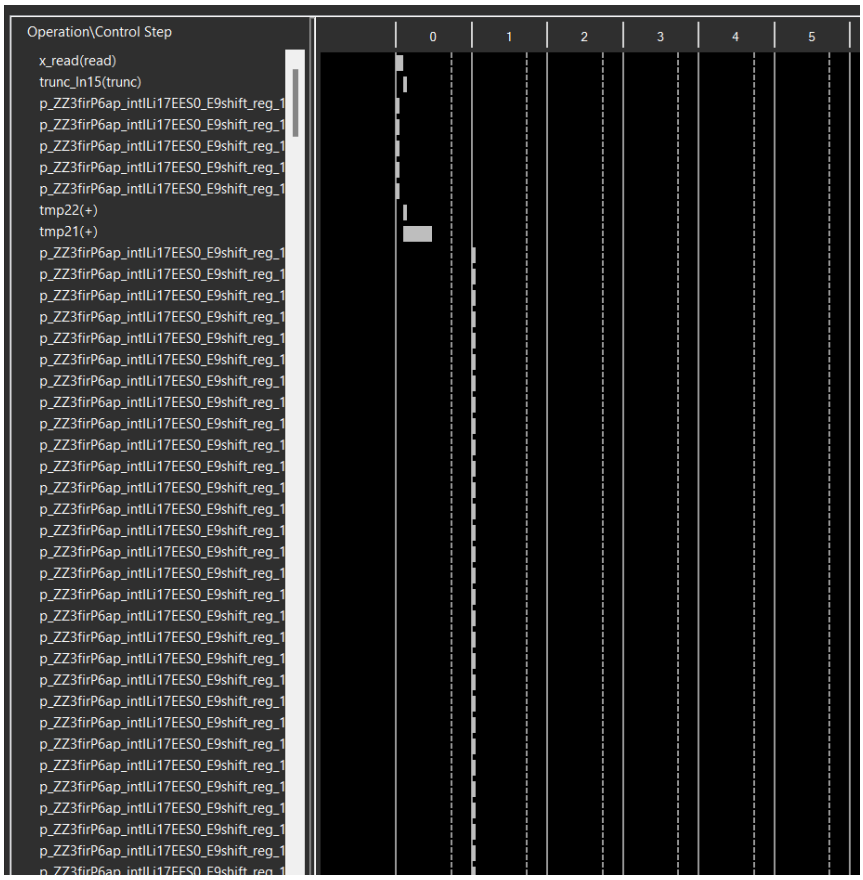
Also, together with pipelining it gives the better performance to 60ns compared to 300ns with complete partitioning. So, proper partitioning gives optimum performance. We didn't try further as it's already good enough to target.



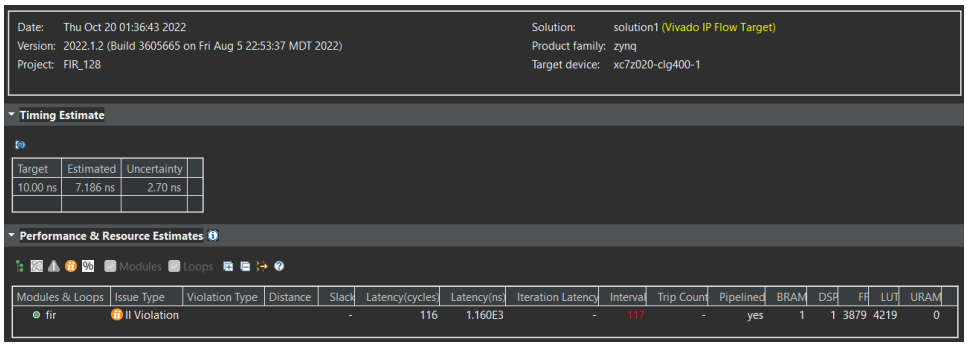
6. Best Design: Memory Partitioning(block:32) + loop unrolling + Pipelining + bit width:60ns

   : c[] / shift_reg[] partitioning(complete), for Shift_Accum_Loop unrolling, AP_int, Pipelining





6b. Other Design: Pipelining + loop unrolling: 1.16us



FIR128 Throughput Calculation

- Throughput (MHz) = 1000 / ((Estimated Clock Period in ns) * (# of clock cycles))

Baseline:

Throughput = 1000 / ((6.912 ns)(641 cycles)) =* ***0.2257 MHz***

1. Variable Bit Width:

Throughput = 1000 / ((6.508 ns)(513 cycles)) =* ***0.2995 MHz***

2. Pipelining:

Throughput = 1000 / ((6.912 ns)(135 cycles)) =* ***1.072 MHz***

3. Code Hoisting (Remove Conditional Statements):

Throughput = 1000 / ((6.912 ns)(636 cycles)) =* ***0.2275 MHz***

4. Loop Partitioning (Unrolling):

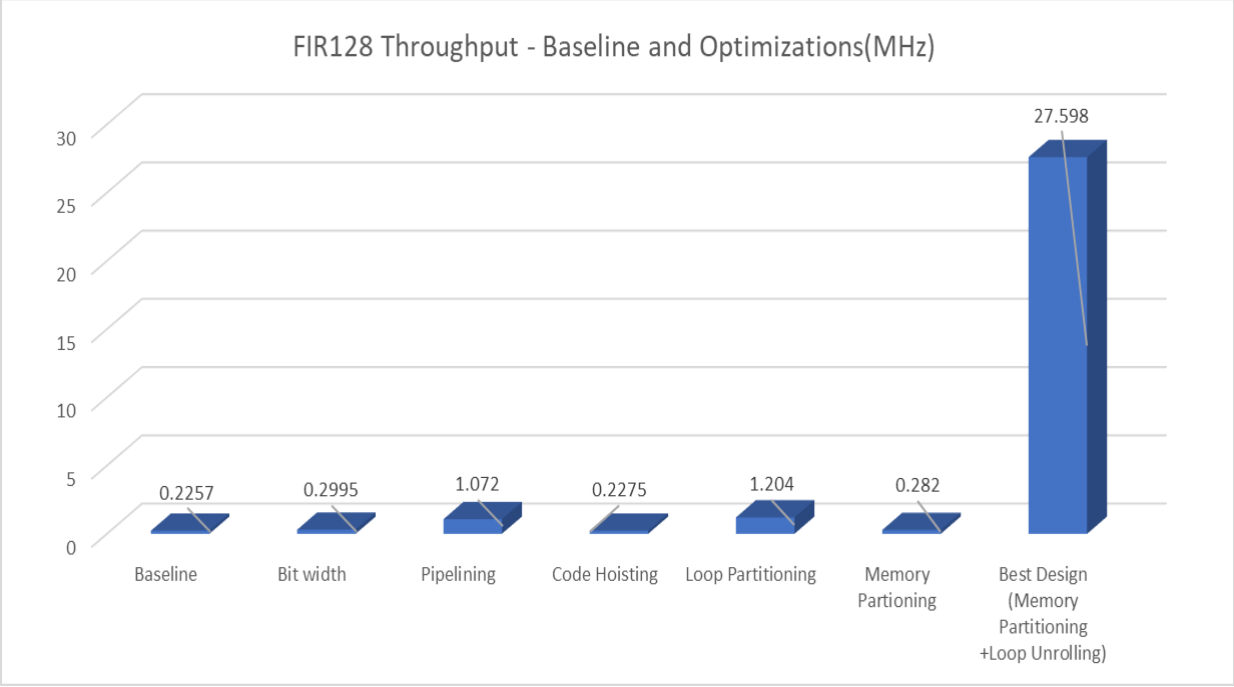
Throughput = 1000 / ((6.923 ns)(120 cycles)) =* ***1.204 MHz***

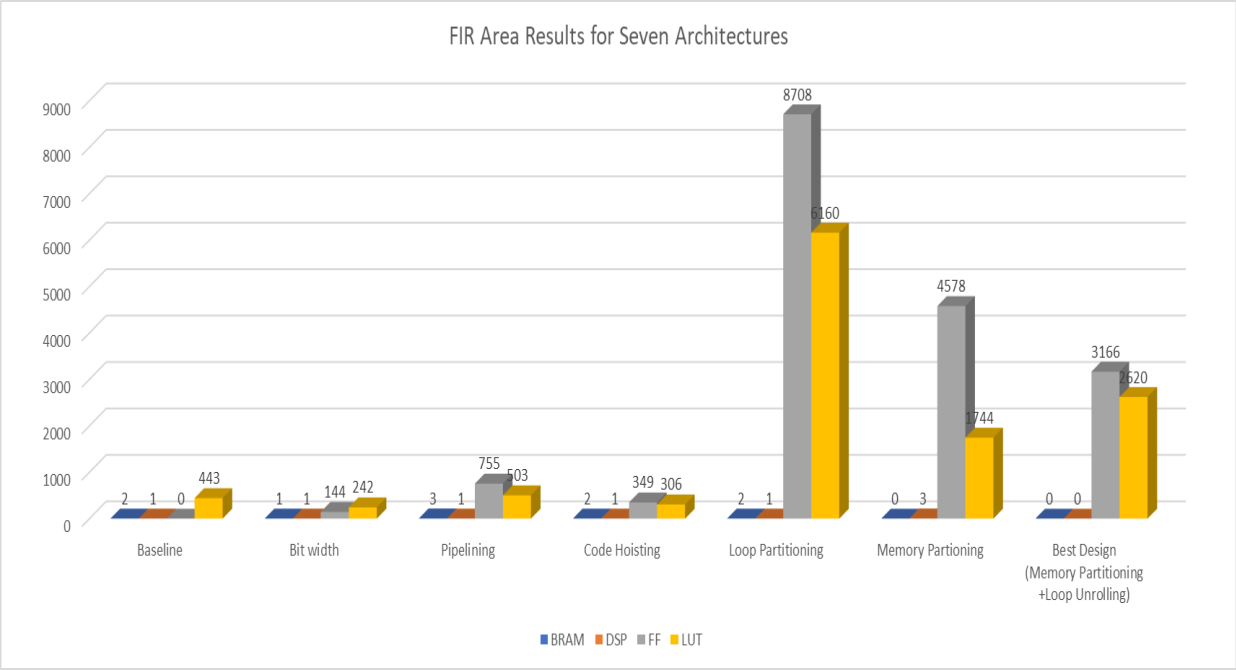
5. Memory Partitioning:

Throughput = 1000 / ((6.912 ns)(513 cycles)) =* ***0.2820 MHz***

6. Best Design (Memory Partitioning[block:32] + loop unrolling + Pipelining + bit width):

Throughput = 1000 / ((6.039 ns)(6 cycles)) =* ***27.598 MHz***




FIR Area Results for Seven Architectures
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
2 1 0 443
1 1 144 242
3 1 755 503
2 1 349 306
2 1 8708 6160
0 3 4578 1744
0 0 3166 2520
Baseline
Bit width
Pipelining
Code Hoisting
Loop Partitioning
Memory Partioning
Best Design (Memory Partitioning +Loop Unrolling)
BRAM DSP FF LUT