HLS Lab B – FIR Optimization

R10943119 蔡岳峰

FIR-11

1. Baseline

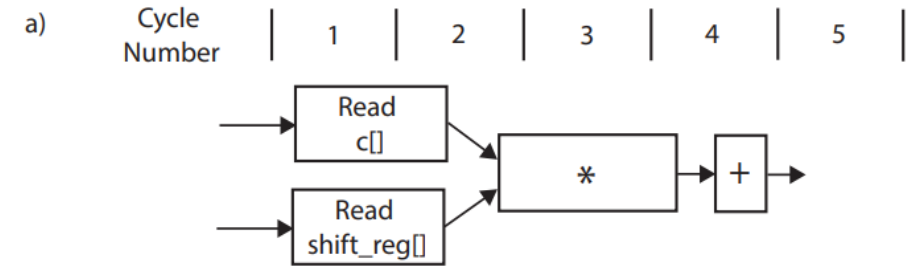
<table>
<tr><th></th><th>Vivado 2020.1</th><th>Vitis 2022.1</th></tr>
<tr><td>Performance</td><td>
Performance Estimates
Timing
Summary
<table>
<tr><th>Clock</th><th>Target</th><th>Estimated</th><th>Uncertainty</th></tr>
<tr><td>ap_clk</td><td>10.00 ns</td><td>8.510 ns</td><td>1.25 ns</td></tr>
</table>
Latency
Summary
<table>
<tr><th colspan="2">Latency (cycles)</th><th colspan="2">Latency (absolute)</th><th colspan="2">Interval (cycles)</th><th></th></tr>
<tr><th>min</th><th>max</th><th>min</th><th>max</th><th>min</th><th>max</th><th>Type</th></tr>
<tr><td>66</td><td>66</td><td>0.660 us</td><td>0.660 us</td><td>66</td><td>66</td><td>none</td></tr>
</table>
Detail
Instance
Loop
<table>
<tr><th></th><th colspan="2">Latency (cycles)</th><th></th><th colspan="2">Initiation Interval</th><th></th><th></th></tr>
<tr><th>Loop Name</th><th>min</th><th>max</th><th>Iteration Latency</th><th>achieved</th><th>target</th><th>Trip Count</th><th>Pipelined</th></tr>
<tr><td>- shift_reg</td><td>20</td><td>20</td><td>2</td><td>-</td><td>-</td><td>10</td><td>no</td></tr>
<tr><td>- forward</td><td>44</td><td>44</td><td>4</td><td>-</td><td>-</td><td>11</td><td>no</td></tr>
</table>
</td><td>
Performance Estimates
Timing
Summary
<table>
<tr><th>Clock</th><th>Target</th><th>Estimated</th><th>Uncertainty</th></tr>
<tr><td>ap_clk</td><td>10.00 ns</td><td>6.912 ns</td><td>2.70 ns</td></tr>
</table>
Latency
Summary
<table>
<tr><th colspan="2">Latency (cycles)</th><th colspan="2">Latency (absolute)</th><th colspan="2">Interval (cycles)</th><th></th></tr>
<tr><th>min</th><th>max</th><th>min</th><th>max</th><th>min</th><th>max</th><th>Type</th></tr>
<tr><td>33</td><td>33</td><td>0.330 us</td><td>0.330 us</td><td>34</td><td>34</td><td>no</td></tr>
</table>
Detail
Instance
<table>
<tr><th></th><th></th><th colspan="2">Latency (cycles)</th><th colspan="2">Latency (absolute)</th><th colspan="2">Interval (cycles)</th><th></th></tr>
<tr><th>Instance</th><th>Module</th><th>min</th><th>max</th><th>min</th><th>max</th><th>min</th><th>max</th><th>Type</th></tr>
<tr><td>grp_fir_Pipeline_shift_reg_fu_60</td><td>fir_Pipeline_shift_reg</td><td>12</td><td>12</td><td>0.120 us</td><td>0.120 us</td><td>12</td><td>12</td><td>no</td></tr>
<tr><td>grp_fir_Pipeline_forward_fu_66</td><td>fir_Pipeline_forward</td><td>16</td><td>16</td><td>0.160 us</td><td>0.160 us</td><td>16</td><td>16</td><td>no</td></tr>
</table>
Loop
N/A
</td></tr>
<tr><td>Utilization</td><td>
Utilization Estimates
Summary
<table>
<tr><th>Name</th><th>BRAM_18K</th><th>DSP48E</th><th>FF</th><th>LUT</th><th>URAM</th></tr>
<tr><td>DSP</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Expression</td><td>-</td><td>2</td><td>0</td><td>103</td><td>-</td></tr>
<tr><td>FIFO</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Instance</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Memory</td><td>0</td><td>-</td><td>74</td><td>8</td><td>0</td></tr>
<tr><td>Multiplexer</td><td>-</td><td>-</td><td>-</td><td>110</td><td>-</td></tr>
<tr><td>Register</td><td>-</td><td>-</td><td>129</td><td>-</td><td>-</td></tr>
<tr><td>Total</td><td>0</td><td>2</td><td>203</td><td>221</td><td>0</td></tr>
<tr><td>Available</td><td>280</td><td>220</td><td>106400</td><td>53200</td><td>0</td></tr>
<tr><td>Utilization (%)</td><td>0</td><td>~0</td><td>~0</td><td>~0</td><td>0</td></tr>
</table>
</td><td>
Utilization Estimates
Summary
<table>
<tr><th>Name</th><th>BRAM_18K</th><th>DSP</th><th>FF</th><th>LUT</th><th>URAM</th></tr>
<tr><td>DSP</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Expression</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>FIFO</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Instance</td><td>0</td><td>2</td><td>414</td><td>256</td><td>-</td></tr>
<tr><td>Memory</td><td>0</td><td>-</td><td>64</td><td>6</td><td>0</td></tr>
<tr><td>Multiplexer</td><td>-</td><td>-</td><td>-</td><td>114</td><td>-</td></tr>
<tr><td>Register</td><td>-</td><td>-</td><td>8</td><td>-</td><td>-</td></tr>
<tr><td>Total</td><td>0</td><td>2</td><td>486</td><td>376</td><td>0</td></tr>
<tr><td>Available</td><td>280</td><td>220</td><td>106400</td><td>53200</td><td>0</td></tr>
<tr><td>Utilization (%)</td><td>0</td><td>~0</td><td>~0</td><td>~0</td><td>0</td></tr>
</table>
</td></tr>
</table>

```c
#include "fir.h"

void fir (
  data_t *y,
  data_t x
  )
{

    char c[N] = {10, 11, 11, 8, 3, -3, -8, -11,

    // Write your cSode here
//  static acc_t reg[N] = {0};
    static char reg[N] = {0};

    shift_reg: for(int i = N-1; i > 0; i--){
        reg[i] = reg[i-1];
    }
    reg[0] = (char)x;
    *y = 0;
    forward: for(int i = 0; i < N; i++){
        *y += c[i] * reg[i];
    }
}
```

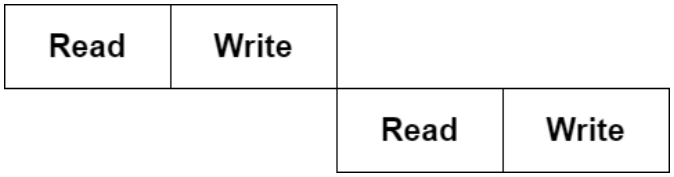
不同版本的 IDE，他們底下就會自動做了很多優化。因此這邊拿了新版優化過後的版本作為接下來的優化方向的參考。

可以看到 2020.1 版本的就是很陽春的做法。我將 shift register loop 以及 MAC forward loop 分開來寫，可以看到一個乘加運算需要 4 個 cycle，總共需要做 11 次因此需要 44 個 cycle。而 shift register 則是總共需要 shift 10 次。再考慮到讀取以及寫入，也需要 2 個 cycle。因此最後的 latency 為 66。

而在 2022.1 優化過後的版本，可以發現他已經把 loop 全部拆掉了，轉變成兩個不同的 instance。這兩個 instance 都有做相對應的再次優化，因此最後 latency 分別為 12 及 16。還考慮到需要進出兩個 instance 以及本身的讀寫，因此最後 latency 為 33。也因為有做了一些優化，造成需要更多的硬體資源才能達到這樣的結果，因此在 FF 以及 LUT 的使用上都大約為 2020.1 版本的兩倍。



這張圖可以說明大致上的喔個 for loop 內做的事情。必須先讀取係數以及暫存器內的變數並且做乘加運算。總共會需要 4 個 cycle。



而 shift register 若是尚未優化過，IDE 會講讀取與寫入給分開寫。因此每次都需要 2 個 cycle 才能做完一個暫存器內的值的更新。

FIR128

1. Baseline

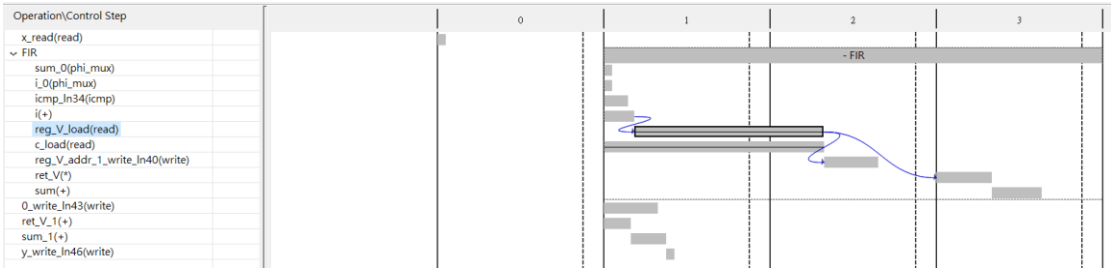
<table>
<tr><th></th><th>Vivado 2020.1</th><th>Vitis 2022.1</th></tr>
<tr><td>Perf.</td><td>
Performance Estimates
Timing
Summary
<table>
<tr><th>Clock</th><th>Target</th><th>Estimated</th><th>Uncertainty</th></tr>
<tr><td>ap_clk</td><td>10.00 ns</td><td>8.510 ns</td><td>1.25 ns</td></tr>
</table>
Latency
Summary
Throughput (MHz): 0.20
<table>
<tr><th colspan="2">Latency (cycles)</th><th colspan="2">Latency (absolute)</th><th colspan="2">Interval (cycles)</th><th></th></tr>
<tr><th>min</th><th>max</th><th>min</th><th>max</th><th>min</th><th>max</th><th>Type</th></tr>
<tr><td>509</td><td>509</td><td>5.090 us</td><td>5.090 us</td><td>509</td><td>509</td><td>none</td></tr>
</table>
Detail
Instance
N/A
Loop
<table>
<tr><th></th><th colspan="2">Latency (cycles)</th><th></th><th colspan="2">Initiation Interval</th><th></th><th></th></tr>
<tr><th>Loop Name</th><th>min</th><th>max</th><th>Iteration Latency</th><th>achieved</th><th>target</th><th>Trip Count</th><th>Pipelined</th></tr>
<tr><td>- Loop 1</td><td>508</td><td>508</td><td>4</td><td>-</td><td>-</td><td>127</td><td>no</td></tr>
</table>
</td><td>
Performance Estimates
Timing
Summary
<table>
<tr><th>Clock</th><th>Target</th><th>Estimated</th><th>Uncertainty</th></tr>
<tr><td>ap_clk</td><td>10.00 ns</td><td>6.912 ns</td><td>2.70 ns</td></tr>
</table>
Latency
Summary
<table>
<tr><th colspan="2">Latency (cycles)</th><th colspan="2">Latency (absolute)</th><th colspan="2">Interval (cycles)</th><th></th></tr>
<tr><th>min</th><th>max</th><th>min</th><th>max</th><th>min</th><th>max</th><th>Type</th></tr>
<tr><td>134</td><td>134</td><td>1.340 us</td><td>1.340 us</td><td>135</td><td>135</td><td>no</td></tr>
</table>
Detail
Instance
<table>
<tr><th></th><th></th><th colspan="2">Latency (cycles)</th><th colspan="2">Latency (absolute)</th><th colspan="2">Interval (cycles)</th><th></th></tr>
<tr><th>Instance</th><th>Module</th><th>min</th><th>max</th><th>min</th><th>max</th><th>min</th><th>max</th><th>Type</th></tr>
<tr><td>grp_fir_Pipeline_FIR_fu_62</td><td>fir_Pipeline_FIR</td><td>132</td><td>132</td><td>1.320 us</td><td>1.320 us</td><td>132</td><td>132</td><td>no</td></tr>
</table>
Loop
N/A
</td></tr>
<tr><td>Util.</td><td>
Utilization Estimates
Summary
<table>
<tr><th>Name</th><th>BRAM_18K</th><th>DSP48E</th><th>FF</th><th>LUT</th><th>URAM</th></tr>
<tr><td>DSP</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Expression</td><td>-</td><td>2</td><td>0</td><td>149</td><td>-</td></tr>
<tr><td>FIFO</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Instance</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Memory</td><td>1</td><td>-</td><td>5</td><td>10</td><td>0</td></tr>
<tr><td>Multiplexer</td><td>-</td><td>-</td><td>-</td><td>87</td><td>-</td></tr>
<tr><td>Register</td><td>-</td><td>-</td><td>127</td><td>-</td><td>-</td></tr>
<tr><td>Total</td><td>1</td><td>2</td><td>132</td><td>246</td><td>0</td></tr>
<tr><td>Available</td><td>280</td><td>220</td><td>106400</td><td>53200</td><td>0</td></tr>
<tr><td>Utilization (%)</td><td>~0</td><td>~0</td><td>~0</td><td>~0</td><td>0</td></tr>
</table>
</td><td>
Utilization Estimates
Summary
<table>
<tr><th>Name</th><th>BRAM_18K</th><th>DSP</th><th>FF</th><th>LUT</th><th>URAM</th></tr>
<tr><td>DSP</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Expression</td><td>-</td><td>-</td><td>0</td><td>64</td><td>-</td></tr>
<tr><td>FIFO</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Instance</td><td>1</td><td>1</td><td>381</td><td>192</td><td>-</td></tr>
<tr><td>Memory</td><td>1</td><td>-</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>Multiplexer</td><td>-</td><td>-</td><td>-</td><td>76</td><td>-</td></tr>
<tr><td>Register</td><td>-</td><td>-</td><td>4</td><td>-</td><td>-</td></tr>
<tr><td>Total</td><td>2</td><td>1</td><td>385</td><td>332</td><td>0</td></tr>
<tr><td>Available</td><td>280</td><td>220</td><td>106400</td><td>53200</td><td>0</td></tr>
<tr><td>Utilization (%)</td><td>~0</td><td>~0</td><td>~0</td><td>~0</td><td>0</td></tr>
</table>
</td></tr>
</table>

我這邊換成了只用一個 for loop 的寫法，這樣在後面的討論會比較統一，在比較上也會比較公平。

這邊可以看到與 FIR-11 一樣的內容，比較不一樣的是他已經將 shift register 需要做的事情藏在 MAX 底下了，因此並不會再多消耗額外的時間來完成。一個 loop 裡面做 MAC 還是需要 4 個 cycle，並且需要 loop 127 次。因此總共為 508 個 cycle。最後在迴圈外面需要再做一次讀取與寫入，因此最後 latency 為 509。

在 2022.1 版本中也是將 for loop 給拔掉換成 instance。並且也是使用硬體資源換取 latency 的下降。

以下的時序圖可以大致說明每個迴圈需要做的事情，其中包含 reg 的 load 及 c array 的 load。最後再接上乘加運算的時間。



2. Variable Bit widths

- **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)?

這邊使用 ap_fixed.h header file 來提供 arbitrary precision 的運算，此 header file 與 ap_int.h 在這邊都可以正常的運作。

```
#include "fir.h"
#include "ap_fixed.h"

typedef ap_int<5> mytype;

void fir (
  data_t *y,
  data_t x
  )
{

    mytype c[N] = {10, 11, 11, 8, 3, -3, -8,

    // Write your cSode here
//  static acc_t reg[N] = {0};
    static char reg[N] = {0};

    shift_reg: for(int i = N-1; i > 0; i--){
        reg[i] = reg[i-1];
    }
    reg[0] = (char)x;
    *y = 0;
    forward: for(int i = 0; i < N; i++){
        *y += c[i] * reg[i];
    }
}
```

最後實驗的結果，是 C array 需要 5 個 bit、register 需要 8 bit 才能保持運算結果的正確性。這邊我叫他 AP best。我將不同的 precision 以及 data type 互相比較。

Int

Performance

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
509	509	5.090 us	5.090 us	509	509	none

Detail

Instance

N/A

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	508	508	4	-	-	127	no

Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	2	0	149	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	87	-
Register	-	-	127	-	-
Total	1	2	132	246	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

Long long

Performance

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
509	509	5.090 us	5.090 us	509	509	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- FIR	508	508	4	-	-	127	no

Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	2	0	149	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	87	-
Register	-	-	127	-	-
Total	1	2	132	246	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

AP best

Performance

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	6.508 ns	1.25 ns

Latency

Throughput (MHz): 0.26

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
382	382	3.820 us	3.820 us	382	382	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	381	381	3	-	-	127	no

Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	64	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	81	-
Register	-	-	57	-	-
Total	1	1	62	155	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

Float

Performance

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.303 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
1663	1663	16.630 us	16.630 us	1663	1663	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- FIR	1651	1651	13	-	-	127	no

Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	981	-
FIFO	-	-	-	-	-
Instance	-	5	688	1265	-
Memory	2	-	0	0	0
Multiplexer	-	-	-	220	-
Register	-	-	335	-	-
Total	2	5	1023	2466	0
Available	280	220	106400	53200	0
Utilization (%)	~0	2	~0	4	0

double

Performance

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.232 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
1919	1919	19.190 us	19.190 us	1919	1919	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- FIR	1905	1905	15	-	-	127	no

Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1617	-
FIFO	-	-	-	-	-
Instance	-	14	1174	2372	-
Memory	4	-	0	0	0
Multiplexer	-	-	-	236	-
Register	-	-	531	-	-
Total	4	14	1705	4225	0
Available	280	220	106400	53200	0
Utilization (%)	1	6	1	7	0

這邊的 critical path 是由一個乘法及一個加法的產生的。因此 data type 以及他的寬度都會造成速度上的差異。
在相同都是 integer 表示式的 int (32 bit)、long long (64 bit)、AP best (8 bit)裡面。Int 與 long long 之間是沒有差異的，主要應該是因為後面的 bit 都已經變成 redundant bit 了，因此 design compiler 已經自動將 64 優化成與 32 bit 相同的電路了，因此在 latency 以及 utility 上都沒有差別。
而因為 AP best 已經是最小可求出正確解的大小了，因此運算不需要使用到 32 bit，只需要 8 bit 就可以了。使用的電路就可以進一步優化。可以看到 8 bit 的 MAC 運算只需要 3 個 cycle 就能完成，導致 latency 也可以進步很多。且因為 register 只需要用到 8 bit 去存每個值，因此所需的 FF 以及 LUT 都變小了。同時要考量到一些 design compiler 做的電路優化，FF 的數量並沒有直接除以 4 那麼簡單。
再來是 float 以及 int 之間的比較，可以發現 float 的運算 iteration latency 都上升很多，可以說明 float 的運算確實是比起 integer 還要複雜的。且 FF 以及 LUT 的用量都會上升很多。
Float 與 double 之間的差異只差在一個是 32 bit 一個是 64 bit。可以想見 double 的運算資源會比 float 還要再更多，且 iteration latency 也會比較大。

3. Pipelining

- **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?

#pragma HLS PIPELINE II = N
只要下這個 directive 指令，HLS tool 就會自動幫我們做相對應的 pipeline 優化。

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

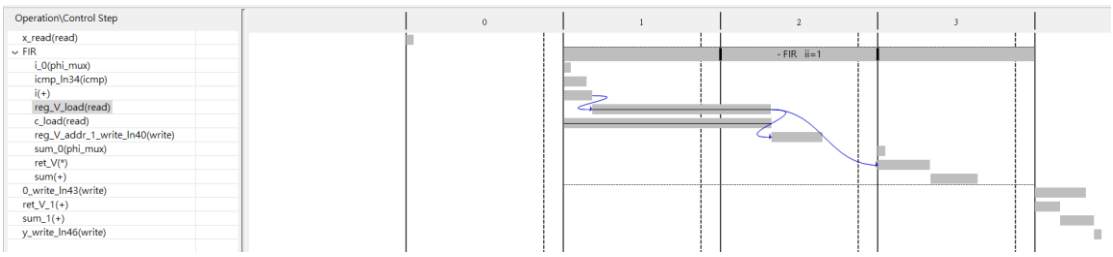
	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- FIR	128	128	3	1	1	127	yes

Utilization Estimates

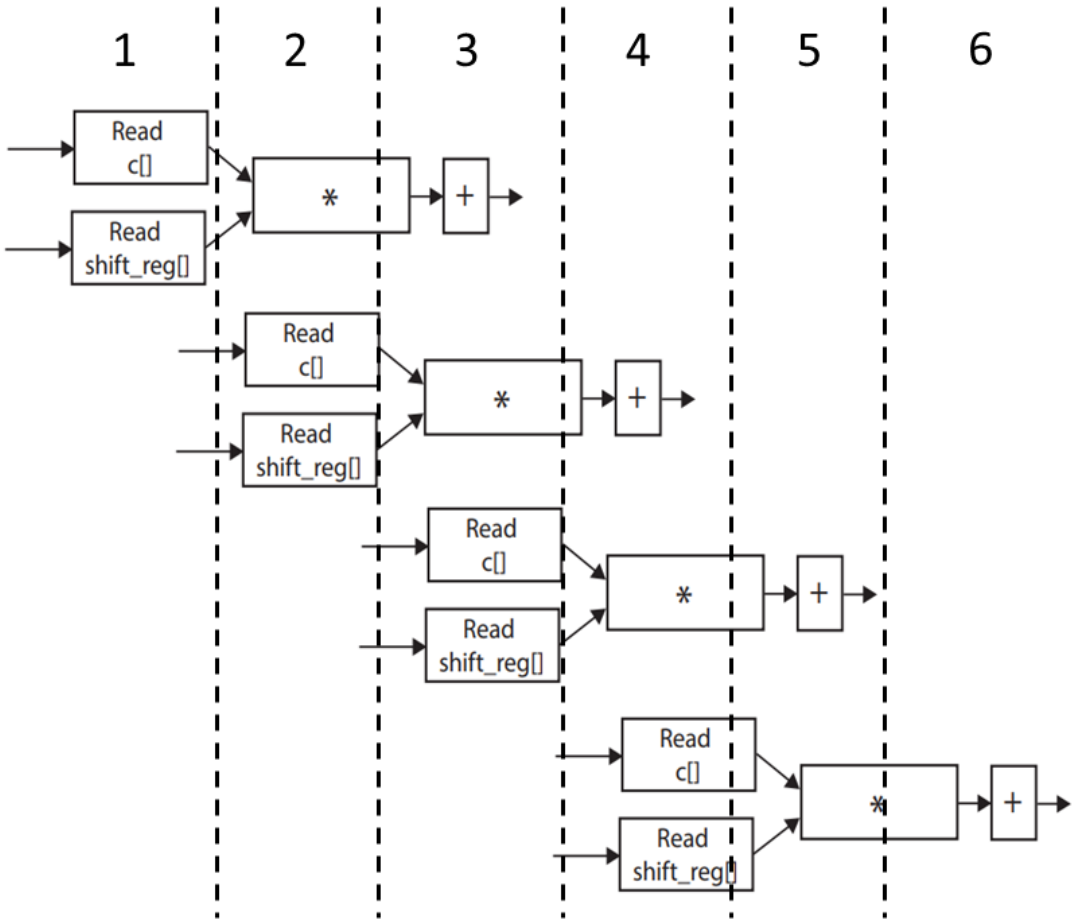
Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	68	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	81	-
Register	-	-	61	-	-
Total	1	1	66	159	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

這是做完 pipeline 且設定 II=1 的結果。可以看到確實可以做到 II=1。latency 則是因為 II=1，trip count 為 127，再加上 Iteration latency。最後可以達到 130 latency 的結果。



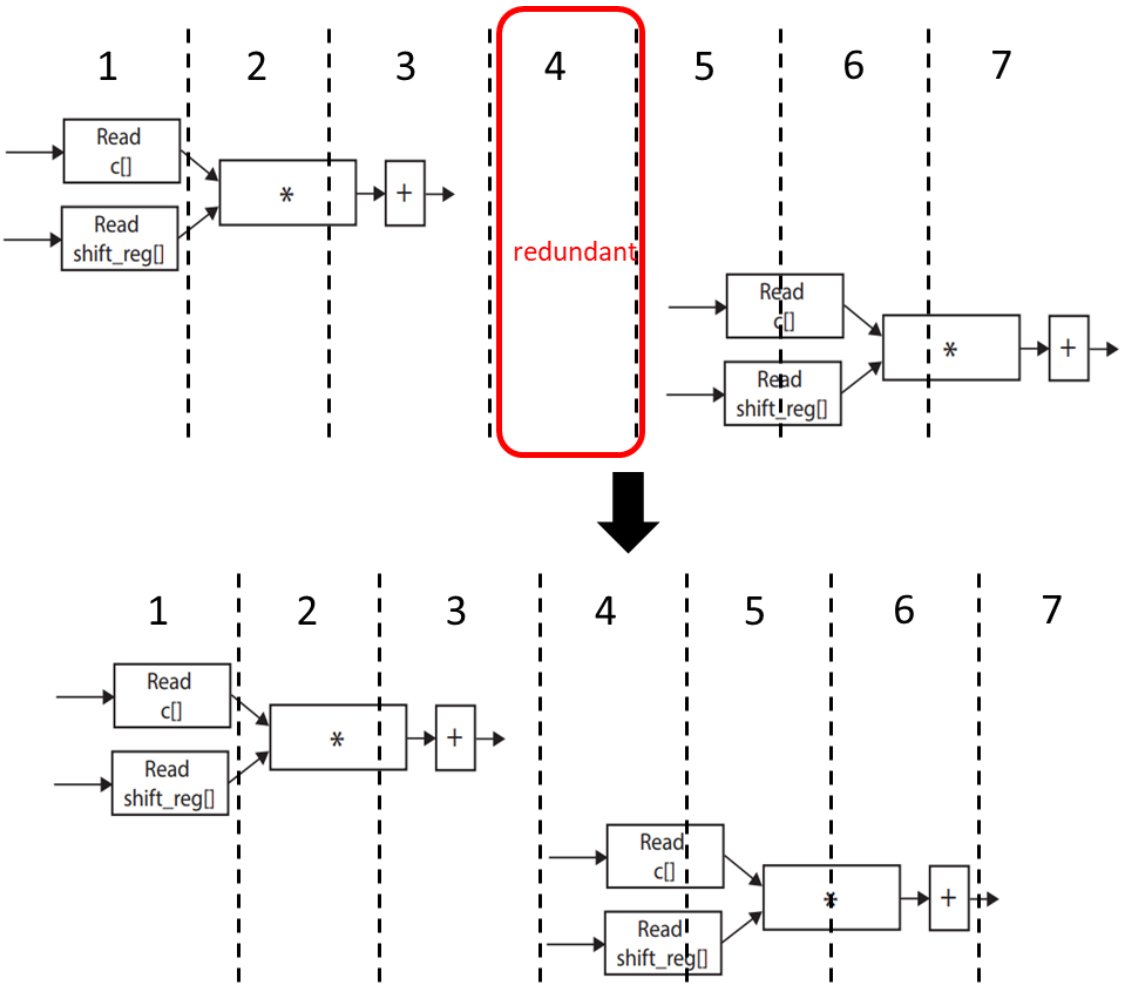
這是 pipeline 後 II=1 的時序圖。一個 loop 做的事情是差不多的。
考量到現在 MAC 只需要 3 個 cycle 就能做完整個運算，且 II=1，每個 iteration 時序圖可以表達成以下。



這樣可以達到很高的 throughput。
若是改變不同的 II value，latency 也會有不同。

	Performance	Utilization
Unpipe	Performance Estimates Timing Summary Clock | Target | Estimated | Uncertainty ap_clk | 10.00 ns | 6.508 ns | 1.25 ns Latency Summary Latency (cycles) min | Latency (cycles) max | Latency (absolute) min | Latency (absolute) max | Interval (cycles) min | Interval (cycles) max | Type 382 | 382 | 3.820 us | 3.820 us | 382 | 382 | none Detail Instance Loop Loop Name | Latency (cycles) min | Latency (cycles) max | Iteration Latency | Initiation Interval achieved | Initiation Interval target | Trip Count | Pipelined - Loop 1 | 381 | 381 | 3 | - | - | 127 | no	Utilization Estimates Summary Name | BRAM_18K | DSP48E | FF | LUT | URAM DSP | - | 1 | - | - | - Expression | - | - | 0 | 64 | - FIFO | - | - | - | - | - Instance | - | - | - | - | - Memory | 1 | - | 5 | 10 | 0 Multiplexer | - | - | - | 81 | - Register | - | - | 57 | - | - Total | 1 | 1 | 62 | 155 | 0 Available | 280 | 220 | 106400 | 53200 | 0 Utilization (%) | ~0 | ~0 | ~0 | ~0 | 0
II=1	Performance Estimates Timing Summary Clock | Target | Estimated | Uncertainty ap_clk | 10.00 ns | 6.508 ns | 1.25 ns Latency Summary Throughput (MHz): 0.76 Latency (cycles) min | Latency (cycles) max | Latency (absolute) min | Latency (absolute) max | Interval (cycles) min | Interval (cycles) max | Type 130 | 130 | 1.300 us | 1.300 us | 130 | 130 | none Detail Instance Loop Loop Name | Latency (cycles) min | Latency (cycles) max | Iteration Latency | Initiation Interval achieved | Initiation Interval target | Trip Count | Pipelined - FIR | 128 | 128 | 3 | 1 | 1 | 127 | yes	Utilization Estimates Summary Name | BRAM_18K | DSP48E | FF | LUT | URAM DSP | - | 1 | - | - | - Expression | - | - | 0 | 68 | - FIFO | - | - | - | - | - Instance | - | - | - | - | - Memory | 1 | - | 5 | 10 | 0 Multiplexer | - | - | - | 81 | - Register | - | - | 61 | - | - Total | 1 | 1 | 66 | 159 | 0 Available | 280 | 220 | 106400 | 53200 | 0 Utilization (%) | ~0 | ~0 | ~0 | ~0 | 0
II=2	Performance Estimates Timing Summary Clock | Target | Estimated | Uncertainty ap_clk | 10.00 ns | 6.508 ns | 1.25 ns Latency Summary Latency (cycles) min | Latency (cycles) max | Latency (absolute) min | Latency (absolute) max | Interval (cycles) min | Interval (cycles) max | Type 256 | 256 | 2.560 us | 2.560 us | 256 | 256 | none Detail Instance Loop Loop Name | Latency (cycles) min | Latency (cycles) max | Iteration Latency | Initiation Interval achieved | Initiation Interval target | Trip Count | Pipelined - FIR | 254 | 254 | 3 | 2 | 2 | 127 | yes	Utilization Estimates Summary Name | BRAM_18K | DSP48E | FF | LUT | URAM DSP | - | 1 | - | - | - Expression | - | - | 0 | 66 | - FIFO | - | - | - | - | - Instance | - | - | - | - | - Memory | 1 | - | 5 | 10 | 0 Multiplexer | - | - | - | 99 | - Register | - | - | 60 | - | - Total | 1 | 1 | 65 | 175 | 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 | 6.508 ns | 1.25 ns Latency Summary Latency (cycles) min | Latency (cycles) max | Latency (absolute) min | Latency (absolute) max | Interval (cycles) min | Interval (cycles) max | Type 383 | 383 | 3.830 us | 3.830 us | 383 | 383 | none Detail Instance Loop Loop Name | Latency (cycles) min | Latency (cycles) max | Iteration Latency | Initiation Interval achieved | Initiation Interval target | Trip Count | Pipelined - FIR | 381 | 381 | 3 | 3 | 3 | 127 | yes	Utilization Estimates Summary Name | BRAM_18K | DSP48E | FF | LUT | URAM DSP | - | 1 | - | - | - Expression | - | - | 0 | 64 | - FIFO | - | - | - | - | - Instance | - | - | - | - | - Memory | 1 | - | 5 | 10 | 0 Multiplexer | - | - | - | 87 | - Register | - | - | 58 | - | - Total | 1 | 1 | 63 | 161 | 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 | 6.508 ns | 1.25 ns Latency Summary Latency (cycles) min | Latency (cycles) max | Latency (absolute) min | Latency (absolute) max | Interval (cycles) min | Interval (cycles) max | Type 383 | 383 | 3.830 us | 3.830 us | 383 | 383 | none Detail Instance Loop Loop Name | Latency (cycles) min | Latency (cycles) max | Iteration Latency | Initiation Interval achieved | Initiation Interval target | Trip Count | Pipelined - FIR | 381 | 381 | 3 | 3 | 4 | 127 | yes	Utilization Estimates Summary Name | BRAM_18K | DSP48E | FF | LUT | URAM DSP | - | 1 | - | - | - Expression | - | - | 0 | 64 | - FIFO | - | - | - | - | - Instance | - | - | - | - | - Memory | 1 | - | 5 | 10 | 0 Multiplexer | - | - | - | 87 | - Register | - | - | 58 | - | - Total | 1 | 1 | 63 | 161 | 0 Available | 280 | 220 | 106400 | 53200 | 0 Utilization (%) | ~0 | ~0 | ~0 | ~0 | 0

從表格比較表來看，從 unpipelined、II=1、II=2、II=3 的結果都蠻合理的。Latency 都是 II * trip count + 2。其中硬體的使用率其實沒有差太多。比較可以注意的是在 II=4 的時候，會發現 initiation interval 的 target=4，但 achieved=3。這是因為現在一個 loop 只需要 3 個 cycle 就可以計算完全，不需要等到第 4 個 cycle。若是故意讓他 II=4，則會有一個 cycle 完全 idle。因此只要做簡單的 cut-set retiming 分析就可以將一整層的 pipeline register 拿掉。



4. Removing Conditional Statements

- **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?

<table>
<tr><th></th><th>w/conditional</th><th>Wo/conditional</th></tr>
<tr><td>Code</td><td>

```
    FIR: for(int i = N-1; i >= 0; i--){
#pragma HLS PIPELINE II=1
        if(i==0)
            reg[i] = (regbit)x;
        else
            reg[i] = reg[i-1];
        *y += c[i] * reg[i];
    }
```

</td><td>

```
    static regbit reg[N] = {0};

    *y = 0;
    FIR: for(int i = N-1; i > 0; i--){
#pragma HLS PIPELINE II=1
//      if(i==0)
//          reg[i] = (regbit)x;
//      else
        reg[i] = reg[i-1];
        *y += c[i] * reg[i];
    }
    reg[0] = (regbit)x;
    *y += c[0] * reg[0];
```

</td></tr>
<tr><td>Perf.</td><td>
Performance Estimates
Timing
Summary
<table>
<tr><th>Clock</th><th>Target</th><th>Estimated</th><th>Uncertainty</th></tr>
<tr><td>ap_clk</td><td>10.00 ns</td><td>6.508 ns</td><td>1.25 ns</td></tr>
</table>
Latency
Summary
Throughput (MHz): 0.38
<table>
<tr><th colspan="2">Latency (cycles)</th><th colspan="2">Latency (absolute)</th><th colspan="2">Interval (cycles)</th><th></th></tr>
<tr><th>min</th><th>max</th><th>min</th><th>max</th><th>min</th><th>max</th><th>Type</th></tr>
<tr><td>260</td><td>260</td><td>2.600 us</td><td>2.600 us</td><td>260</td><td>260</td><td>none</td></tr>
</table>
Detail
Instance
Loop
<table>
<tr><th></th><th colspan="2">Latency (cycles)</th><th></th><th colspan="2">Initiation Interval</th><th></th><th></th></tr>
<tr><th>Loop Name</th><th>min</th><th>max</th><th>Iteration Latency</th><th>achieved</th><th>target</th><th>Trip Count</th><th>Pipelined</th></tr>
<tr><td>- FIR</td><td>258</td><td>258</td><td>5</td><td>2</td><td>1</td><td>128</td><td>yes</td></tr>
</table>
</td><td>
Performance Estimates
Timing
Summary
<table>
<tr><th>Clock</th><th>Target</th><th>Estimated</th><th>Uncertainty</th></tr>
<tr><td>ap_clk</td><td>10.00 ns</td><td>6.508 ns</td><td>1.25 ns</td></tr>
</table>
Latency
Summary
<table>
<tr><th colspan="2">Latency (cycles)</th><th colspan="2">Latency (absolute)</th><th colspan="2">Interval (cycles)</th><th></th></tr>
<tr><th>min</th><th>max</th><th>min</th><th>max</th><th>min</th><th>max</th><th>Type</th></tr>
<tr><td>130</td><td>130</td><td>1.300 us</td><td>1.300 us</td><td>130</td><td>130</td><td>none</td></tr>
</table>
Detail
Instance
Loop
<table>
<tr><th></th><th colspan="2">Latency (cycles)</th><th></th><th colspan="2">Initiation Interval</th><th></th><th></th></tr>
<tr><th>Loop Name</th><th>min</th><th>max</th><th>Iteration Latency</th><th>achieved</th><th>target</th><th>Trip Count</th><th>Pipelined</th></tr>
<tr><td>- FIR</td><td>128</td><td>128</td><td>3</td><td>1</td><td>1</td><td>127</td><td>yes</td></tr>
</table>
</td></tr>
<tr><td>Util.</td><td>
Utilization Estimates
Summary
<table>
<tr><th>Name</th><th>BRAM_18K</th><th>DSP48E</th><th>FF</th><th>LUT</th><th>URAM</th></tr>
<tr><td>DSP</td><td>-</td><td>1</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Expression</td><td>-</td><td>-</td><td>0</td><td>30</td><td>-</td></tr>
<tr><td>FIFO</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Instance</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Memory</td><td>1</td><td>-</td><td>5</td><td>10</td><td>0</td></tr>
<tr><td>Multiplexer</td><td>-</td><td>-</td><td>-</td><td>123</td><td>-</td></tr>
<tr><td>Register</td><td>-</td><td>-</td><td>66</td><td>-</td><td>-</td></tr>
<tr><td>Total</td><td>1</td><td>1</td><td>71</td><td>163</td><td>0</td></tr>
<tr><td>Available</td><td>280</td><td>220</td><td>106400</td><td>53200</td><td>0</td></tr>
<tr><td>Utilization (%)</td><td>~0</td><td>~0</td><td>~0</td><td>~0</td><td>0</td></tr>
</table>
</td><td>
Utilization Estimates
Summary
<table>
<tr><th>Name</th><th>BRAM_18K</th><th>DSP48E</th><th>FF</th><th>LUT</th><th>URAM</th></tr>
<tr><td>DSP</td><td>-</td><td>1</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Expression</td><td>-</td><td>-</td><td>0</td><td>68</td><td>-</td></tr>
<tr><td>FIFO</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Instance</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>Memory</td><td>1</td><td>-</td><td>5</td><td>10</td><td>0</td></tr>
<tr><td>Multiplexer</td><td>-</td><td>-</td><td>-</td><td>81</td><td>-</td></tr>
<tr><td>Register</td><td>-</td><td>-</td><td>61</td><td>-</td><td>-</td></tr>
<tr><td>Total</td><td>1</td><td>1</td><td>66</td><td>159</td><td>0</td></tr>
<tr><td>Available</td><td>280</td><td>220</td><td>106400</td><td>53200</td><td>0</td></tr>
<tr><td>Utilization (%)</td><td>~0</td><td>~0</td><td>~0</td><td>~0</td><td>0</td></tr>
</table>
</td></tr>
</table>

為了加上 conditional statement，我將我的 code 改寫 for loop 的條件。
加上了 if/else 之後，在每一次進入迴圈，都需要做一次 if 判斷式的比較，造成有可能會需要 branch。而就是因為有 branch 的可能，II 就無法壓在 1。且因為每次迴圈都要判斷 if statement，因此 iteration latency 就會比起原本的 3 上升為 5。多出來的 2 一個會是 if 判斷、一個會是 read/write 運算。造成整體的 latency 變成 260。若是將 code 寫成不需要 if 判斷式的形式，還可以將 FF 以及 LUT 的使用率下降一些。

5. Loop Partitioning

- **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?

為了要做到 loop partition (loop unmerge)，我將原本的 code 改寫成兩個迴圈的形式。兩個 for loop 分別負責 shift register 運算以及 FIR forward。

```
    shift_reg: for(int i = N-1; i > 0; i--){
#pragma HLS PIPELINE II=1
        reg[i] = reg[i-1];
    }
    reg[0] = (regbit)x;
    forward: for(int i = 0; i < N; i++){
#pragma HLS PIPELINE II=1
        *y += c[i] * reg[i];
    }
}
```

Unmerge

Performance

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
- shift_reg	127	127	2	1	1	127	yes
- forward	129	129	3	1	1	128	yes

Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	62	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	147	-
Register	-	-	67	-	-
Total	1	1	72	219	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

Unmerge+unroll

Performance

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	9.871 ns	1.25 ns

Latency

Throughput (MHz): 0.79

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
126	126	1.260 us	1.260 us	126	126	none

Detail

Instance

N/A

Loop

N/A

Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	24	-	-	-
Expression	-	-	0	3508	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	0	0	0
Multiplexer	-	-	-	1726	-
Register	-	-	1575	-	-
Total	1	24	1575	5234	0
Available	280	220	106400	53200	0
Utilization (%)	~0	10	1	9	0

Merged

Performance

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
- FIR	128	128	3	1	1	127	yes

Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	68	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	81	-
Register	-	-	61	-	-
Total	1	1	66	159	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

Merged+unroll

Performance

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
126	126	1.260 us	1.260 us	126	126	none

Detail

Instance

N/A

Loop

N/A

Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	24	-	-	-
Expression	-	-	0	3496	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	0	0	0
Multiplexer	-	-	-	1726	-
Register	-	-	1575	-	-
Total	1	24	1575	5222	0
Available	280	220	106400	53200	0
Utilization (%)	~0	10	1	9	0

Merged+unroll=2

Performance

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
132	132	1.320 us	1.320 us	132	132	none

Detail

Instance

Loop

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

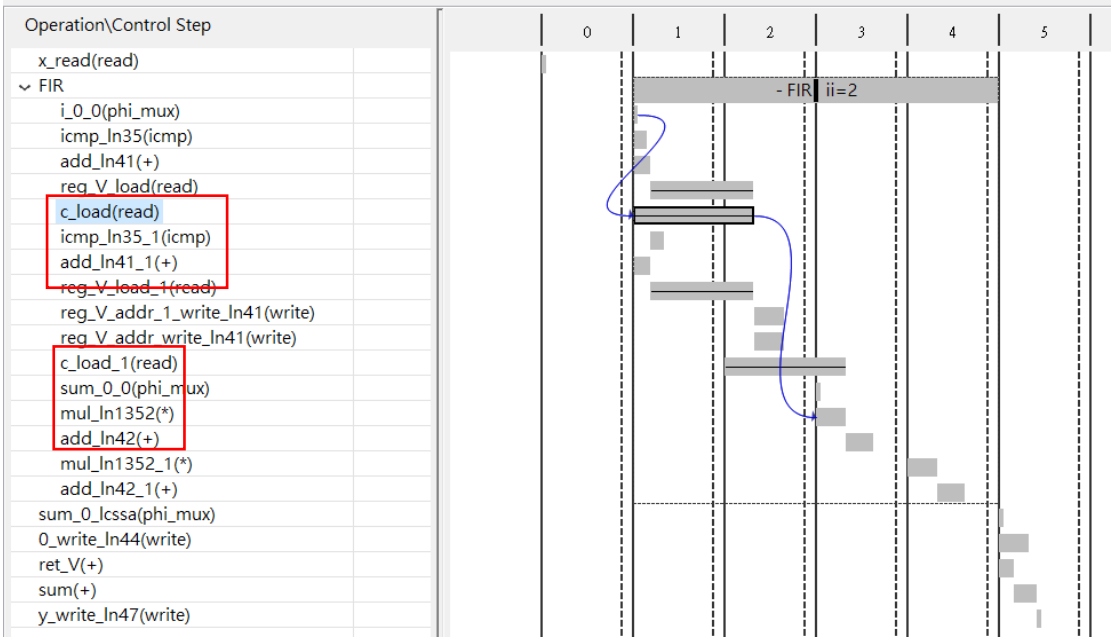
Utilization

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	-	0	109	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	1	-	5	10	0
Multiplexer	-	-	-	138	-
Register	-	-	166	-	-
Total	1	2	171	257	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

若是 loop unmerge 後沒有做任何的優化，HLS tool 會將兩個 loop 分開來做。因此就算兩個 loop 的 II 皆為 1，但整體 latency 還是會有 260。
因此將 unmerge 加上 unroll 的優化之後，會發現 HLS tool 已經將 loop 以及 instance 全部拆掉，變成一大堆的 FF 以及 LUT。如此一來確實能將 latency 降為 126。我再次嘗試將 merged loop 做 unroll 的優化之後，tool 自動將 loop 及 instance 拆掉，一樣是可以達到 latency 126。整體來說，有做 loop unmerge 或是 loop merge 看起來差異不大。
我另外嘗試了 loop merged + unroll factor = 2，也就是硬體會複製 2 份。可以發現到 II 只能變為 2，但因為 trip count 降低為 64 了因此整體 latency 並沒有太大差別。若是沒有將 data partition 成我們想要的方式的話，就算複製再多的硬體都沒辦法幫助我們繼續降低 latency。



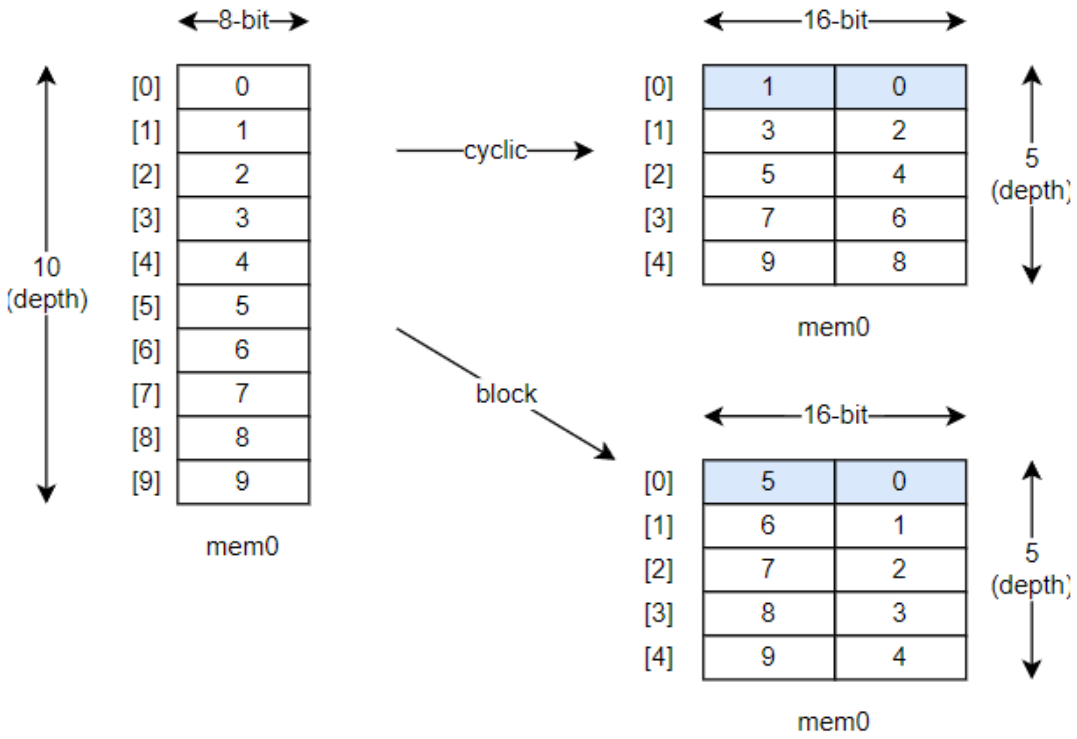
可以看到確實將硬體複製兩份了，但是卻無法成功平行運算。

6. Memory Partition

- **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?

經過實驗之後，只有設 directive 成 Cyclic ARRAY_RESHAPE factor = 64，才能將 II 達到 1。並且讓 latency 降低大約一半。



如此的資料排列，HLS 才能夠順利的將資料讀取進來。達成 II=1。

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	39.629 ns	1.25 ns

Latency

Summary

Throughput (MHz): 1.47

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
68	68	2.695 us	2.695 us	68	68	none

Detail

Instance

Loop

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

Utilization Estimates

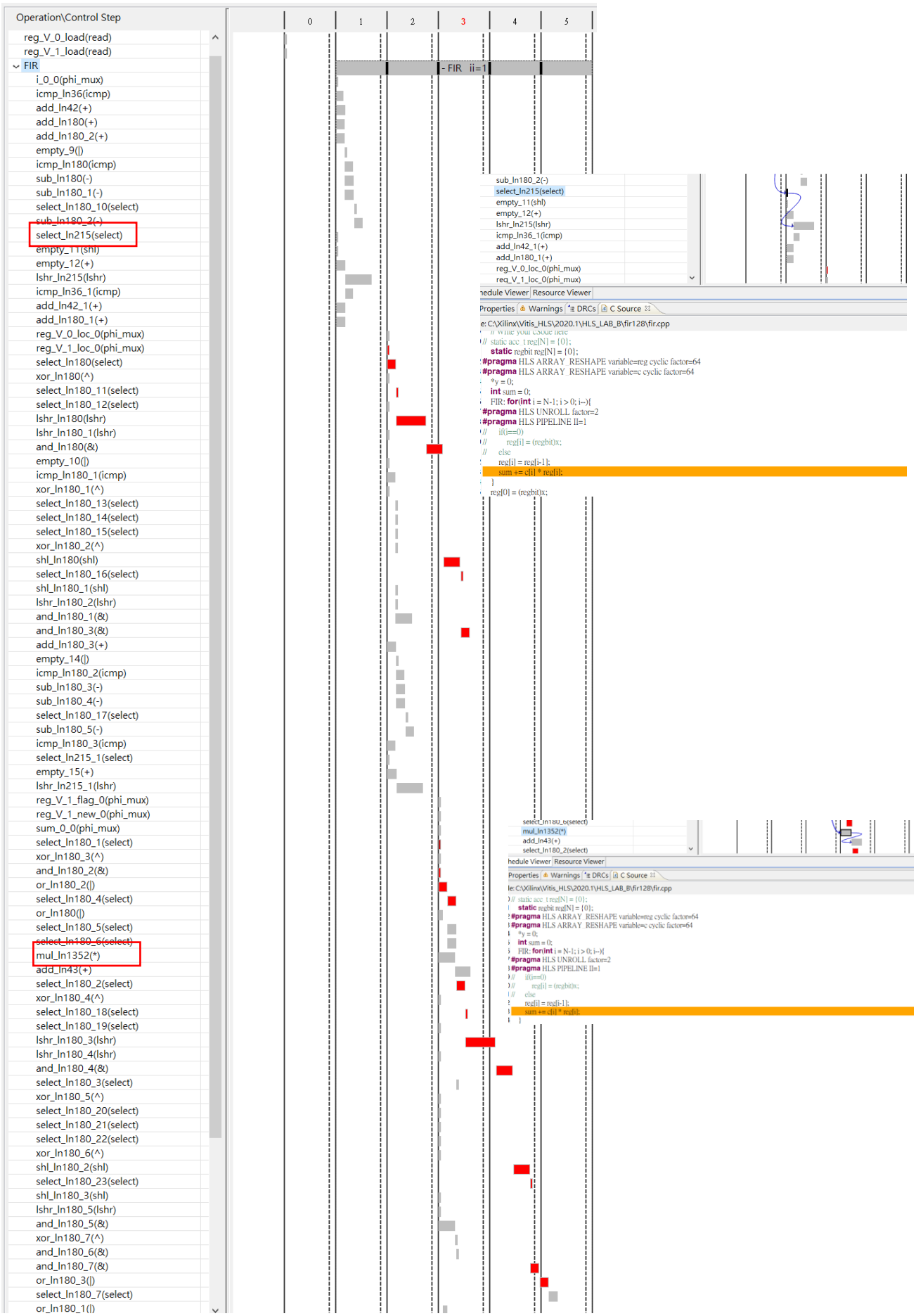
Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	-	0	35516	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	0	-	10	10	-
Multiplexer	-	-	-	129	-
Register	-	-	4779	-	-
Total	0	2	4789	35655	0
Available	280	220	106400	53200	0
Utilization (%)	0	~0	4	67	0

Latency 可以達到 68，換來的代價就是 FF 及 LUT 的用量又變得更大了。

但同時看到 estimated clock 會需要 39.629ns，這邊有機會造成 timing violation。

從時序圖可以看到 HLS tool 已經將電路拆分得很細了，已經很難看懂內部是怎麼做的了。但是還是可以盡量去看到說大致上還是將電路複製成兩份(如圖中兩個乘法器位子)。並且可以同時運算。


Operation\Control Step
0
1
2
3
4
5
reg_V_0_load(read)
reg_V_1_load(read)
FIR
i_0_0(phi_mux)
icmp_ln36(icmp)
add_ln42(+)
add_ln180(+)
add_ln180_2(+)
empty_9(|)
icmp_ln180(icmp)
sub_ln180(-)
sub_ln180_1(-)
select_ln180_10(select)
sub_ln180_2(-)
select_ln215(select)
empty_11(shl)
empty_12(+)
lshr_ln215(lshr)
icmp_ln36_1(icmp)
add_ln42_1(+)
add_ln180_1(+)
reg_V_0_loc_0(phi_mux)
reg_V_1_loc_0(phi_mux)
select_ln180(select)
xor_ln180(^)
select_ln180_11(select)
select_ln180_12(select)
lshr_ln180(lshr)
lshr_ln180_1(lshr)
and_ln180(&)
empty_10(|)
icmp_ln180_1(icmp)
xor_ln180_1(^)
select_ln180_13(select)
select_ln180_14(select)
select_ln180_15(select)
xor_ln180_2(^)
shl_ln180(shl)
select_ln180_16(select)
shl_ln180_1(shl)
lshr_ln180_2(lshr)
and_ln180_1(&)
and_ln180_3(&)
add_ln180_3(+)
empty_14(|)
icmp_ln180_2(icmp)
sub_ln180_3(-)
sub_ln180_4(-)
select_ln180_17(select)
sub_ln180_5(-)
icmp_ln180_3(icmp)
select_ln215_1(select)
empty_15(+)
lshr_ln215_1(lshr)
reg_V_1_flag_0(phi_mux)
reg_V_1_new_0(phi_mux)
sum_0_0(phi_mux)
select_ln180_1(select)
xor_ln180_3(^)
and_ln180_2(&)
or_ln180_2(|)
select_ln180_4(select)
or_ln180(|)
select_ln180_5(select)
select_ln180_6(select)
mul_ln1352(*)
add_ln43(+)
select_ln180_2(select)
xor_ln180_4(^)
select_ln180_18(select)
select_ln180_19(select)
lshr_ln180_3(lshr)
lshr_ln180_4(lshr)
and_ln180_4(&)
select_ln180_3(select)
xor_ln180_5(^)
select_ln180_20(select)
select_ln180_21(select)
select_ln180_22(select)
xor_ln180_6(^)
shl_ln180_2(shl)
select_ln180_23(select)
shl_ln180_3(shl)
lshr_ln180_5(lshr)
and_ln180_5(&)
xor_ln180_7(^)
and_ln180_6(&)
and_ln180_7(&)
or_ln180_3(|)
select_ln180_7(select)
or_ln180_1(|)
- FIR ii=1
sub_ln180_2(-)
select_ln215(select)
empty_11(shl)
empty_12(+)
lshr_ln215(lshr)
icmp_ln36_1(icmp)
add_ln42_1(+)
add_ln180_1(+)
reg_V_0_loc_0(phi_mux)
reg_V_1_loc_0(phi_mux)
Properties
Warnings
DRCs
C Source
e:C:\Xilinx\Vitis_HLS\2020.1\HLS_LAB_B\fir128\fir.cpp
// static acc_t reg[N] = {0};
static regbit reg[N] = {0};
#pragma HLS ARRAY_RESHAPE variable=reg cyclic factor=64
#pragma HLS ARRAY_RESHAPE variable=c cyclic factor=64
*y = 0;
int sum = 0;
FIR: for(int i = N-1; i > 0; i--){
#pragma HLS UNROLL factor=2
#pragma HLS PIPELINE II=1
// if(i==0)
// reg[i] = (regbit)x;
// else
reg[i] = reg[i-1];
sum += c[i] * reg[i];
}
reg[0] = (regbit)x;
mul_ln1352(*)
add_ln43(+)
select_ln180_2(select)
Properties
Warnings
DRCs
C Source
le:C:\Xilinx\Vitis_HLS\2020.1\HLS_LAB_B\fir128\fir.cpp
static regbit reg[N] = {0};
#pragma HLS ARRAY_RESHAPE variable=reg cyclic factor=64
#pragma HLS ARRAY_RESHAPE variable=c cyclic factor=64
*y = 0;
int sum = 0;
FIR: for(int i = N-1; i > 0; i--){
#pragma HLS UNROLL factor=2
#pragma HLS PIPELINE II=1
// if(i==0)
// reg[i] = (regbit)x;
// else
reg[i] = reg[i-1];
sum += c[i] * reg[i];
}

7. Best Design

- **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).

目前最佳解應該就是上面的優化方式總合起來。將 pipeline II 設為 1、做 loop unroll、以及 array reshape factor=2。Directive 設定如下。

```
    mytype c[N] = {10, 11, 11, 8, 3, -3, -8, -11, -11, -1

    // Write your cSode here
//  static acc_t reg[N] = {0};
    static regbit reg[N] = {0};
#pragma HLS ARRAY_RESHAPE variable=reg cyclic factor=2
#pragma HLS ARRAY_RESHAPE variable=c cyclic factor=2
    *y = 0;
    int sum = 0;
    FIR: for(int i = N-1; i > 0; i--){
#pragma HLS UNROLL
#pragma HLS PIPELINE II=1
//      if(i==0)
//          reg[i] = (regbit)x;
//      else
        reg[i] = reg[i-1];
        sum += c[i] * reg[i];
    }
    reg[0] = (regbit)x;
    sum += (c[0] * reg[0]);

    *y = sum;
```

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	9.871 ns	1.25 ns

Latency

Throughput (MHz): 50

Summary

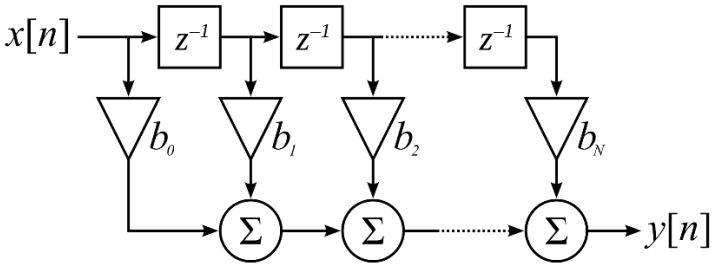
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
2	2	20.000 ns	20.000 ns	2	2	none

Utilization Estimates

Summary

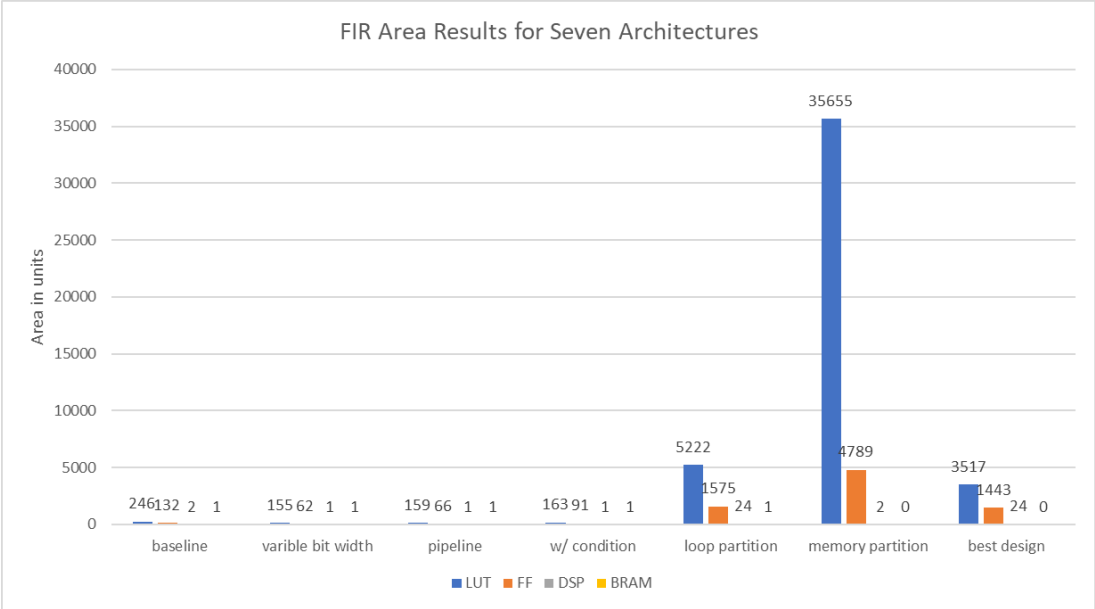
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	24	-	-	-
Expression	-	-	0	3496	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	21	-
Register	-	-	1443	-	-
Total	0	24	1443	3517	0
Available	280	220	106400	53200	0
Utilization (%)	0	10	1	6	0

用這樣的優化設定，可以讓整體 latency 只需要 2。這邊已經將所有的 loop、instance 全部拆解。且 FF 與 LUT 用量算是在合理範圍。我認為這樣子的設定應該是最接近我們一般在電路上設計的 FIR 架構。

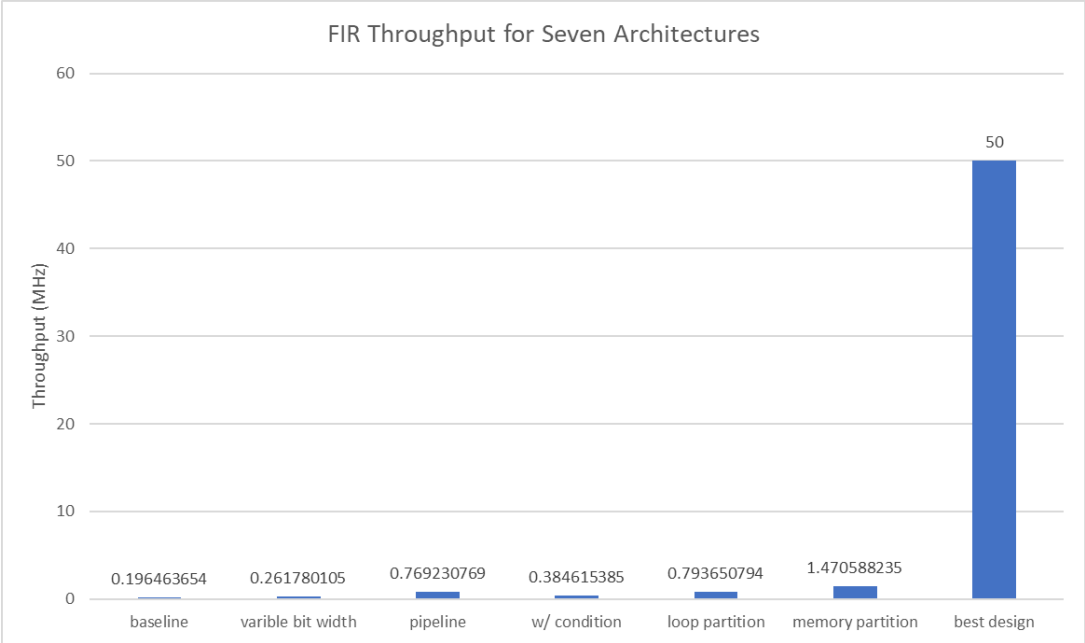


如此的架構應該是每個 cycle 都能夠輸出一個結果，考量到也許需要 data read/write 的 overhead，latency 大約為 2。這才是真正達到最佳平行的設計。在這個架構下，確實可以讓 shift register 全部平行運算(平移)；全部的乘加器同時運算。每次輸入都能輸出一個結果。因此這麼小的 latency 是有機會可以達成的。

FIR Area Results for Seven Architectures



FIR Throughput for Seven Architectures



就前面 4 種方式來說，使用的硬體資源其實都差不多，但 throughput 也不高。前面 4 個來說最好的還是單純 pipeline 是最好的。而就 loop partition 以及 memory partition 來說，loop partition 雖然 throughput 略輸，但是硬體使用卻是比較合理的。而且 loop partition 比起單純 pipeline 還要再好一點。當然其中最好的是 best design，將所有硬體展開；將 memory 切分成理想形式，能夠有最高的 throughput。同時展開方式沒有造成過多的浪費，達成最接近 designer 想像中的 FIR filter。可以有最佳的 throughput 以及合理的硬體使用率。

GitHub link: https://github.com/yuehfeng1114/2022HLS-LAB-B-FIR-design.git