HLS

CH4 : Fundamentals of High Level Synthesis

https://github.com/Arthurddd/HLS_CH4

Outline

Preliminary

Scheduling & Clock period

Loop Pipelining

Loop Unrolling

Loop with conditional bounds

Sequential Loop

Pipeline Feedback

Preliminary

In our presentation, we will go through several concepts and examples in the textbook.

code architecture for demos with “MAIN_LOOP” in text book :

- tester.cpp : testbench
- test.h : header file
- test.cpp
  - top() : main loop in text book
  - test() : the target funciton to be tested

For simplicity, sub() is always set inline to top() in demo.

Scheduling

Scheduling takes the operations described in the DFG and decides when (in which clock cycle) they are performed. It depends on the target device and the clock period.

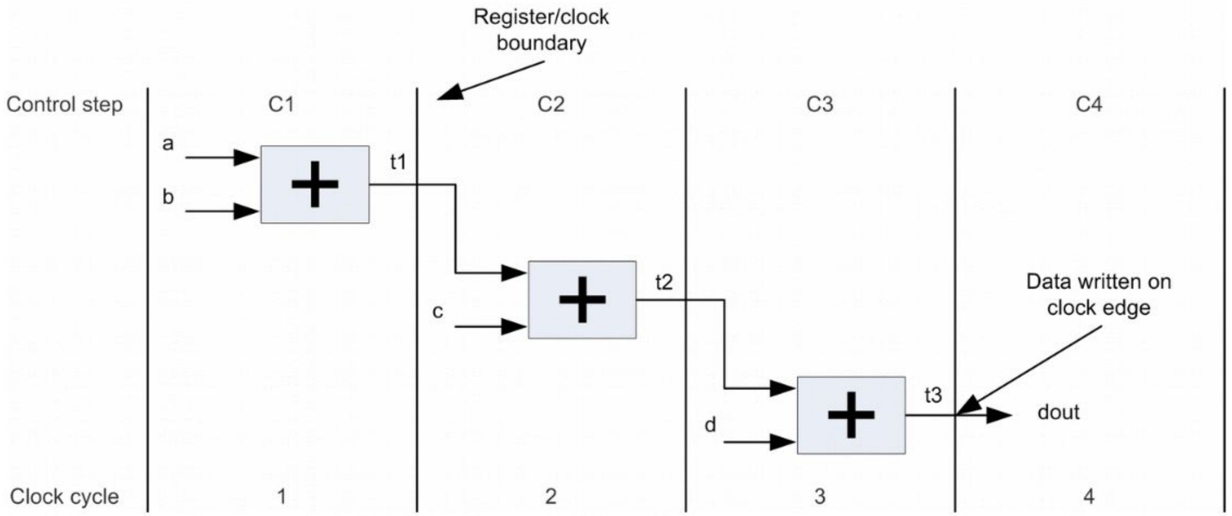


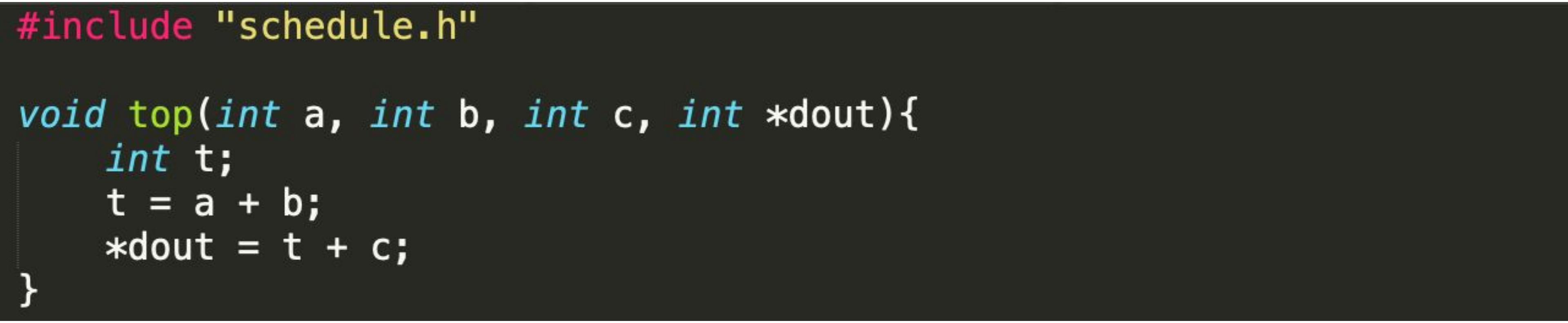
Illustration 19: Scheduled Design

Scheduling with different clock period time

higher clock period : No significant effect, but it is under the risk wasting time to wait for operations about clock edge such as load operations.

lower clock period : The operation is unable to finish in one clock cycle such as multiplication operations.

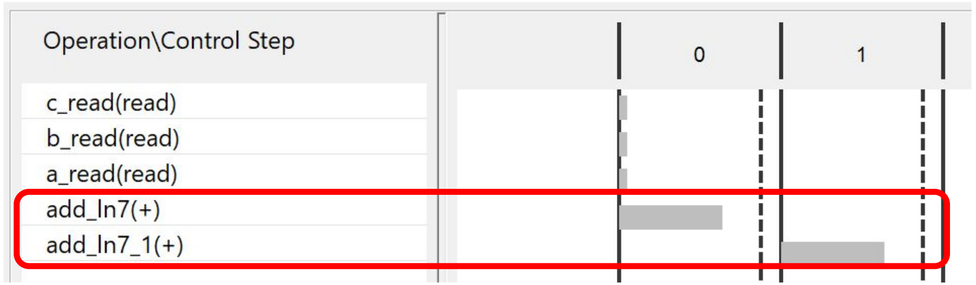
Scheduling under different clock period time demo



```
#include "schedule.h"

void top(int a, int b, int c, int *dout){
    int t;
    t = a + b;
    *dout = t + c;
}
```

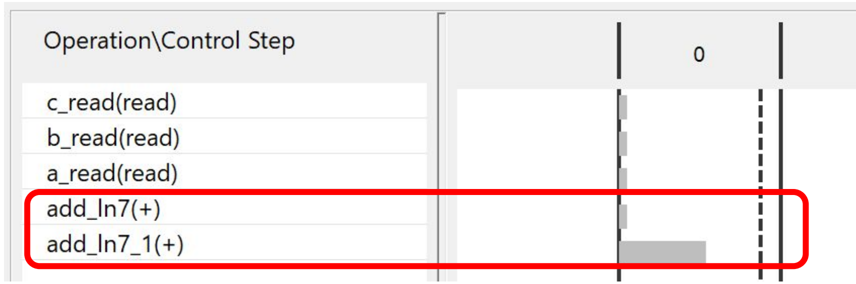
Scheduling with different clock period time demo

Clock period = 4



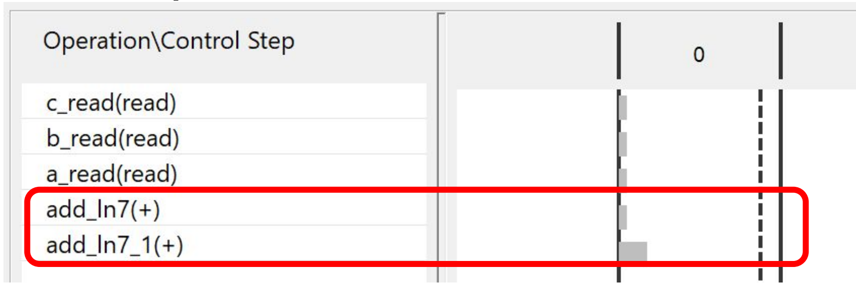
Operation\Control Step	0	1
c_read(read)		
b_read(read)		
a_read(read)		
add_ln7(+)		
add_ln7_1(+)		

Clock period = 8



Operation\Control Step	0
c_read(read)	
b_read(read)	
a_read(read)	
add_ln7(+)	
add_ln7_1(+)	

Clock period = 24



Operation\Control Step	0
c_read(read)	
b_read(read)	
a_read(read)	
add_ln7(+)	
add_ln7_1(+)	

Loop Pipelining

```
#pragma HLS pipeline II=n
```

Loop Pipelining allows a new iteration of a loop to be started before the current iteration has finished.

II : how many cycles are taken before starting the next loop iteration

L : latency, from first input to the first ouput in clock cycles

TP : refers to how often in clock cycles a function call can complete

Loop without Pipelining

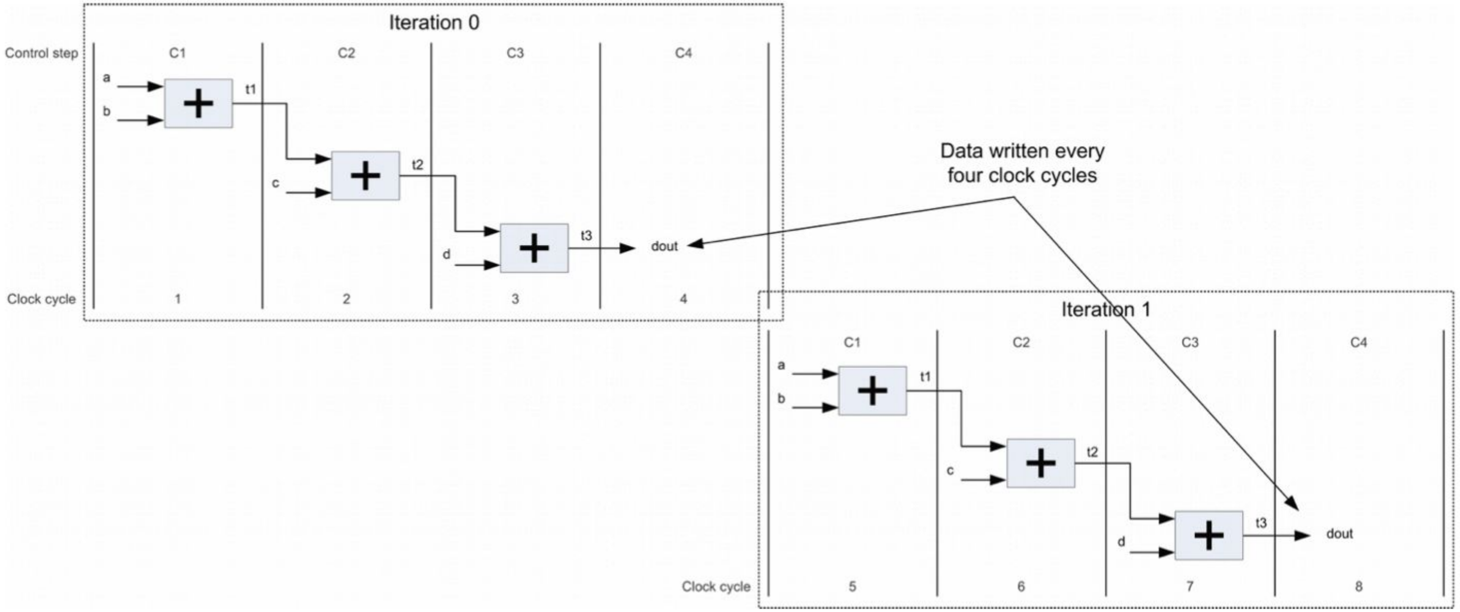


Illustration 23: No Pipelining, L=3, TP=4

Loop Pipelining with II=3

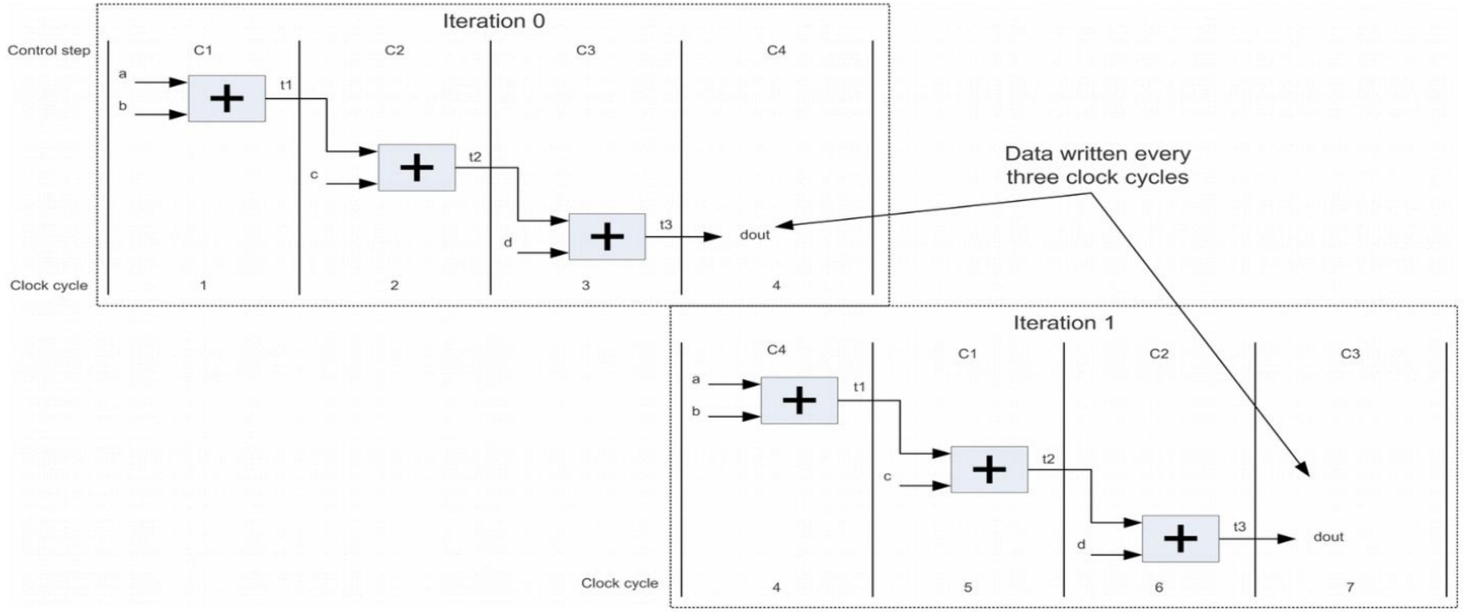


Illustration 24: Pipeline II=3, L=3, TP=3

Loop Pipelining with II=2

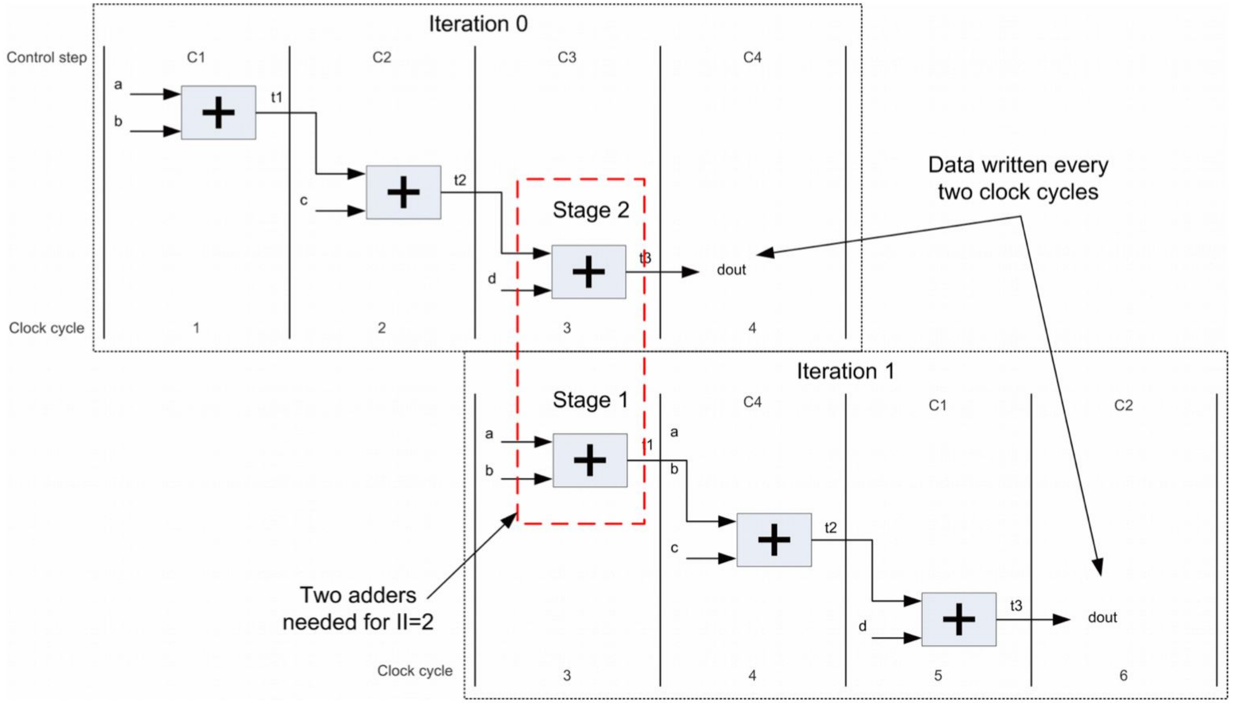


Illustration 25: Pipeline II=2, L=3, TP=2

Loop Pipelining with II=1

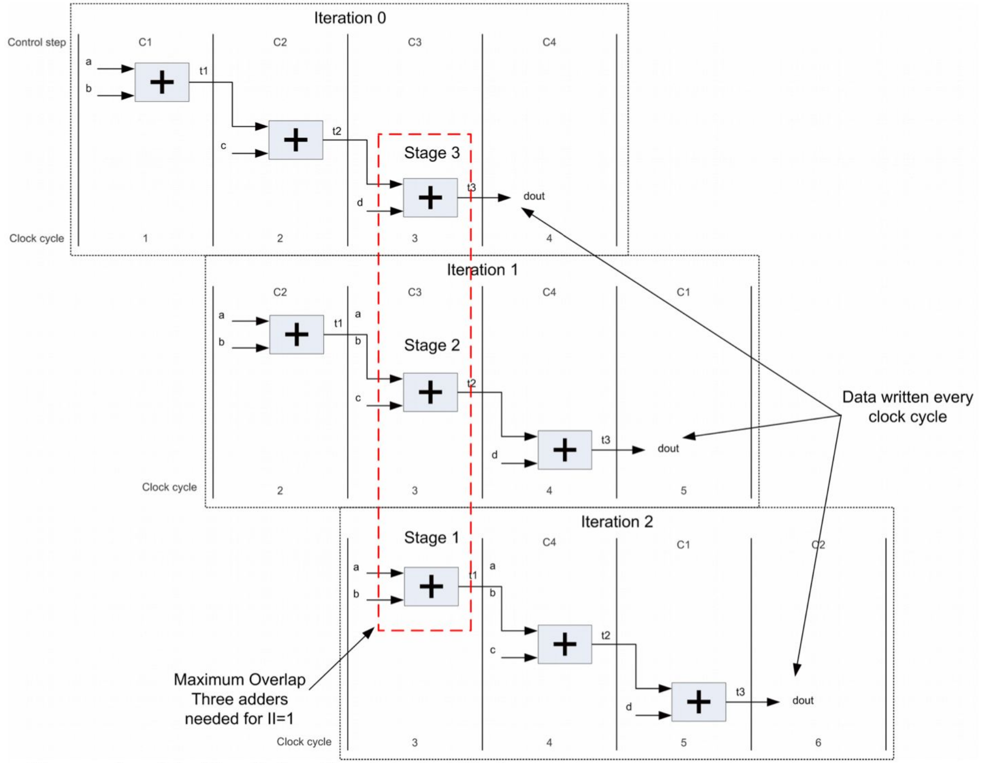
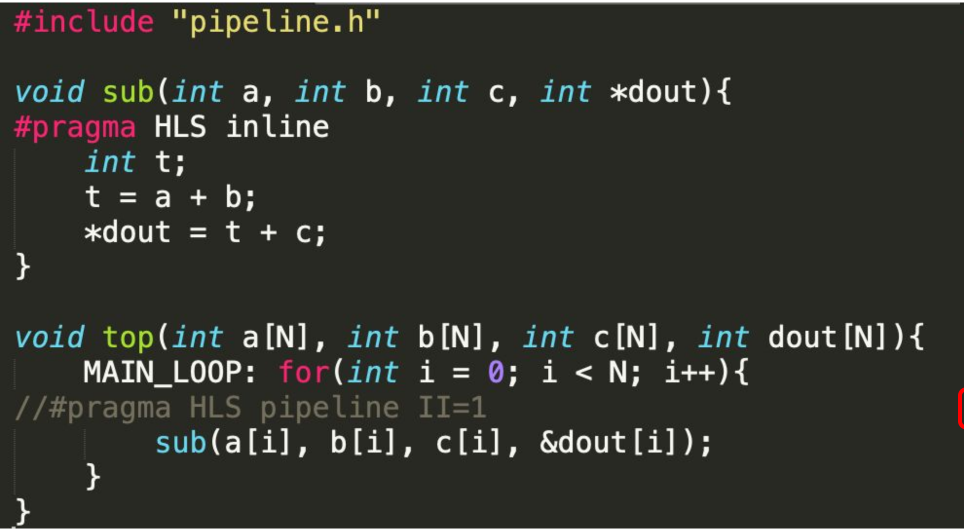
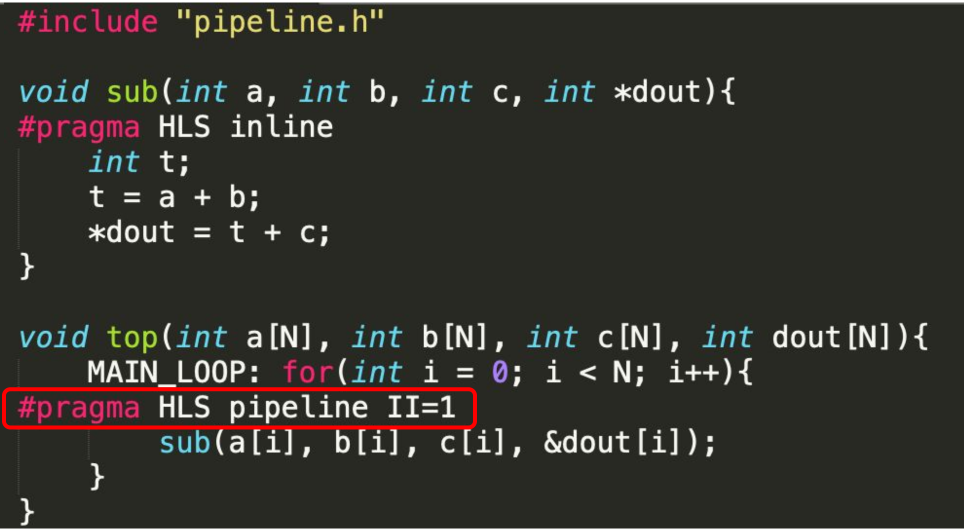


Illustration 26: Pipelining II = 1, L=3, TP=1

Loop Pipelining demo



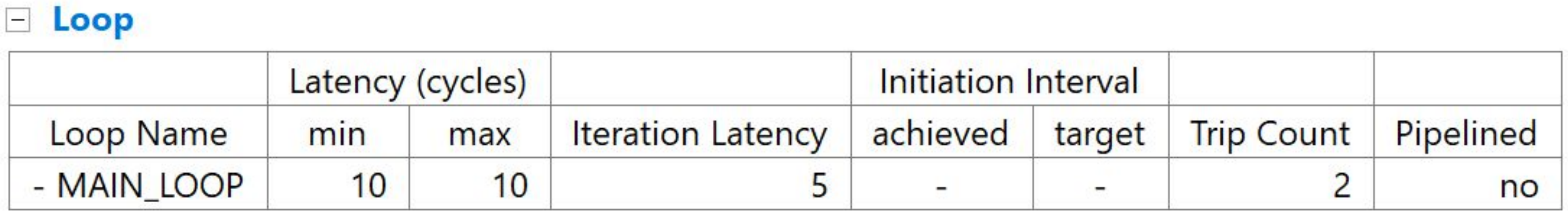
```
#include "pipeline.h"

void sub(int a, int b, int c, int *dout){
#pragma HLS inline
    int t;
    t = a + b;
    *dout = t + c;
}

void top(int a[N], int b[N], int c[N], int dout[N]){
    MAIN_LOOP: for(int i = 0; i < N; i++){
//#pragma HLS pipeline II=1
        sub(a[i], b[i], c[i], &dout[i]);
    }
}
```



```
#include "pipeline.h"

void sub(int a, int b, int c, int *dout){
#pragma HLS inline
    int t;
    t = a + b;
    *dout = t + c;
}

void top(int a[N], int b[N], int c[N], int dout[N]){
    MAIN_LOOP: for(int i = 0; i < N; i++){
#pragma HLS pipeline II=1
        sub(a[i], b[i], c[i], &dout[i]);
    }
}
```

Loop Pipelining demo - latency

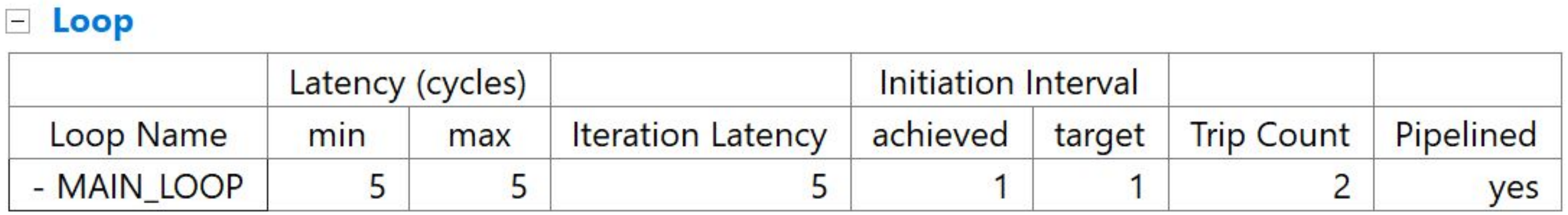
None



Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- MAIN_LOOP	10	10	5	-	-	2	no

II=1

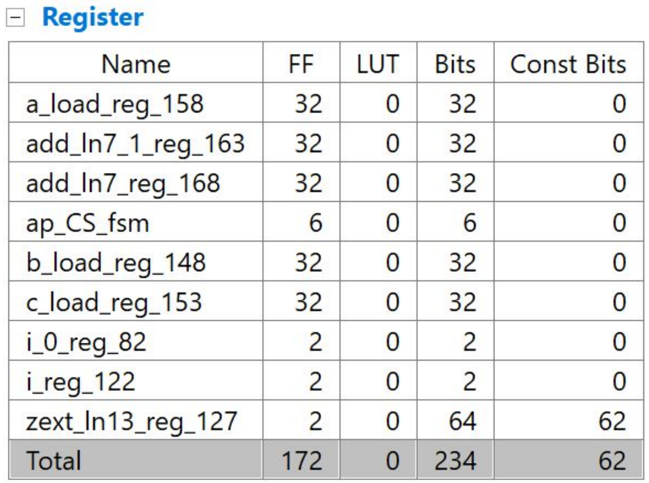


Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- MAIN_LOOP	5	5	5	1	1	2	yes

Loop Pipelining demo - resource

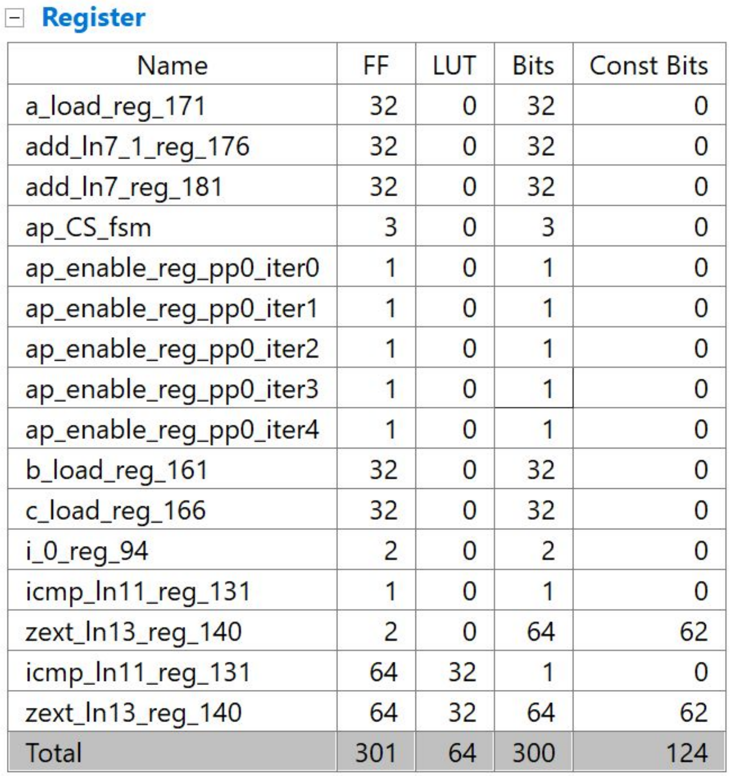
None



Register

Name	FF	LUT	Bits	Const Bits
a_load_reg_158	32	0	32	0
add_ln7_1_reg_163	32	0	32	0
add_ln7_reg_168	32	0	32	0
ap_CS_fsm	6	0	6	0
b_load_reg_148	32	0	32	0
c_load_reg_153	32	0	32	0
i_0_reg_82	2	0	2	0
i_reg_122	2	0	2	0
zext_ln13_reg_127	2	0	64	62
Total	172	0	234	62

II=1



Register

Name	FF	LUT	Bits	Const Bits
a_load_reg_171	32	0	32	0
add_ln7_1_reg_176	32	0	32	0
add_ln7_reg_181	32	0	32	0
ap_CS_fsm	3	0	3	0
ap_enable_reg_pp0_iter0	1	0	1	0
ap_enable_reg_pp0_iter1	1	0	1	0
ap_enable_reg_pp0_iter2	1	0	1	0
ap_enable_reg_pp0_iter3	1	0	1	0
ap_enable_reg_pp0_iter4	1	0	1	0
b_load_reg_161	32	0	32	0
c_load_reg_166	32	0	32	0
i_0_reg_94	2	0	2	0
icmp_ln11_reg_131	1	0	1	0
zext_ln13_reg_140	2	0	64	62
icmp_ln11_reg_131	64	32	1	0
zext_ln13_reg_140	64	32	64	62
Total	301	64	300	124

Loop Unrolling

#pragma HLS unroll factor=n

Loop unrolling is the primary mechanism to add parallelism into a design

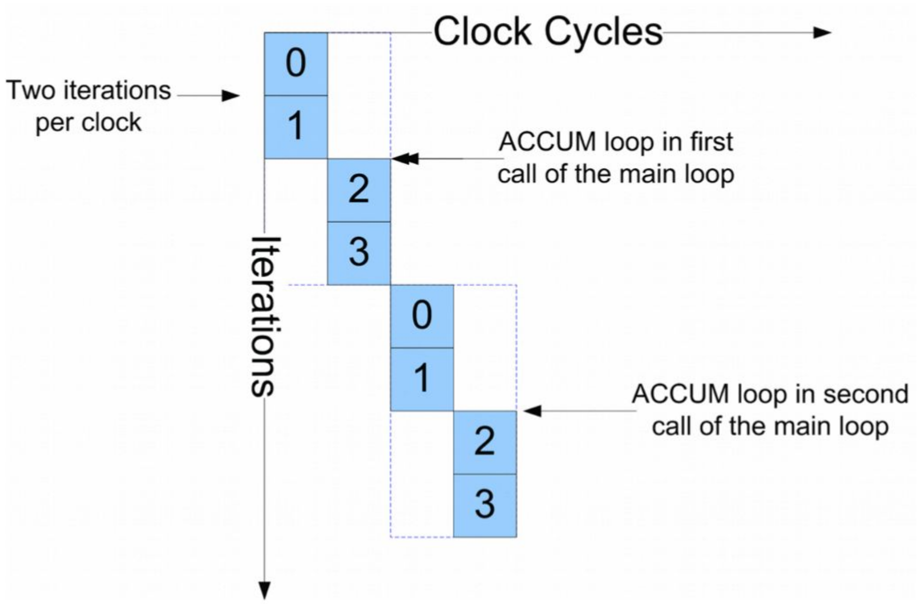
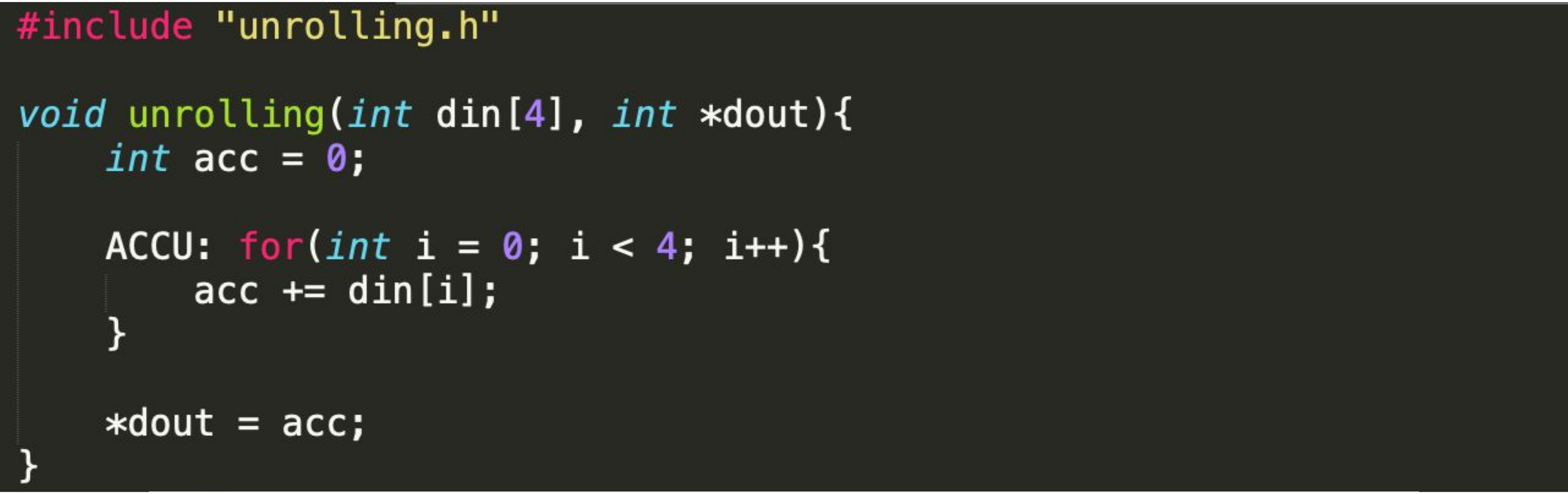
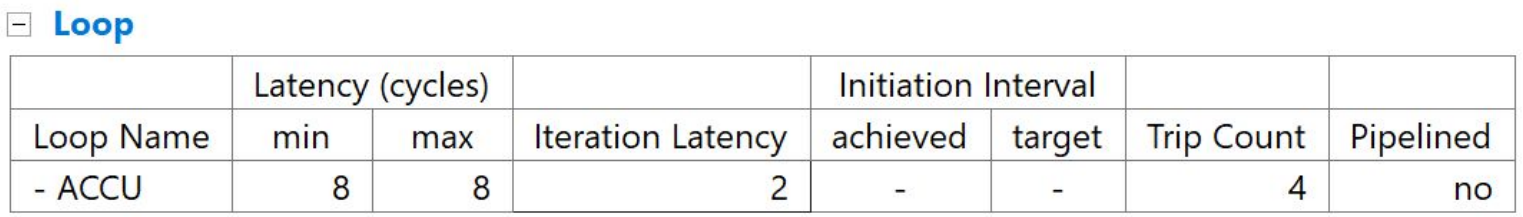

Clock Cycles
Two iterations per clock
0
1
2
3
ACCUM loop in first call of the main loop
Iterations
0
1
2
3
ACCUM loop in second call of the main loop

Illustration 29: Schedule for Accumulate Unroll by 2

Loop without unrolling



```
#include "unrolling.h"

void unrolling(int din[4], int *dout){
    int acc = 0;

    ACCU: for(int i = 0; i < 4; i++){
        acc += din[i];
    }

    *dout = acc;
}
```

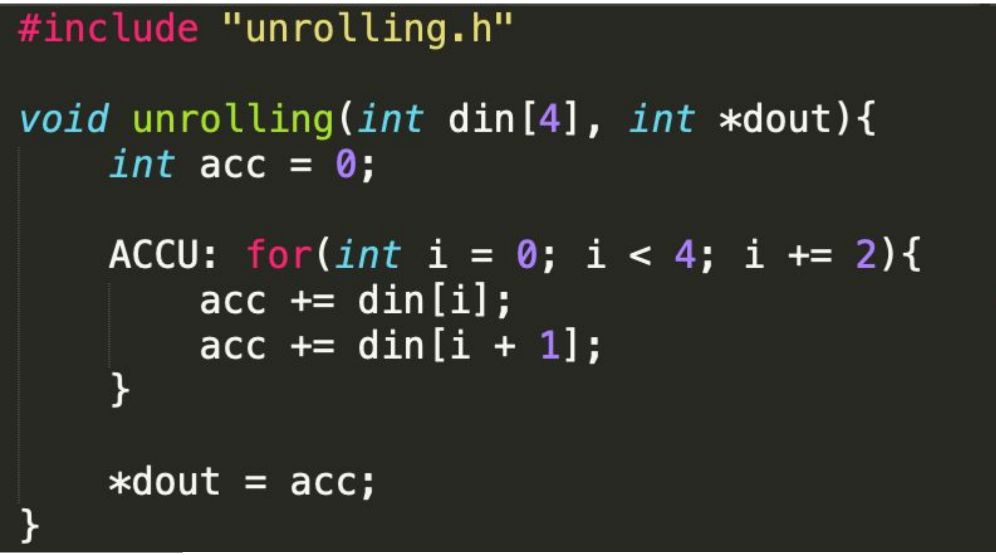


Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- ACCU	8	8	2	-	-	4	no

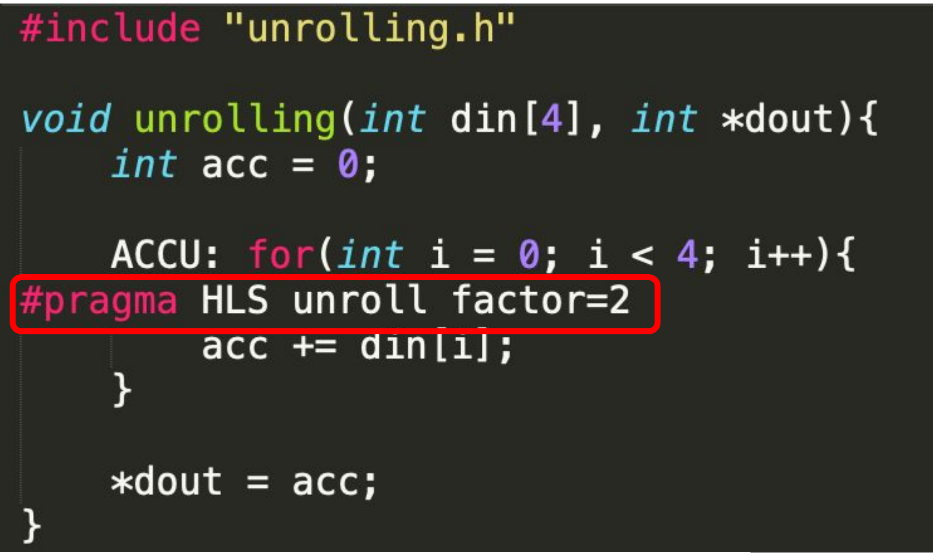
Loop Unrolling with factor=2

manual

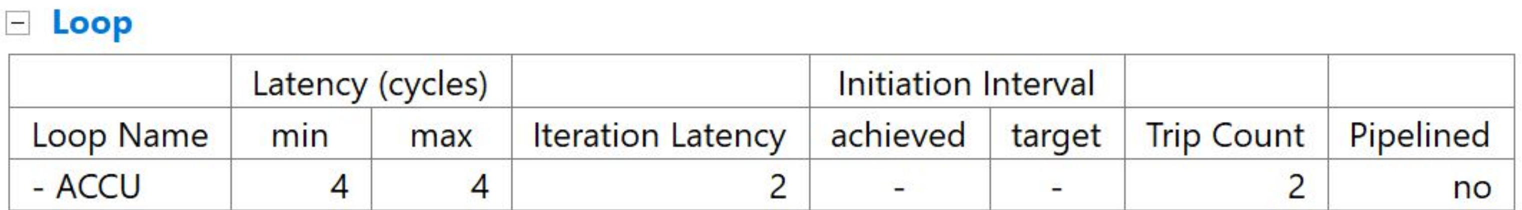


```
#include "unrolling.h"

void unrolling(int din[4], int *dout){
    int acc = 0;

    ACCU: for(int i = 0; i < 4; i += 2){
        acc += din[i];
        acc += din[i + 1];
    }

    *dout = acc;
}
```

pragma



```
#include "unrolling.h"

void unrolling(int din[4], int *dout){
    int acc = 0;

    ACCU: for(int i = 0; i < 4; i++){
#pragma HLS unroll factor=2
        acc += din[i];
    }

    *dout = acc;
}
```

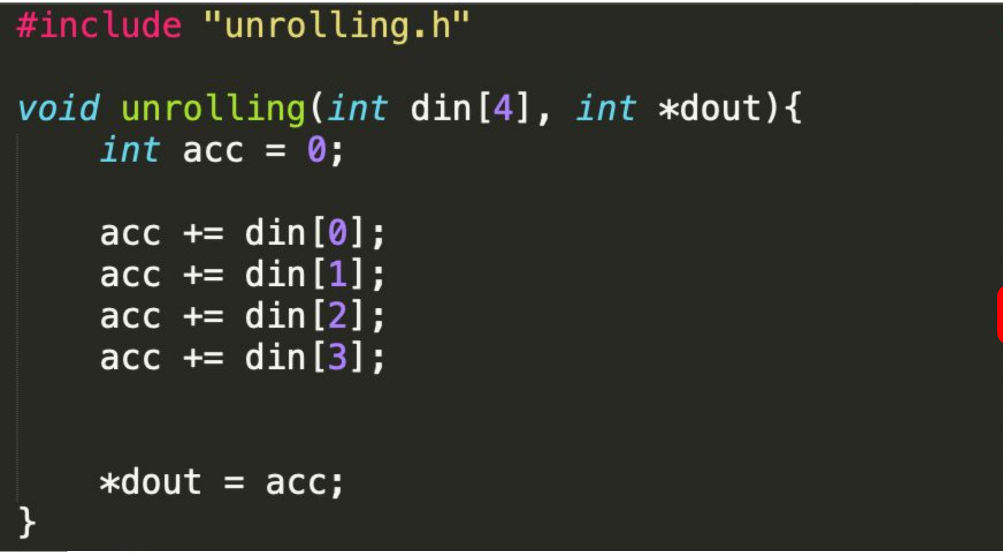


Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- ACCU	4	4	2	-	-	2	no

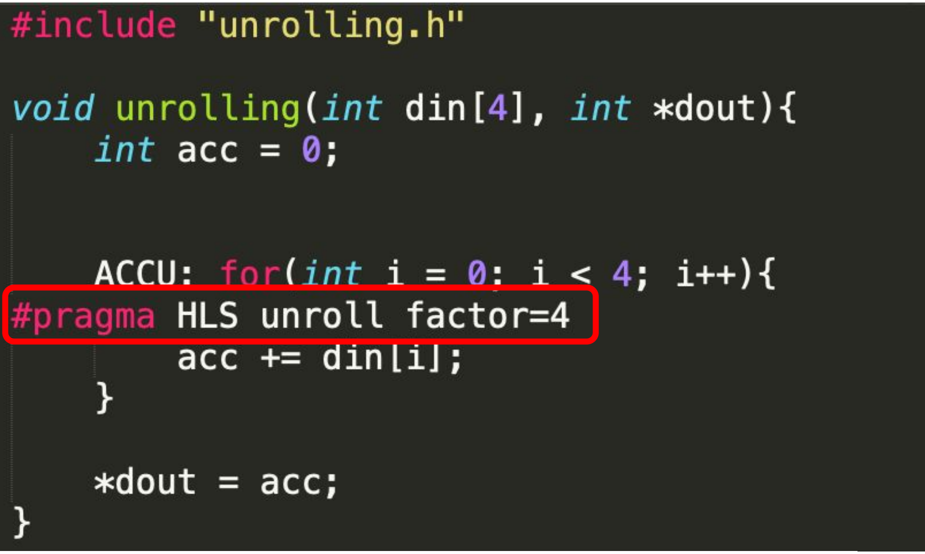
Loop Unrolling manual code with factor=4

manual

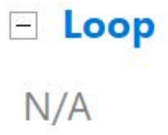


```
#include "unrolling.h"

void unrolling(int din[4], int *dout){
    int acc = 0;

    acc += din[0];
    acc += din[1];
    acc += din[2];
    acc += din[3];


    *dout = acc;
}
```

pragma



```
#include "unrolling.h"

void unrolling(int din[4], int *dout){
    int acc = 0;


    ACCU: for(int i = 0; i < 4; i++){
#pragma HLS unroll factor=4
        acc += din[i];
    }

    *dout = acc;
}
```

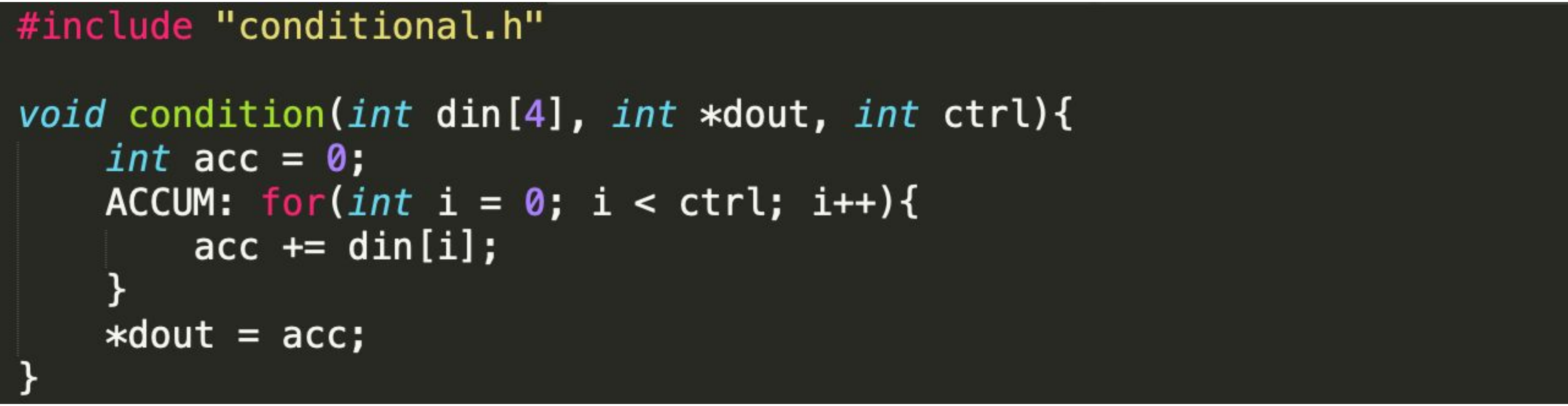


Loop

N/A

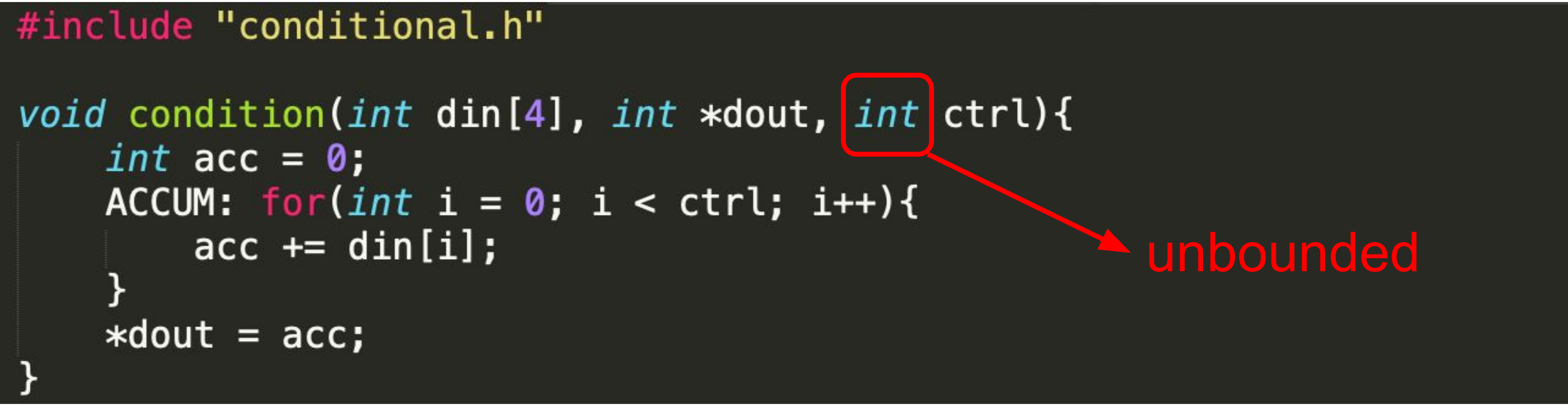
Loop with conditional bound

Sometimes loops are bounded with variable value which could have difficulties in calculating the latency .

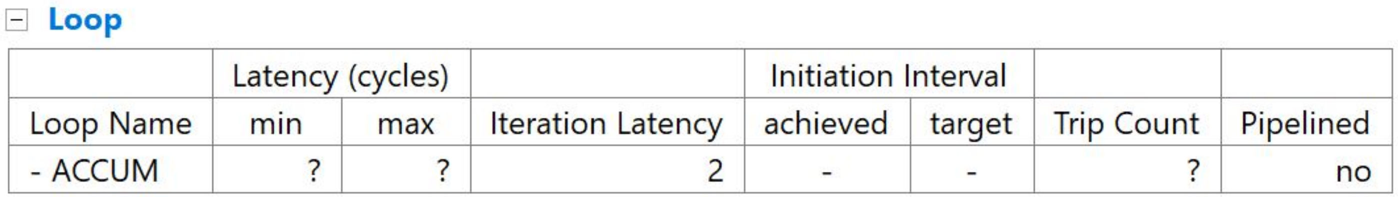


```
#include "conditional.h"

void condition(int din[4], int *dout, int ctrl){
    int acc = 0;
    ACCUM: for(int i = 0; i < ctrl; i++){
        acc += din[i];
    }
    *dout = acc;
}
```

Loops unbounded



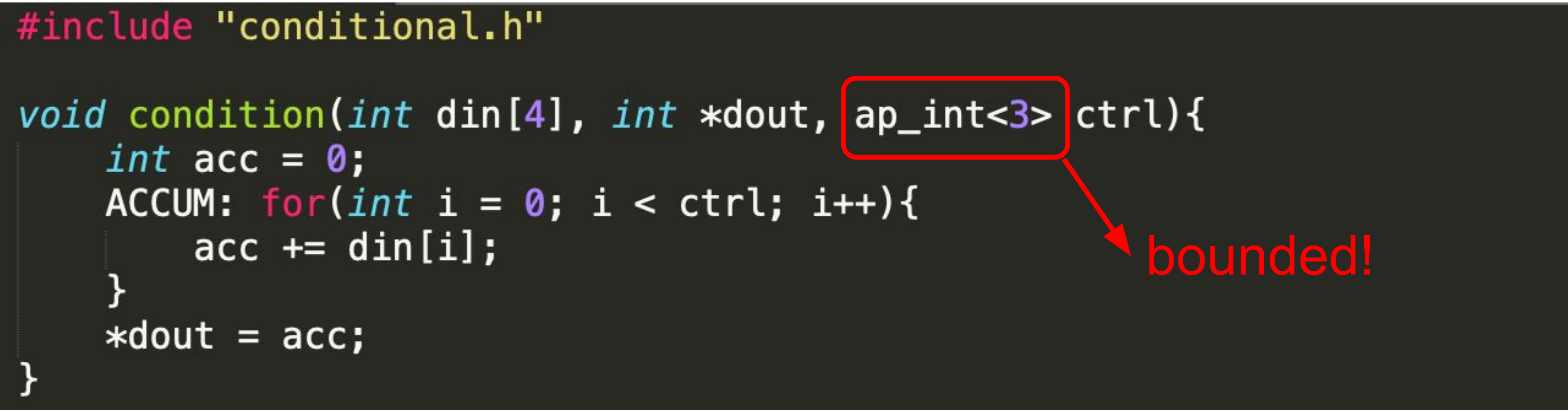
```
#include "conditional.h"

void condition(int din[4], int *dout, int ctrl){
    int acc = 0;
    ACCUM: for(int i = 0; i < ctrl; i++){
        acc += din[i];
    }
    *dout = acc;
}
```



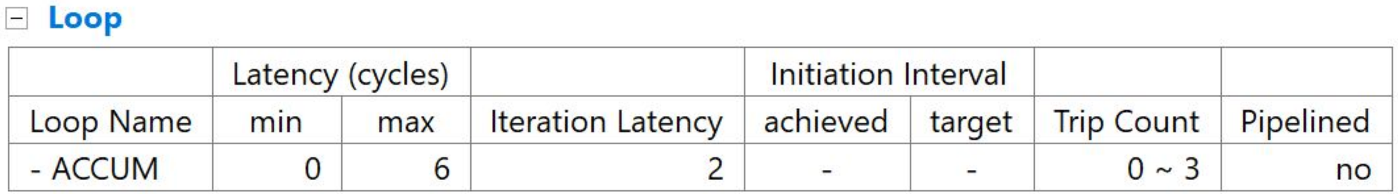
Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- ACCUM	?	?	2	-	-	?	no

Loops bounded



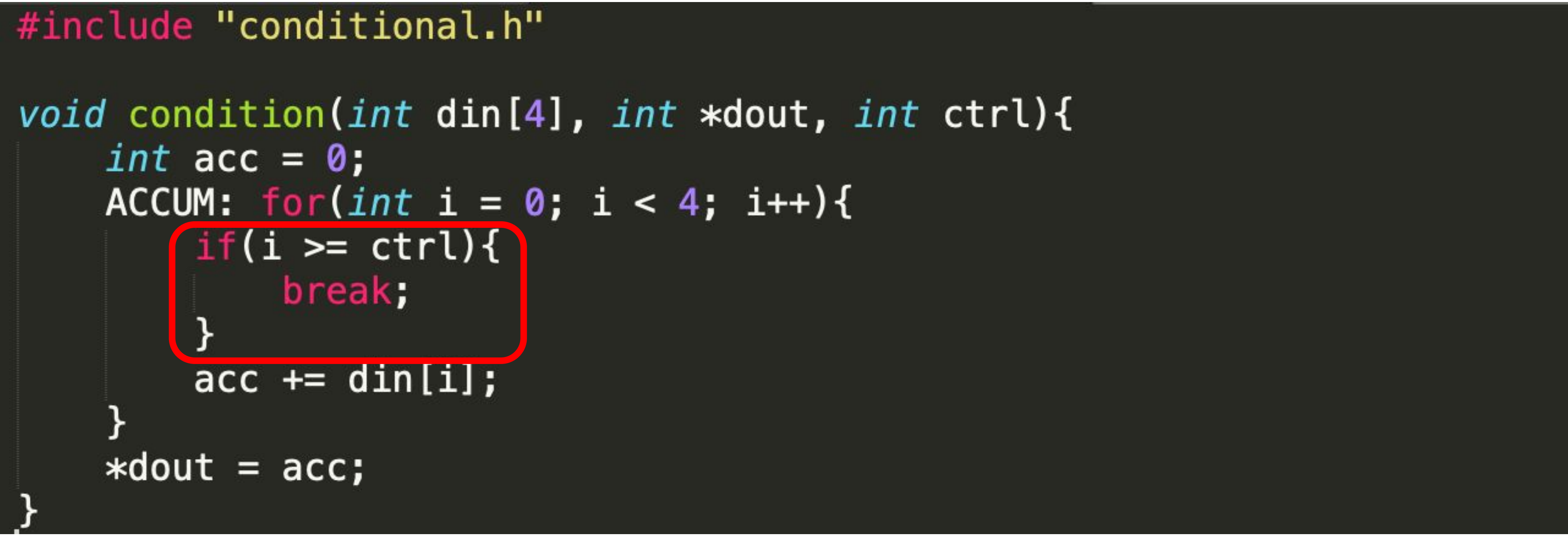
```
#include "conditional.h"

void condition(int din[4], int *dout, ap_int<3> ctrl){
    int acc = 0;
    ACCUM: for(int i = 0; i < ctrl; i++){
        acc += din[i];
    }
    *dout = acc;
}
```



Loop

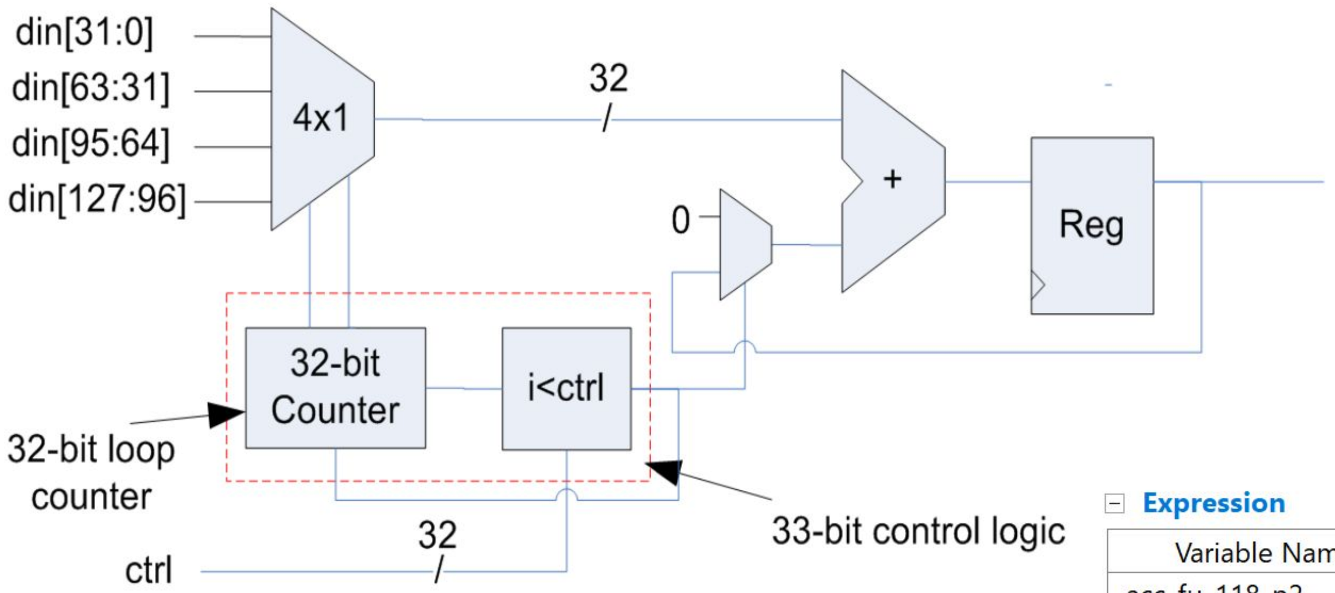
	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- ACCUM	0	6	2	-	-	0 ~ 3	no

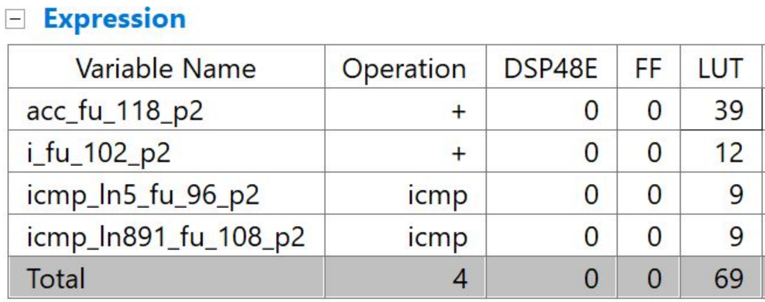
Loops bounded with another solution



```
#include "conditional.h"

void condition(int din[4], int *dout, int ctrl){
    int acc = 0;
    ACCUM: for(int i = 0; i < 4; i++){
        if(i >= ctrl){
            break;
        }
        acc += din[i];
    }
    *dout = acc;
}
```

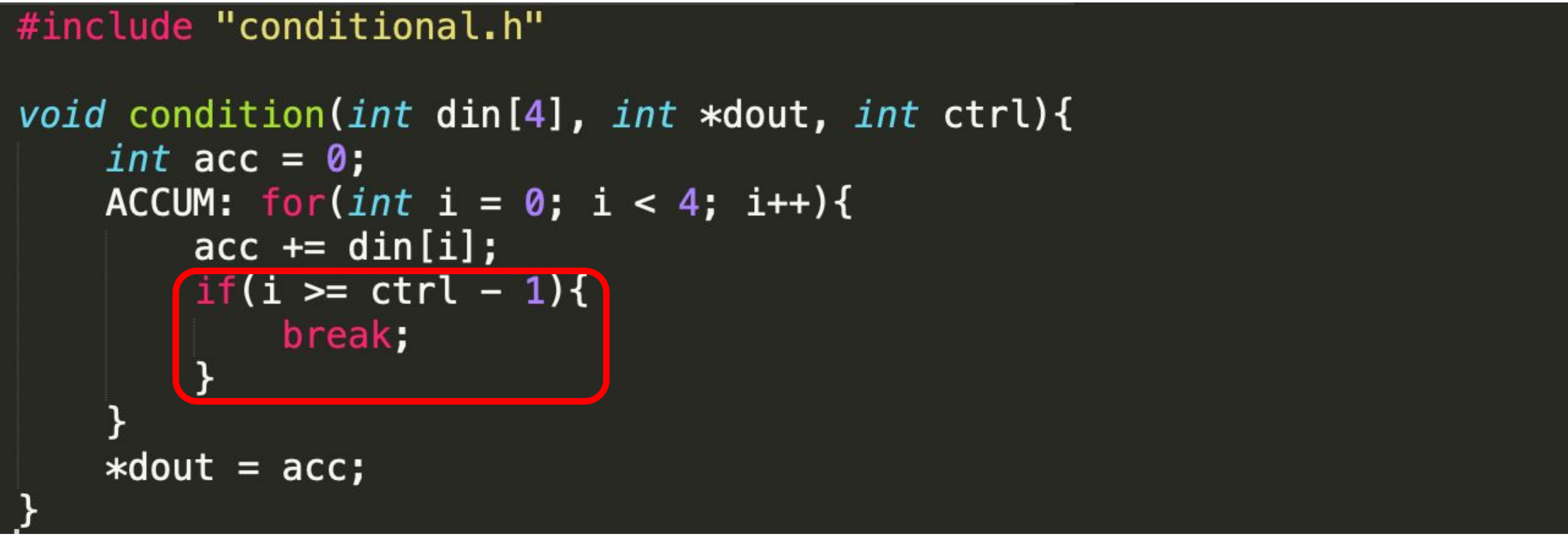
Loops bounded with another solution




Expression

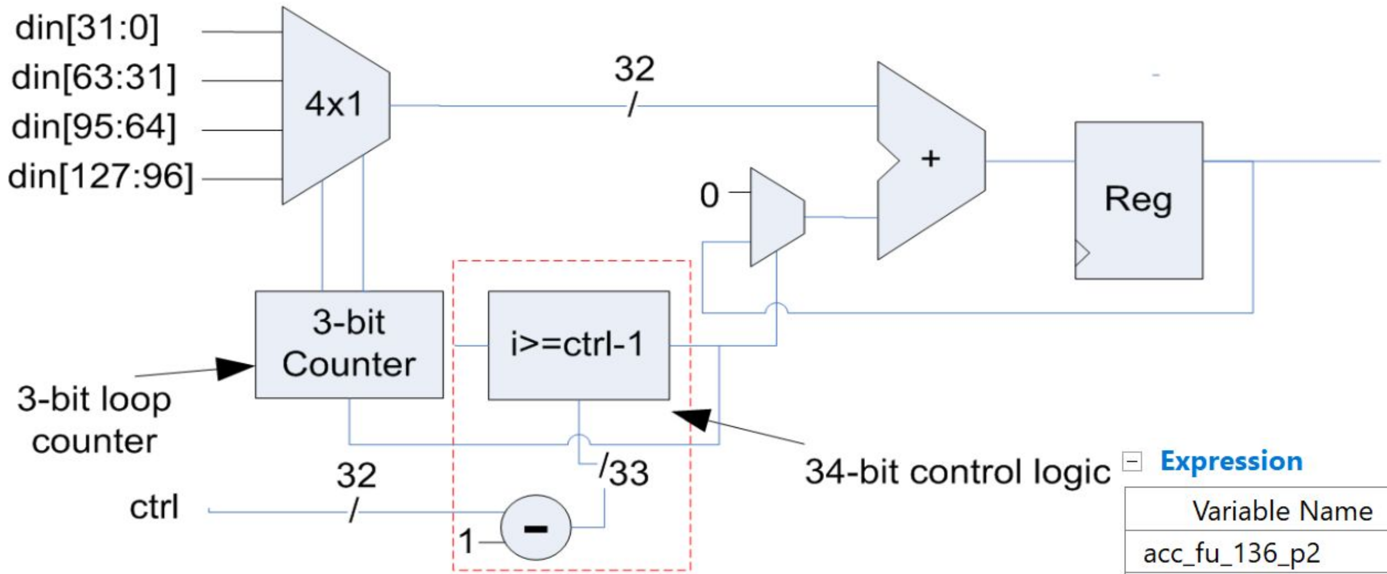
Variable Name	Operation	DSP48E	FF	LUT
acc_fu_118_p2	+	0	0	39
i_fu_102_p2	+	0	0	12
icmp_ln5_fu_96_p2	icmp	0	0	9
icmp_ln891_fu_108_p2	icmp	0	0	9
Total	4	0	0	69

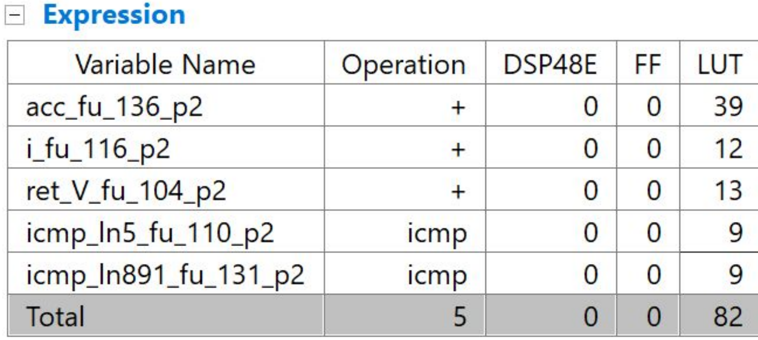
Loops bounded with another solution



```
#include "conditional.h"

void condition(int din[4], int *dout, int ctrl){
    int acc = 0;
    ACCUM: for(int i = 0; i < 4; i++){
        acc += din[i];
        if(i >= ctrl - 1){
            break;
        }
    }
    *dout = acc;
}
```

Loops bounded with another solution




Expression

Variable Name	Operation	DSP48E	FF	LUT
acc_fu_136_p2	+	0	0	39
i_fu_116_p2	+	0	0	12
ret_V_fu_104_p2	+	0	0	13
icmp_ln5_fu_110_p2	icmp	0	0	9
icmp_ln891_fu_131_p2	icmp	0	0	9
Total	5	0	0	82

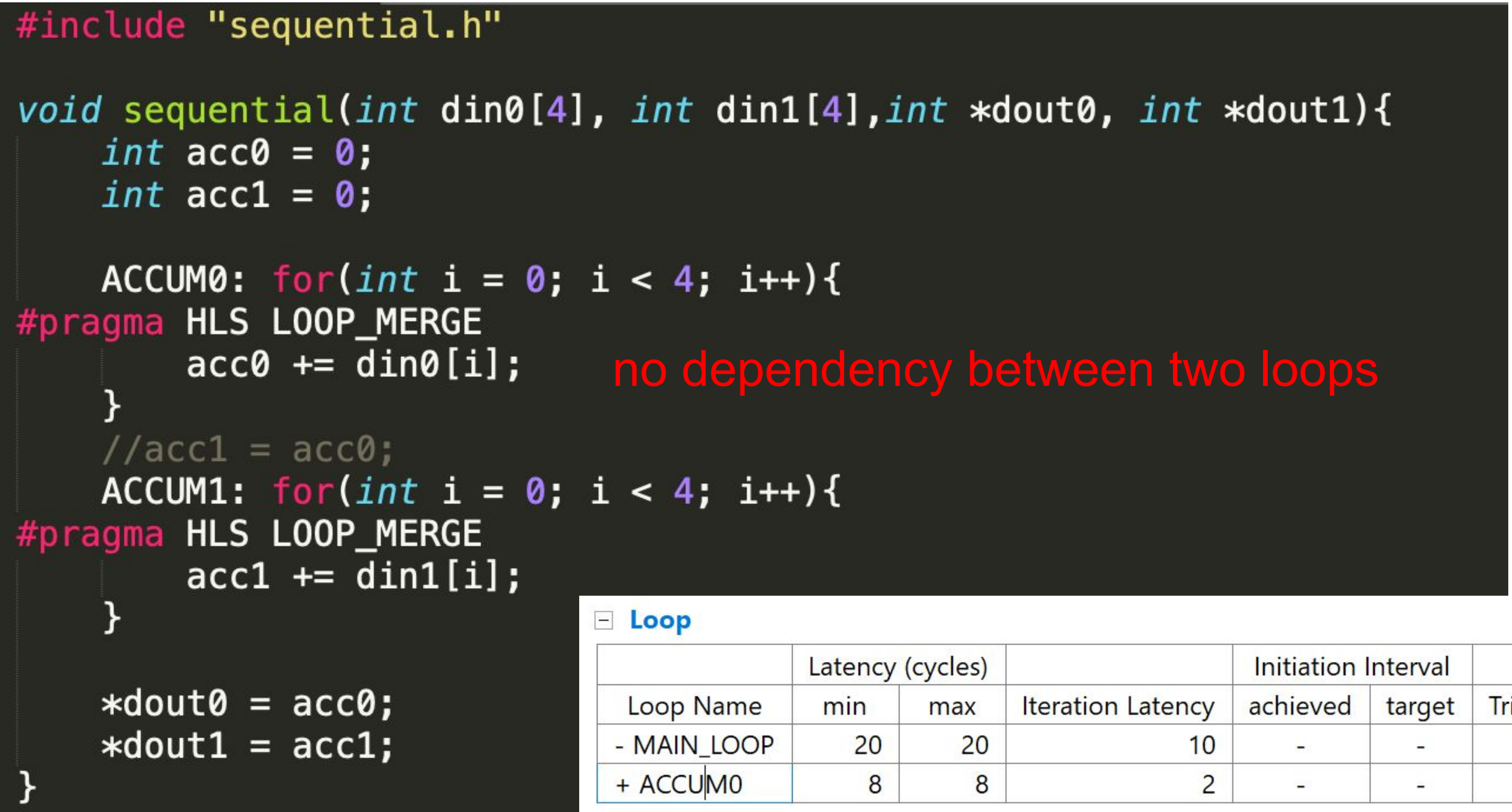
Sequential Loop Merge

```
#pragma HLS
LOOP_MERGE
```

Sometimes the loops are independent and they are feasible to be executed in parallel in hardware.

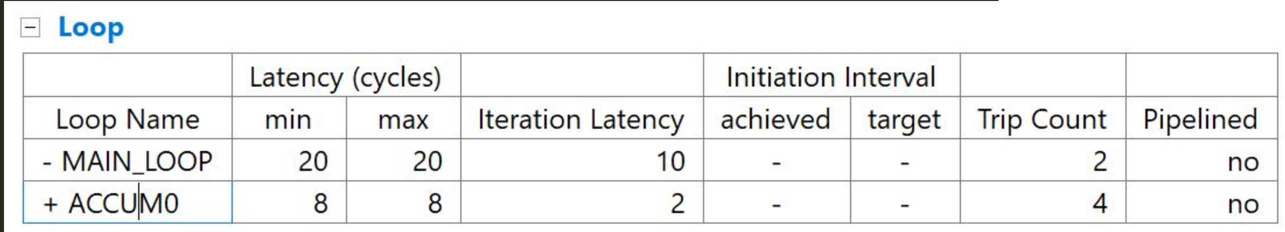
Caution : Data dependancy influence the degree of merging loops.

Sequential Loop Merge without dependency



```
#include "sequential.h"

void sequential(int din0[4], int din1[4],int *dout0, int *dout1){
    int acc0 = 0;
    int acc1 = 0;

    ACCUM0: for(int i = 0; i < 4; i++){
#pragma HLS LOOP_MERGE
        acc0 += din0[i];
    }
    //acc1 = acc0;
    ACCUM1: for(int i = 0; i < 4; i++){
#pragma HLS LOOP_MERGE
        acc1 += din1[i];
    }

    *dout0 = acc0;
    *dout1 = acc1;
}
```

no dependency between two loops



Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- MAIN_LOOP	20	20	10	-	-	2	no
+ ACCUM0	8	8	2	-	-	4	no

Sequential Loop Merge without dependency

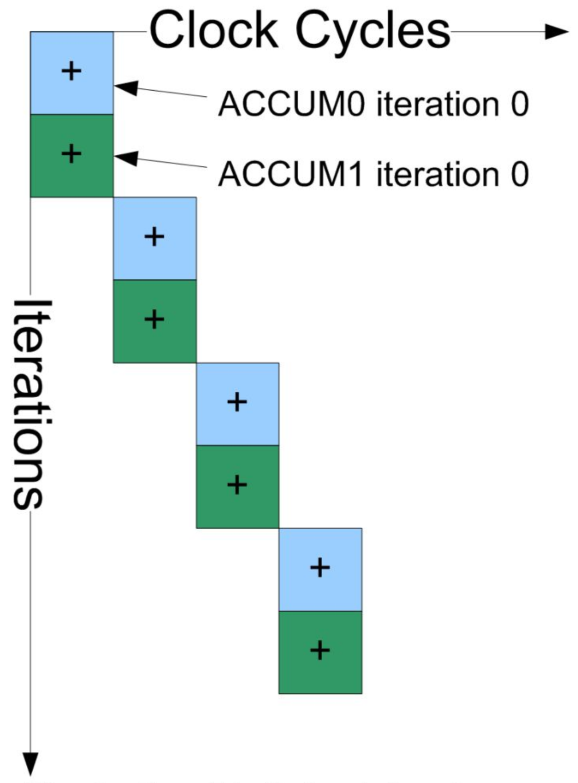
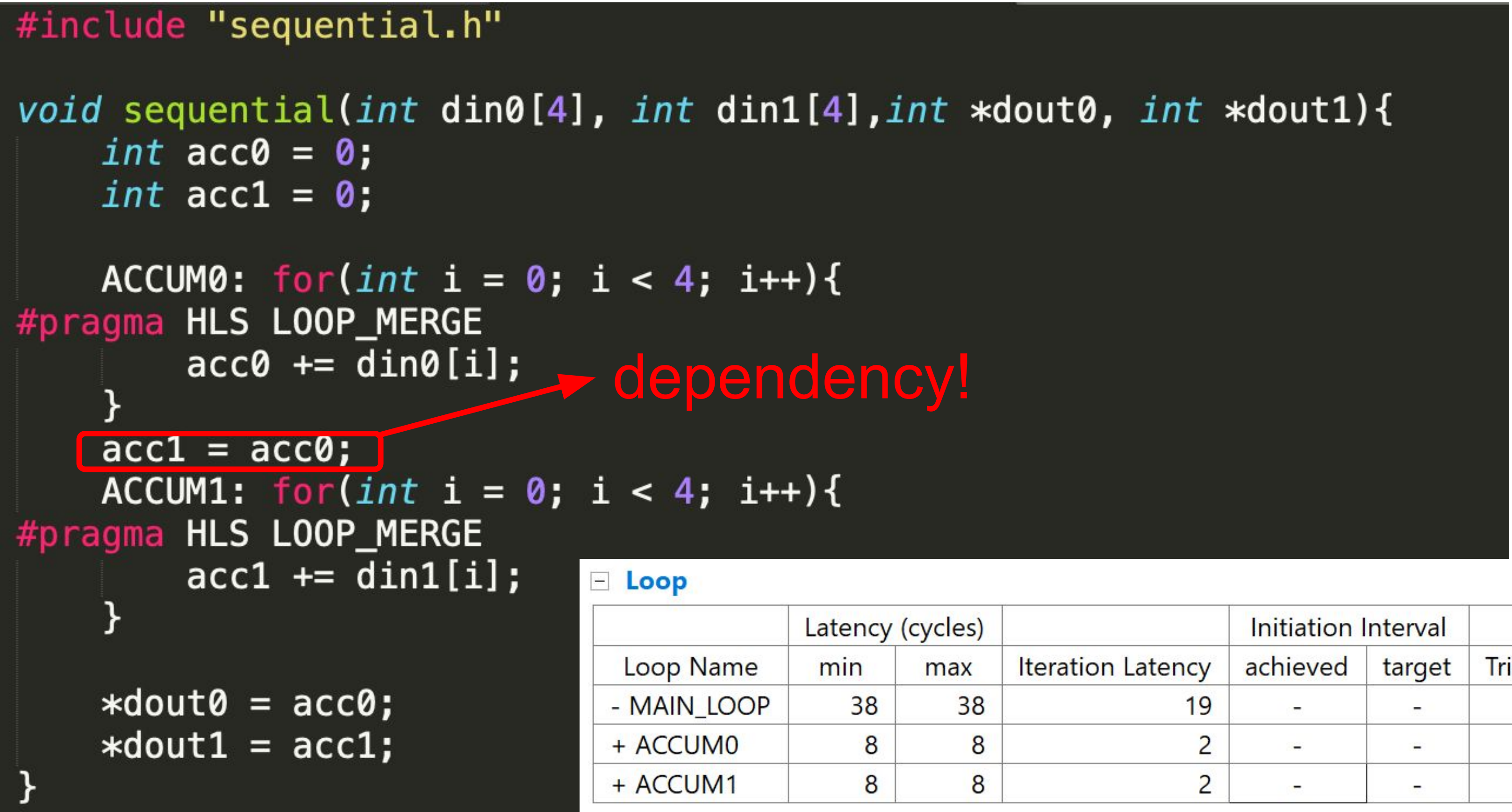
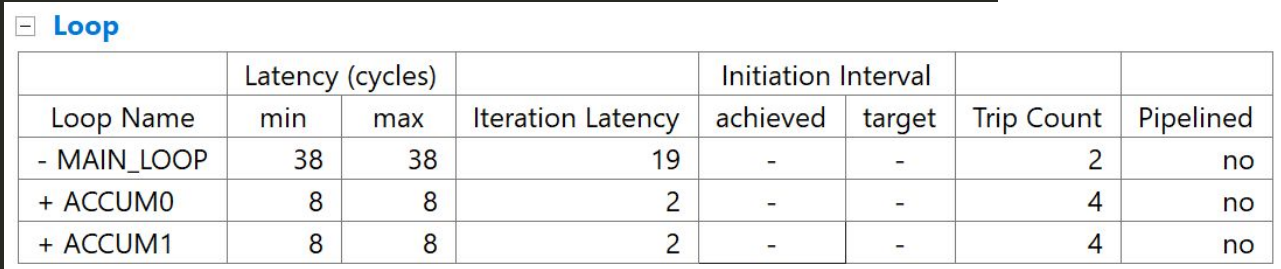


Illustration 51: Schedule of Merged Sequential Loops

Sequential Loop Merge with dependency



```
#include "sequential.h"

void sequential(int din0[4], int din1[4],int *dout0, int *dout1){
    int acc0 = 0;
    int acc1 = 0;

    ACCUM0: for(int i = 0; i < 4; i++){
#pragma HLS LOOP_MERGE
        acc0 += din0[i];
    }
    acc1 = acc0;
    ACCUM1: for(int i = 0; i < 4; i++){
#pragma HLS LOOP_MERGE
        acc1 += din1[i];
    }

    *dout0 = acc0;
    *dout1 = acc1;
}
```



Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- MAIN_LOOP	38	38	19	-	-	2	no
+ ACCUM0	8	8	2	-	-	4	no
+ ACCUM1	8	8	2	-	-	4	no

Pipeline Feedback

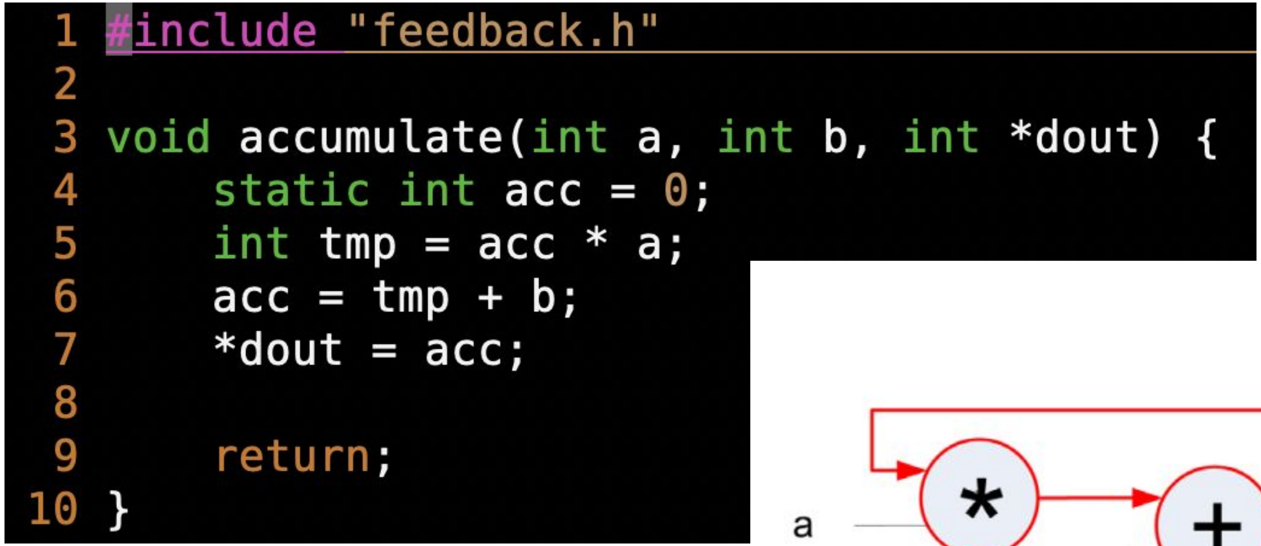
- Data Feedback

- Control Feedback

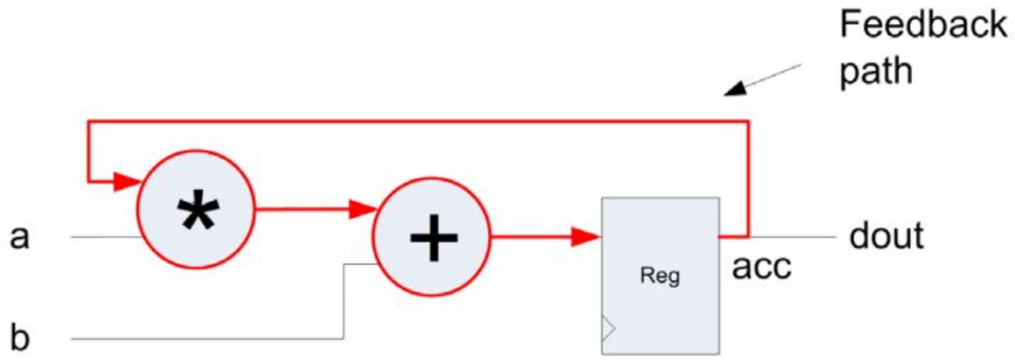
Data Feedback

Data feedback occurs when the input to a data path operation is dependent on a variable computed in the previous loop iteration. If the only loop in the design is the main loop the

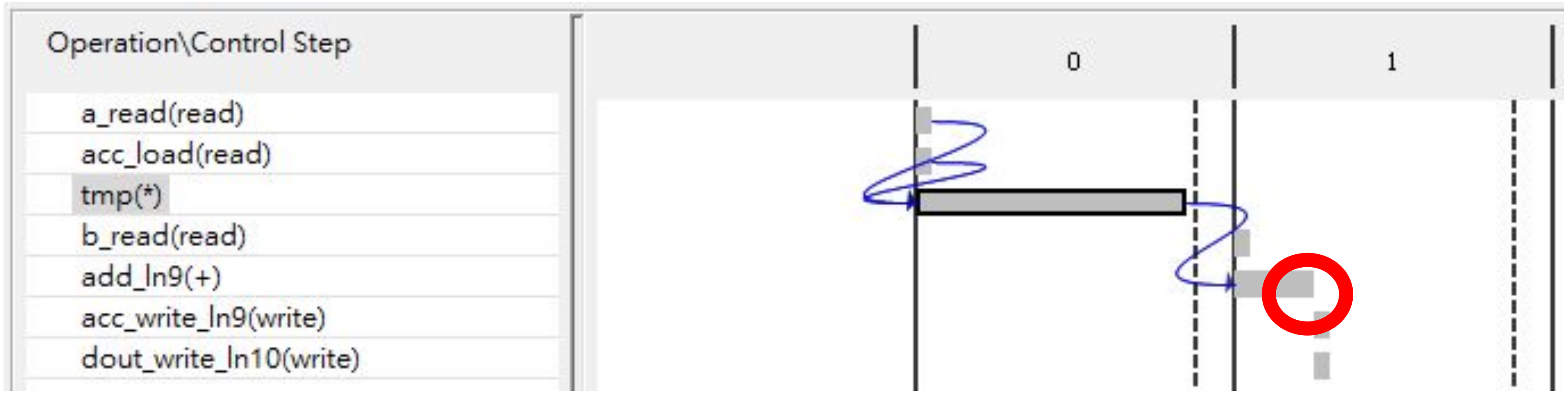
Data Feedback



```
#include "feedback.h"

void accumulate(int a, int b, int *dout) {
    static int acc = 0;
    int tmp = acc * a;
    acc = tmp + b;
    *dout = acc;

    return;
}
```

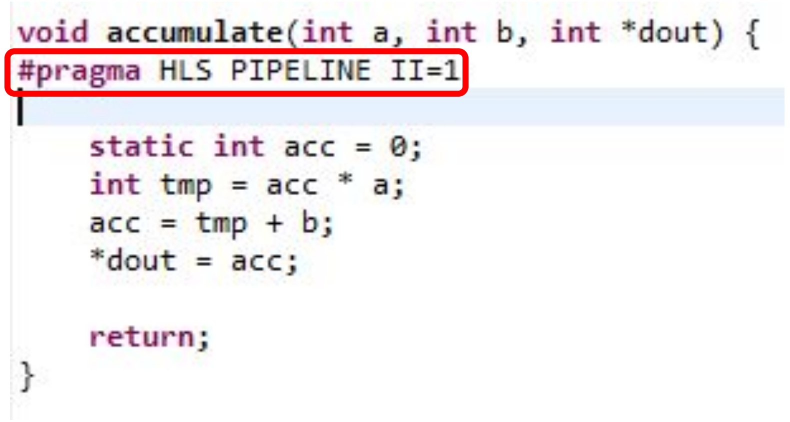


1. acc is computed in clock cycle 1



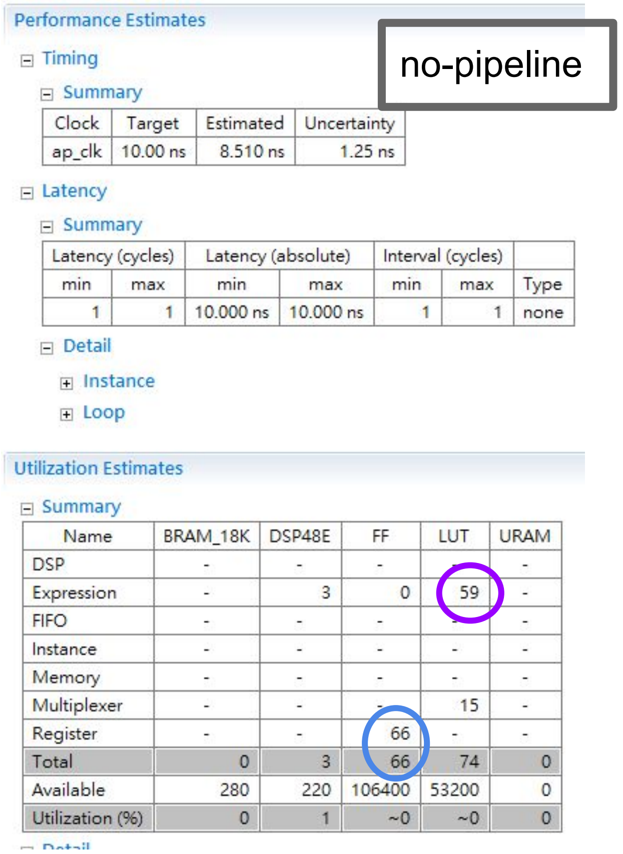
Can pipeline with II = 1

1. acc is computed in clock cycle 1



```
void accumulate(int a, int b, int *dout) {
#pragma HLS PIPELINE II=1

    static int acc = 0;
    int tmp = acc * a;
    acc = tmp + b;
    *dout = acc;

    return;
}
```

1. acc is computed in clock cycle 1



no-pipeline

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
1	1	10.000 ns	10.000 ns	1	1	none

Detail

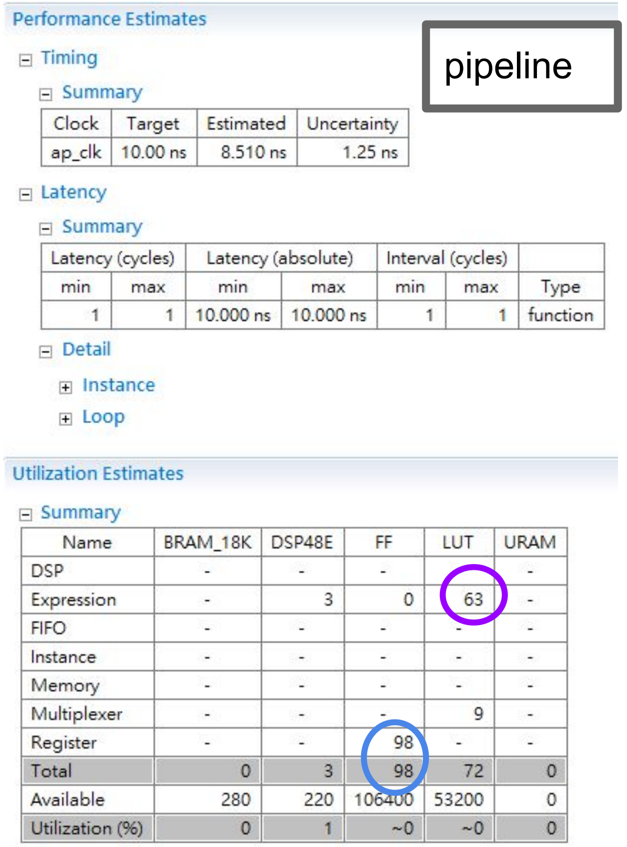
Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	3	0	59	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	15	-
Register	-	-	66	-	-
Total	0	3	66	74	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	~0	~0	0



pipeline

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
1	1	10.000 ns	10.000 ns	1	1	function

Detail

Instance

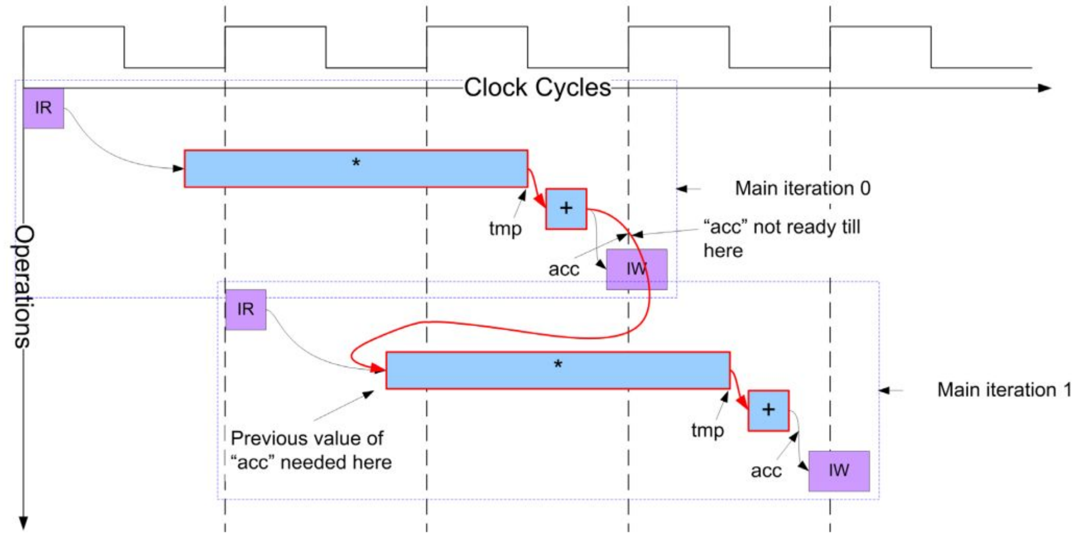
Loop

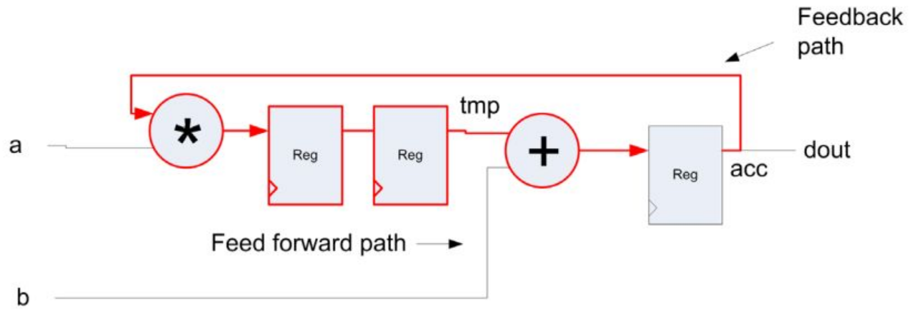
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	3	0	63	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	9	-
Register	-	-	98	-	-
Total	0	3	98	72	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	~0	~0	0

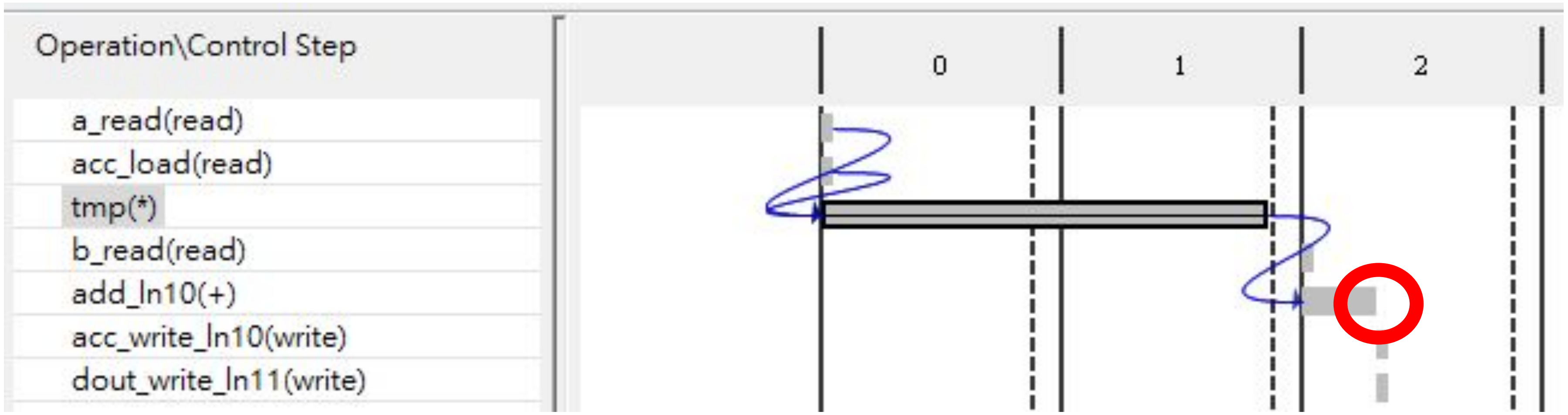
2. acc computed in longer clock cycles (fast clock rate)





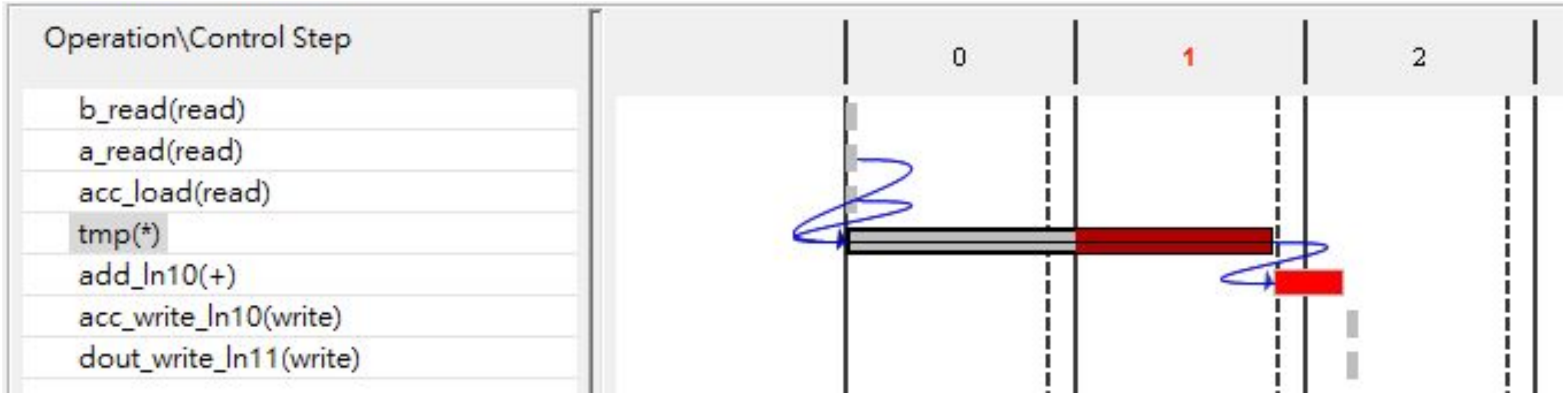
2. acc computed in longer clock cycles (fast clock rate)

調高 clock rate (減少clock period)

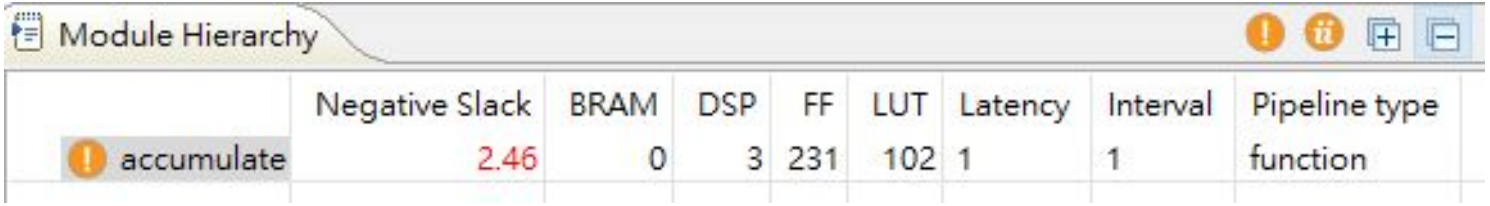


Can’t pipeline with II = 1

Pipeline with II = 1



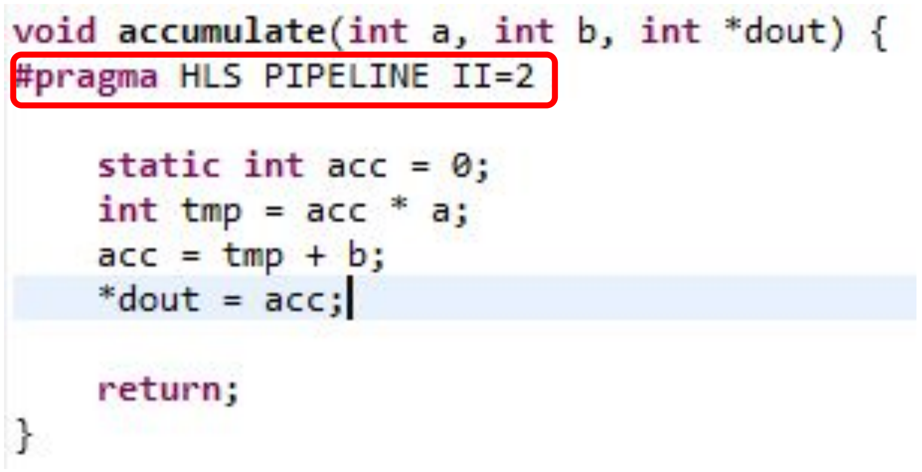
Operation\Control Step	0	1	2
b_read(read)			
a_read(read)			
acc_load(read)			
tmp(*)			
add_ln10(+)			
acc_write_ln10(write)			
dout_write_ln11(write)			



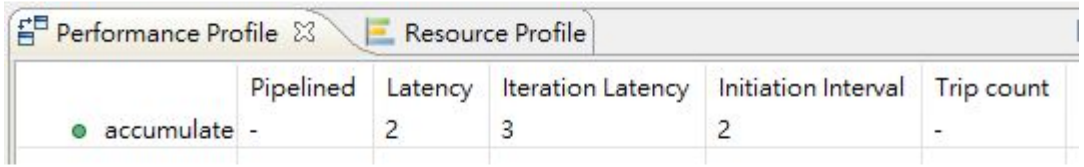
Module Hierarchy

	Negative Slack	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
accumulate	2.46	0	3	231	102	1	1	function

最多 pipeline with II = 2

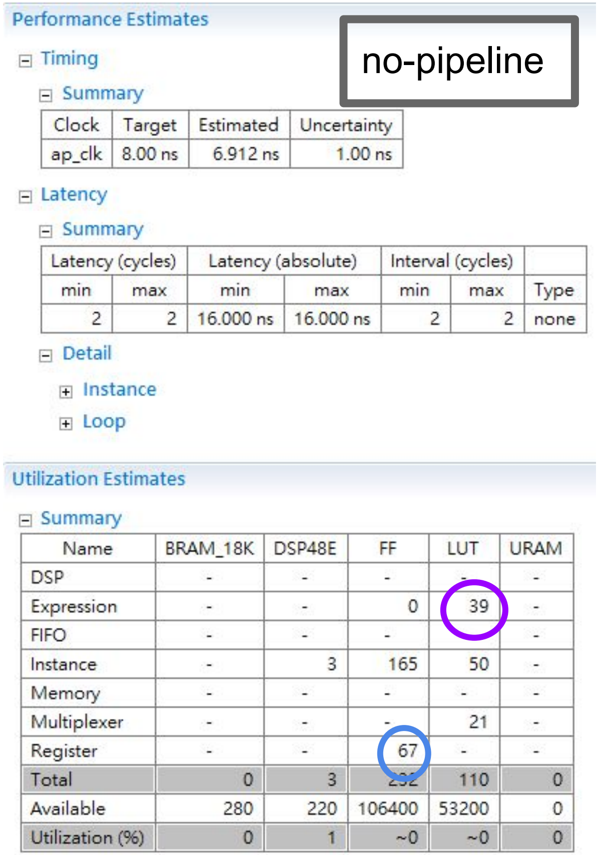


```
void accumulate(int a, int b, int *dout) {
#pragma HLS PIPELINE II=2

    static int acc = 0;
    int tmp = acc * a;
    acc = tmp + b;
    *dout = acc;

    return;
}
```



	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
● accumulate	-	2	3	2	-

Pipeline with II = 2



no-pipeline

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	8.00 ns	6.912 ns	1.00 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
2	2	16.000 ns	16.000 ns	2	2	none

Detail

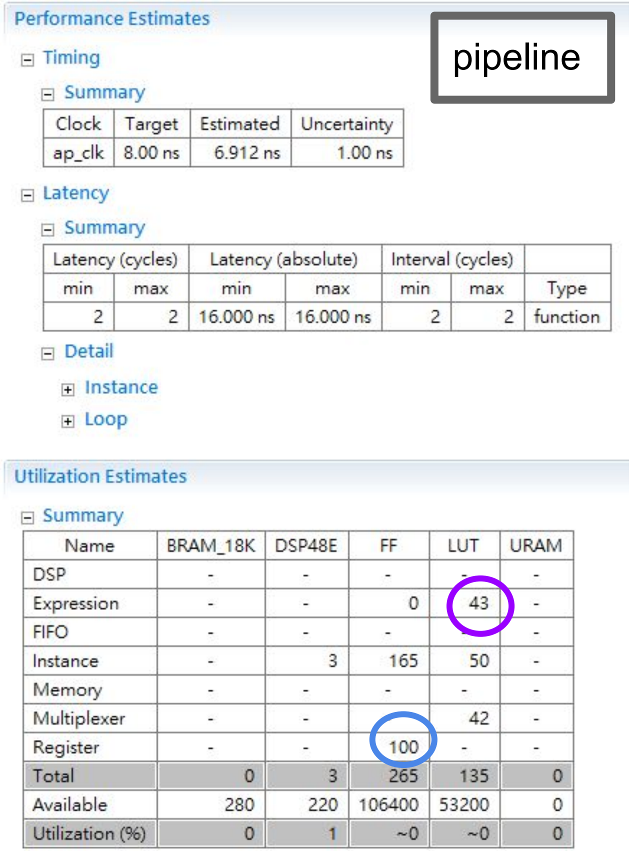
Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	39	-
FIFO	-	-	-	-	-
Instance	-	3	165	50	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	21	-
Register	-	-	67	-	-
Total	0	3	232	110	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	~0	~0	0



pipeline

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	8.00 ns	6.912 ns	1.00 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
2	2	16.000 ns	16.000 ns	2	2	function

Detail

Instance

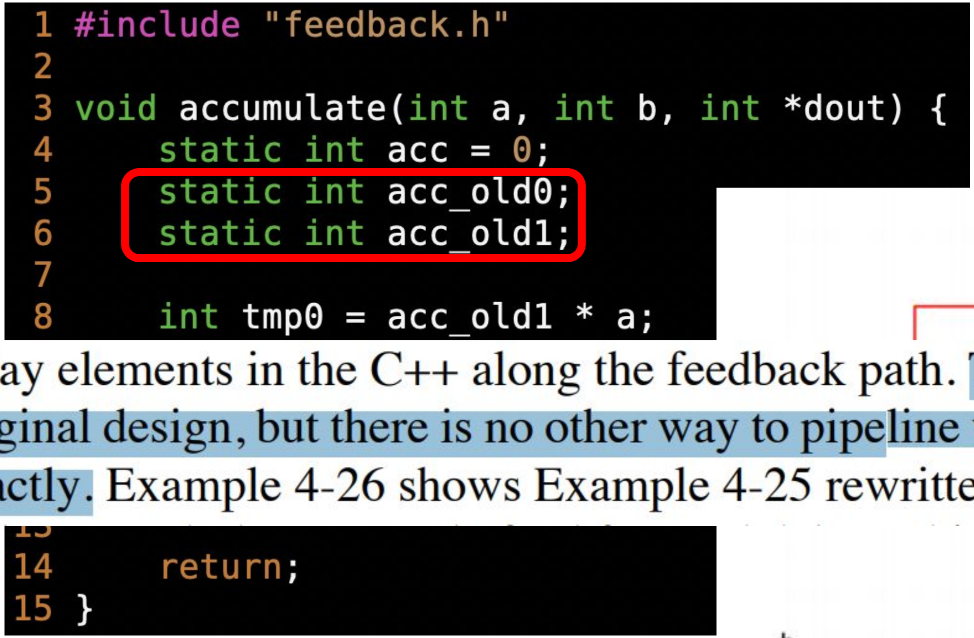
Loop

Utilization Estimates

Summary

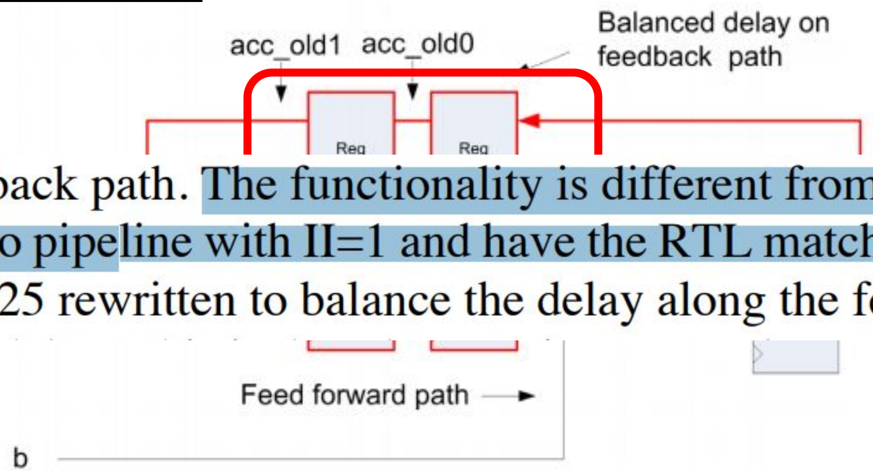
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	43	-
FIFO	-	-	-	-	-
Instance	-	3	165	50	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	42	-
Register	-	-	100	-	-
Total	0	3	265	135	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	~0	~0	0

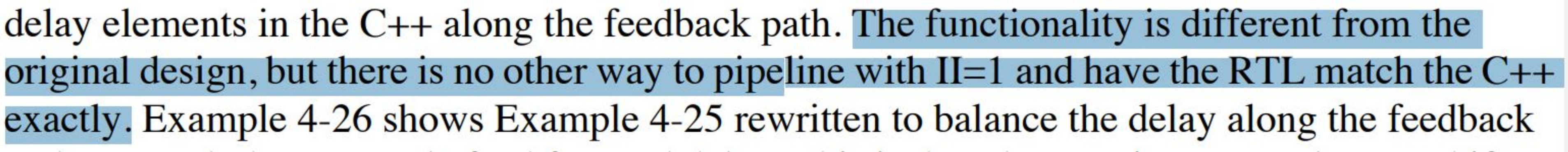
3. 硬要pipeline with II = 1 (fast clock rate)



```
#include "feedback.h"

void accumulate(int a, int b, int *dout) {
    static int acc = 0;
    static int acc_old0;
    static int acc_old1;

    int tmp0 = acc_old1 * a;
```

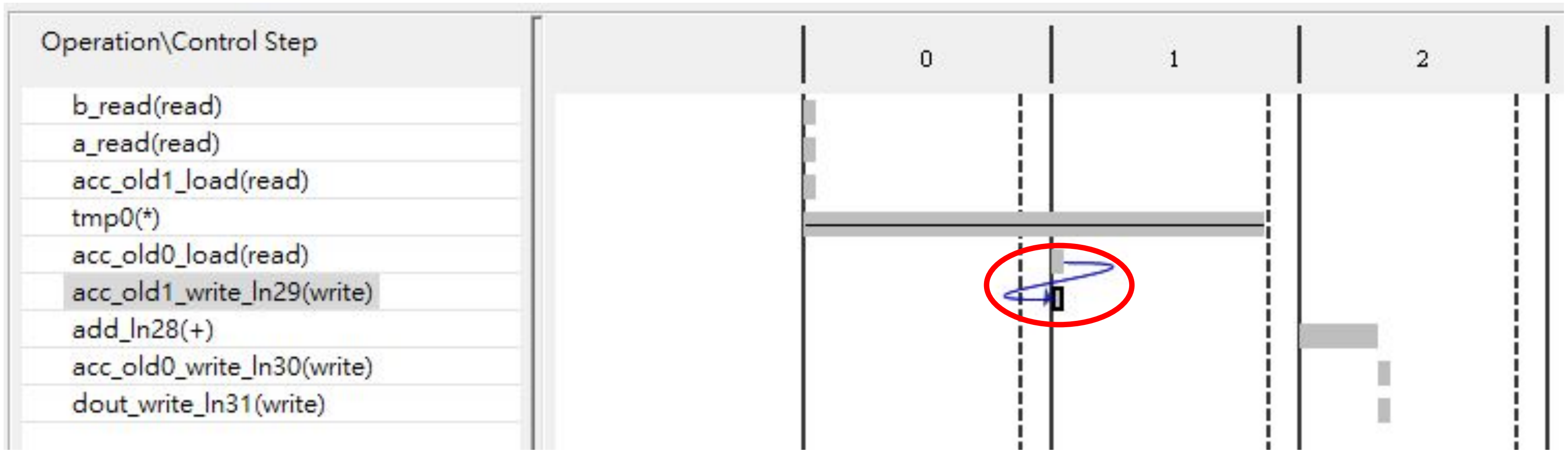
```
    return;
}
```



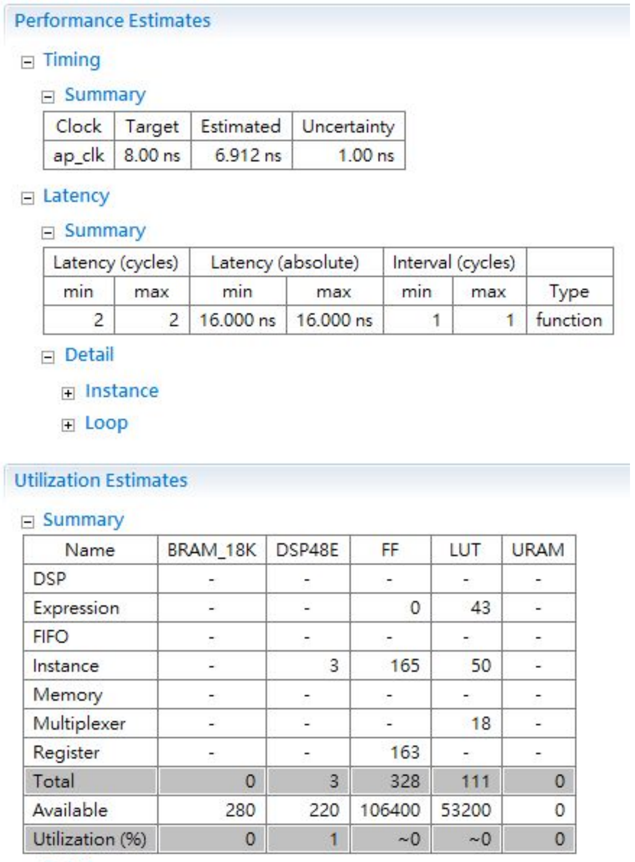


delay elements in the C++ along the feedback path. The functionality is different from the original design, but there is no other way to pipeline with II=1 and have the RTL match the C++ exactly. Example 4-26 shows Example 4-25 rewritten to balance the delay along the feedback

3. 硬要pipeline with II = 1 (faster clock rate)



3. 硬要pipeline with II = 1 (faster clock rate)



Performance Estimates

⊟ Timing

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	8.00 ns	6.912 ns	1.00 ns

⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
2	2	16.000 ns	16.000 ns	1	1	function

⊟ Detail

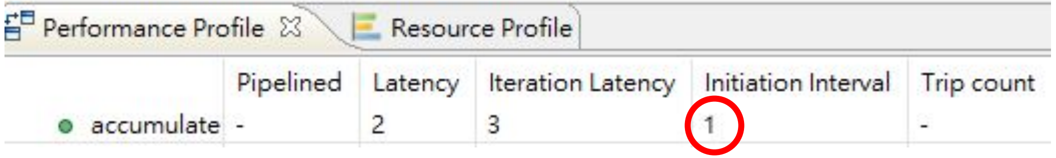
⊞ Instance

⊞ Loop

Utilization Estimates

⊟ Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	43	-
FIFO	-	-	-	-	-
Instance	-	3	165	50	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	18	-
Register	-	-	163	-	-
Total	0	3	328	111	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	~0	~0	0



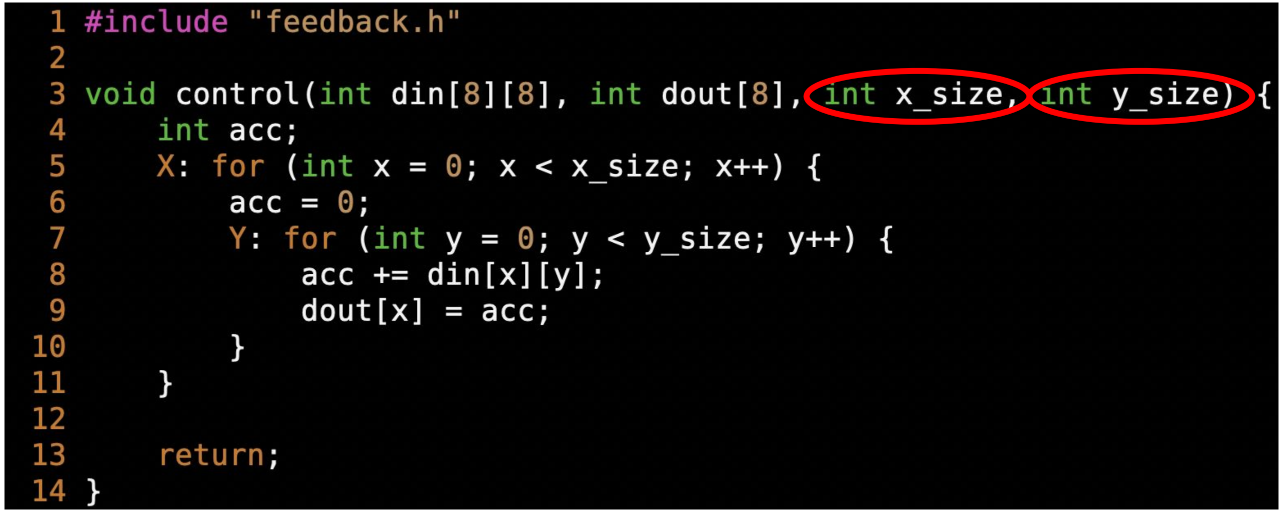
Performance Profile | Resource Profile

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
● accumulate	-	2	3	1	-

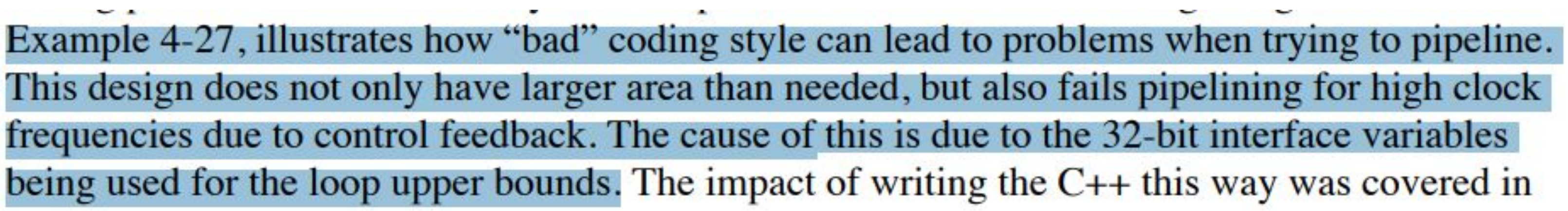
Control Feedback

Pipelining failures due to feedback are also possible due to the loop control in a design. The deeper the nesting of loops in a design, the more complicated the control becomes, which in turn limits the clock frequency and ability to pipeline a design. Adhering to the recommended

Control Feedback with bad coding style

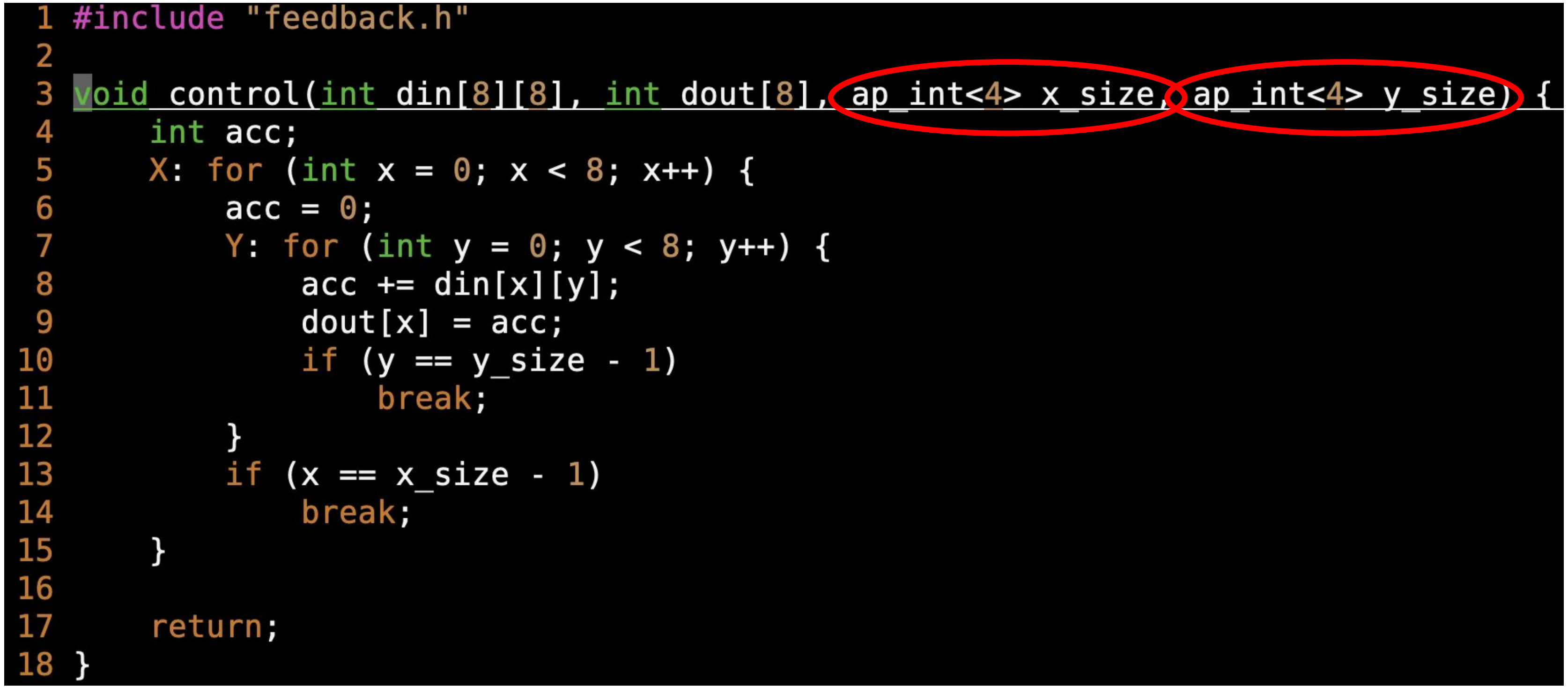


```
#include "feedback.h"

void control(int din[8][8], int dout[8], int x_size, int y_size) {
    int acc;
    X: for (int x = 0; x < x_size; x++) {
        acc = 0;
        Y: for (int y = 0; y < y_size; y++) {
            acc += din[x][y];
            dout[x] = acc;
        }
    }

    return;
}
```



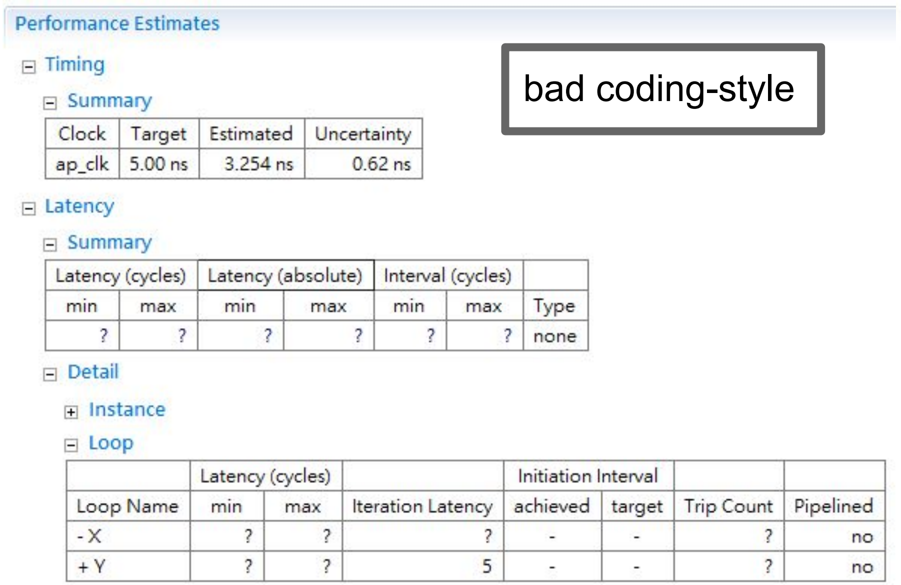
Example 4-27, illustrates how "bad" coding style can lead to problems when trying to pipeline. This design does not only have larger area than needed, but also fails pipelining for high clock frequencies due to control feedback. The cause of this is due to the 32-bit interface variables being used for the loop upper bounds. The impact of writing the C++ this way was covered in

Control Feedback with good coding style



```
#include "feedback.h"

void control(int din[8][8], int dout[8], ap_int<4> x_size, ap_int<4> y_size) {
    int acc;
    X: for (int x = 0; x < 8; x++) {
        acc = 0;
        Y: for (int y = 0; y < 8; y++) {
            acc += din[x][y];
            dout[x] = acc;
            if (y == y_size - 1)
                break;
        }
        if (x == x_size - 1)
            break;
    }

    return;
}
```

Control Feedback performance



bad coding-style

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	3.254 ns	0.62 ns

Latency

Summary

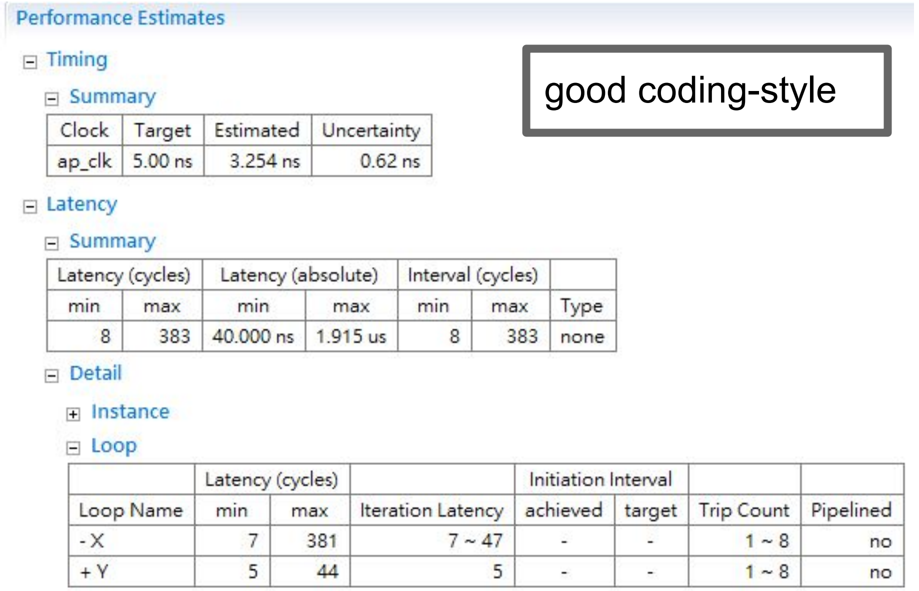
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
?	?	?	?	?	?	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- X	?	?	?	-	-	?	no
+ Y	?	?	5	-	-	?	no



good coding-style

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	3.254 ns	0.62 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
8	383	40.000 ns	1.915 us	8	383	none

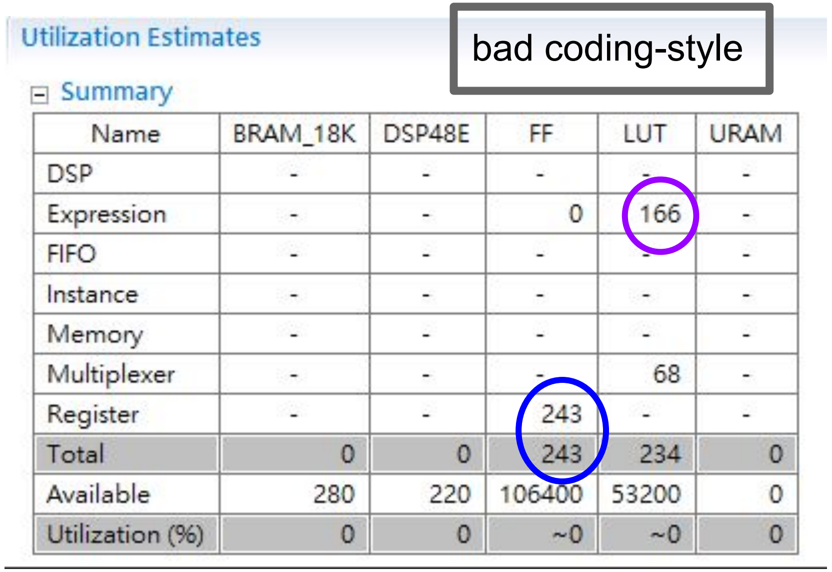
Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- X	7	381	7 ~ 47	-	-	1 ~ 8	no
+ Y	5	44	5	-	-	1 ~ 8	no

Control Feedback usage

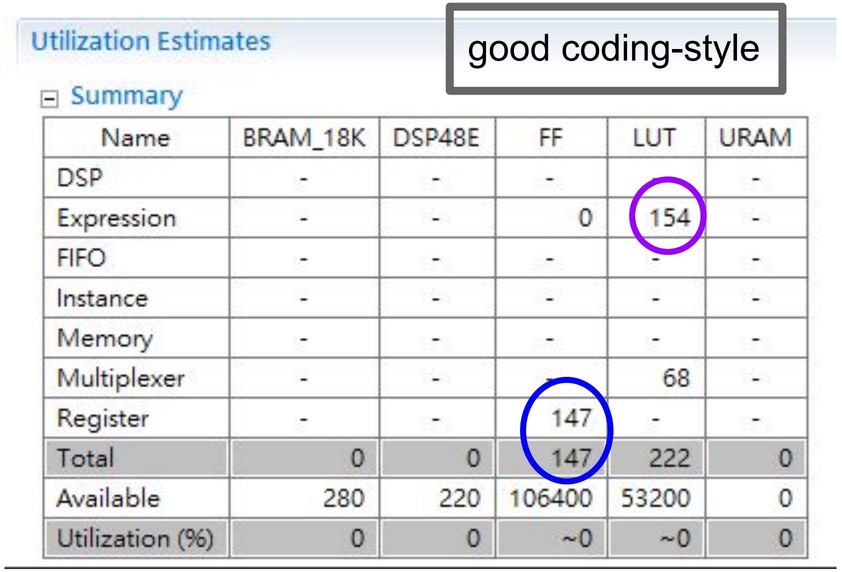


bad coding-style

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	166	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	68	-
Register	-	-	243	-	-
Total	0	0	243	234	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	~0	0



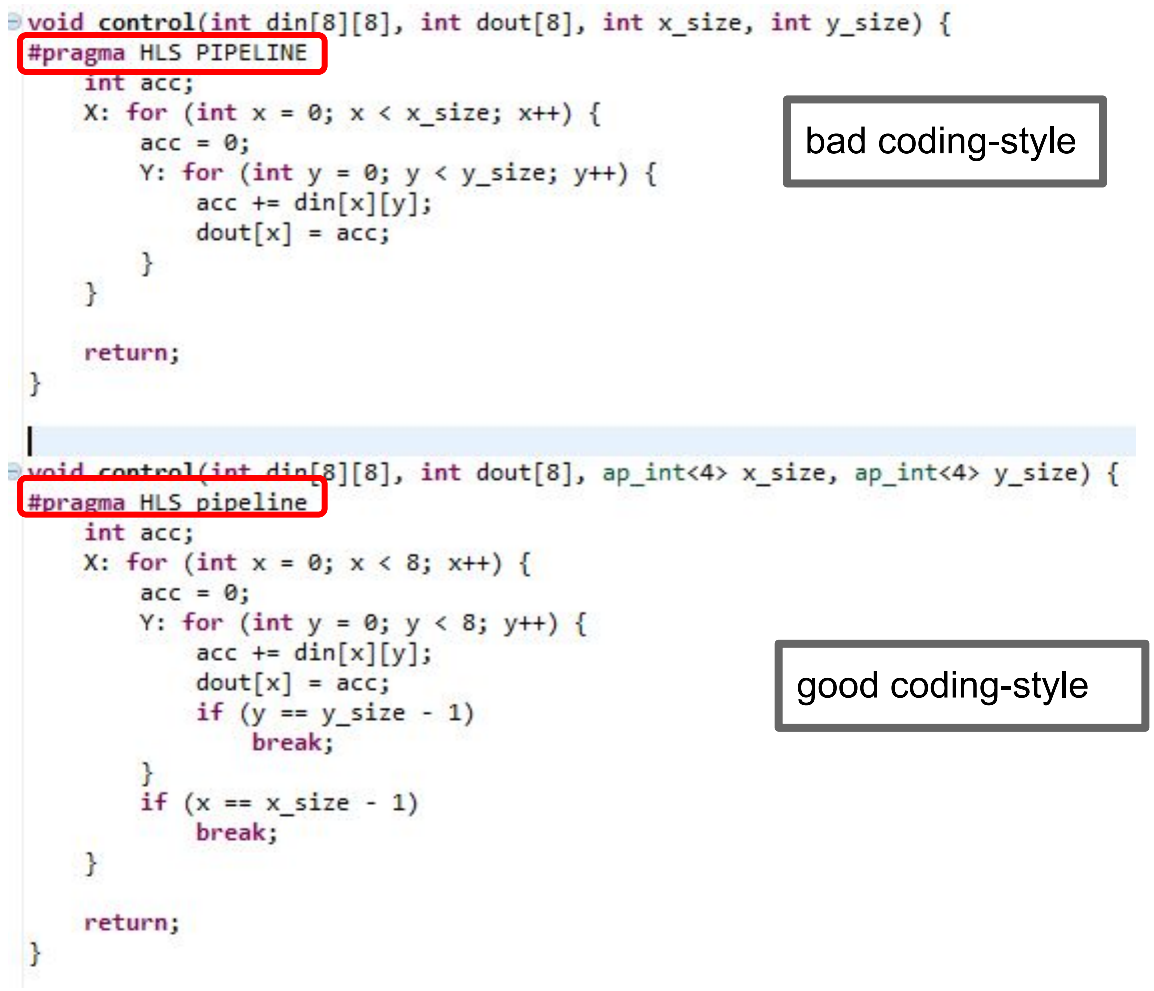
good coding-style

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	154	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	68	-
Register	-	-	147	-	-
Total	0	0	147	222	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	~0	0

Control Feedback pipeline



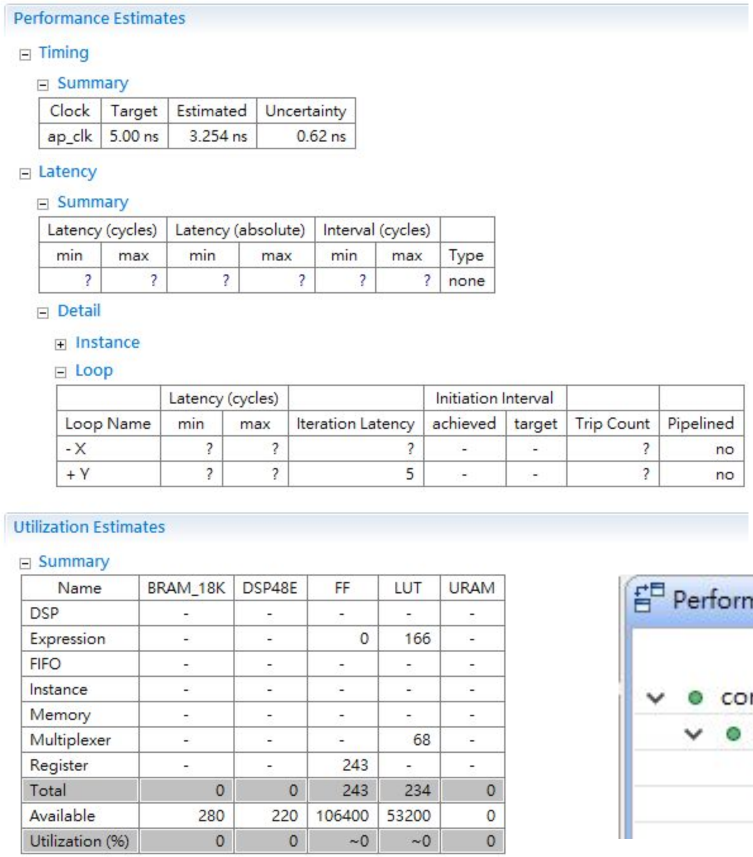
```
void control(int din[8][8], int dout[8], int x_size, int y_size) {
#pragma HLS PIPELINE
    int acc;
    X: for (int x = 0; x < x_size; x++) {
        acc = 0;
        Y: for (int y = 0; y < y_size; y++) {
            acc += din[x][y];
            dout[x] = acc;
        }
    }

    return;
}
```

bad coding-style

```
void control(int din[8][8], int dout[8], ap_int<4> x_size, ap_int<4> y_size) {
#pragma HLS pipeline
    int acc;
    X: for (int x = 0; x < 8; x++) {
        acc = 0;
        Y: for (int y = 0; y < 8; y++) {
            acc += din[x][y];
            dout[x] = acc;
            if (y == y_size - 1)
                break;
        }
        if (x == x_size - 1)
            break;
    }

    return;
}
```

good coding-style

Control Feedback -pipeline (bad coding-style)



Performance Estimates

⊟ Timing

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	3.254 ns	0.62 ns

⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
?	?	?	?	?	?	none

⊟ Detail

⊞ Instance

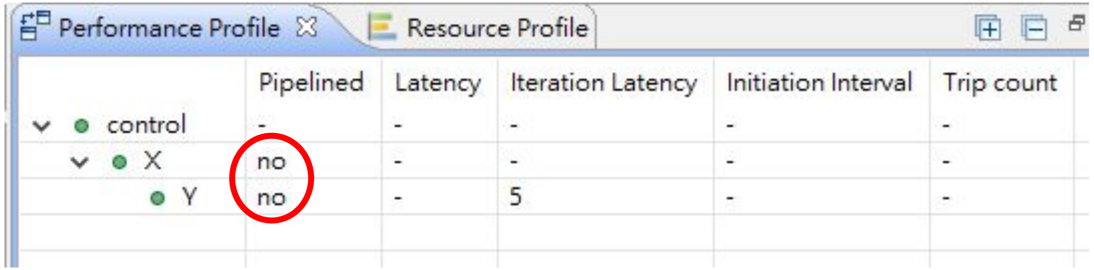
⊟ Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- X	?	?	?	-	-	?	no
+ Y	?	?	5	-	-	?	no

Utilization Estimates

⊟ Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	166	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	68	-
Register	-	-	243	-	-
Total	0	0	243	234	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	~0	0

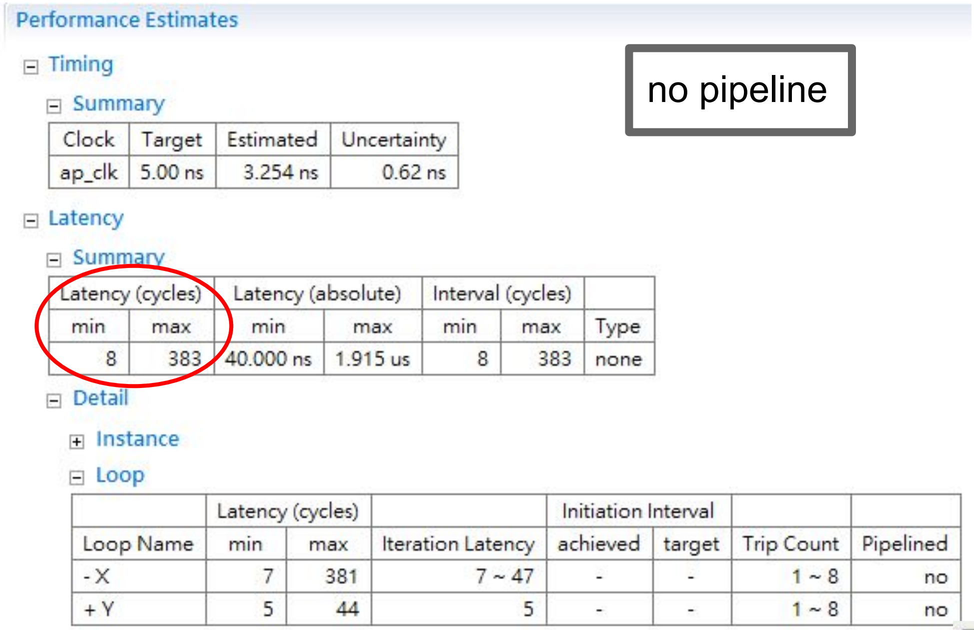


Performance Profile | Resource Profile

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
control	-	-	-	-	-
X	no	-	-	-	-
Y	no	-	5	-	-

Control Feedback -pipeline (good coding-style)

no pipeline


Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	3.254 ns	0.62 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
8	383	40.000 ns	1.915 us	8	383	none

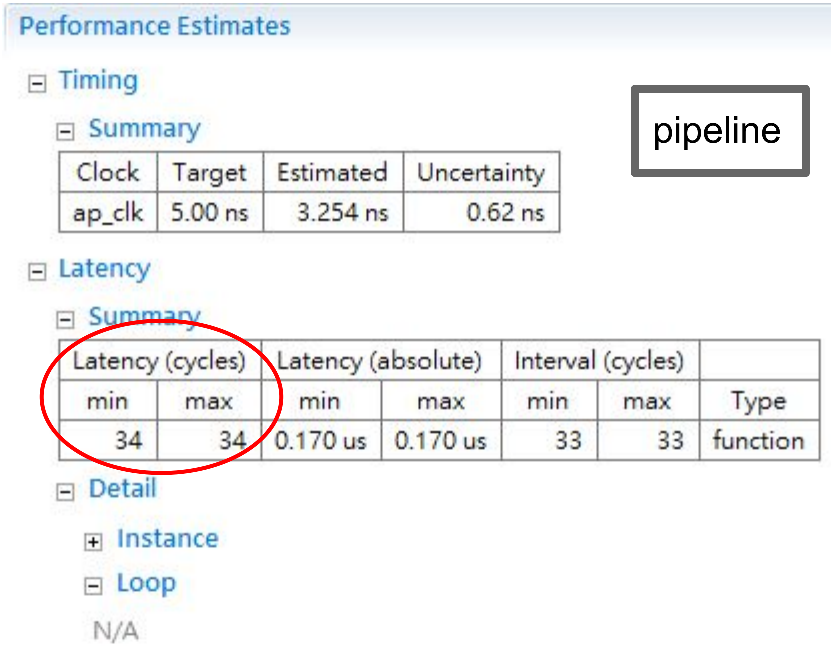
Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- X	7	381	7 ~ 47	-	-	1 ~ 8	no
+ Y	5	44	5	-	-	1 ~ 8	no

pipeline


Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	3.254 ns	0.62 ns

Latency

Summary

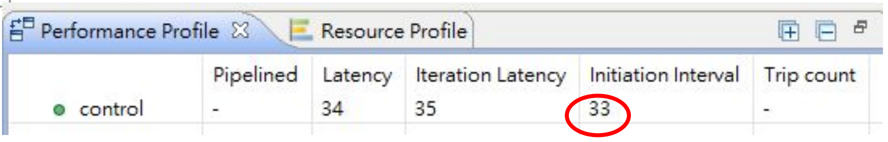
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
34	34	0.170 us	0.170 us	33	33	function

Detail

Instance

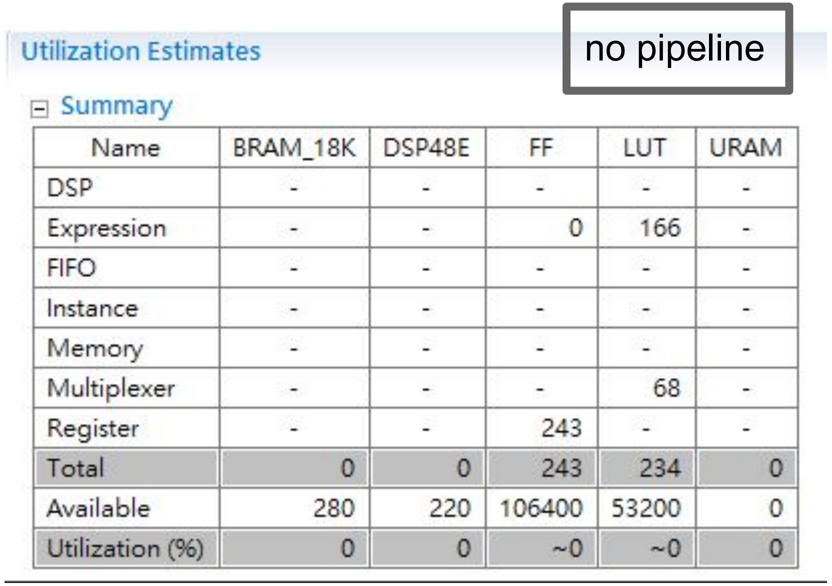
Loop

N/A



	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
● control	-	34	35	33	-

Control Feedback -pipeline (good coding-style)

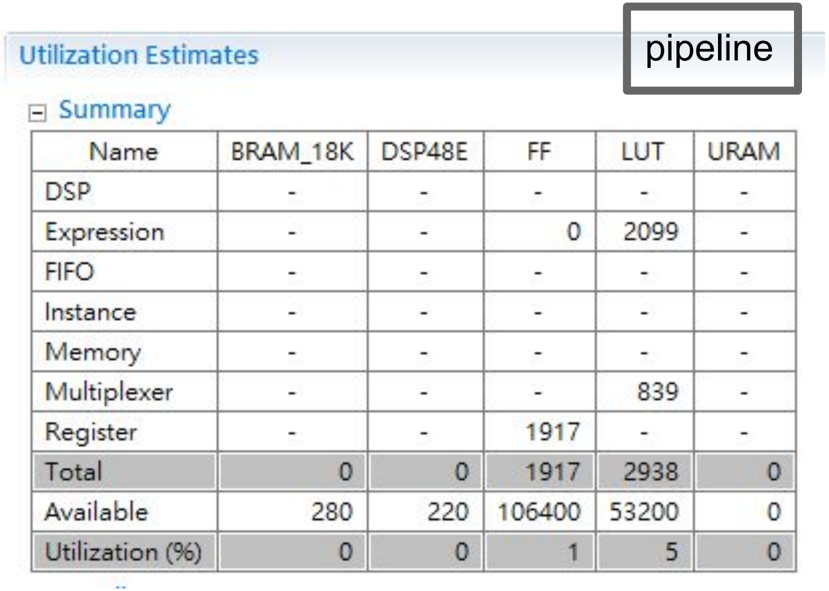


no pipeline

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	166	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	68	-
Register	-	-	243	-	-
Total	0	0	243	234	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	~0	~0	0



pipeline

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2099	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	839	-
Register	-	-	1917	-	-
Total	0	0	1917	2938	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	1	5	0

끝