Starting with HLS

Snippets and blocks

Writing code and understanding it



US AHEAD bootcamp - June 9th-13th 2025

Marcos Oliveira

Table of contents

01

Combinational

Connecting blocks together

02

Quantization

Make it as big as it needs to be

03

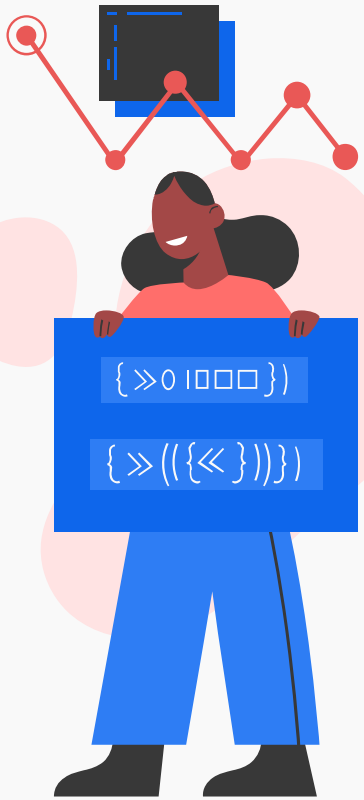
Sequential

Let’s add more fun to it

04

Summary

Wrapping up



01 Combinational logic

Connecting blocks together



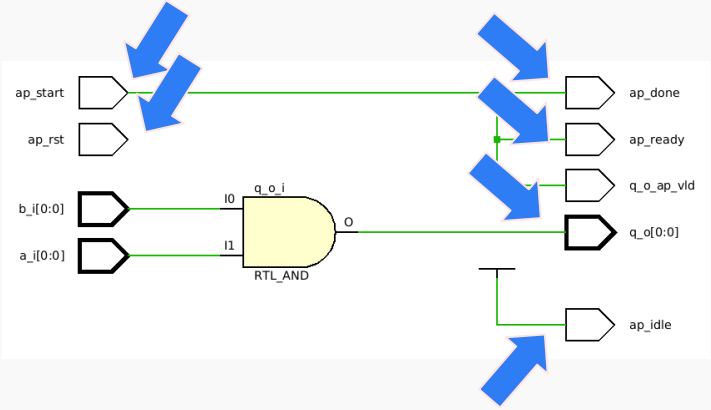
00 – Logic operator

What should we expect with the following code?

😲

```
void logic_operator(bool a_i, bool b_i, bool &q_o) {
                    q_o = a_i & b_i;
}
```

Interface ports added by



You need to explicitly ask to not add default interface signals

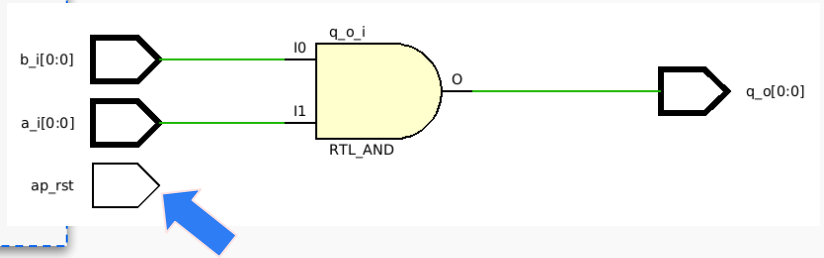
00 – Logic operator (simplified)

```
void logic_operator(bool a_i, bool b_i, bool &q_o)
 {
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=b_i
                #pragma HLS INTERFACE ap_none port=q_o

                q_o = a_i & b_i;

}
```

Most ports are removed except



Reset is a crucial part of synchronous circuits and Vitis implement them by default

01 - Arithmetic operator

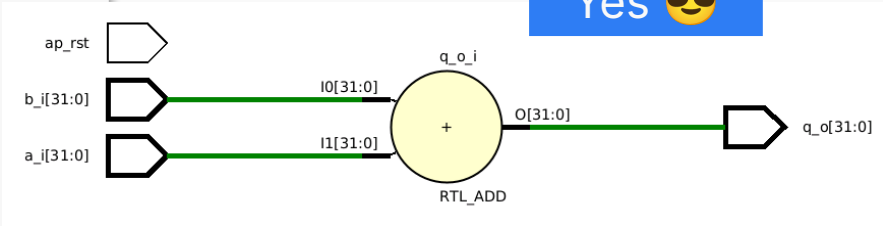
```
void arithmetic_operator(int a_i, int b_i, int &q_o) {
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=b_i
                #pragma HLS INTERFACE ap_none port=q_o

                q_o = a_i + b_i;

}
```

Can we write different operations instead of just logic gates?

Yes 😎



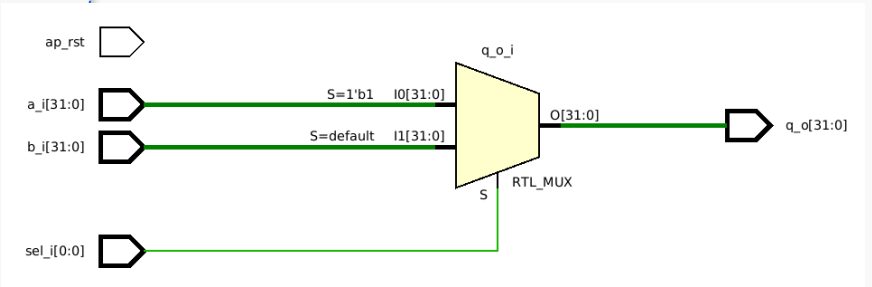
Integers are 32-bit signals by default

02 – If-else

What an if else statement translates

```
void if_else(int a_i, int b_i, bool sel_i, int &q_o) {
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=b_i
                #pragma HLS INTERFACE ap_none port=sel_i
                #pragma HLS INTERFACE ap_none port=q_o

                if (sel_i) {
                q_o = b_i;
                } else {
                q_o = a_i;
                }
}
```

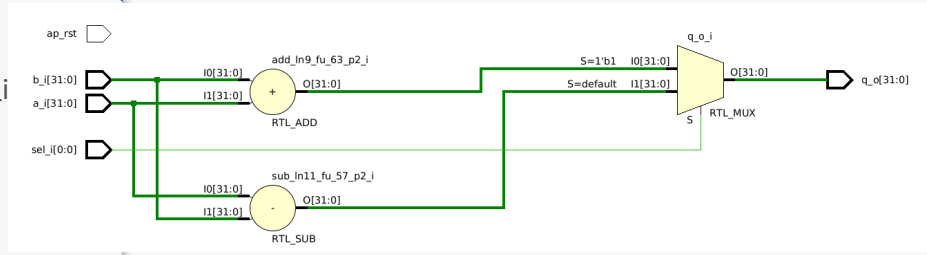


03 – If-else-arithmetic

What should we expect with the following code?

```
void if_else(int a_i, int b_i,
        bool sel_i, int &q_o) {
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=b_i
                #pragma HLS INTERFACE ap_none port=sel_i
                #pragma HLS INTERFACE ap_none port=q_o

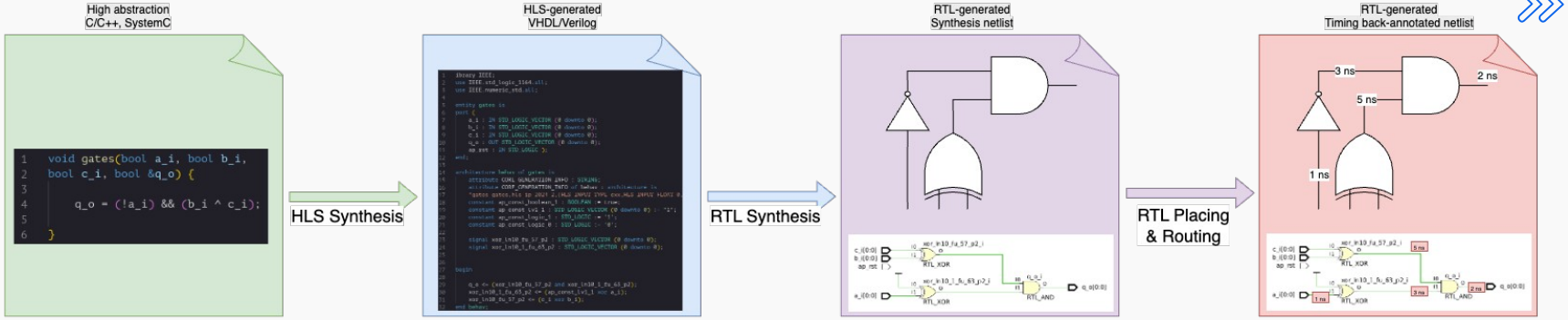
                if (sel_i) {
                q_o = a_i + b_i;
                } else {
                q_o = a_i - b_i;
                }
}
```



Performance & Resource Estimates

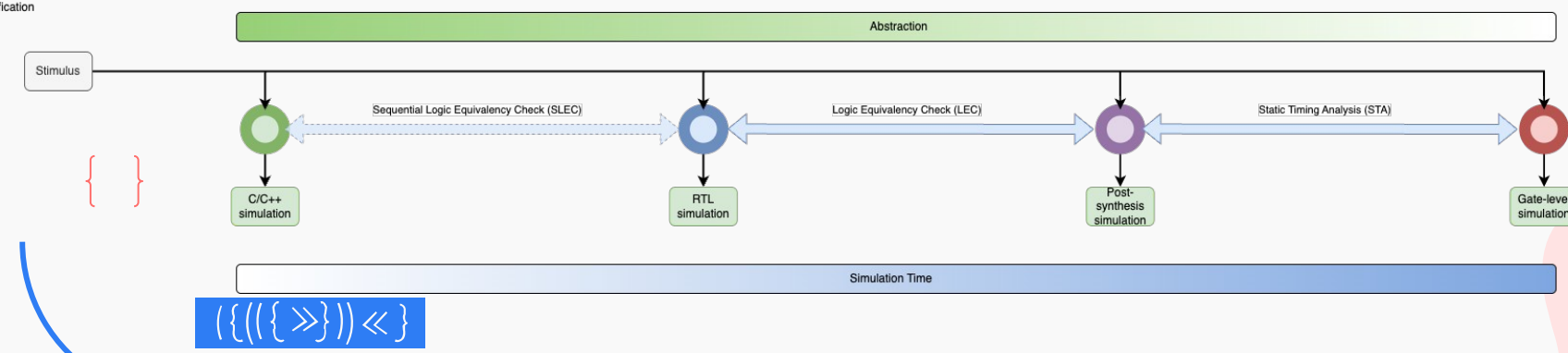
MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
if_else	0	0.0	1	no	0	0	0	110	0

Design flow



Implementation

Verification





02

Quantization

Make it as big as it needs to be

Quantization

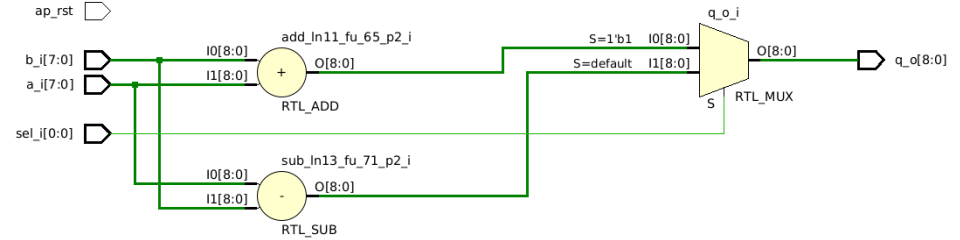
```
#include <ap_int.h>

void if_else(ap_int<8> a_i, ap_int<8> b_i,
        bool sel_i, ap_int<9> &q_o)
{
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=b_i
                #pragma HLS INTERFACE ap_none port=sel_i
                #pragma HLS INTERFACE ap_none port=q_o

                if (sel_i) {
                q_o = a_i + b_i;
                } else {
                q_o = a_i - b_i;
                }
}
```

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
if_else	0	0.0	1	no	0	0	0	41	0



Quantization

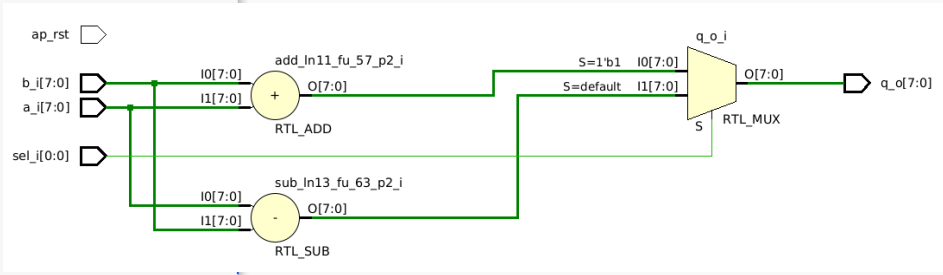
```
#include <ap_int.h>

void if_else(ap_int<8> a_i, ap_int<8> b_i,
        bool sel_i, ap_int<8> &q_o)
{
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=b_i
                #pragma HLS INTERFACE ap_none port=sel_i
                #pragma HLS INTERFACE ap_none port=q_o

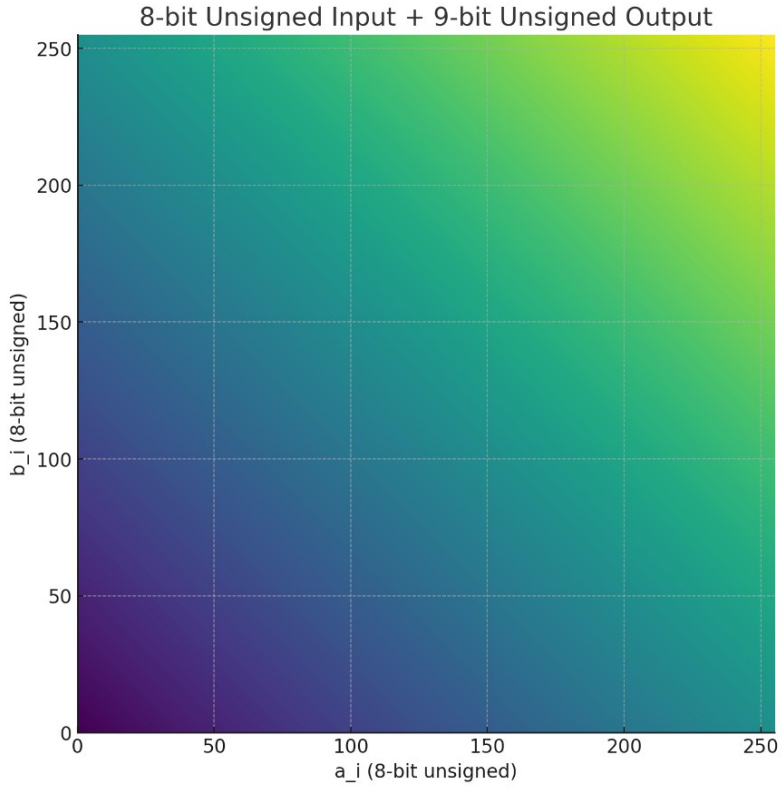
                if (sel_i) {
                q_o = a_i + b_i;
                } else {
                q_o = a_i - b_i;
                }
}
```

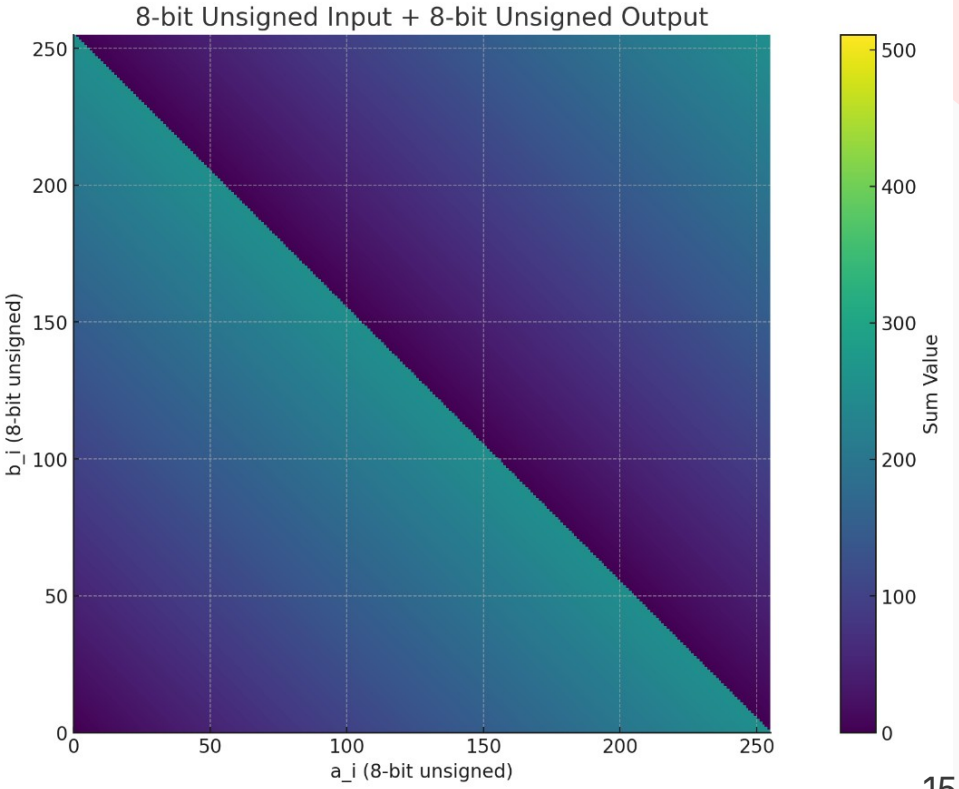
Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
if_else	0	0.0	1	no	0	0	0	38	0

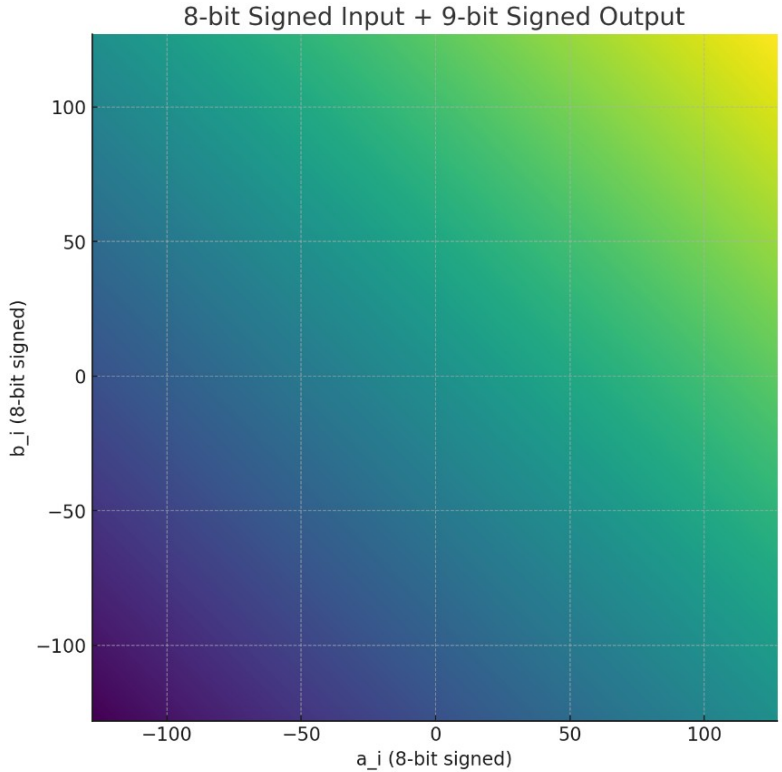


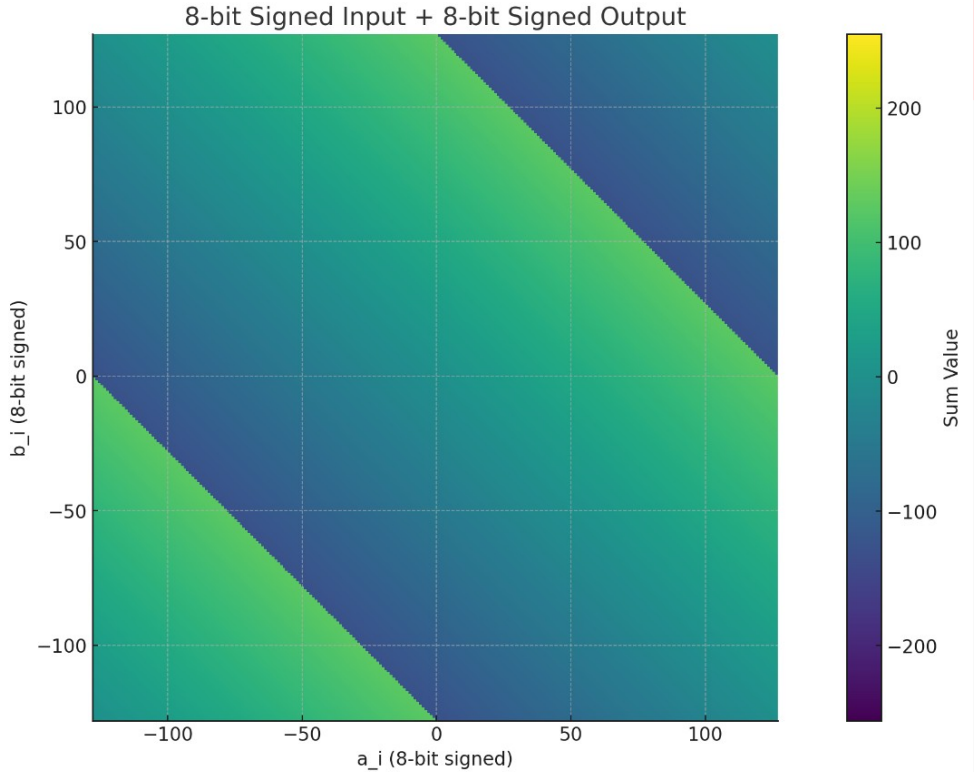
Quantization





Quantization





Quantization

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
if_else	0	0.0	1	no	0	0	0	40	0

TARGET	ESTIMATED	UNCERTAINTY
10.00 ns	1.550 ns	2.70 ns

```
#include <ap_int.h>

void multiplier(ap_int<8> a_i, ap_int<8> b_i,
        ap_int<8> &q_o) {
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=b_i
                #pragma HLS INTERFACE ap_none port=q_o

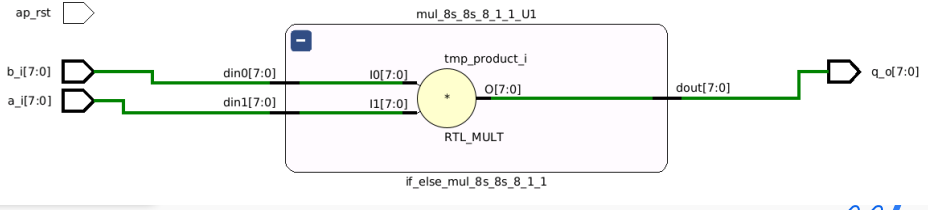
                q_o = a_i * b_i;

}
```

Utilization — Post-Synthesis | Post-Implementation

Graph | Table

Resource	Estimation	Available	Utilization %
LUT	35	1182240	0.01



Quantization

```
#include <ap_int.h>

void if_else(ap_int<8> a_i, ap_int<8> b_i,
        ap_int<16> &q_o) {
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=b_i
                #pragma HLS INTERFACE ap_none port=q_o

                q_o = a_i * b_i;

}
```

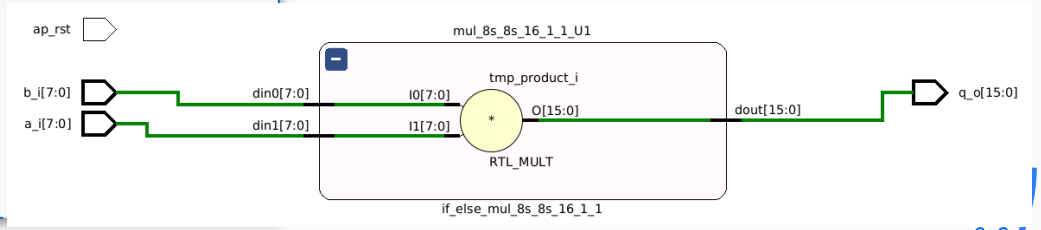
Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
if_else	0	0.0	1	no	0	0	0	40	0

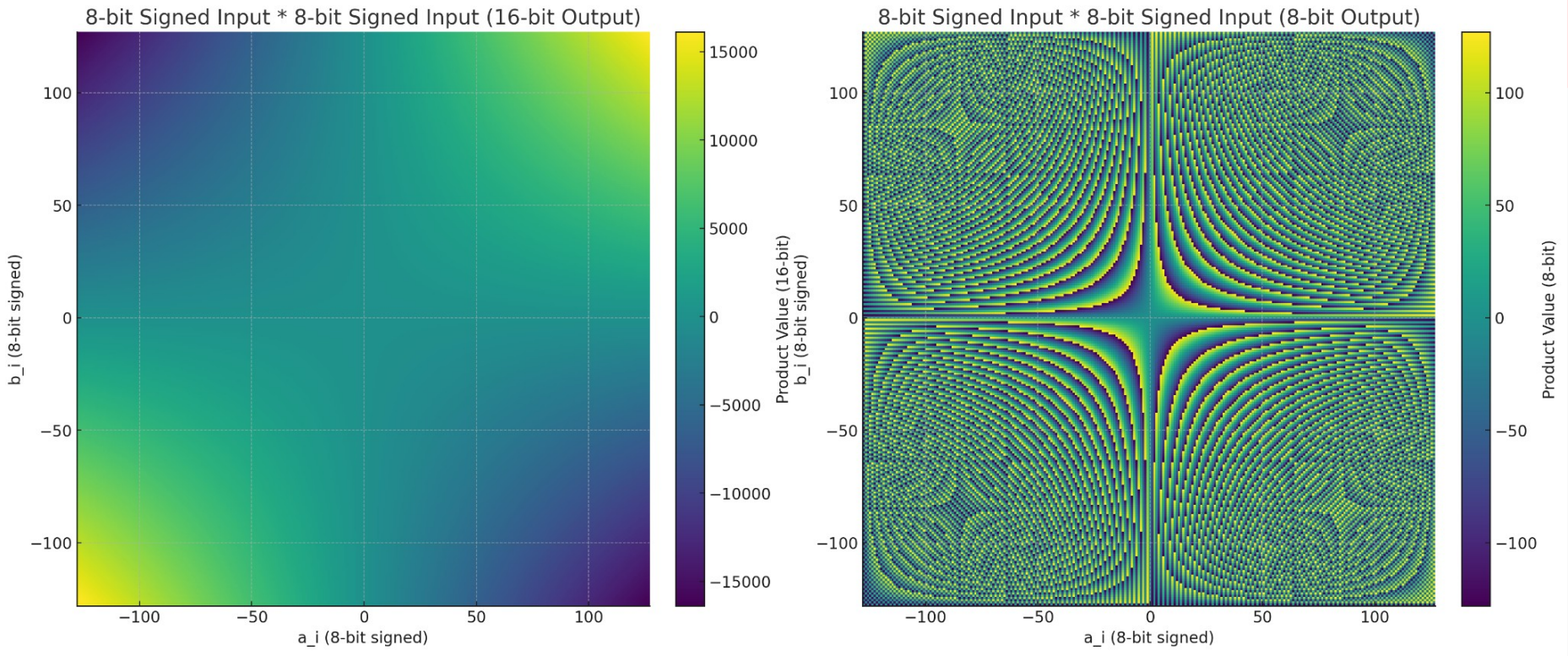
Utilization — **Post-Synthesis** | Post-Implementation

Graph | **Table**

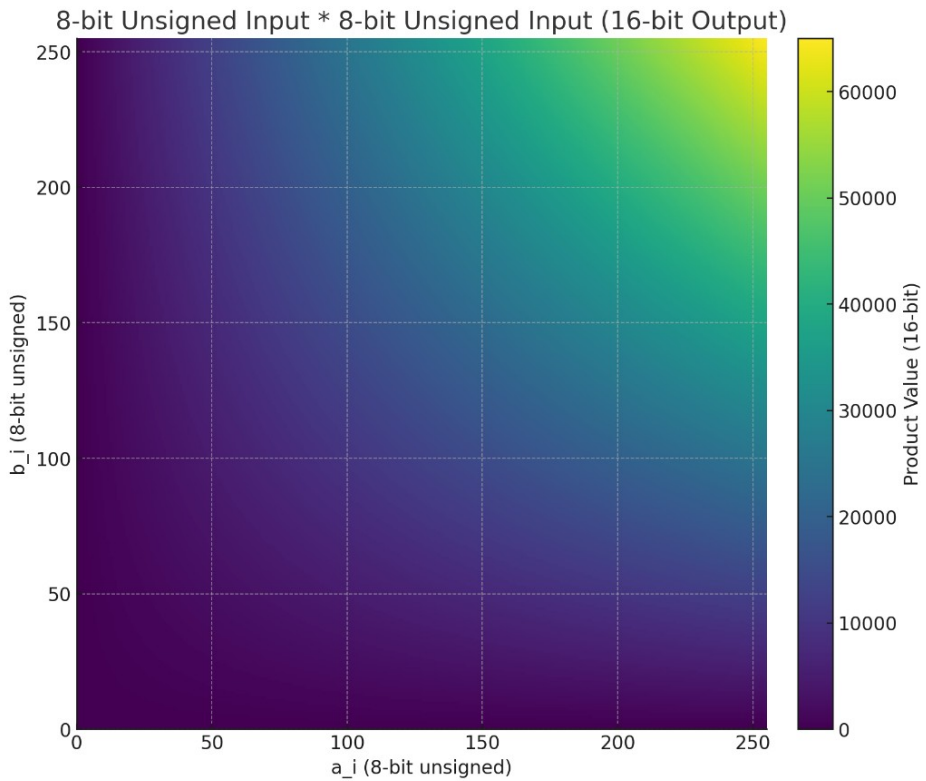
Resource	Estimation	Available	Utilization %
LUT	61	1182240	0.01

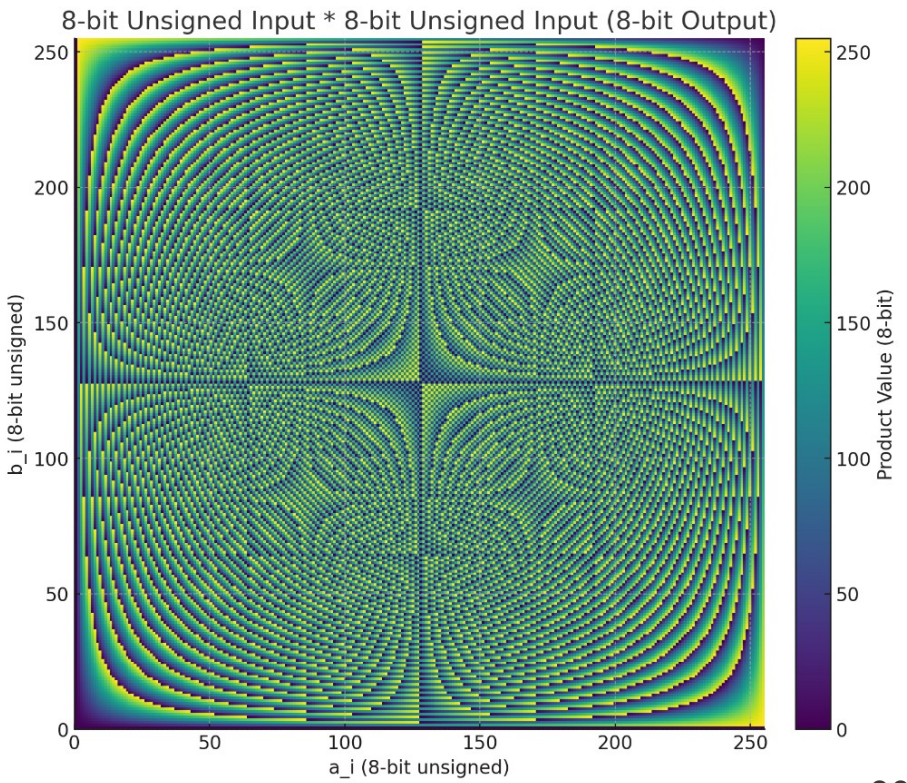


Quantization

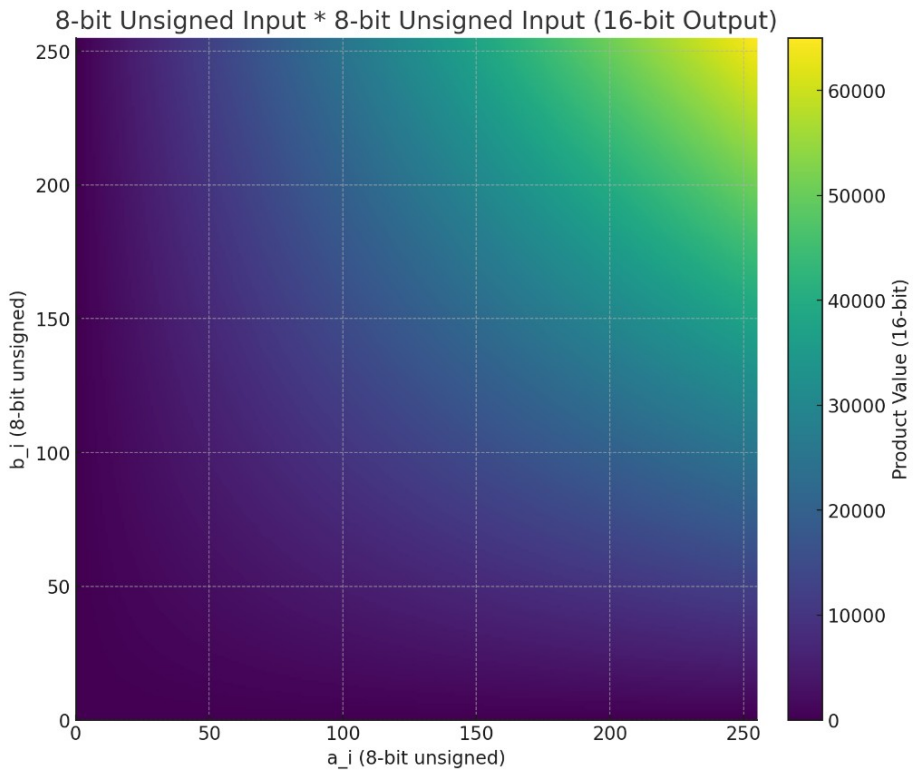


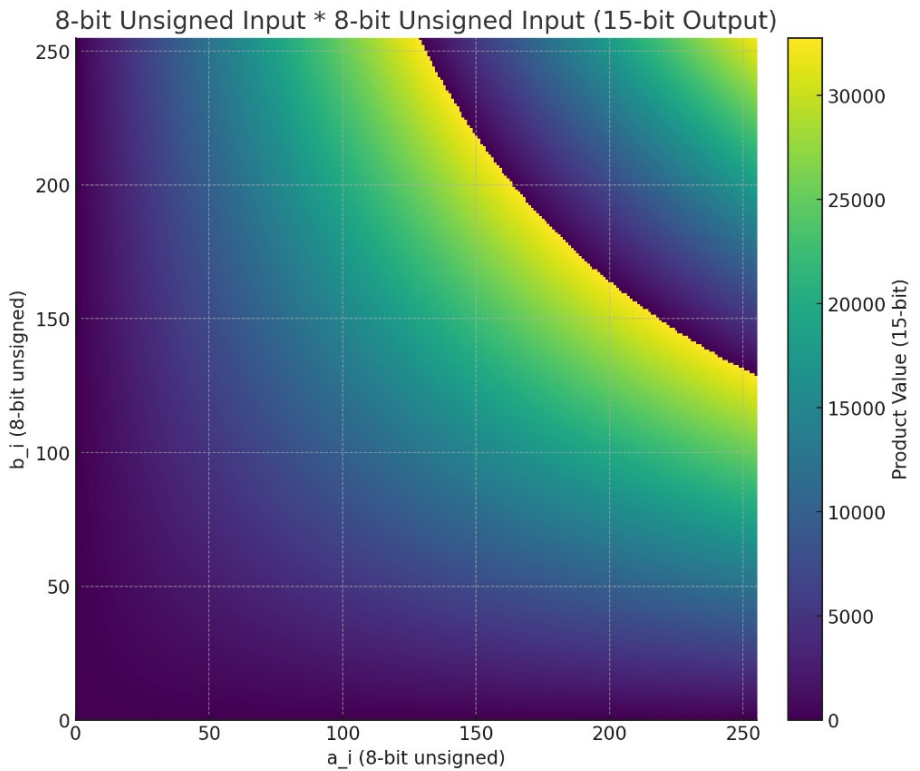
Quantization





Quantization





Recap

To stay on the linear side without overflow, make sure:

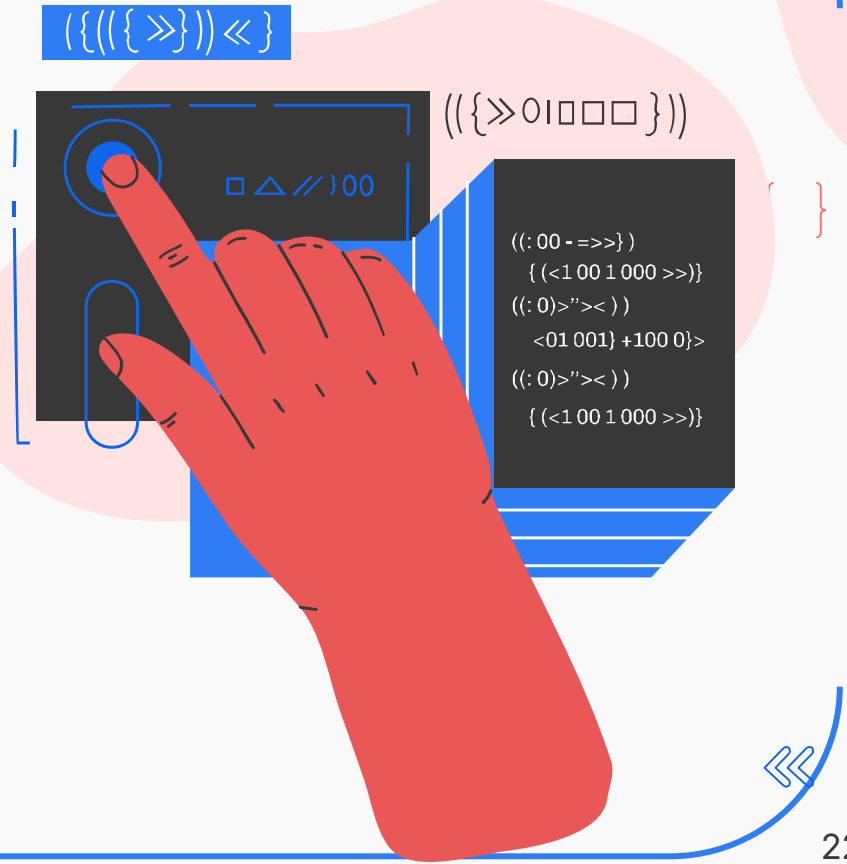


Q=max(A,B)+1



Q=A+B

when A, B, and Q are the width of a_i, b_i, and q_o respectively



03 Sequential logic

Let’s add more fun to it



Registered logic

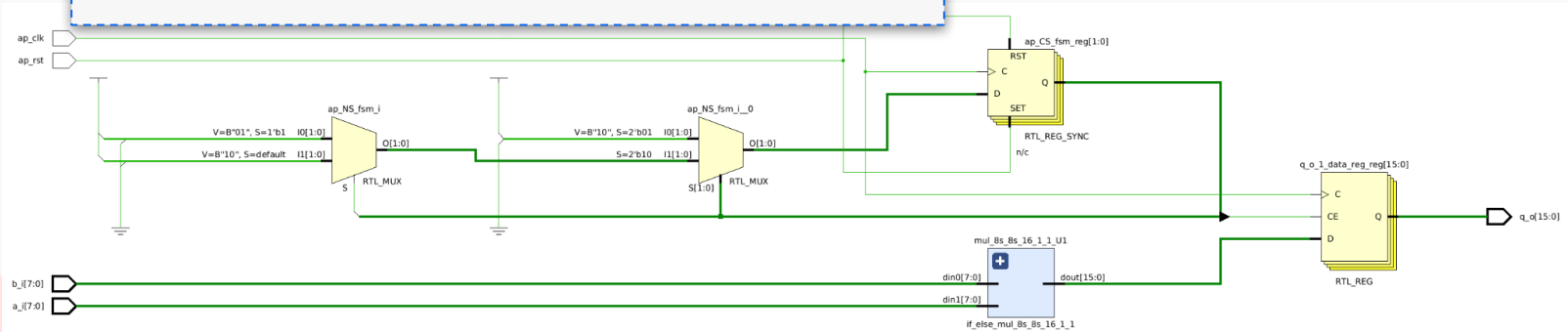
```
#include <ap_int.h>

void multiplier(ap_int<8> a_i, ap_int<8> b_i,
        ap_int<16> &q_o) {
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=b_i
                #pragma HLS INTERFACE ap_none port=q_o
register

                q_o = a_i * b_i;
}
```

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
if_else	1	10.000	2	no	0	0	18	54	0



Registered logic

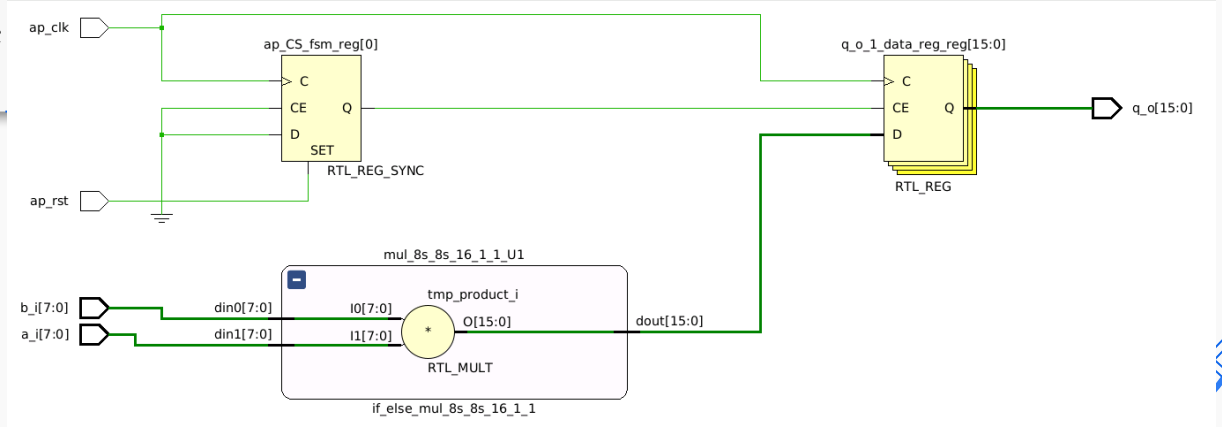
```
#include <ap_int.h>

void multiplier(ap_int<8> a_i, ap_int<8> b_i,
        ap_int<16> &q_o) {
    #pragma HLS PIPELINE
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=b_i
                #pragma HLS INTERFACE ap_none port=q_o
register

                q_o = a_i * b_i;
}
```

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
if_else	1	10.000	1	yes	0	0	18	44	0



Sequential logic

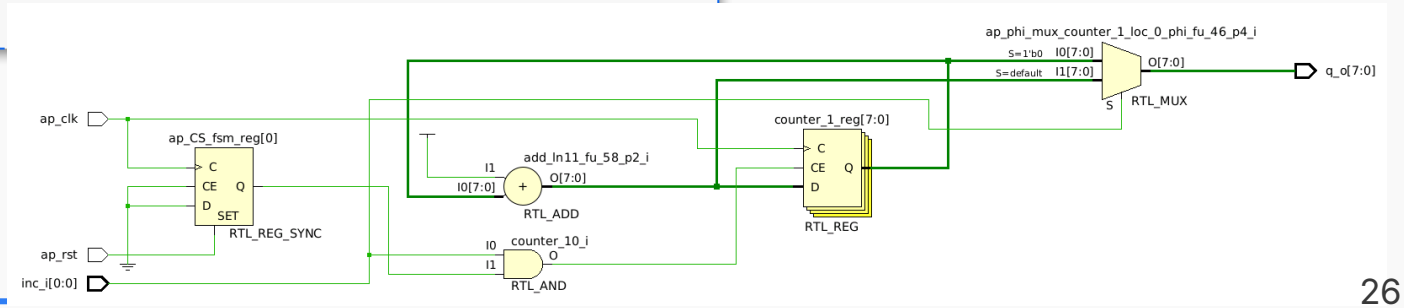
```
#include <ap_int.h>

void counter(bool inc_i, ap_uint<8> &q_o) {
        #pragma HLS INTERFACE ap_ctrl_none port=return
        #pragma HLS INTERFACE ap_none port=inc_i
        #pragma HLS INTERFACE ap_none port=q_o
        // Use a static variable to retain state between calls
        static ap_uint<8> counter = 0;

        if (inc_i) {
        counter++;
        }
        q_o = counter;
}
```

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
counter	0	0.0	1	no	0	0	9	29	0



Multiple input adder v1

```
#include <ap_int.h>

void adder(ap_uint<8> a_i[4], ap_uint<10> &q_o) {
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=q_o

                // Sum the elements, using a 10-bit accumulator
                ap_uint<10> sum = 0;
                for (int i = 0; i < 4; i++) {
                sum += a_i[i];
                }

                q_o = sum;
}
```

HW Interfaces

AP_MEMORY

PORT	DIRECTION	BITWIDTH
a_i_address0	out	2
a_i_q0	in	8

Other Ports

PORT	MODE	DIRECTION	BITWIDTH
q_o	ap_none	out	10

TOP LEVEL CONTROL

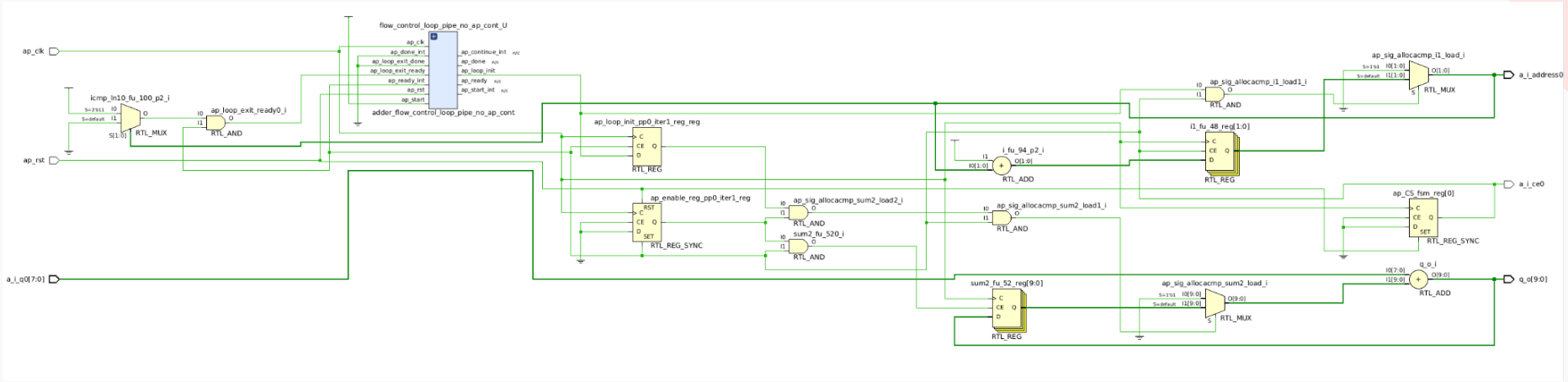
INTERFACE	TYPE	PORTS
ap_clk	clock	ap_clk
ap_rst	reset	ap_rst
ap_ctrl	ap_ctrl_none	

Performance & Resource Estimates

Modules Loops Hide empty columns

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	ITERATION LATENCY	INTERVAL	TRIP COUNT	PIPELINED	BRAM	DSP	FF	LUT	URAM
adder (1)	6	60.000	-	4	-	loop auto-rew	0	0	17	86	0

Multiple input adder v1



Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	24	1182240	0.01
FF	15	2364480	0.01

Multiple input adder v2

```
#include <ap_int.h>

void adder(ap_uint<8> a_i[4], ap_uint<10> &q_o) {
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=q_o
    #pragma HLS ARRAY_PARTITION variable=a_i dim=1 type=complete

                // Sum the elements, using a 10-bit accumulator
                ap_uint<10> sum = 0;
                for (int i = 0; i < 4; i++) {
                sum += a_i[i];
                }

                q_o = sum;
}
```

HW Interfaces

Other Ports

PORT	MODE	DIRECTION	BITWIDTH
a_i_0	ap_none	in	8
a_i_1	ap_none	in	8
a_i_2	ap_none	in	8
a_i_3	ap_none	in	8
q_o	ap_none	out	10

TOP LEVEL CONTROL

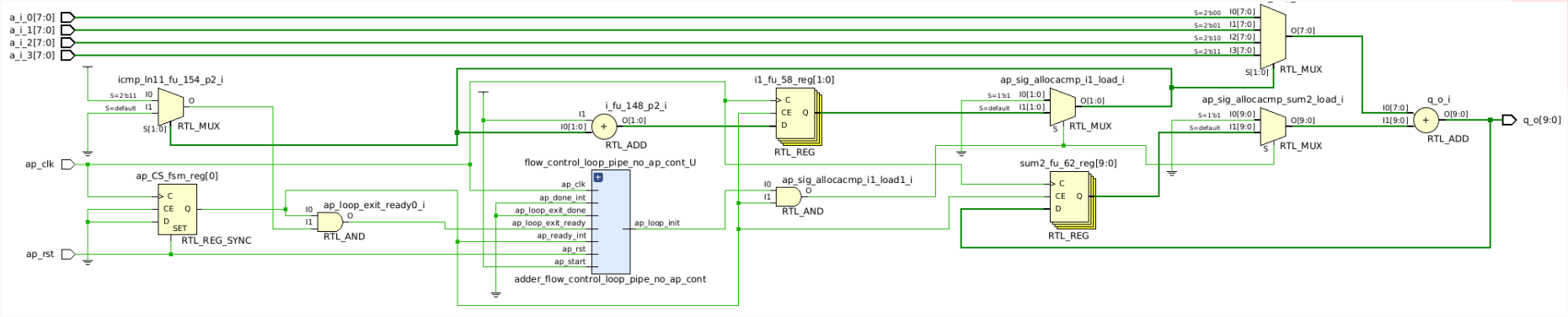
INTERFACE	TYPE	PORTS
ap_clk	clock	ap_clk
ap_rst	reset	ap_rst
ap_ctrl	ap_ctrl_none	

Performance & Resource Estimates

Modules Loops Hide empty columns

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	ITERATION LATENCY	INTERVAL	TRIP COUNT	PIPELINED	BRAM	DSP	FF	LUT	URAM
adder (1)	5	50.000	-	4	-	loop auto-rew	0	0	13	91	0

Multiple input adder v2



Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	45	1182240	0.01
FF	13	2364480	0.01

Multiple input adder v3

```
#include <ap_int.h>

void adder(ap_uint<8> a_i[4], ap_uint<10> &q_o) {
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=q_o
   #pragma HLS ARRAY_PARTITION variable=a_i dim=1 type=complete

                // Sum the elements, using a 10-bit accumulator
                ap_uint<10> sum = 0;
                for (int i = 0; i < 4; i++) {
      #pragma HLS UNROLL factor=2
                sum += a_i[i];
                }

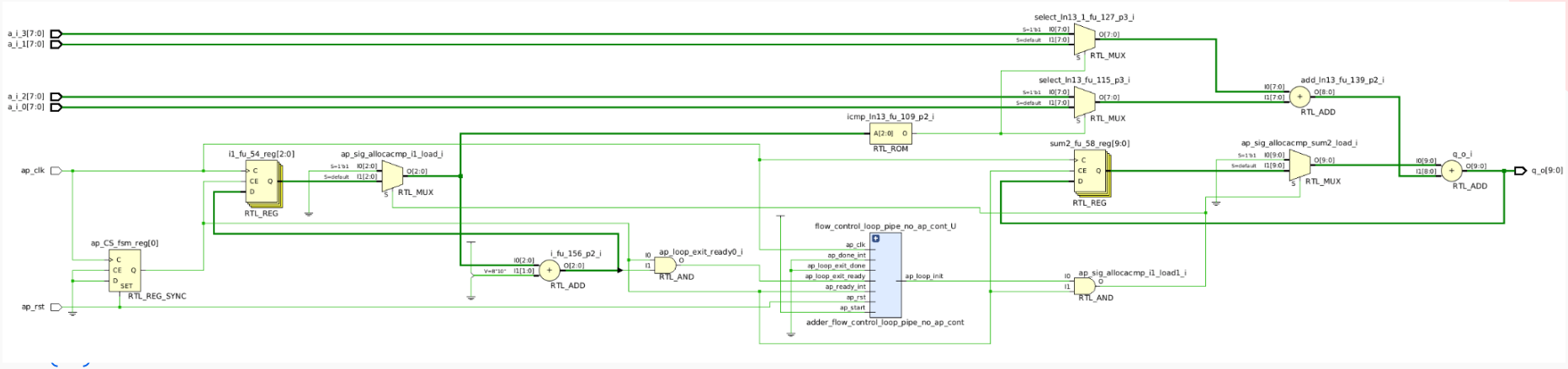
                q_o = sum;
}
```

Performance & Resource Estimates

Modules Loops Hide empty columns

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	ITERATION LATENCY	INTERVAL	TRIP COUNT	PIPELINED	BRAM	DSP	FF	LUT	URAM
adder (1)	3	30.000	-	2	-	loop auto-rew	0	0	14	105	0

Multiple input adder v3



Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	34	1182240	0.01
FF	13	2364480	0.01

Multiple input adder v4

```
#include <ap_int.h>

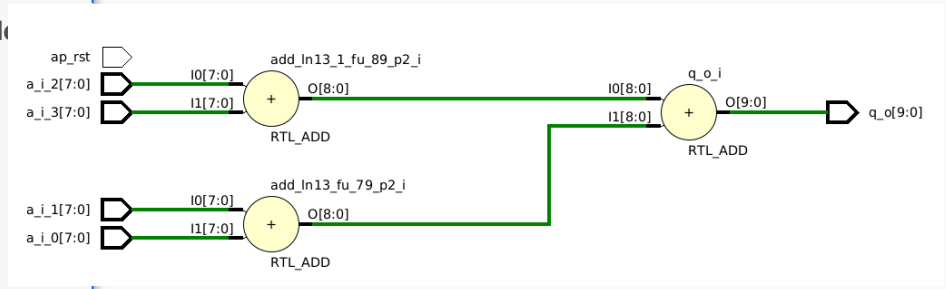
void adder(ap_uint<8> a_i[4], ap_uint<10> &q_o) {
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=q_o
    #pragma HLS ARRAY_PARTITION variable=a_i dim=1 type=compl

                // Sum the elements, using a 10-bit accumulator
                ap_uint<10> sum = 0;
                for (int i = 0; i < 4; i++) {
        #pragma HLS UNROLL
                sum += a_i[i];
                }

                q_o = sum;
}
```

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
adder	0	0.0	1	no	0	0	0	49	0



Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	23	1182240	0.01

Multiple input adder v5

```
#include <ap_int.h>

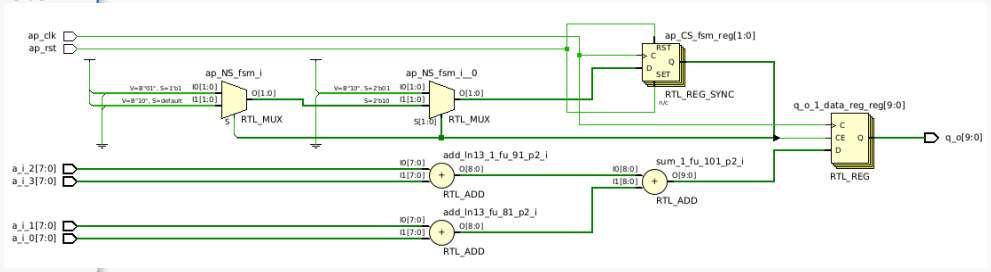
void adder(ap_uint<8> a_i[4], ap_uint<10> &q_o) {
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=q_o register
    #pragma HLS ARRAY_PARTITION variable=a_i dim=1 type=complete

                // Sum the elements, using a 10-bit accumulator
                ap_uint<10> sum = 0;
                for (int i = 0; i < 4; i++) {
        #pragma HLS UNROLL
                sum += a_i[i];
                }

                q_o = sum;
}
```

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
adder	1	10.000	2	no	0	0	12	63	0



Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	24	1182240	0.01
FF	11	2364480	0.01

Multiple input adder v6

```
#include <ap_int.h>

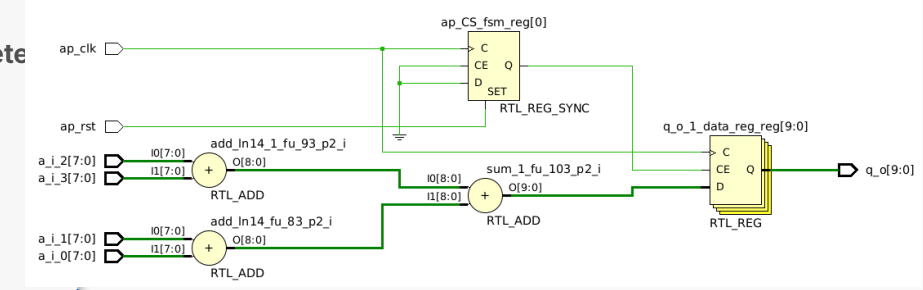
void adder(ap_uint<8> a_i[4], ap_uint<10> &q_o) {
    #pragma HLS PIPELINE
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=q_o register
    #pragma HLS ARRAY_PARTITION variable=a_i dim=1 type=complete

                // Sum the elements, using a 10-bit accumulator
                ap_uint<10> sum = 0;
                for (int i = 0; i < 4; i++) {
        #pragma HLS UNROLL
                sum += a_i[i];
                }

                q_o = sum;
}
```

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
adder	1	10.000	1	yes	0	0	12	53	0



Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	23	1182240	0.01
FF	10	2364480	0.01

FIR v1

```
#include <ap_int.h>

void fir(
                ap_uint<8> a_i,             // Data input
                ap_uint<8> h0_i,            // Coefficient 0
                ap_uint<8> h1_i,            // Coefficient 1
                ap_uint<8> h2_i,            // Coefficient 2
                ap_uint<8> h3_i,            // Coefficient 3
                ap_uint<18> &q_o)           // Output
{
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=h0_i
                #pragma HLS INTERFACE ap_none port=h1_i
                #pragma HLS INTERFACE ap_none port=h2_i
                #pragma HLS INTERFACE ap_none port=h3_i
                #pragma HLS INTERFACE ap_none port=q_o register
```

```
 static ap_uint<8> shift_reg[4] = {0, 0, 0, 0};

                // Shift register: update samples
                shift_reg[3] = shift_reg[2];
                shift_reg[2] = shift_reg[1];
                shift_reg[1] = shift_reg[0];
                shift_reg[0] = a_i;

                // Multiply-accumulate (MAC) with 18-bit
accumulator
                ap_uint<18> acc = 0;
                acc += shift_reg[0] * h0_i;
                acc += shift_reg[1] * h1_i;
                acc += shift_reg[2] * h2_i;
                acc += shift_reg[3] * h3_i;

                q_o = acc;
}
```

FIR v1

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
fir	4	40.000	5	no	0	2	55	136	0

FIR v2

```
#include <ap_int.h>

void fir(
                ap_uint<8> a_i,             // Data input
                ap_uint<8> h0_i,            // Coefficient 0
                ap_uint<8> h1_i,            // Coefficient 1
                ap_uint<8> h2_i,            // Coefficient 2
                ap_uint<8> h3_i,            // Coefficient 3
                ap_uint<18> &q_o)           // Output
{
    #pragma HLS PIPELINE
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=h0_i
                #pragma HLS INTERFACE ap_none port=h1_i
                #pragma HLS INTERFACE ap_none port=h2_i
                #pragma HLS INTERFACE ap_none port=h3_i
                #pragma HLS INTERFACE ap_none port=q_o register
```

```
 static ap_uint<8> shift_reg[4] = {0, 0, 0, 0};

                // Shift register: update samples
                shift_reg[3] = shift_reg[2];
                shift_reg[2] = shift_reg[1];
                shift_reg[1] = shift_reg[0];
                shift_reg[0] = a_i;

                // Multiply-accumulate (MAC) with 18-bit
accumulator
                ap_uint<18> acc = 0;
                acc += shift_reg[0] * h0_i;
                acc += shift_reg[1] * h1_i;
                acc += shift_reg[2] * h2_i;
                acc += shift_reg[3] * h3_i;

                q_o = acc;
}
```

FIR v2

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
fir	4	40.000	1	yes	0	2	95	109	0

Utilization Post-Synthesis | **Post-Implementation**

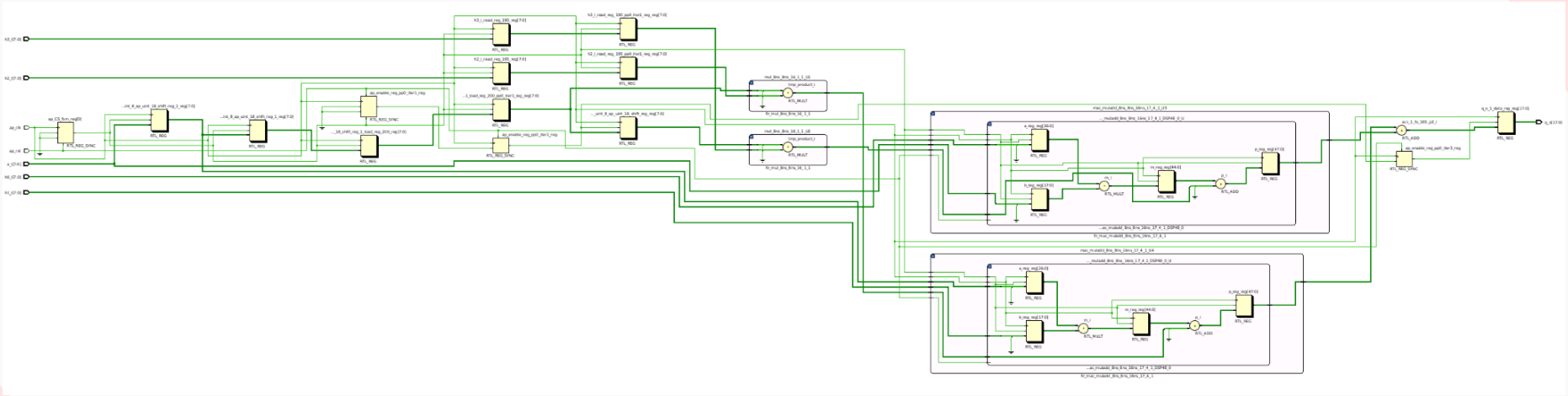
Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	17	1182240	0.01
FF	45	2364480	0.01
DSP	4	6840	0.06

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.318 ns	Worst Hold Slack (WHS):	0.050 ns	Worst Pulse Width Slack (WPWS):	4.725 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	337	Total Number of Endpoints:	337	Total Number of Endpoints:	50

All user specified timing constraints are met.

FIR v2



FIR v3

```
#include <ap_int.h>

void fir(
                ap_uint<8> a_i,
                ap_uint<8> h_i[4],
                ap_uint<18> &q_o)
{
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=h_i
                #pragma HLS INTERFACE ap_none port=q_o register

                static ap_uint<8> shift_reg[4] = {0, 0, 0, 0};
```

```
// Shift register
                for (int i = 3; i > 0; i--) {
                shift_reg[i] = shift_reg[i-1];
                }
                shift_reg[0] = a_i;

                // MAC operation
                ap_uint<18> acc = 0;
                for (int i = 0; i < 4; i++) {
                acc += shift_reg[i] * h_i[i];
                }

                q_o = acc;
}
```

FIR v3

Performance & Resource Estimates

Modules Loops Hide empty columns

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	ITERATION LATENCY	INTERVAL	TRIP COUNT	PIPELINED	BRAM	DSP	FF	LUT	URAM
fir (2)	19	190.000	-	20	-	no	0	1	167	244	0
fir_Pipeline_VITIS_LOOP_16_1 (1)	5	50.000	-	4	-	loop auto-rew	0	0	4	77	0
fir_Pipeline_VITIS_LOOP_23_2 (1)	9	90.000	-	5	-	loop auto-rew	0	1	97	130	0

HW Interfaces

AP_MEMORY

PORT	DIRECTION	BITWIDTH
h_i_address0	out	2
h_i_q0	in	8

Other Ports

PORT	MODE	DIRECTION	BITWIDTH
a_i	ap_none	in	8
q_o	ap_none	out	18

TOP LEVEL CONTROL

INTERFACE	TYPE	PORTS
ap_clk	clock	ap_clk
ap_rst	reset	ap_rst
ap_ctrl	ap_ctrl_none	

FIR v4

```
#include <ap_int.h>

void fir(
                ap_uint<8> a_i,
                ap_uint<8> h_i[4],
                ap_uint<18> &q_o)
{
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=h_i
                #pragma HLS INTERFACE ap_none port=q_o register
    #pragma HLS ARRAY_PARTITION variable=h_i dim=1 type=complete

                static ap_uint<8> shift_reg[4] = {0, 0, 0, 0};
```

```
// Shift register
                for (int i = 3; i > 0; i--) {
                shift_reg[i] = shift_reg[i-1];
                }
                shift_reg[0] = a_i;

                // MAC operation
                ap_uint<18> acc = 0;
                for (int i = 0; i < 4; i++) {
                acc += shift_reg[i] * h_i[i];
                }

                q_o = acc;
}
```

FIR v4

Performance & Resource Estimates

Modules Loops Hide empty columns

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	ITERATION LATENCY	INTERVAL	TRIP COUNT	PIPELINED	BRAM	DSP	FF	LUT	URAM
fir (2)	18	180.000	-	19	-	no	0	1	132	232	0
fir_Pipeline_VITIS_LOOP_17_1 (1)	5	50.000	-	4	-	loop auto-rew	0	0	4	77	0
fir_Pipeline_VITIS_LOOP_24_2 (1)	8	80.000	-	5	-	loop auto-rew	0	1	30	118	0

HW Interfaces

Other Ports

PORT	MODE	DIRECTION	BITWIDTH
a_i	ap_none	in	8
h_i_0	ap_none	in	8
h_i_1	ap_none	in	8
h_i_2	ap_none	in	8
h_i_3	ap_none	in	8
q_o	ap_none	out	18

FIR v5

```
#include <ap_int.h>

void fir(
                ap_uint<8> a_i,
                ap_uint<8> h_i[4],
                ap_uint<18> &q_o)
{
    #pragma HLS PIPELINE
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=h_i
                #pragma HLS INTERFACE ap_none port=q_o register
    #pragma HLS ARRAY_PARTITION variable=h_i dim=1 type=complete

                static ap_uint<8> shift_reg[4] = {0, 0, 0, 0};
```

```
// Shift register
                for (int i = 3; i > 0; i--) {
                shift_reg[i] = shift_reg[i-1];
                }
                shift_reg[0] = a_i;

                // MAC operation
                ap_uint<18> acc = 0;
                for (int i = 0; i < 4; i++) {
                acc += shift_reg[i] * h_i[i];
                }

                q_o = acc;
}
```

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
fir	4	40.000	1	yes	0	2	111	109	0

FIR v6

```
#include <ap_int.h>

void fir(
                ap_uint<8> a_i,
                ap_uint<8> h_i[4],
                ap_uint<18> &q_o)
{
    #pragma HLS PIPELINE
                #pragma HLS INTERFACE ap_ctrl_none port=return
                #pragma HLS INTERFACE ap_none port=a_i
                #pragma HLS INTERFACE ap_none port=h_i
                #pragma HLS INTERFACE ap_none port=q_o register
    #pragma HLS ARRAY_PARTITION variable=h_i dim=1 type=complete

                static ap_uint<8> shift_reg[4] = {0, 0, 0, 0};
```

```
// Shift register
                for (int i = 3; i > 0; i--) {
    #pragma UNROLL
                shift_reg[i] = shift_reg[i-1];
                }
                shift_reg[0] = a_i;

                // MAC operation
                ap_uint<18> acc = 0;
                for (int i = 0; i < 4; i++) {
    #pragma UNROLL
                acc += shift_reg[i] * h_i[i];
                }

                q_o = acc;
}
```

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(CYCLES)	LATENCY(NS)	INTERVAL	PIPELINED	BRAM	DSP	FF	LUT	URAM
fir	4	40.000	1	yes	0	2	111	109	0

FIR v6

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	17	1182240	0.01
FF	37	2364480	0.01
DSP	4	6840	0.06

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.391 ns	Worst Hold Slack (WHS):	0.045 ns	Worst Pulse Width Slack (WPWS):	4.725 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	336	Total Number of Endpoints:	336	Total Number of Endpoints:	42

All user specified timing constraints are met.

