Exercise 1

The following function describes this expression. The function has seven 1-bit input arguments and one output argument.

```
void combinational_circuit_exercise_01(
    bool a,
    bool b,
    bool c,
    bool d,
    bool e,
    bool f,
    bool g,
    bool &h
) {
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE ap_none port=a
#pragma HLS INTERFACE ap_none port=b
#pragma HLS INTERFACE ap_none port=c
#pragma HLS INTERFACE ap_none port=d
#pragma HLS INTERFACE ap_none port=e
#pragma HLS INTERFACE ap_none port=f
#pragma HLS INTERFACE ap_none port=g
#pragma HLS INTERFACE ap_none port=h



      h = (a | (b & c)) ^ (e | (f & g));
}
```

The following figure shows part of the synthesis report.

The propagation delay of the design is about $0.978\ ns$ which is labelled by 1.

The synthesised circuit is combinational as it only utilises LUTs as annotated by 2.

The design ports are also the top-function arguments without any extra signals.

Performance Estimates

Timing

1

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	0.978 ns	1.25 ns

Latency

Utilization Estimates

2

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	10	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	0	0	0	10	0
Available	100	90	41600	20800	0
Utilization (%)	0	0	0	~0	0

Detail

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
a	in	1	ap_none	a	scalar
b	in	1	ap_none	b	scalar
c	in	1	ap_none	c	scalar
d	in	1	ap_none	d	scalar
e	in	1	ap_none	e	scalar
f	in	1	ap_none	f	scalar
g	in	1	ap_none	g	scalar
h	out	1	ap_none	h	pointer

3

Exercise 2

Note that arithmetic operation in HLS is straightforward and we do not need to go into the bit-level implementation. However, to review the concepts explained in this section, I have implemented the 4-bit adders using full-adder here. Later along the course, when we will learn about arbitrary-precision data type and how to use them in implementing arithmetic operations.

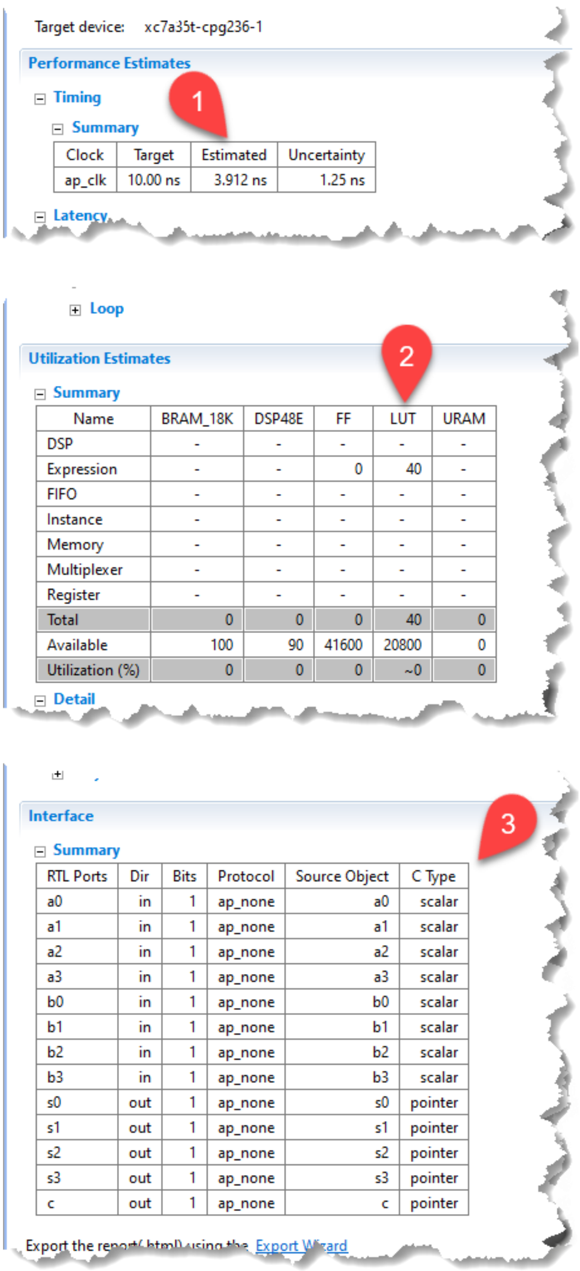
First, we should define a sub-function to implement the full adder.

```
void full_adder(bool a, bool b, bool cin, bool *sum, bool *cout) {
  *sum  = a^b^cin;
  *cout = (a&&b)||(b&&cin)||(a&&cin);
}
```

Then we write the top-function that uses this full adder.

```
void fourbit_adder(
     bool  a0, bool  a1, bool  a2, bool  a3,
     bool  b0, bool  b1, bool  b2, bool  b3,

     bool *s0, bool *s1, bool *s2, bool *s3,
     bool *c
) {

#pragma HLS INTERFACE ap_ctrl_none port=return

#pragma HLS INTERFACE ap_none port=a0
#pragma HLS INTERFACE ap_none port=a1
#pragma HLS INTERFACE ap_none port=a2
#pragma HLS INTERFACE ap_none port=a3

#pragma HLS INTERFACE ap_none port=b0
#pragma HLS INTERFACE ap_none port=b1
#pragma HLS INTERFACE ap_none port=b2
#pragma HLS INTERFACE ap_none port=b3

#pragma HLS INTERFACE ap_none port=s0
#pragma HLS INTERFACE ap_none port=s1
#pragma HLS INTERFACE ap_none port=s2
#pragma HLS INTERFACE ap_none port=s3

#pragma HLS INTERFACE ap_none port=c

  bool c0, c1, c2, c3, c4;

  full_adder(a0, b0,  0, s0, &c0);
  full_adder(a1, b1, c0, s1, &c1);
  full_adder(a2, b2, c1, s2, &c2);
  full_adder(a3, b3, c2, s3, &c3);

  *c = c3;
}
```

This figure shows parts of the synthesis report. As can be seen, the design propagation delay is *3.912 ns*. The design uses only 40 LUTs. Moreover, the design ports only includes the top function arguments.



Target device: xc7a35t-cpg236-1

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	3.912 ns	1.25 ns

Latency

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	40	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	0	0	0	40	0
Available	100	90	41600	20800	0
Utilization (%)	0	0	0	~0	0

Detail

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
a0	in	1	ap_none	a0	scalar
a1	in	1	ap_none	a1	scalar
a2	in	1	ap_none	a2	scalar
a3	in	1	ap_none	a3	scalar
b0	in	1	ap_none	b0	scalar
b1	in	1	ap_none	b1	scalar
b2	in	1	ap_none	b2	scalar
b3	in	1	ap_none	b3	scalar
s0	out	1	ap_none	s0	pointer
s1	out	1	ap_none	s1	pointer
s2	out	1	ap_none	s2	pointer
s3	out	1	ap_none	s3	pointer
c	out	1	ap_none	c	pointer

Export the report(.html) using the Export Wizard

Exercise 3

Before solving this exercise, I should mention that this is an easier way to solve this problem in HLS however because I have not explained all the concepts yet, I only use Boolean data type to represent the data and solve the problem. This also helps us to practice the concepts explained throughout this section.

The following function explains the design. It compares 4-bit two numbers bit by bit starting from the MSB.

```
void comparator(
             bool a0,
             bool a1,
             bool a2,
             bool a3,

             bool b0,
             bool b1,
             bool b2,
             bool b3,

             bool &M,
             bool &N,
             bool &P
             )
{
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE ap_none port=a0
#pragma HLS INTERFACE ap_none port=a1
#pragma HLS INTERFACE ap_none port=a2
#pragma HLS INTERFACE ap_none port=a3


#pragma HLS INTERFACE ap_none port=b0
#pragma HLS INTERFACE ap_none port=b1
#pragma HLS INTERFACE ap_none port=b2
#pragma HLS INTERFACE ap_none port=b3

#pragma HLS INTERFACE ap_none port=M
#pragma HLS INTERFACE ap_none port=N
#pragma HLS INTERFACE ap_none port=P

      if (a3 == b3 ) {
             if (a2 == b2) {
                   if (a1 == b1) {
                          if (a0 == b0) {
```

```
                        M=1;
                        N=0;
                        P=0;
                    } else {
                        if (a0 == 1) {
                            M=0;
                            N=1;
                            P=0;
                        } else {
                            M=0;
                            N=0;
                            P=1;
                        }
                    }
                } else {
                    if (a1 == 1) {
                        M=0;
                        N=1;
                        P=0;
                    } else {
                        M=0;
                        N=0;
                        P=1;
                    }
                }

            } else {
                if (a2 == 1) {
                    M=0;
                    N=1;
                    P=0;
                } else {
                    M=0;
                    N=0;
                    P=1;
                }
            }
        } else {
            if (a3 == 1) {
                M=0;
                N=1;
                P=0;
            } else {
                M=0;
                N=0;
                P=1;
            }
        }
}
```

If we synthesise this code, the following figure shows the report.

Performance Estimates

⊟ **Timing**

⊟ **Summary**

1

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	3.942 ns	1.25 ns

⊟ **Latency**

Utilization Estimates

⊟ **Summary**

2

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	46	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	0	0	0	46	0
Available	100	90	41600	20800	0
Utilization (%)	0	0	0	~0	0

⊟ **Detail**

Interface

3

⊟ **Summary**

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
a0	in	1	ap_none	a0	scalar
a1	in	1	ap_none	a1	scalar
a2	in	1	ap_none	a2	scalar
a3	in	1	ap_none	a3	scalar
b0	in	1	ap_none	b0	scalar
b1	in	1	ap_none	b1	scalar
b2	in	1	ap_none	b2	scalar
b3	in	1	ap_none	b3	scalar
M	out	1	ap_none	M	pointer
N	out	1	ap_none	N	pointer
P	out	1	ap_none	P	pointer

1) The propagation delay is about 3.942 ns
2) The synthesised design is combinational as it only utilises LUTs.
3) In addition, the design ports are only the top-function arguments.