DK2

Using the logic estimator

Authors: SB

Document number: 1

Customer Support at http://www.celoxica.com/support/

Celoxica in Europe	**Celoxica in Japan**	**Celoxica in Asia Pacific**	**Celoxica in the Americas**
T: +44 (0) 1235 863 656	T: +81 (0) 45 331 0218	T: +65 6896 4838	T: +1 800 570 7004
E: sales.emea@celoxica.com	E: sales.japan@celoxica.com	E: sales.apac@celoxica.com	E: sales.america@celoxica.com

Contents

Conventions

A number of conventions are used in this document. These conventions are detailed below.

Warning Message. These messages warn you that actions may damage your hardware.

Handy Note. These messages draw your attention to crucial pieces of information.

Hexadecimal numbers will appear throughout this document. The convention used is that of prefixing the number with '0x' in common with standard C syntax.

Sections of code or commands that you must type are given in typewriter font like this:

```
void main();
```

Information about a type of object you must specify is given in italics like this:

copy ***SourceFileName DestinationFileName***

Optional elements are enclosed in square brackets like this:

```
struct [type_Name]
```

Curly brackets around an element show that it is optional but it may be repeated any number of times.

string ::= "{***character***}"

1 Tutorial: Using the logic estimator

The following examples illustrate the use of the DK Logic Estimator to produce smaller and faster designs. A basic knowledge of Handel-C is assumed, and some knowledge of digital electronics and design techniques will also be helpful.

New users are recommended to work through the following topics in order:

Enabling the logic estimator (see page 3)

Using the logic estimator results (see page 4)

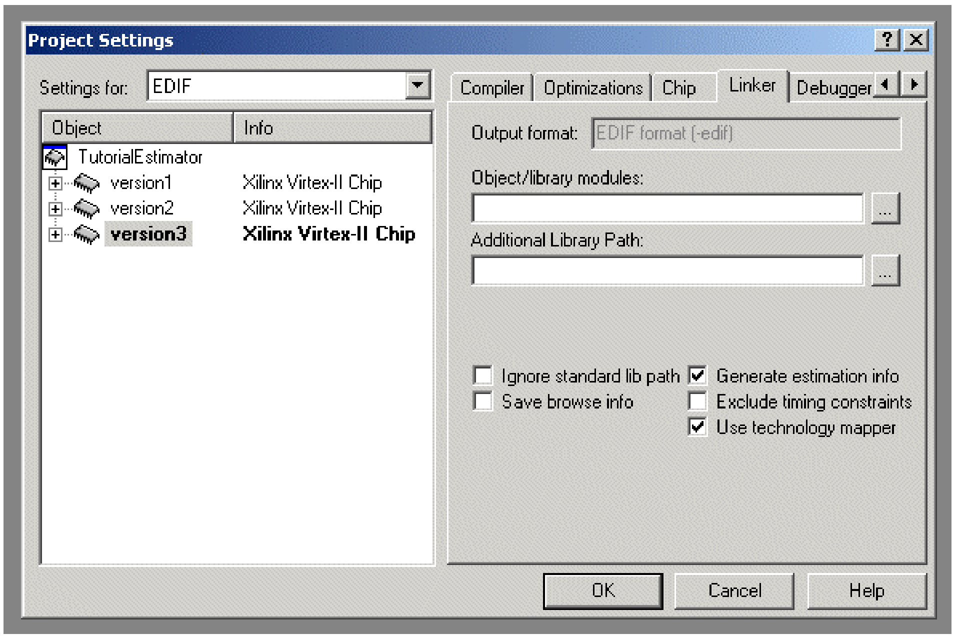
Reducing the logic delay (see page 7)

Reducing the logic area (see page 12)

1.1 Enabling the logic estimator

The logic estimator is a tool included in DK which generates an HTML based report on the expected logic area and delay of the Handel-C code in the current project. This information can be very useful to increase the speed and reduce the size of a Handel-C design. Further information on the detailed operation of the logic estimator can be found in the DK online help, the information below is to instruct you in enabling it correctly, and the link at the end will take you to information on using the results it generates.

To enable the logic estimator for a given project, select the **Project->Settings** menu, and select the **Linker** tab. Make sure that the **Settings for** drop-down list is set to **EDIF**. Check the boxes for **Generate estimation info** and **Use technology mapper**, as shown below:



ENABLING THE LOGIC ESTIMATOR

Enabling the technology mapper allows the logic estimator to produce more accurate results. If the mapper is not enabled, logic estimation can still be used, but the timing and resource usage information will be expressed in general terms, rather specific components and delays.

Next: ***Using the logic estimator results*** (see page 4)

1.2 Using the logic estimator results

The results from the logic estimator can help you to improve the speed and reduce the size of a Handel-C design. The following examples show you how to do this, the code is contained in the **TutorialEstimator** workspace, and the screenshots are from this workspace.

The **version1** project in the **TutorialEstimator** workspace contains the following simple piece of code:

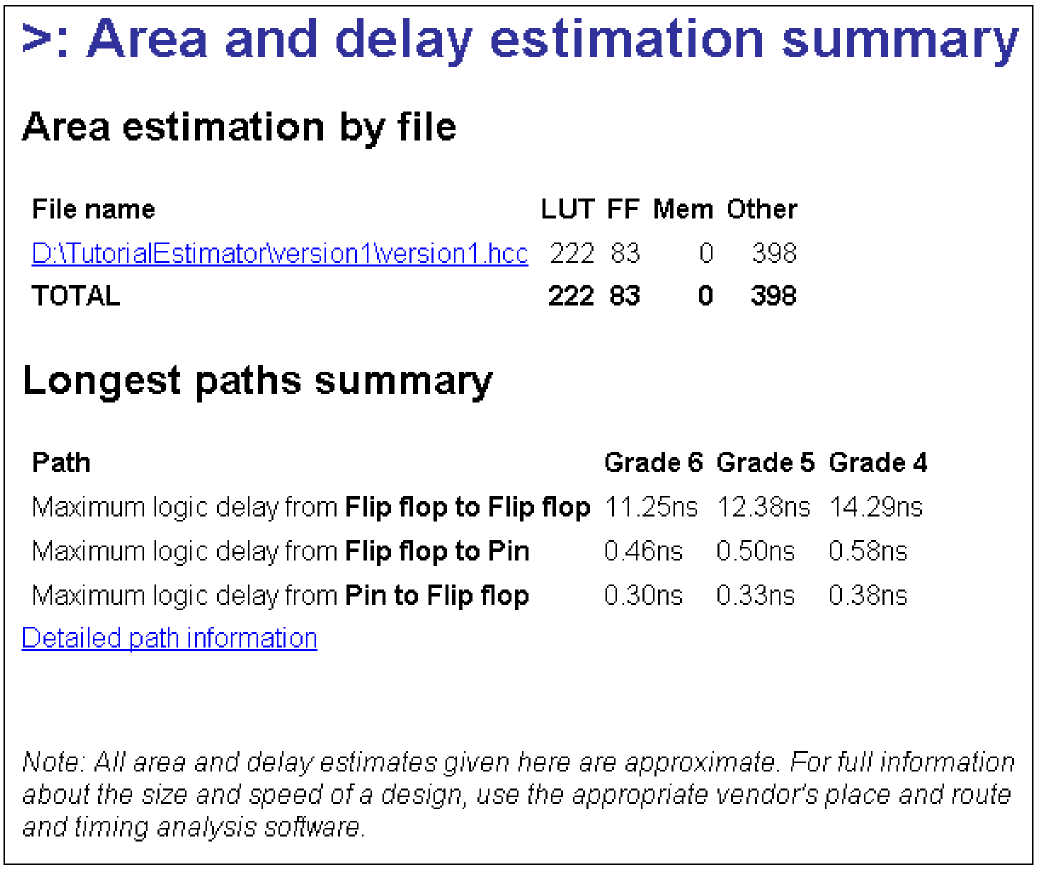
```
set clock = external;
void main(void)
{
   interface bus_in(unsigned 16) InBusA();
   interface bus_in(unsigned 16) InBusB();
   unsigned 16 A, B, C, D, Output;
   unsigned 32 Index;
   interface bus_out() OutBus(Output);

   while(1)
   {
      par
      {
         A = InBusA.in;
         B = InBusB.in;
         Index = 0;
      }

      do
      {
         par
         {
            C = A * B;
            D = A + B;
         }
         par
         {
            Output = C + D;
            Index++;
         }
      } while (Index < 10000);
   }
}
```

Build the above code in the **version1** project in the **TutorialEstimator** workspace. The logic estimator will save its results in the **TutorialEstimator\version1\EDIF** directory. Open the file named **Summary.html** by double-clicking on it (this should load your computers default web browser).

It should appear as below:



>: Area and delay estimation summary

Area estimation by file

File name	LUT	FF	Mem	Other
D:\TutorialEstimator\version1\version1.hcc	222	83	0	398
TOTAL	**222**	**83**	**0**	**398**

Longest paths summary

Path	Grade 6	Grade 5	Grade 4
Maximum logic delay from **Flip flop to Flip flop**	11.25ns	12.38ns	14.29ns
Maximum logic delay from **Flip flop to Pin**	0.46ns	0.50ns	0.58ns
Maximum logic delay from **Pin to Flip flop**	0.30ns	0.33ns	0.38ns

Detailed path information

Note: All area and delay estimates given here are approximate. For full information about the size and speed of a design, use the appropriate vendor's place and route and timing analysis software.

ESTIMATION SUMMARY FROM VERSION1 PROJECT

The first section of the summary provides an estimation of the logic area, described in terms of LUTs, FFs, memory bits and miscellaneous other components. The numbers of these components are listed per source file in the project, with a total at the end. Clicking on the link to the source file will take you to a page providing more detail on how the logic area is distributed within the source file.

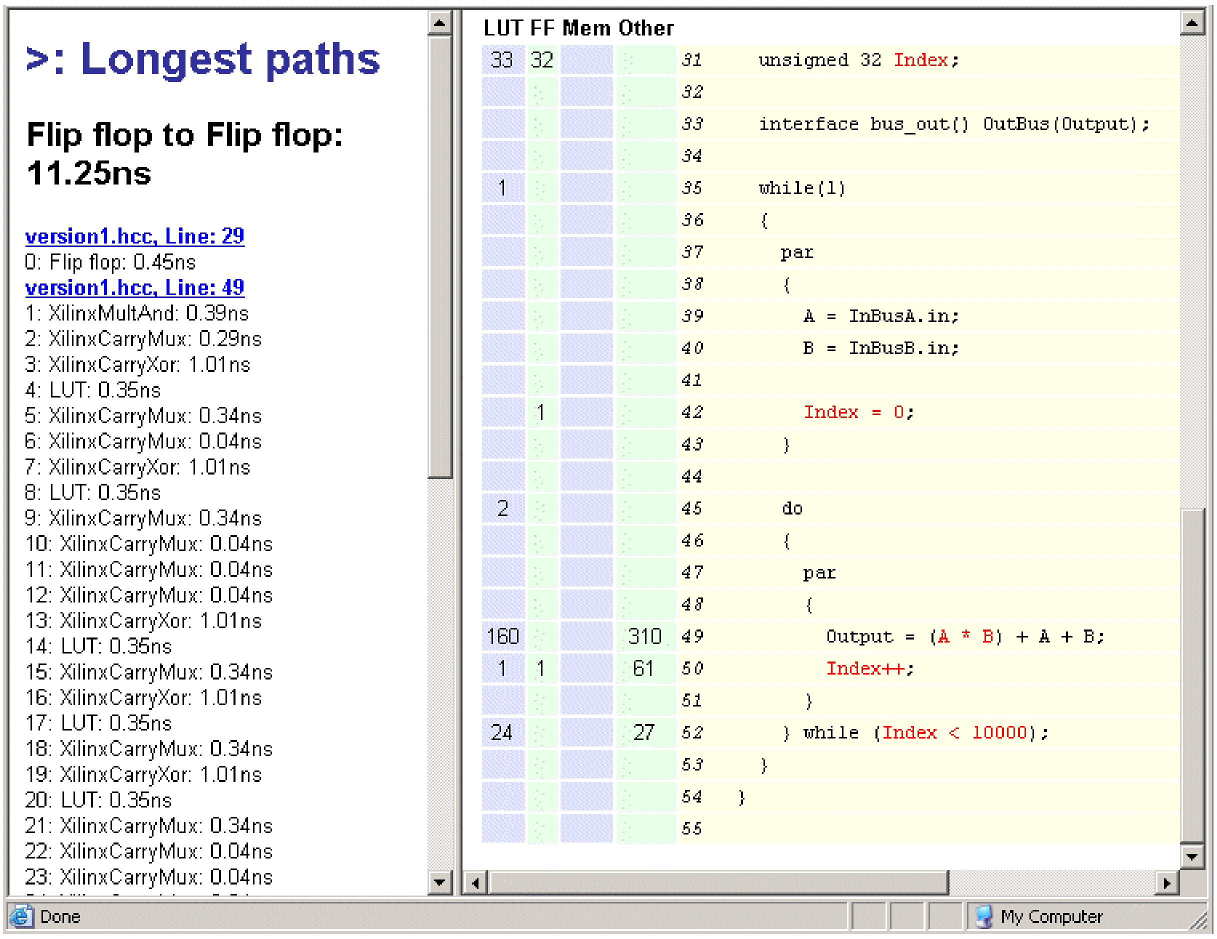
The second section of the summary provides an estimate of the logic delay for the project, giving times for the available speed grades for the selected target device. Clicking on the link to **Detailed path information** takes you to a page showing which lines of Handel-C source code contributed to the longest path in the design. It is important to note that the delays shown here are purely for logic elements, and do not make any allowances for routing when the design is implemented in an FPGA. The exact delay can only be found by implementing the design using the FPGA vendors Place and Route tools. An approximate value for the total delay can be obtained by doubling the logic delay, as in many designs the logic and routing delays are roughly equal.

Note that if the Technology Mapper is not turned on (as described in ***Enabling the logic estimator*** (see page 3)), the information provided will not be as detailed or accurate as that shown here.

The following sections include instructions for reducing the logic delay and area of the design in the **version1** project in the **TutorialEstimator** workspace.

1.3 Reducing the logic delay

Build the above code in the **version1** project in the **TutorialEstimator** workspace. The logic estimator will save its results in the **TutorialEstimator\version1\EDIF** directory. Open the file named **Summary.html** by double-clicking on it (this should load your computers default web browser). Click on the link to Detailed path information, which will take you to a page showing which lines of Handel-C source code contributed to the longest path in the design, as shown below:



>: Longest paths

Flip flop to Flip flop: 11.25ns

version1.hcc, Line: 29
0: Flip flop: 0.45ns
version1.hcc, Line: 49
1: XilinxMultAnd: 0.39ns
2: XilinxCarryMux: 0.29ns
3: XilinxCarryXor: 1.01ns
4: LUT: 0.35ns
5: XilinxCarryMux: 0.34ns
6: XilinxCarryMux: 0.04ns
7: XilinxCarryXor: 1.01ns
8: LUT: 0.35ns
9: XilinxCarryMux: 0.34ns
10: XilinxCarryMux: 0.04ns
11: XilinxCarryMux: 0.04ns
12: XilinxCarryMux: 0.04ns
13: XilinxCarryXor: 1.01ns
14: LUT: 0.35ns
15: XilinxCarryMux: 0.34ns
16: XilinxCarryXor: 1.01ns
17: LUT: 0.35ns
18: XilinxCarryMux: 0.34ns
19: XilinxCarryXor: 1.01ns
20: LUT: 0.35ns
21: XilinxCarryMux: 0.34ns
22: XilinxCarryMux: 0.04ns
23: XilinxCarryMux: 0.04ns

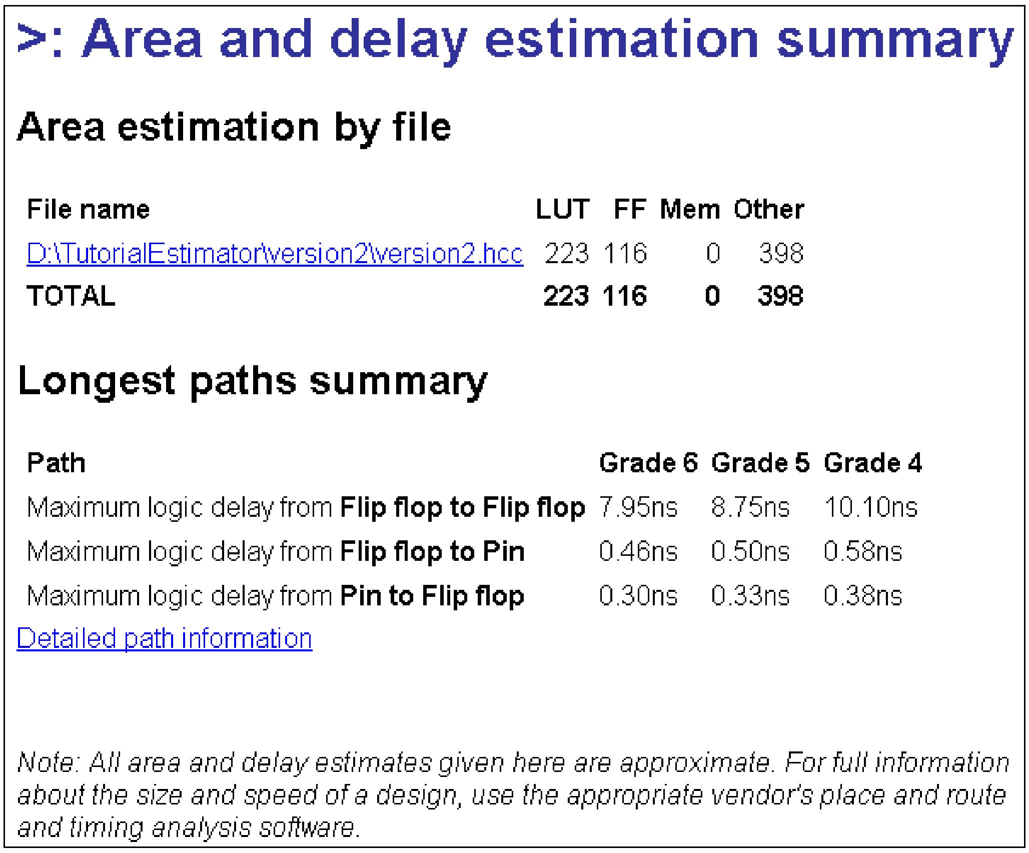
```
LUT FF  Mem Other
 33 32            31    unsigned 32 Index;
                  32
                  33    interface bus_out() OutBus(Output);
                  34
  1               35    while(1)
                  36    {
                  37      par
                  38      {
                  39        A = InBusA.in;
                  40        B = InBusB.in;
                  41
     1            42        Index = 0;
                  43      }
                  44
  2               45      do
                  46      {
                  47        par
                  48        {
160          310  49          Output = (A * B) + A + B;
  1  1        61  50          Index++;
                  51        }
 24           27  52      } while (Index < 10000);
                  53    }
                  54  }
                  55
```

Done   My Computer

This information shows that most of the logic in the longest path is on line number 49 in the Handel-C source, and also that this line is associated with a large number of LUTs and other logic elements. The high logic delay is due to line 49 including a multiply and two adds in a single cycle, and it can be reduced by creating two extra variables **C** and **D**, and performing the calculation over two cycles, as shown below:

```
do
{
    par
    {
        C = A * B;
        D = A + B;
    }
    par
    {
        Output = C + D;
        Index++;
    }
} while (Index < 10000);
```

The **while()** loop now takes two cycles to execute, but the logic delay has been reduced from 14.29ns to 10.1ns (for a grade 4 part), as shown in the new estimation summary below (comapre to results in ***Using the logic estimator results*** (see page 4)). You can generate this summary yourself by building the **version2** project in the **TutorialEstimator** workspace. The logic estimator will save its results in the **TutorialEstimator\version2\EDIF** directory. Open the file named **Summary.html** by double-clicking on it (this should load your computers default web browser).



>: Area and delay estimation summary

Area estimation by file

File name	LUT	FF	Mem	Other
D:\TutorialEstimator\version2\version2.hcc	223	116	0	398
TOTAL	**223**	**116**	**0**	**398**

Longest paths summary

Path	Grade 6	Grade 5	Grade 4
Maximum logic delay from **Flip flop to Flip flop**	7.95ns	8.75ns	10.10ns
Maximum logic delay from **Flip flop to Pin**	0.46ns	0.50ns	0.58ns
Maximum logic delay from **Pin to Flip flop**	0.30ns	0.33ns	0.38ns

Detailed path information

Note: All area and delay estimates given here are approximate. For full information about the size and speed of a design, use the appropriate vendor's place and route and timing analysis software.

ESTIMATION SUMMARY FROM VERSION2 PROJECT

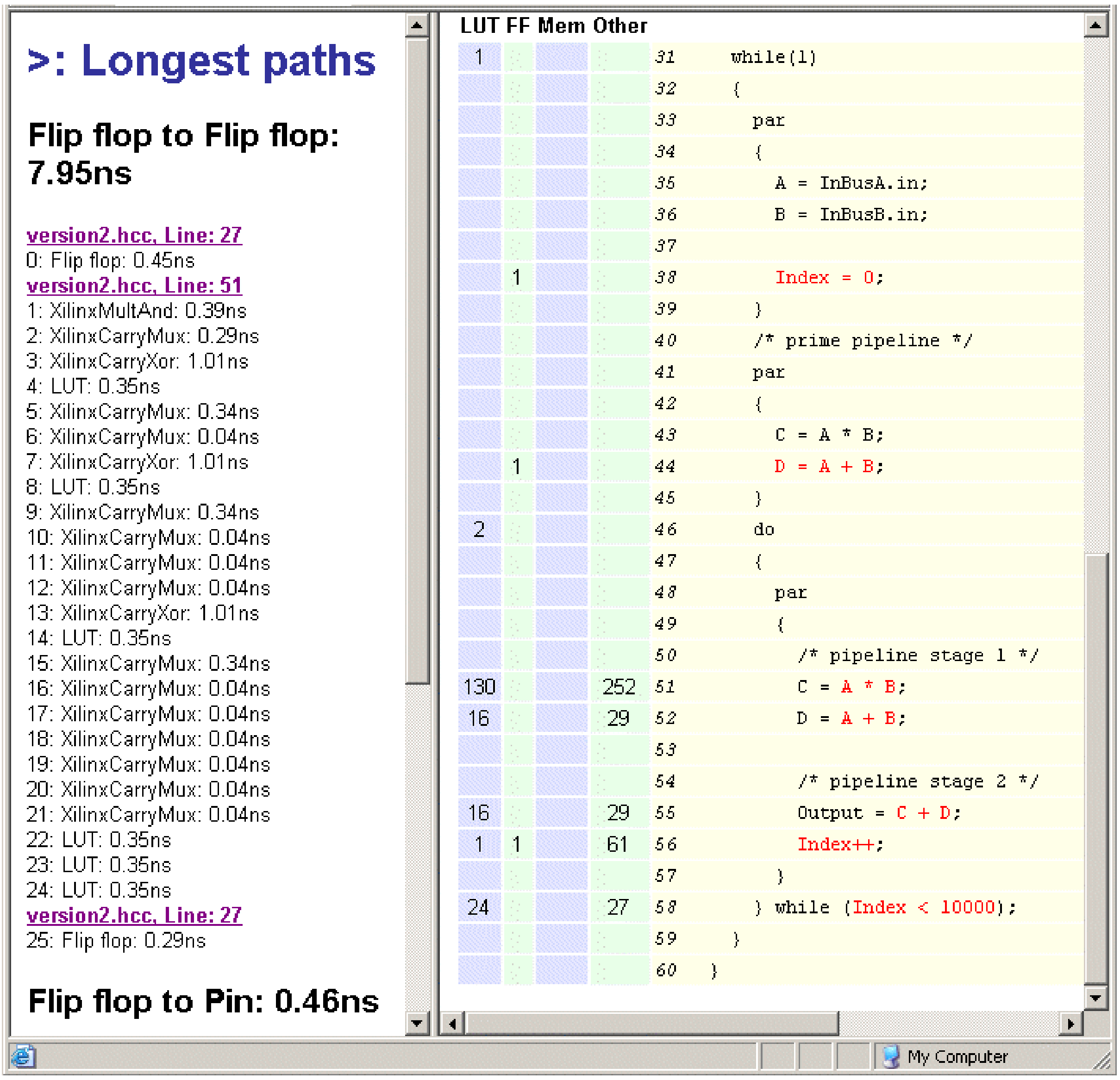
The code can be altered to allow the loop to execute in one cycle again by implementing a two-stage pipeline, where the first stage calculates the values of **C** and **D**, and the second stage adds them together. The pipeline must be primed before the **while()** loop begins executing, as shown below:

```
/* prime pipeline */
par
{
    C = A * B;
    D = A + B;
}

do
{
    par
    {
        /* pipeline stage 1 */
        C = A * B;
        D = A + B;

        /* pipeline stage 2 */
        Output = C + D;
        Index++;
    }
} while (Index < 10000);
```

Try modifying the code in the **version2** project in the **TutorialEstimator** workspace to use this pipeline, rebuild it, and open the estimation summary again. You will see that the logic delay is unchanged, and there has been no significant change in the number of LUTs or other logic elements used, despite calculating the values for **C** and **D** in two separate places. This is because the optimizations in DK include identifying common expressions which do not execute at the same time, and sharing hardware between them. The detail on the new longest paths when using the pipeline are shown below:



>: Longest paths

Flip flop to Flip flop: 7.95ns

version2.hcc, Line: 27
0: Flip flop: 0.45ns
version2.hcc, Line: 51
1: XilinxMultAnd: 0.39ns
2: XilinxCarryMux: 0.29ns
3: XilinxCarryXor: 1.01ns
4: LUT: 0.35ns
5: XilinxCarryMux: 0.34ns
6: XilinxCarryMux: 0.04ns
7: XilinxCarryXor: 1.01ns
8: LUT: 0.35ns
9: XilinxCarryMux: 0.34ns
10: XilinxCarryMux: 0.04ns
11: XilinxCarryMux: 0.04ns
12: XilinxCarryMux: 0.04ns
13: XilinxCarryXor: 1.01ns
14: LUT: 0.35ns
15: XilinxCarryMux: 0.34ns
16: XilinxCarryMux: 0.04ns
17: XilinxCarryMux: 0.04ns
18: XilinxCarryMux: 0.04ns
19: XilinxCarryMux: 0.04ns
20: XilinxCarryMux: 0.04ns
21: XilinxCarryMux: 0.04ns
22: LUT: 0.35ns
23: LUT: 0.35ns
24: LUT: 0.35ns
version2.hcc, Line: 27
25: Flip flop: 0.29ns

Flip flop to Pin: 0.46ns

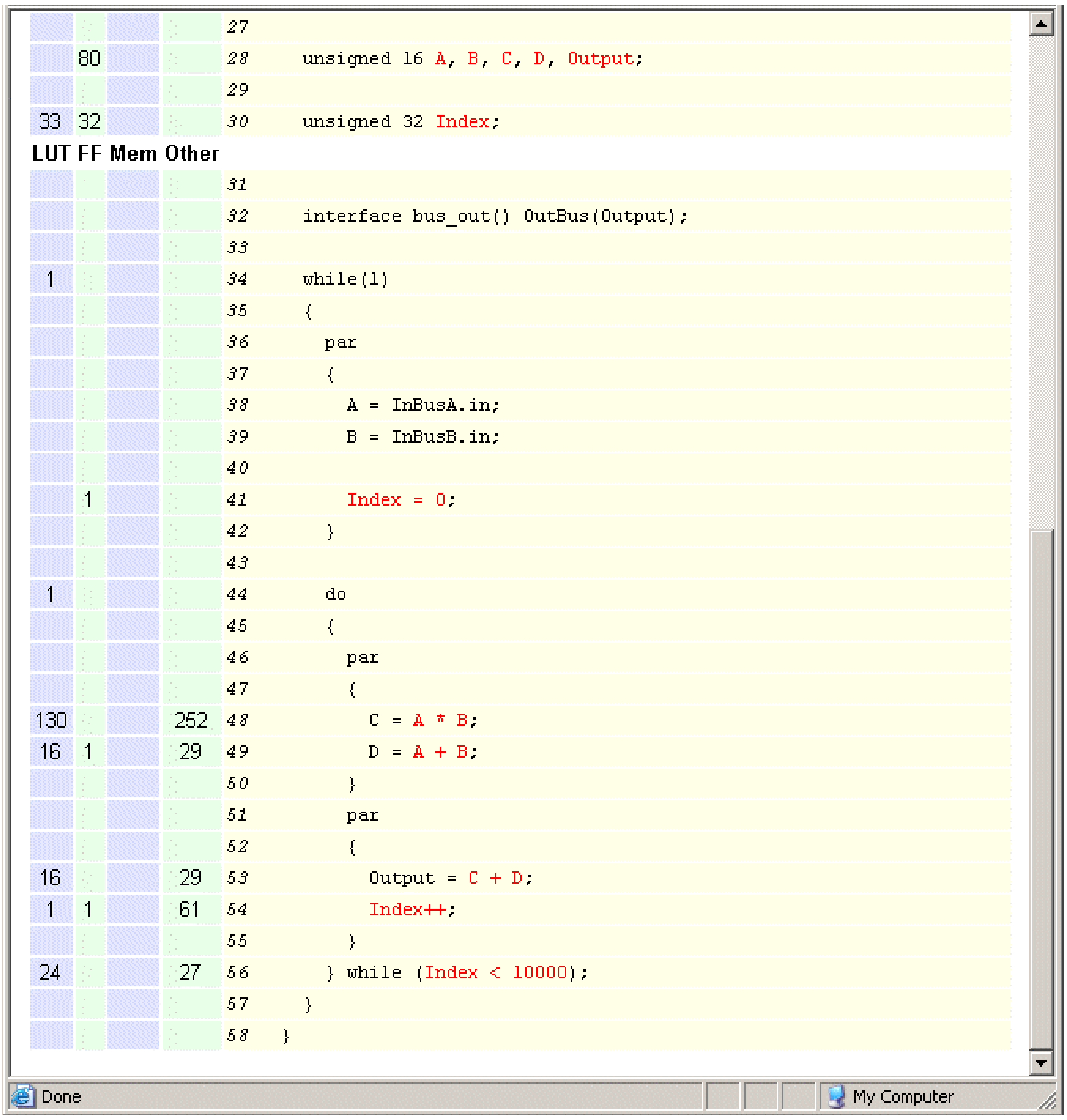
LUT	FF	Mem	Other		
1				31	`while(1)`
				32	`{`
				33	`  par`
				34	`  {`
				35	`    A = InBusA.in;`
				36	`    B = InBusB.in;`
				37	
	1			38	`    Index = 0;`
				39	`  }`
				40	`  /* prime pipeline */`
				41	`  par`
				42	`  {`
				43	`    C = A * B;`
	1			44	`    D = A + B;`
				45	`  }`
2				46	`  do`
				47	`  {`
				48	`    par`
				49	`    {`
				50	`      /* pipeline stage 1 */`
130			252	51	`      C = A * B;`
16			29	52	`      D = A + B;`
				53	
				54	`      /* pipeline stage 2 */`
16			29	55	`      Output = C + D;`
1	1		61	56	`      Index++;`
				57	`    }`
24			27	58	`  } while (Index < 10000);`
				59	`}`
				60	`}`

My Computer

1.4 Reducing the logic area

The section on ***Reducing the logic delay*** (see page 7) looked at using the Logic Estimator to help increase the maximum clock rate at which a design could run. This section looks at how you can use the Estimator to reduce the logic area of a design.

Build the above code in the **version2** project in the **TutorialEstimator** workspace. The logic estimator will save its results in the **TutorialEstimator\version2\EDIF** directory. Open the file named **Summary.html** by double-clicking on it (this should load your computers default web browser). Click on the link to **version3.hcc**, which will take you to a page showing the logic used to implement each line of Handel-C source code, as shown below:

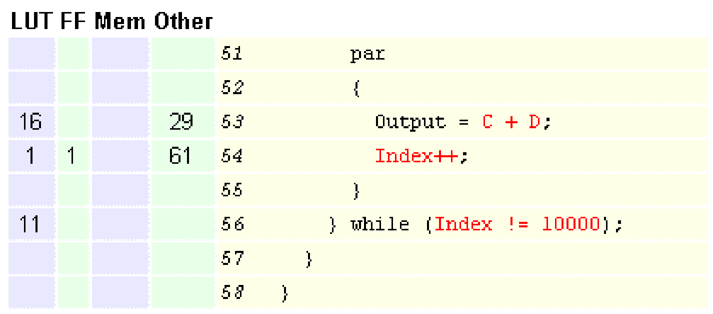


				27	
	80			28	`unsigned 16 A, B, C, D, Output;`
				29	
33	32			30	`unsigned 32 Index;`
LUT	**FF**	**Mem**	**Other**		
				31	
				32	`interface bus_out() OutBus(Output);`
				33	
1				34	`while(1)`
				35	`{`
				36	`par`
				37	`{`
				38	`A = InBusA.in;`
				39	`B = InBusB.in;`
				40	
	1			41	`Index = 0;`
				42	`}`
				43	
1				44	`do`
				45	`{`
				46	`par`
				47	`{`
130			252	48	`C = A * B;`
16	1		29	49	`D = A + B;`
				50	`}`
				51	`par`
				52	`{`
16			29	53	`Output = C + D;`
1	1		61	54	`Index++;`
				55	`}`
24			27	56	`} while (Index < 10000);`
				57	`}`
				58	`}`

From this it is clear which lines of Handel-C source contribute most to the logic area of the design. Some of the logic is associated with the calculation of the values of **C**, **D** and

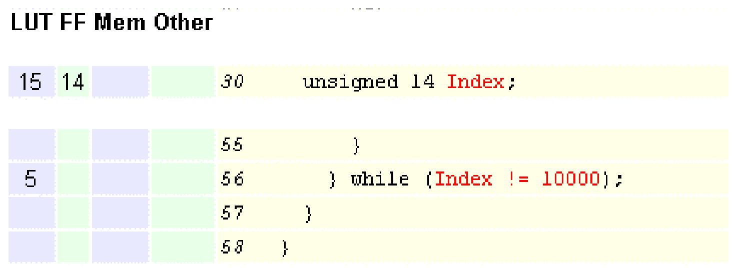
Output, and there is no opportunity to eliminate this, unless the widths of the variables was reduced. However, the loop control code is not as efficient as it could be, so we will now look at how to improve it.

First, the **while** condition on line number 56 uses a "less than" **<** comparison, when in fact a "not equal" **!=** will perform the same function, as **Index** is only incremented by 1 each time through the loop. Try changing this line of code from < to != in the in the **version2** project in the **TutorialEstimator** workspace, rebuild it, and look at the Estimator output again. You will notice that the logic associated with line number has now reduced, as shown below:



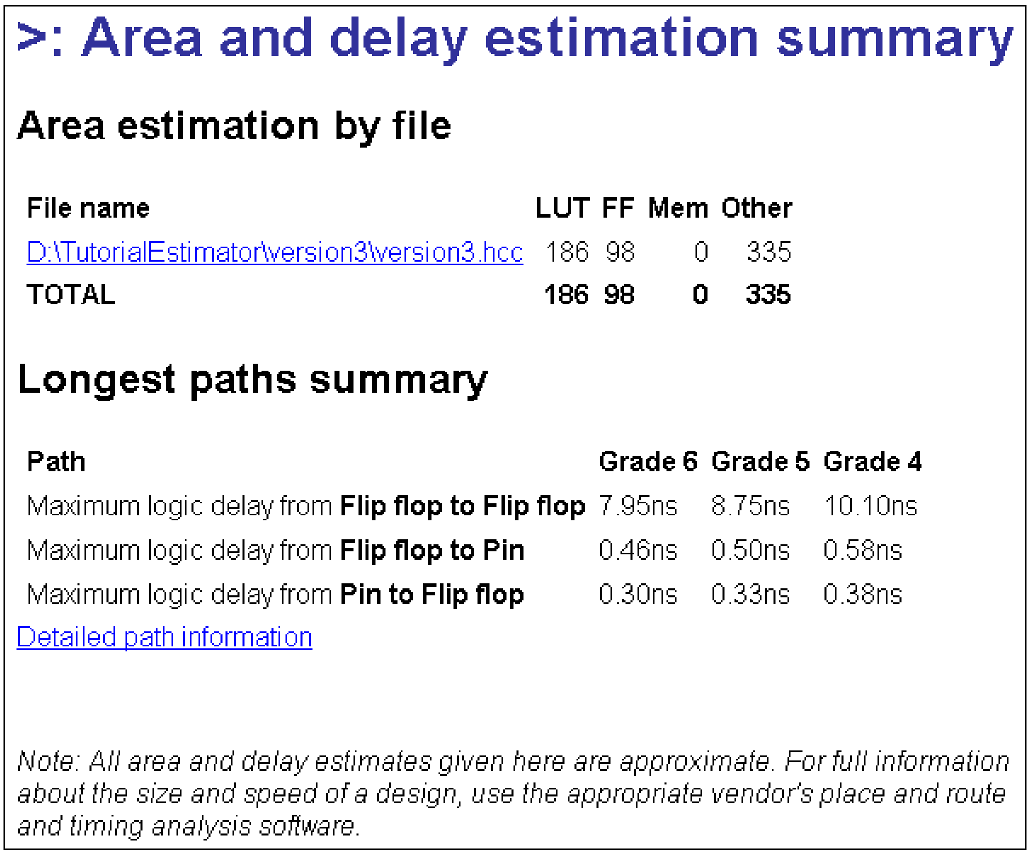
LUT	FF	Mem	Other		
				51	`par`
				52	`{`
16			29	53	`Output = C + D;`
1	1		61	54	`Index++;`
				55	`}`
11				56	`} while (Index != 10000);`
				57	`}`
				58	`}`

A further reduction in logic area is possible because the **Index** variable is 32 bits wide, but is never incremented above 10,000, which only requires a width of 14 bits. Try changing the width of **Index** in the **version2** project in the **TutorialEstimator** workspace, rebuild it, and look at the Estimator output again. You will notice that the logic associated with line numbers 30 and 56 has now reduced, as shown below:



LUT	FF	Mem	Other		
15	14			30	`unsigned 14 Index;`
				55	`}`
5				56	`} while (Index != 10000);`
				57	`}`
				58	`}`

The **version3** project in the **TutorialEstimator** workspace contains these changes (but not the pipelining described in ***Reducing the logic delay*** (see page 7)), and the estimation summary from building it is show below. Comparing with the summary in ***Reducing the logic delay*** (see page 7), it can be seen that a logic area reduction of over 15% has been achieved by changing only two lines of code.



>: Area and delay estimation summary

Area estimation by file

File name	LUT	FF	Mem	Other
D:\TutorialEstimator\version3\version3.hcc	186	98	0	335
TOTAL	**186**	**98**	**0**	**335**

Longest paths summary

Path	Grade 6	Grade 5	Grade 4
Maximum logic delay from **Flip flop to Flip flop**	7.95ns	8.75ns	10.10ns
Maximum logic delay from **Flip flop to Pin**	0.46ns	0.50ns	0.58ns
Maximum logic delay from **Pin to Flip flop**	0.30ns	0.33ns	0.38ns

Detailed path information

Note: All area and delay estimates given here are approximate. For full information about the size and speed of a design, use the appropriate vendor's place and route and timing analysis software.

ESTIMATION SUMMARY FROM VERSION3 PROJECT

2 Index