# advanced vhdl code to get 160ps

## target

We will use a real design to illustrate the HDL floorplanning design techniques we will discuss in this lab

In the MIG DDR3 core, in the memory write data path, there is a need to move a 64-bit data bus from a BUFG clock domain to a BUFR clock domain

The BUFG and BUFR clocks have the same frequency but the phase relation between them is a little uncertain and affected by PVT variations; to maximize the setup/hold time margins and achieve the fastest possible clock rates, the 64-bit bus must have as little skew as possible and this must be guaranteed form on implementation to another

The challenge design contains 64 FFs clocked by a BUFG, which then drive 64 FFs clocked by a BUFR; all 128 FFs must be located in a single clock region, as close as possible to the IOBs; we would like to achieve as little skew as possible across the 64-bit bus between the two sets of FFs ? less than 20ps would be considered acceptable

## Region UCF

The UCF file contains only an area group constraint to restrict the design to the desired FPGA location, a 4x10 CLB area in the lower left corner of the FPGA:

INST "/" AREA\_GROUP = "AG\_/";

AREA\_GROUP "AG\_/" RANGE = SLICE\_X0Y0:SLICE\_X7Y9;

The top level design has two 64-bit input and output ports and two clocks, one for each set of FFs

library ieee;

use ieee.std\_logic\_1164.all;

entity TEST is

port(CLKI:in STD\_LOGIC;

I:in STD\_LOGIC\_VECTOR(63 downto 0);

CLKO:in STD\_LOGIC;

O:out STD\_LOGIC\_VECTOR(63 downto 0));

end TEST;

architecture BEHAVIORAL of TEST is

signal D:STD\_LOGIC\_VECTOR(I'range):=(others=>'0');

attribute syn\_keep:BOOLEAN;

attribute maxdelay:STRING;

attribute syn\_keep of D:signal is TRUE;

attribute maxdelay of D:signal is "160ps";

begin

process(CLKI)

begin

if rising\_edge(CLKI) then

D<=I;

end if;

end process;

process(CLKO)

begin

if rising\_edge(CLKO) then

O<=D;

end if;

end process;

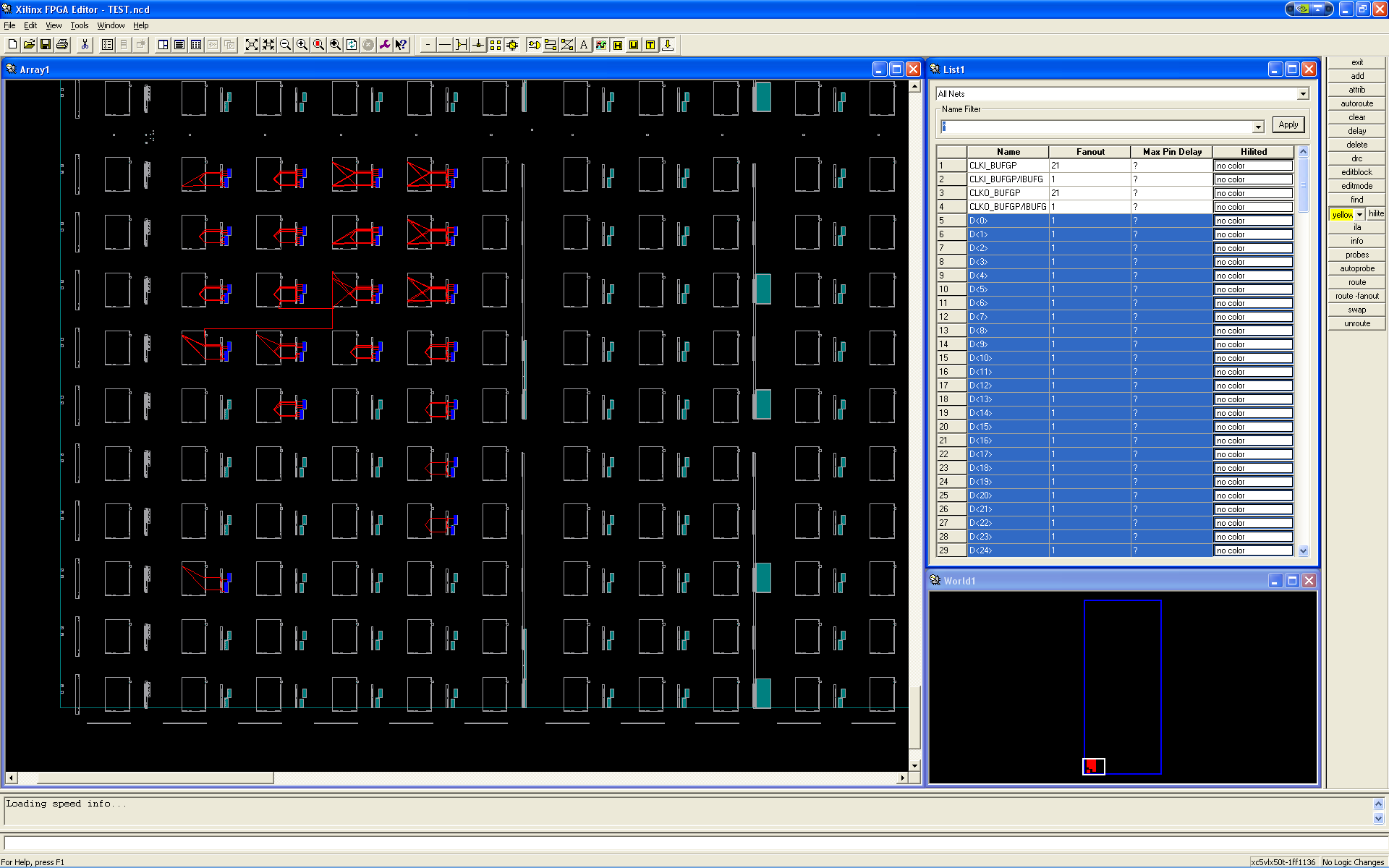
end BEHAVIORAL;

## Par placement is not optimal

Even with MAXDELAY and AREA\_GROUP constraints the tools fail to achieve a consistent placement and routing

This is an otherwise empty FPGA, in reality a normally utilized device would make achieving the timing goals even harder

In this case we have to help the tools achieve the timing closure and then reproduce the result consistently



## Design challenge

To prepare for the lab, you should try to implement this design yourself and see if you can meet the design requirements

We will use XST for synthesis but you can also try SynplifyPro if you want Select a 5VLX50T-1FF1136 target for your implementation and add TEST.vhd and TEST.ucf to your HDL ISE project ? you can use either 9.2.x or 10.1.x

Do not forget to turn off IOB FF mapping in the synthesis tool and in map ? we want fabric, not IOB FFs in this particular case

The VHDL code contains a MAXDELAY attribute, set to 160ps; this is the fastest inter FF routing that could be achieved and if we can meet this constraint on all 64 data lines, we will automatically also achieve a very low skew across the entire bus; this VHDL timing constraint makes it easy to check the result in the par report and see if the design requirement is met ? the final goal is to reduce the timing score to 0 if possible

If you have ideas on how to resolve this design problem, you can try to do it yourself and we can discuss the results during the lab

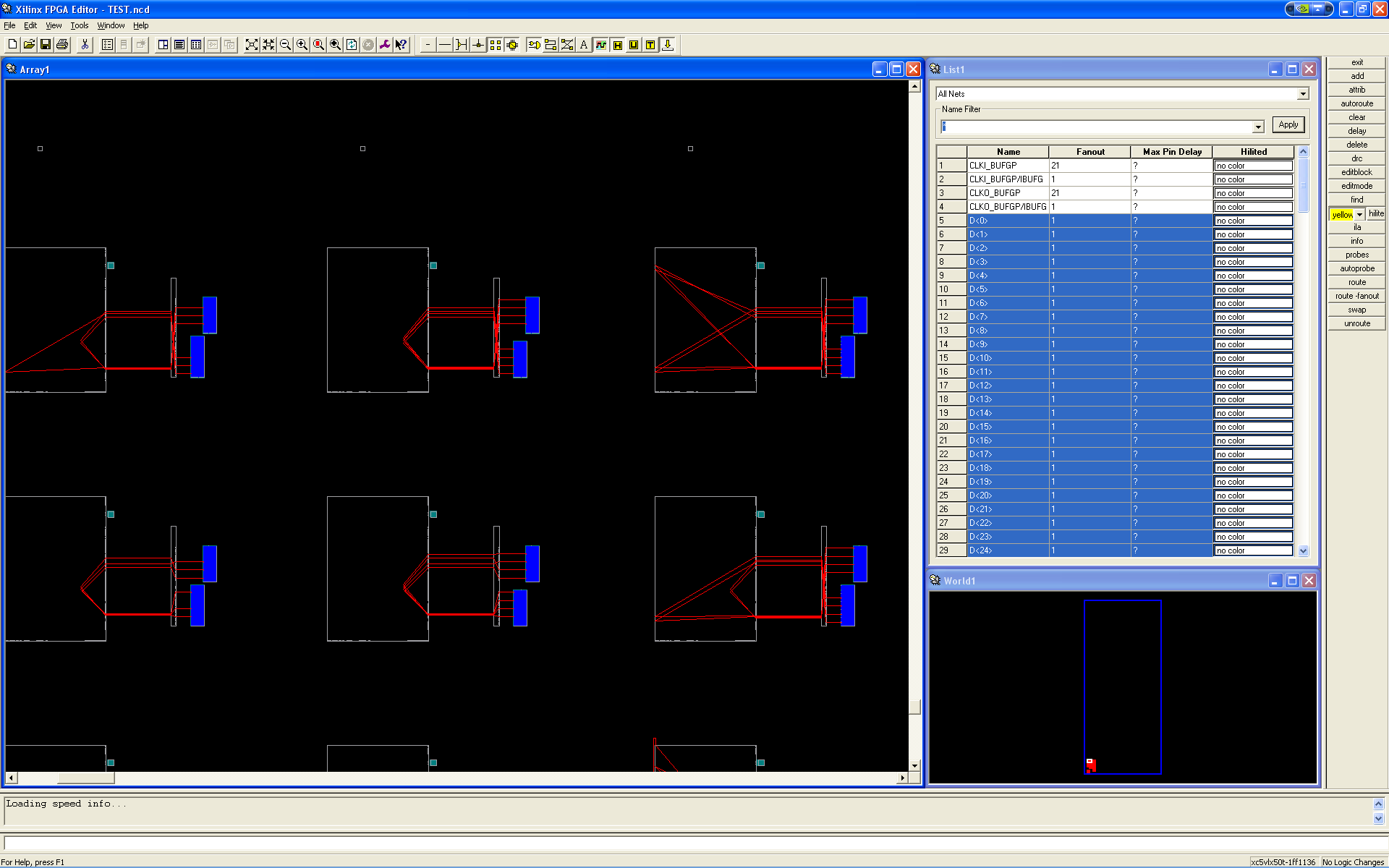
## Par placement is not optimal…

For a few of the 64 data lines par actually finds a very good solution ?

4 FFs in one CLB slice drive 4 FFs in the second CLB slice using very fast direct connects

We will try to capture this placement and routing pattern using an RPM with DIRTs and force the entire 64-bit bus to use this implementation

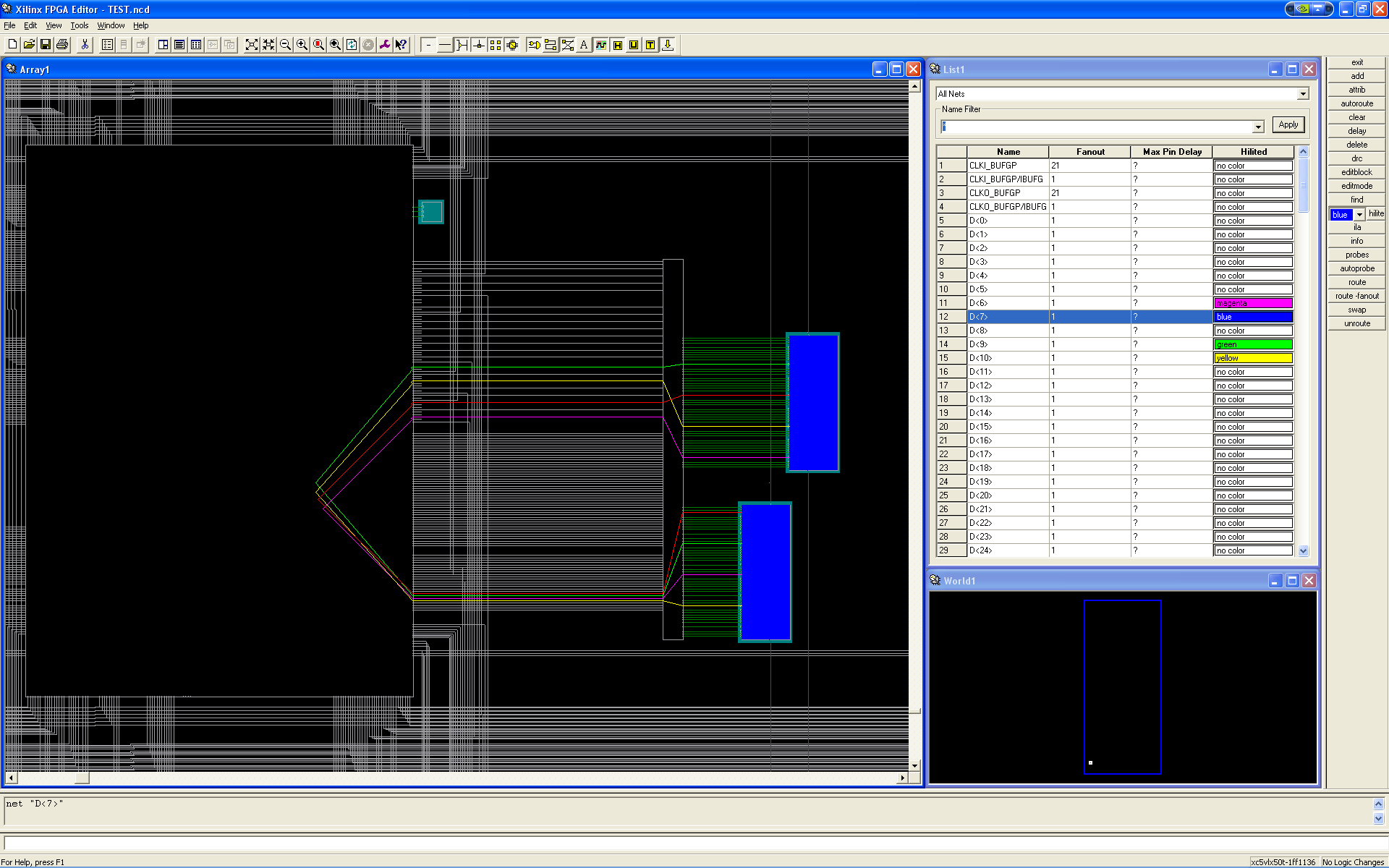
The added bonus is that we will be guaranteed to meet timing independent of the rest of the design and performance will be maintained every time the design is reimplemented



A closer look shows that the best FF placement is not the most natural one

To take advantage of direct connects, FFs ABCD must drive FFs BADC

Same thing is true if the slices for the source and destination FFs are swapped



## RPMx4 Stage

We will create a hierarchical RPM, starting with a 4-bit version, which will then be replicated 16 times to create the 64-bit data bus

The low lever RPM consists of 8 FFs, 4 in each clock domain, which must be locked down using RLOC and BEL constraints and the routing defined with 4 directed routing constraints

The most elegant solution is to pass all these attributes in VHDL using behavioral code and here is where we encounter the first problem ? this flow works well in Synplify/SynplifyPro but not in XST

XST passes RLOC and DIRT behavioral code attributes but ignores BELs ? the workaround is to instantiate DFF primitives and attach the RLOC and BEL attributes to the component label

If you will be using Synplify open RPMx4\_to\_do\_Synplify.vhd – the RLOCs are already defined but we need to add BEL and ROUTE constraints

If you are using XST start with the RPMx4\_to\_do\_XST.vhd file instead

At line 61 we have to address the ABCD to BADC BEL swapping

Implement the design and make sure timing constraints are met

Capture the 4 DIRT strings in FED and paste them into the VHDL code, lines 35-38 (lines 38-41 in the XST version)

Reimplement and verify that the 4 DIRTs succeed

Rename the VHDL file RPMx4.vhd or skip this Lab step and look at the solved example in RPMx4\_snap2\_Synplify.zip or RPMx4\_snap3\_XST.zip

library ieee;

use ieee.std\_logic\_1164.all;

library unisim;

use unisim.vcomponents.all;

entity RPMx4 is

port(CLKI:in STD\_LOGIC;

I:in STD\_LOGIC\_VECTOR(3 downto 0);

CLKO:in STD\_LOGIC;

O:out STD\_LOGIC\_VECTOR(3 downto 0));

end RPMx4;

architecture RPM of RPMx4 is

attribute rloc:STRING;

attribute bel:STRING;

attribute route:STRING;

attribute syn\_keep:BOOLEAN;

attribute maxdelay:STRING;

function FF\_BEL(B:INTEGER) return STRING is

begin

case B is

when 0=>return "FFA";

when 1=>return "FFB";

when 2=>return "FFC";

when 3=>return "FFD";

when others=>return "";

end case;

end;

signal D:STD\_LOGIC\_VECTOR(I'range):=(others=>'0');

begin

li:for K in I'range generate

function Q\_DIRT(K:INTEGER) return STRING is

begin

case K is

when 0=>return "{3;1;5vlx50tff1136;4c4c3c8a!-1;-69680;-165816;S!0;-683;16!1;-404;728!2;404;8!3;843;-152;L!}";

when 1=>return "{3;1;5vlx50tff1136;9a2cba25!-1;-69680;-165504;S!0;-683;-272!1;-404;632!2;404;8!3;843;24;L!}";

when 2=>return "{3;1;5vlx50tff1136;814df02c!-1;-69680;-165608;S!0;-683;-176!1;-404;760!2;404;8!3;843;8;L!}";

when 3=>return "{3;1;5vlx50tff1136;9a2cba25!-1;-69680;-165504;S!0;-683;-272!1;-404;632!2;404;8!3;843;24;L!}";

--when 0=>return "{3;1;5vlx50tff1136;62e6656c!-1;-73832;-18248;S!0;-683;16!1;-404;728!2;404;8!3;843;-152;L!}";

--when 1=>return "{3;1;5vlx50tff1136;d56b2d29!-1;-73832;-18144;S!0;-683;-80!1;-404;600!2;404;8!3;843;-136;L!}";

--when 2=>return "{3;1;5vlx50tff1136;46295ccc!-1;-73832;-18040;S!0;-683;-176!1;-404;760!2;404;8!3;843;8;L!}";

--when 3=>return "{3;1;5vlx50tff1136;27218db1!-1;-73832;-17936;S!0;-683;-272!1;-404;632!2;404;8!3;843;24;L!}";

when others=>return "";

end case;

--A --> B ROUTE="{3;1;5vlx50tff1136;4c4c3c8a!-1;-69680;-165816;S!0;-683;16!1;-404;728!2;404;8!3;843;-152;L!}";

--B --> A ROUTE="{3;1;5vlx50tff1136;9a2cba25!-1;-69680;-165504;S!0;-683;-272!1;-404;632!2;404;8!3;843;24;L!}";

--C --> D ROUTE="{3;1;5vlx50tff1136;814df02c!-1;-69680;-165608;S!0;-683;-176!1;-404;760!2;404;8!3;843;8;L!}";

--D --> C ROUTE="{3;1;5vlx50tff1136;9a2cba25!-1;-69680;-165504;S!0;-683;-272!1;-404;632!2;404;8!3;843;24;L!}";

end;

signal Q:STD\_LOGIC:='0';

attribute rloc of ff:label is "X0Y0";

attribute bel of ff:label is FF\_BEL(K);

attribute syn\_keep of Q:signal is TRUE;

attribute route of Q:signal is Q\_DIRT(K);

attribute maxdelay of Q:signal is "160ps";

begin

ff:FD port map(C=>CLKI,

D=>I(K),

Q=>Q);

D(K)<=Q;

end generate;

lo:for K in O'range generate

signal Q:STD\_LOGIC:='0';

attribute rloc of ff:label is "X1Y0";

attribute bel of ff:label is FF\_BEL(K/2\*2+1-K mod 2);

begin

ff:FD port map(C=>CLKO,

D=>D(K),

Q=>Q);

O(K)<=Q;

end generate;

end RPM;

Synplify/SynplifyPro (but not XST) can be used to create an RPM with DIRTs out of behavioral VHDL code – XST can only use structural code with instantiated primitives

VHDL functions can be used to great advantage to compute attribute values at synthesis time

A syn\_keep attribute must be used with Synplify when net, as opposed to instance attributes like route and maxdelay need to be passed into the EDIF – without the syn\_keep they will be assigned to the instance driving the net instead, like the rloc and bel attributes are

Both RLOC (relative slice location) and BEL (position within slice) values must be specified, especially when DIRTs will be used

DIRT strings can be captured in FPGA Editor and pasted into HDL code

## RPMx4 Lab1 XST Version

li:for K in I'range generate -- generate loop for input FFs

function Q\_DIRT(K:INTEGER) return STRING is -- function to compute DIRT values

begin

case K is -- DIRT strings captured with FED

when 0=>return "{3;1;5vlx50tff1136;62e6656c!-1;-73832;-18248;S!0;-683;16!1;-404;728!2;404;8!3;843;-152;L!}";

when 1=>return "{3;1;5vlx50tff1136;d56b2d29!-1;-73832;-18144;S!0;-683;-80!1;-404;600!2;404;8!3;843;-136;L!}";

when 2=>return "{3;1;5vlx50tff1136;46295ccc!-1;-73832;-18040;S!0;-683;-176!1;-404;760!2;404;8!3;843;8;L!}";

when 3=>return "{3;1;5vlx50tff1136;27218db1!-1;-73832;-17936;S!0;-683;-272!1;-404;632!2;404;8!3;843;24;L!}";

when others=>return "";

end case;

end;

signal Q:STD\_LOGIC:='0';

attribute rloc of ff:label is "X0Y0";

attribute bel of ff:label is FF\_BEL(K);

attribute syn\_keep of Q:signal is TRUE; -- not required for XST but it doesn’t hurt

attribute route of Q:signal is Q\_DIRT(K);

attribute maxdelay of Q:signal is "160ps";

begin

ff:FD port map(C=>CLKI,

D=>I(K),

Q=>Q);

D(K)<=Q;

end generate;

lo:for K in O'range generate -- generate loop for output FFs

signal Q:STD\_LOGIC:='0'; -- with SynplifyPro, RLOC and BEL attributes

attribute rloc of ff:label is "X1Y0"; -- can be used in behavioral code

attribute bel of ff:label is FF\_BEL(K/2\*2+1-K mod 2); -- ABCD to BADC BEL swap

begin

ff:FD port map(C=>CLKO,

D=>D(K),

Q=>Q);

O(K)<=Q;

end generate;

end RPM;

## RPMx8 Stage

The next hierarchical level up is an 8-bit RPM that instantiates two RPMx4 components next to each other

From here on the Synplify and XST design versions are identical

The first RPMx4 has an RLOC value of “X0Y0” and the second one “X0Y4” – we could have used “X0Y2” here but we want to avoid a V5 DIRT bug

DIRTs captured in a SLICEM/SLICEL CLB will fail in a SLICEL/SLICEL CLB and the other way around

library ieee;

use ieee.std\_logic\_1164.all;

entity RPMx8 is

port(CLKI:in STD\_LOGIC;

I:in STD\_LOGIC\_VECTOR(7 downto 0);

CLKO:in STD\_LOGIC;

O:out STD\_LOGIC\_VECTOR(7 downto 0));

end RPMx8;

architecture RPM of RPMx8 is

attribute rloc:STRING;

component RPMx4

port(CLKI:in STD\_LOGIC;

I:in STD\_LOGIC\_VECTOR(3 downto 0);

CLKO:in STD\_LOGIC;

O:out STD\_LOGIC\_VECTOR(3 downto 0));

end component;

attribute rloc of l0:label is "X0Y0"; -- relative position of the first RPMx4

attribute rloc of l1:label is "X4Y0"; -- relative position of the second RPMx4

begin

l0:RPMx4 port map(CLKI=>CLKI,

I=>I(3 downto 0),

CLKO=>CLKO,

O=>O(3 downto 0));

l1:RPMx4 port map(CLKI=>CLKI,

I=>I(7 downto 4),

CLKO=>CLKO,

O=>O(7 downto 4));

end RPM;

The top hierarchical level is the 64-bit RPM that instantiates eight RPMx8 components in a vertical stack

Only one top level sub-RPM receives an RLOC\_ORIGIN attribute, it doesn’t matter which one

## RPMx64 Lab2

Open RPMx64\_to\_do.vhd (the same file will work both for Synplify and XST flows)

At line 23 we need to compute the RLOC value as a function of K

At line 24 we need an RLOC\_ORIGIN=“X0Y1” but only for one loop iteration (when K=0 for example), not all of them

If your DIRTs have been captured in a SLICEL/SLICEL CLB the RLOC origin should be “X2Y1” instead of “X0Y1”, which should be used for SLICEM/SLICEL type DIRTs

Create a Synplify or XST project with all 3 VHDL file, RPMx4, RPMx8 and RPMx64, if all is well you should have 64 successful DIRTs and meet the timing constraints

If you want to skip this step you can look at the final resulut in RPMx64\_snap2\_Synplify.zip or RPMx64\_snap3\_XST.zip

library ieee;

use ieee.std\_logic\_1164.all;

entity RPMx64 is

port(CLKI:in STD\_LOGIC;

I:in STD\_LOGIC\_VECTOR(63 downto 0);

CLKO:in STD\_LOGIC;

O:out STD\_LOGIC\_VECTOR(63 downto 0));

end RPMx64;

architecture RPM of RPMx64 is

attribute rloc:STRING;

attribute rloc\_origin:STRING;

component RPMx8

port(CLKI:in STD\_LOGIC;

I:in STD\_LOGIC\_VECTOR(7 downto 0);

CLKO:in STD\_LOGIC;

O:out STD\_LOGIC\_VECTOR(7 downto 0));

end component;

function RPM\_ORIGIN(B:BOOLEAN;X,Y:INTEGER) return STRING is

begin

if B then

return "X"&INTEGER'image(X)&"Y"&INTEGER'image(Y);

else

return "";

end if;

end;

begin

lk:for K in 0 to 7 generate

attribute rloc of l8:label is "X0Y"&INTEGER'image(K);

attribute rloc\_origin of l8:label is RPM\_ORIGIN(K=0,0,1); -- we can change the

-- RPM location here

begin

l8:RPMx8 port map(CLKI=>CLKI,

I=>I(8\*K+7 downto 8\*K),

CLKO=>CLKO,

O=>O(8\*K+7 downto 8\*K));

end generate;

end RPM;

Only one RPMx8 instance requires an rloc\_origin attribute – to avoid treating one of them different from the others the RPM\_ORIGIN function will return an empty string for all RPMX8 except the first one

The entire RPM will be placed at absolute location X0Y1, the X and Y values could be passed as generics for example at the top entity level

## Conclusions

The implementation tools are generally unable to find the absolute best placement and routing – very high performance designs require detailed floorplanning and use of directed routing constraints to achieve and guarantee the desired results

The development of hierarchical RPMs with DIRTs is best done in HDL, the UCF flow is too limited for that

Both Synplify and XST can be used, with minor differences

– XST doesn’t pass BEL attributes in behavioral code properly, the workaround is to instantiate primitives

– Synplify requires the use of syn\_keep attributes whenever net rather than instance constraints are used (like ROUTE and MAXDELAY for example)

The flow we have described for VHDL also works in Verilog, with the exception of dynamic evaluation of attribute values at synthesis time using functions

DIRTs have issues in V5 related to SLICEM/SLICEL CLB types – an RPM with DIRTs can move freely along the Y axis but it is likely to fail when translated along the X axis; until this is fixed, multiple versions of the RPM for all possible X locations must be used as a workaround