

# Xilinx Tutorial



Pallavi Paliwal  
([pallavip@ee.iitb.ac.in](mailto:pallavip@ee.iitb.ac.in))  
Indian Institute of Technology, Bombay

# Contents

- 1. Introduction to Xilinx**
  - § *Purpose of Xilinx Tool*
  - § *Xilinx Flow Overview*
  - § *Available Xilinx Product Families*
  - § *Selection Consideration for Xilinx Device*
- 2. Creating new ISE Project**
- 3. Synthesizing the Design**
  - § *Understanding Synthesis Process Properties*
  - § *Analyzing Synthesis Report*
  - § *Generating Post-Synthesis Simulation Model*
- 4. Specifying User Constraints**
  - § *Understanding Timing Constraints*
  - § *Assigning Package Pins*
- 5. Translating the Design**
- 6. Mapping the Design**
  - § *Understanding MAP Options*
  - § *Analyzing MAP Report*
- 7. Placing and Routing the Design**
  - § *Understanding PAR options*
  - § *Analyzing PAR Report*
  - § *Asynchronous Delay Report*
  - § *Post PAR Static Timing Analysis*
  - § *Generating Post PAR Simulation Model*
- 8. Generating BitMap File**

# Purpose of Xilinx Tool

## **Requirement of Xilinx Tool :**

Xilinx is a Synthesis Tool which converts Schematic/HDL Design Entry into functionally equivalent logic gates on Xilinx FPGA, with optimized speed & area.

So, after specifying behavioral description for HDL, the designer merely has to select the Library and specify Optimization Criteria; and Xilinx Synthesis Tool determines the netlist to meet the specification; which is then converted into Bit-File to be loaded onto FPGA PROM .

Also, Xilinx Tool generates Post-Process Simulation Model after every Implementation Step, which is used to functionally verify generated netlist after processes, like Map, Place & Route.

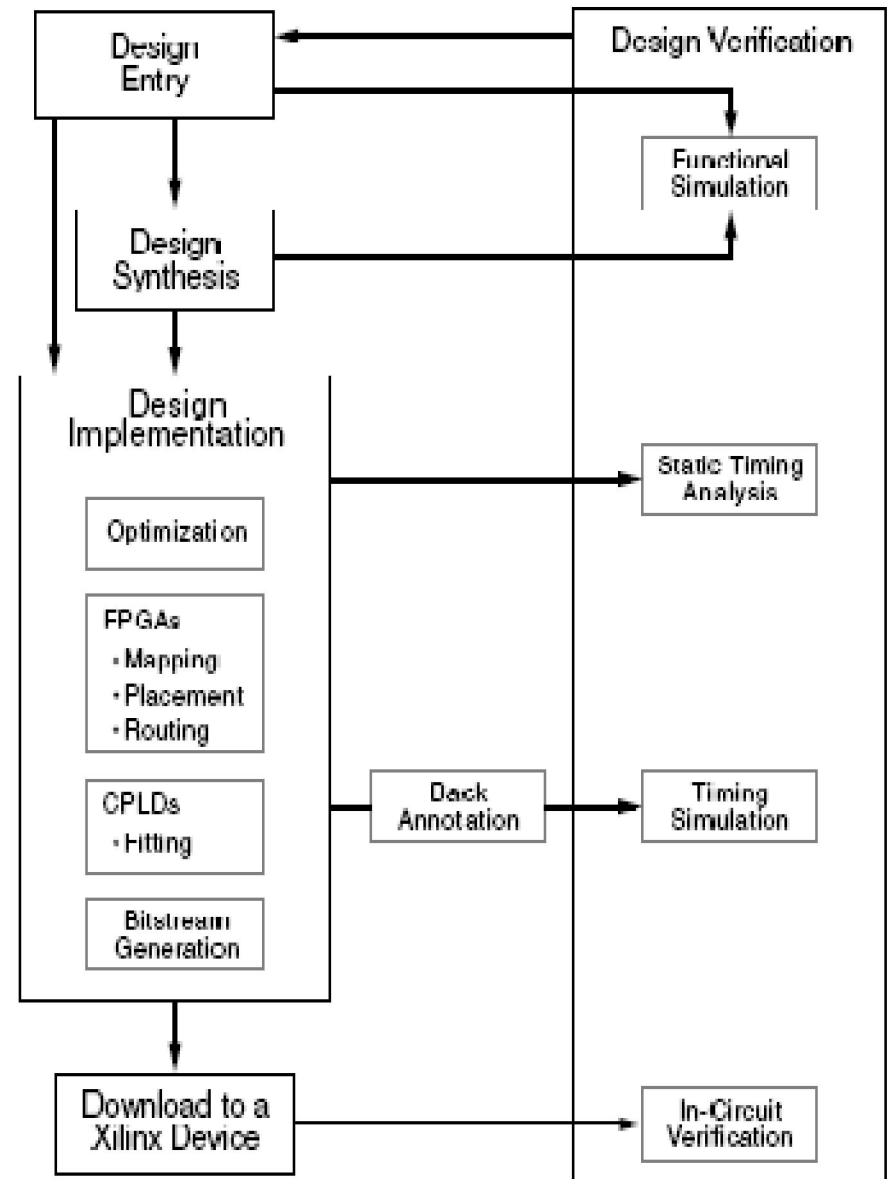
## **Some more Features of Xilinx :**

- Allows Mixed Mode HDL Design Entry
- Xilinx ISE allows integration with other Synthesis Engine from Mentor Graphics/Exemplar, Synopsys and Synplicity. (XST is proprietary Synthesis Tool of Xilinx.)

# Xilinx Flow Overview

- 1) Add HDL/Schematic Design Entry to ISE Project, targeted for a particular Xilinx Product Family.
- 2) **Design Synthesis** : Converts HDL into equivalent boolean equations; according to which logic gates are then correspondingly packed into Logic Cells, LUT's & FF's from Xilinx UNISIM Library
- 3) **Design Implementation** :
  - I. **Translate** : Translate step checks the design and ensures that netlist is consistent with chosen architecture. Translate also checks User-defined Constraint file, for any inconsistencies.

*The above mentioned Process Steps are Technology-Independent Part of Xilinx, since these processes would be carried out successfully by tool, for any product family. (irrespective of the resources available in that particular Xilinx FPGA Product)*



[Figure Source: Xilinx ISE 8 Software Manual]

# Xilinx Flow Overview

*The following mentioned Implementation Process Steps are Technology Dependent Part of Xilinx, wherein Design could be Mapped, Placed & Routed, with desired Speed & Area constraint, only if Targeted Xilinx Product has required speed grade, sufficient Logic Blocks & Interconnect resources available for Design Entry.*

**II. Map :** Calculates & Allocates Physical Combinational Logic Blocks (CLB) & Input Output Block (IOB) Components in Targeted Device, to Logic Element symbols in Netlist that is generated during Translation Process.

**III. Place & Route (PAR):** Places CLBs into logical position and utilizes the routing resources on target device, to connect logic cells on Xilinx Product such that desired Timing Specification are met.

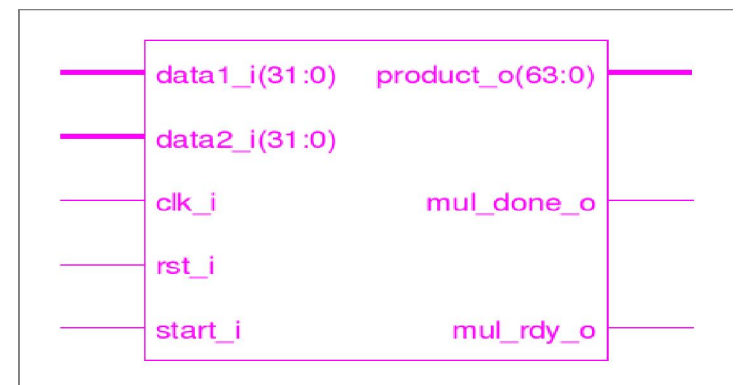
**4) Bit-File Generation :** Creates Bit-Stream file containing Configuration Data for Target FPGA Device

---

*Further in this tutorial, Xilinx Flow will be demonstrated through 32-bit Shift and Add Multiplier Implementation on Spartan xc3s500e-5-pq208 device.*

This 32-bit Shift & Add Multiplier latches value on Input Data Bus, when START control signal is high; and outputs valid 64-bit Product, while asserting DONE control signal high.

DUT asserts READY control signal, when Multiplier FSM is in idle state.



```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity shift_add_multiplier is

    port ( data1_i      : in  std_logic_vector(31 downto 0);-- Multiplicand Input
          data2_i      : in  std_logic_vector(31 downto 0);-- Multiplier Input
          product_o     : out std_logic_vector(63 downto 0);-- Data Product output

          clk_i         : in  std_logic;-- Input Clock Signal
          rst_i         : in  std_logic;-- Input Reset Signal
          start_i       : in  std_logic;-- Input START Control Signal
          mul_done_o    : out std_logic;-- Multiplication DONE Output Control Signal
          mul_rdy_o     : out  std_logic);-- Multiplier Block READY

end shift_add_multiplier;

-----
-- On assertion of Start Input signal, Multiplier Block latches two Input Data
-- Double Word; and after 32 clock cycles, places multiplied value on 64-bit
-- product_o o/p signal, while asserting mul_done_o signal (indicating valid
-- 64-bit data product is available on output signal
-----

architecture shift_add_multiplier_logic of shift_add_multiplier is

    type state_type is (start_st,load_st,shift_add_st); -- Multiplier FSM States

    signal cur_state,nxt_state : state_type := start_st; -- FSM Current/Next State Register

    signal mul1_shift_reg : std_logic_vector(63 downto 0); -- Multiplicand Shift Register
    signal mul2_shift_reg : std_logic_vector(31 downto 0); -- Multiplier Shift Register

```

```

signal product_d1: std_logic_vector(63 downto 0); -- Product O/P Signal used in
-- Sequential Adder
signal product_d   : std_logic_vector(63 downto 0); -- Temporary Product Output from Adder

-- Multiplier O/P is valid only when mul_done_o control signal is asserted
signal mul_done_d  : std_logic; -- DONE Control Signal o/p from Combinational Block

-- Input data could be latched by Multiplier block, only when mul_rdy_o signal is high
signal mul_rdy_d   : std_logic; -- READY Control Signal o/p from Combinational Block

begin

    product_o <= product_d1;
-----
-- Sequential Logic : Output Data/Control Signals are registered
-----

    process(clk_i,rst_i)
    begin

        -- Reset Multiplier Block asynchronously on Assertion of rst_i signal

        if (rst_i = '1') then

            cur_state <= start_st      ;

            product_d1 <= (others=>'0');
            mul_done_o <= '0'          ;
            mul_rdy_o  <= '0'          ;

        elsif(clk_i'event and clk_i = '1') then

            cur_state <= nxt_state ;

            product_d1 <= product_d ;
            mul_done_o <= mul_done_d ;
            mul_rdy_o  <= mul_rdy_d  ;

        end if;

```



```
end process;
```

```
-----  
Sequential Logic : Multiplicand & Multiplier Shift Register  
-----
```

```
process(clk_i,rst_i)  
begin
```

```
    if (rst_i = '1') then                -- Asynchronous Reset
```

```
        mul1_shift_reg <= X"0000_0000_0000_0000" ;  
        mul2_shift_reg <= X"0000_0000" ;
```

```
    elsif (clk_i'event and clk_i = '1') then
```

```
        if (nxt_state = load_st) then --Next State as Load acts as Load Enable  
            --signal for Shift Register
```

```
            mul1_shift_reg <= X"0000_0000" & data1_i;  
            mul2_shift_reg <= data2_i ;
```

```
        elsif (nxt_state = shift_add_st) then --Next State as Shift_Add acts as Shift Enable  
            --signal for Shift Register
```

```
            mul1_shift_reg <= mul1_shift_reg(62 downto 0) & '0';-- Shift Left Multiplicand  
            mul2_shift_reg <= '0' & mul2_shift_reg(31 downto 1);-- Shift Right Multiplier
```

```
        end if;
```

```
    end if;
```

```
end process;
```

```
-----  
FSM Next State Generation Logic  
-----
```

```
process(cur_state,start_i,mul_done_d)  
begin
```



```

case cur_state is

-- On assertion of Start Control I/P , load Input data words
when start_st      => if (start_i = '1') then
                        nxt_state <= load_st ;
                      else
                        nxt_state <= start_st;
                      end if;

-- Load State is asserted only for 1 clock cycle
when load_st       => nxt_state <= shift_add_st ;

-- FSM continues to remain in Shift_Add state, depending on width
-- of smaller data byte
when shift_add_st => if (mul_done_d = '1') then
                        nxt_state <= start_st;
                      else
                        nxt_state <= shift_add_st;
                      end if;

-- when others => nxt_state <= start_st ;
end case;

end process;

```

---

- FSM Output Signal Generation Logic

---

```

process(cur_state,mul2_shift_reg,mul1_shift_reg,product_d1)
begin

case cur_state is

when start_st => product_d  <= (others=>'0');
                 mul_done_d  <= '0'           ;
                 mul_rdy_d   <= '1'           ;

```

```

when load_st => if (mul2_shift_reg(0) = '1') then-- Multiplier LSB checks whether 0 or shifted
    product_d <= mul1_shift_reg ; -- Multiplicand has to be added to temporary
else -- Product Register
    product_d <= X"0000_0000_0000_0000" ;
end if;

mul_done_d <= '0' ;
mul_rdy_d <= '0' ;

when shift_add_st => if (mul2_shift_reg(0) = '1') then
    product_d <= product_d1 + mul1_shift_reg;
else
    product_d <= product_d1 ;
end if;

-- done_d signal asserts when Multiplier Shift Register value is 1

    if (mul2_shift_reg = X"0000_0001") then
        mul_done_d <= '1';
        mul_rdy_d <= '1';
    else
        mul_done_d <= '0';
        mul_rdy_d <= '0';
    end if;

when others => product_d <= X"0000_0000_0000_0000" ;
mul_done_d <= '0' ;
mul_rdy_d <= '1' ;

end case;

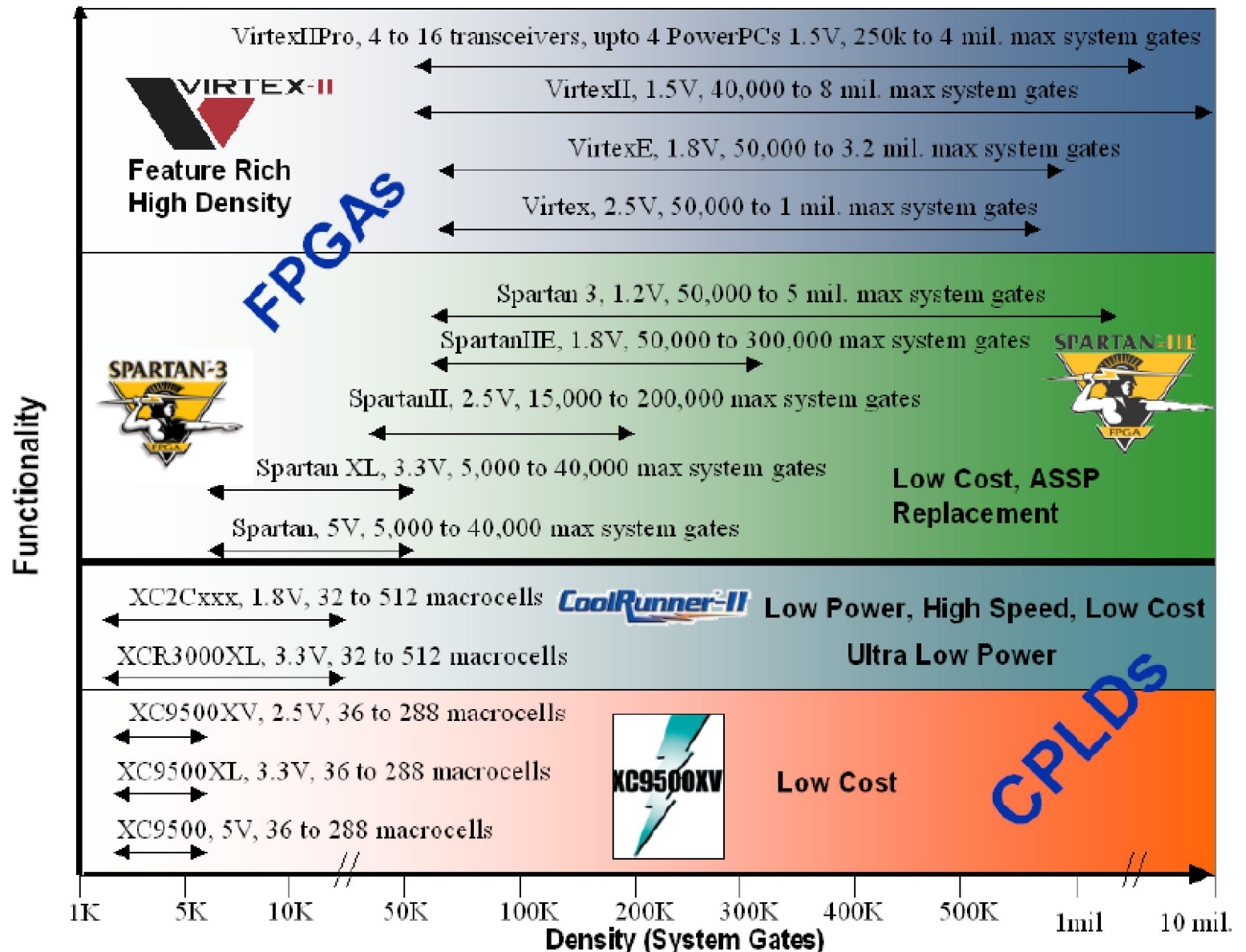
end process;

end shift_add_multiplier_logic;

```

# Available Xilinx Product Families

- Spartan
- Virtex
- Coolrunner
- XC9500



[Figure Source: Programmable Logic Design Quick Start Handbook, By Karen Pawell & Nick Mehta]

# Selection Consideration for Xilinx Device

According to Design Specification, decide which device best meets the design criteria.

***Selection Consideration in order of general priority are :-***

1. ***Speed Requirement*** :- If the Maximum Frequency requirement for the design is not met with a particular Xilinx device, choose a Xilinx device of higher Speed Grade (or) switch to next generation Xilinx Product Family.

*For eg. Following is the Speed Comparison of Shift & Add Multiplier synthesized on Spartan Device (with different Speed Grade) & Virtex Device.*

Target Device	Maximum Frequency (of Multiplier design)
<i>Spartan</i> :- xc3s500e-4-pq208	119 MHz
<i>Spartan</i> :- xc3s500e-5-pq208	137 MHz
<i>Virtex</i> :- xc5vlx30-2-ff324	259.639 MHz
<i>Virtex</i> :-xc5vlx30-3-ff324	295.683 MHz

Note :-

- i) In Target Device Name, -2/-3/-4/-5 indicates **Speed Grade** available for the device. (Higher the indicated Speed Grade number, higher is the maximum frequency obtained through that device)
- ii) Virtex and Spartan Product Family are indicated by text “**vlx**” and “**s**” respectively, in Xilinx Device name.

# Selection Consideration for Xilinx Device

2. **Logic Density** :- Choose the device which has Gate Count and Macro-Cells to meet the Logic Density of the design.

For e.g. Following is Device Utilization Summary of Multiplier Implementation on xc3s**500**e-5-pq208 device.  
(Here, number **500** in device name indicates that this device has **500k** gates)

## *Device utilization summary:*

-----  
*Number of Slices:*                   198 out of 4656 4%  
*Number of Slice Flip Flops:*   165 out of 9312 1%  
*Number of 4 input LUTs:*       379 out of 9312 4%  
*Number of IOs:*                   133  
*Number of bonded IOBs:*       133 out of 158 84%  
*Number of GCLKs:*               1 out of 24 4%

(Small Designs like Multiplier can be implemented on CPLD Series having minimum number of gates, amongst available Xilinx Product families)

3. **Package Type and Number of IO Pins** :- Based on Number of Input/Output Ports of the design, decide on device having sufficient number of IO Pins.

For e.g. 32-bit Multiplier Design cannot be implemented using device 3s500e**cp132**-5, since, this **Chip-Scale Package** only has **92** I/O pins, whereas Multiplier design here has 133 I/O ports.

xc3s500e-5-**pq208** device can be used for Multiplier Design because this **Plastic Quad Flat Package** has **158** I/O Pins.

# Creating new ISE Project

1. From Project Navigator, select File > New Project.

(Specify the **Project Name**, **Project Location**; and select **Top Level Source Type** as HDL)

2. Click *Next* & describe **Device Properties** in New Project Wizard. (Based on the Selection Design Consideration mentioned earlier, select **Xilinx Product Family/ Device Type/ Package Type & Speed Grade**)

*Following image window indicates the Device Properties selected for Shift-Add Multiplier Implementation :-*

Property Name	Value
Product Category	All
Family	Spartan3E
Device	XC3S500E
Package	PQ208
Speed	-5
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim-SE VHDL
Preferred Language	VHDL
Enable Enhanced Design Summary	<input checked="" type="checkbox"/>
Enable Message Filtering	<input type="checkbox"/>
Display Incremental Messages	<input type="checkbox"/>

3. Click *Next* & then click Add Source tab in New Project Wizard, to browse and add existing HDL source files to current ISE Project

# Synthesizing Design

## Synthesis Tool Functionality :

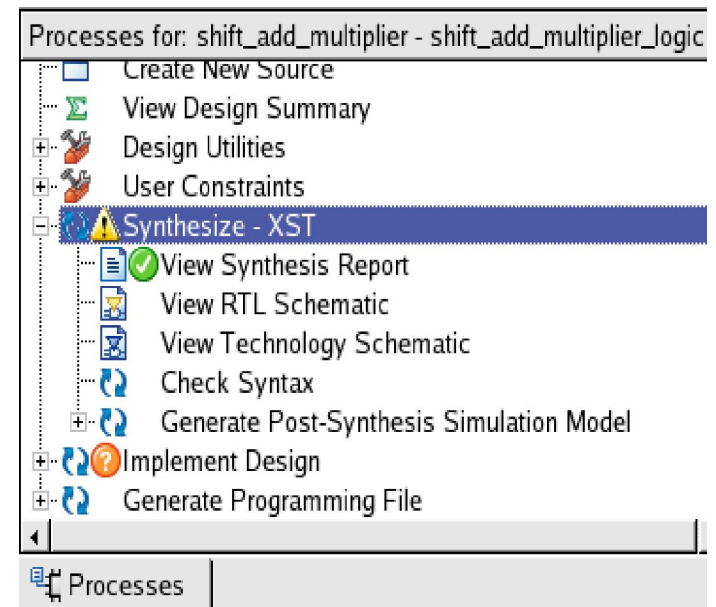
During synthesis, HDL files are translated into gates and optimized for the target architecture.

Thus, XST Synthesis tool uses design's HDL code and generates a supported netlist type (NGC) for Xilinx implementation tools, by performing following general steps :-

- **Analyze / Check Syntax** of the Source Code.
- **Compile** : Translates and optimizes the HDL code into a set of components that the synthesis tool can recognize.
- **Map** : Translates the components from the compile stage into the target technology's primitive components from UNISIM Library.

## Steps to Synthesize HDL Design :

- Select top-level HDL design in the *Sources* window.
- To set Synthesis options, right-click **Synthesize - XST** in the *Processes* window; select *Properties* to display the *Process Properties* dialog box.
- With the top-level source file selected, right-click **Synthesize - XST** in the *Processes* window & select *Run* option.
- Synthesis Report file is stored with extension <project\_name>.syr in ISE Project Directory.





# Synthesizing Design – Understanding Synthesis Options

[Source : Xilinx Synthesis and Simulation Design Guide]

Synthesis options enables designer to modify the behaviour of the synthesis tool, to make optimizations according to the needs of design.

- **Optimization Effort** : constraint allows to choose synthesis optimization level as **Normal** or **High** optimization.
- **Optimization Goal** : constraint allows to choose synthesis optimization strategy as **Speed** or **Area**.  
*For e.g. In Shift Add Multiplier implementation, when **Optimization Goal** is selected as **Area**, though Number of Slices and LUTs utilized reduces by **1%**, but the maximum frequency for the design also reduces by **8MHz**, in comparison to Synthesis done with default Optimization Goal (i.e. Speed).*
- **Use Synthesis Constraint File** : option allows to include (or) exclude **.xcf** Constraint File ,during synthesis process.
- **Keep Hierarchy** : is a synthesis and implementation constraint. If hierarchy is maintained during Synthesis, the Implementation tools will use this constraint to preserve the hierarchy throughout the implementation process and allow a simulation netlist to be created with the desired hierarchy. *Though preserving the hierarchy gives the advantage of fast processing; but, merging the hierarchy blocks improves the fitting results (i.e. fewer device macrocells & better frequency).*
- **Global Optimization Goal** : allows to optimize speed in different regions (register to register, inpad to register, register to outpad, and inpad to outpad) of the design.
- **Write Timing Constraints** : enables or disables propagation of timing constraints to the NGC file, which will be used during place and route, as well as synthesis optimization.
- **Slice Utilization Ratio** : defines the area size in absolute number or percent of total number of slices that XST must not exceed, during timing optimization.

# Synthesizing Design – Understanding HDL Options

[Source : Xilinx Synthesis and Simulation Design Guide]

- **RAM / ROM / MUX / DECODER / PRIORITY ENCODER / LOGICAL SHIFTER / SHIFT REGISTER / LOGICAL SHIFTER Extraction** : constraints enable or disable corresponding macro inference.
- *Following table indicates how various constraints control the way, the macro-generator implements the inferred various macros:-*

HDL Options	Allowed Values
RAM Style	Auto/Block/Distributed/Pipe-Distributed RAM
ROM Style	Auto/Block/Distributed ROM
MUX Style	Auto/MUXF/MUXCY (Value AUTO indicates that XST looks for best implementation for each macro inference)
Multiplier Style	Auto/Block/LUT/Pipe_Block/KCM/CSD/Pipe_LUT
FSM Style	BRAM/LUT ; [ Large FSMs can be made more compact and faster by implementing them in Block RAM resources (instead of LUTs) provided in Virtex and later technologies].
<b>Case Implementation Style</b> (This option instructs XST how to interpret Verilog Case statements )	None/Full/Parallel/Full-Parallel  [ - If full is used, XST assumes that the case statements are complete and avoids latch creation. - If parallel is used, XST assumes that the branches cannot occur in parallel and does not use a priority encoder.]
<b>FSM Encoding Algorithm</b> (This constraint selects FSM encoding technique to use)	Auto/One-Hot/Compact/Sequential/Gray/Johnson/Speed1/User (FPGA State Machines are usually One-Hot encoded.)

# Synthesizing Design – Understanding HDL Options

*For e.g. In Shift Add Multiplier Synthesis, leaving FSM Encoding Algorithm as “Auto” causes One-Hot encoding to be implemented for <cur\_state> and <nxt\_state> signal.*

Using one-hot encoding for signal <cur\_state>.

INFO:Xst:2117 - HDL ADVISOR - Mux Selector <cur\_state> of Case statement line 121 was re-encoded using one-hot encoding. The case statement will be optimized (default statement optimization), but this optimization may lead to design initialization problems. To ensure the design works safely, you can:

- add an 'INIT' attribute on signal <cur\_state> (optimization is then done without any risk)
- use the attribute 'signal\_encoding user' to avoid onehot optimization
- use the attribute 'safe\_implementation yes' to force XST to perform a safe (but less efficient) optimization

Using one-hot encoding for signal <nxt\_state>.

- **Safe Implementation :** constraint implements FSM with additional logic, that forces FSM to a valid state (recovery state), if FSM gets into an invalid state.
- **XOR Collapsing :** controls collapsing of cascaded XORs into single XOR.
- **Resource Sharing :** constraint enables or disables resource sharing of arithmetic operators.

# Synthesizing Design – Understanding Xilinx Options

[Source : Xilinx Synthesis and Simulation Design Guide]

- **Add I/O Buffers** : enables or disables I/O buffer insertion, to all the port names in the top level entity of the design
- **Max Fanout** : constraint limits the fanout of nets or signals. Large fanouts can cause routability problems, therefore XST tries to limit fanout by duplicating gates or by inserting buffers. *(These buffers will be protected against logic trimming at the implementation level by defining a KEEP attribute in the NGC file.)*
- **Number Of Clock Buffers** : controls maximum number of Clock Buffers created by XST.
- **Register Duplication** : enables or disables register replication, during timing optimization and fanout control.
- **Equivalent Register Removal** : enables or disables removal of equivalent registers, described at the RTL Level.
- **Register Balancing** : option allows to move flip-flops and latches across logic to increase clock frequency.  
*(Forward Register Balancing will move a set of flip-flops that are at the inputs of a LUT to a single flip-flop at its output. Backward Register Balancing will move a flip-flop which is at the output of a LUT to a set of flip-flops at its inputs. )*
- **Move First Stage** : constraint controls the retiming of registers with paths coming from primary inputs.
- **Move Last Stage** : constraint controls the retiming of registers with paths going to primary outputs.
- **Pack I/O Registers into IOBs** : constraint packs flip-flops in the I/Os to improve input/output path timing.
- **Slice Packing** : option enables the XST internal packer, which attempts to pack critical LUT-to-LUT connections within a slice or a CLB.
- **Use Clock Enable** : enables or disables the use of clock enable function in flip-flops. The disabling of the clock enable function is typically used for ASIC prototyping on FPGAs. In auto mode, XST tries to estimate a trade off between using a dedicated clock enable input of a flip-flop input and putting clock enable logic on the D input of a flip-flop.
- **Use Synchronous Set/Reset** : constraint enables or disables the use of synchronous set/reset function in flip-flops.
- **Optimize Instantiated Primitives** : Constraint allows XST to optimize Xilinx library primitives that have been instantiated in HDL.

# Synthesizing Design – Analyzing Synthesis Report

- **HDL Compilation & Analysis**

During HDL Compilation and HDL Analysis, XST parses and analyzes VHDL/Verilog files and gives the name of the libraries into which they are compiled. During this step, XST may report potential mismatches between synthesis and simulation results, potential multi-sources, and other issues.

```
=====
*                               HDL Compilation                               *
=====
Compiling vhd1 file "/home/users/pallavip/multiply/shift_add_multiplier_synth/shift_add_multiplier/shift_add_multiplier.vhd" in Library work.
=====
*                               HDL Analysis                               *
=====
Analyzing Entity <shift_add_multiplier> in library <work> (Architecture <shift_add_multiplier_logic>).
Entity <shift_add_multiplier> analyzed. Unit <shift_add_multiplier> generated.
```

- **HDL Synthesis**

During this step, XST tries to recognize as many basic macros as possible ,to create a technology specific implementation.

```
=====
*                               HDL Synthesis                               *
=====

Performing bidirectional port resolution...

Synthesizing Unit <shift_add_multiplier>.
  Related source file is "/home/users/pallavip/multiply/shift_add_multiplier_synth/shift_add_multiplier/shift_add_multiplier.vhd"
  Using one-hot encoding for signal <cur_state>.
  Using one-hot encoding for signal <nxt_state>.
WARNING:Xst - Property "use_dsp48" is not applicable for this technology.
  Found 1-bit register for signal <mul_done_o>.
  Found 1-bit register for signal <mul_rdy_o>.
  Found 3-bit register for signal <cur_state>.
  Found 64-bit register for signal <mul1_shift_reg>.
  Found 32-bit register for signal <mul2_shift_reg>.
  Found 64-bit register for signal <product_d1>.
  Found 64-bit adder for signal <product_d1$addsub0000> created at line 174.
Summary:
  inferred 162 D-type flip-flop(s).
  inferred 1 Adder/Subtractor(s).
Unit <shift_add_multiplier> synthesized.
```

# Synthesizing Design – Analyzing Synthesis Report

Following is the sectional details of Synthesis Report (.syr) file :-

- **Advanced HDL Synthesis** : During this step, XST performs advanced macro recognition and inference like recognizing dynamic shift registers, implementing pipelined multipliers, coding state machines, etc.

(Following is the Advanced Synthesis Report section generated for Shift & Add Multiplier)

```
=====
*                               Advanced HDL Synthesis                               *
=====

Loading device for application Rf_Device from file '3s500e.nph' in environment /cad/Xilinx.

=====
Advanced HDL Synthesis Report

Macro Statistics
# Adders/Subtractors      : 1
  64-bit adder            : 1
# Registers               : 165
Flip-Flops               : 165
```

- **Low Level Synthesis** : During this step XST reports the potential removal of equivalent flip-flops, register replication, etc

```
Final Results
RTL Top Level Output File Name : shift_add_multiplier.ngc
Top Level Output File Name     : shift_add_multiplier
Output Format                   : NGC
Optimization Goal               : Speed
Keep Hierarchy                 : NO
```

```
Design Statistics
# IOs                          : 133
```

- **Final Report** : includes (NGC) output file name (having extension .ngc), target family and cell usage.

```
Cell Usage :
# BELS      : 643
# GND       : 1
# LUT2      : 104
# LUT3      : 20
# LUT4      : 256
# MUXCY     : 71
# MUXF5     : 127
# VCC       : 1
# XORCY     : 63
# FlipFlops/Latches : 165
# FDC       : 164
# FDP       : 1
# Clock Buffers : 1
# BUFGP     : 1
# IO Buffers  : 132
# IBUF      : 66
# OBUF      : 66
```



# Synthesizing Design – Analyzing Synthesis Report

Final Report also includes following information :-

- Device Utilization Summary: where XST estimates the number of slices, gives the number of flip-flops, IOBs, BRAMS, etc. This report is very close to the one produced by MAP.

*Following is the Device Utilization Summary & Clock Information for Shift Add Multiplier Synthesis :*

Device utilization summary:

-----				
Selected Device : 3s500epq208-5				
Number of Slices:	198	out of	4656	4%
Number of Slice Flip Flops:	165	out of	9312	1%
Number of 4 input LUTs:	380	out of	9312	4%
Number of IOs:	133			
Number of bonded IOBs:	133	out of	158	84%
Number of GCLKs:	1	out of	24	4%

Clock Information:

-----		
Clock Signal	Clock buffer(FF name)	Load
-----		
clk_i	BUFGP	165
-----		

Asynchronous Control Signals Information:

-----		
Control Signal	Buffer(FF name)	Load
-----		
rst_i	IBUF	165
-----		

- Clock Information: gives information about the number of clocks in the design, how each clock is buffered and how many loads it has.
- Timing report: Timing Summary section gives a summary of following timing paths :-
  - (i) The path from any clock to any clock (*i.e. flop to flop delay*) in the design:
  - (ii) The maximum path from all primary inputs to the sequential elements.
  - (iii) The maximum path from the sequential elements to all primary outputs.
  - (iv) The maximum path from input to output pad.

Timing Summary:

-----  
Speed Grade: -5

Minimum period: 7.295ns (Maximum Frequency: 137.079MHz)  
Minimum input arrival time before clock: 5.269ns  
Maximum output required time after clock: 4.182ns  
Maximum combinational path delay: No path found

*Timing Summary for Shift Add Multiplier Synthesis ->*



# Synthesizing Design– Generating Post-Synthesis Simulation Model

## Requirement & Generation of Post-Synthesis Simulation Model :

To verify whether the correct functionality of the design is retained, after synthesizing it into netlist; HDL Design's equivalent simulation model can be generated, by clicking on **Generate Post-Synthesis Simulation Model** option within **Synthesis-XST** process list. (Netlist simulation model is generated in **netgen/synthesis** directory)

## Post-Synthesis Simulation using Modelsim :

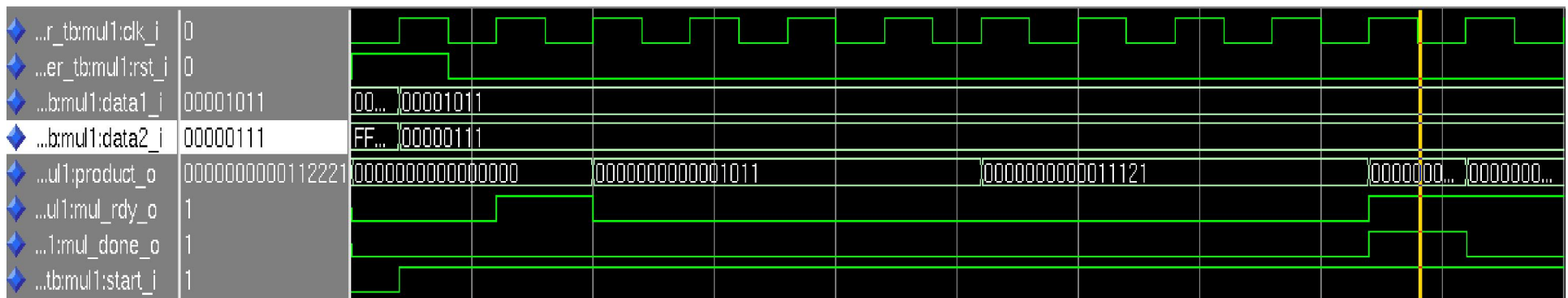
Post-Synthesis Simulation Model can be compiled & simulated using the same HDL testbench, that was used for HDL behavioral code verification.

In Modelsim, after creating a new project for Post Synthesis Simulation, include Netlist (**\_synthesis.vhd**) from **<ise\_project>/netgen/synthesis** directory. This netlist file is compiled along with testbench, instead of HDL behavioral Code being compiled.

## Post-Synthesis Simulation Result :

For the generated Post-Synthesis Simulation Model, no standard delay file (.sdf file) is back-annotated during simulation. (Thus, UNISIM Library primitives, included in the synthesis generated netlist, do not have any delay associated with it)

Therefore, expected Post-Synthesis Simulation result is same as Functional Verification result of HDL Design, as can be seen in following Post-Synthesis simulation waveform for Shift-Add Multiplier design :-



# Specifying User Constraints

## Need for Setting Constraints :

For Design to meet desired Area/Frequency Specification on FPGA, it is required to tell the Implementation Tool for what performance it should optimize the design implementation processes like Map, Place & Route. Thus, User Defined Constraints allows to specify desired Clock period/ Pad to Setup/Clock to Pad delay & assign areas to hierarchical blocks of logic.

Physical User Constraint also allows to allocate HDL design's I/O signals to specific package pins.

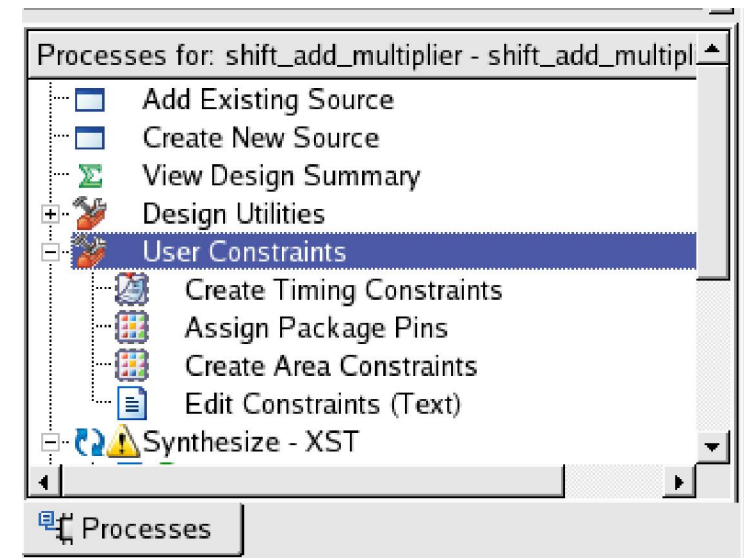
## Adding Constraints to Design :

User can specify Constraints for ISE Design Project , either through GUI by double-clicking on following options available within *User Constraints* label in *Processes* Window:-

- (i) *Create Timing Constraints*
- (ii) *Assign Package Pins*
- (iii) *Create Area Constraints*

(or) the User can specify constraints in **.ucf** file, through any text editor.

While opening Constraint Editor window, Translate step runs automatically because implementation stage must see the netlist before it can offer the user the chance to constraint sections of design.



# Specifying User Constraints – Timing Constraints

## (i) Timing Constraints :

The *Global Clock Domain* and *Input/Output Ports* tab of *Create Timing Constraints* window automatically displays all clock nets in the design, and enables designer to define the associated *Period*, *Pad to Setup*, and *Clock to Pad* values.

**Xilinx Constraints Editor**

File Edit View Window Help

Global\*

Clock Net Name	Period	Pad to Setup	Clock to Pad
clk_i	7.5 ns HIGH 50 %	2 ns	2 ns

Pad to Pad...

Ports\*

Port Name	Port Direction	Pad to Setup	Clock to Pad
data2_i<6>	INPUT		N/A
data2_i<7>	INPUT		N/A
data2_i<8>	INPUT		N/A
data2_i<9>	INPUT		N/A
mul_done_o	OUTPUT	N/A	
mul_rdy_o	OUTPUT	N/A	
product_o<0>	OUTPUT	N/A	
product_o<1>	OUTPUT	N/A	

Global\* Ports\* Advanced\* Misc\*

These constraints can be specified in .ucf text file, as an alternative to using Constraint Editor GUI Window

```
NET "clk_i" TNM_NET = "clk_i";
TIMESPEC "TS_clk_i" = PERIOD "clk_i" 7.5 ns HIGH 50 %;
OFFSET = IN 2 ns BEFORE "clk_i" ;
OFFSET = OUT 2 ns AFTER "clk_i" ;
```

Syntax for Clock Period Constraint  
Syntax for Pad to Setup Constraint  
Syntax for Clock to Pad Constraint

(Since, there are no combinatorial paths in Shift Add Multiplier design, Pad to Pad constraint is not specified for this case.)

ISE tool Timing Analyzer is used to analyze the results of these timing specifications for the design.

# Specifying User Constraints – Timing Constraints

[Source : Xilinx Timing Constraint User Guide]

**Clock Period** constraint ensures that the internal paths starting and ending at synchronous points (Flip-Flops /RAM / Latches ) have logic delay less than Maximum Delay allowed in the design specification.

**Pad to Setup** is the path starting at Input Port of the design and ending at an input to a flip-flop/latch/RAM—wherever there is a setup time against a control signal.

The Pad to Setup constraint defines the maximum time required for the data to enter the FPGA, travel through logic and routing, and arrive at the input before the clock or control signal arrives.

*Pad-to-Setup Delay Constraint can be calculated as follows :-*

$$T_{\text{Data}} + T_{\text{Setup}} - T_{\text{Clock}} \leq T_{\text{Offset\_IN\_BEFORE}}$$

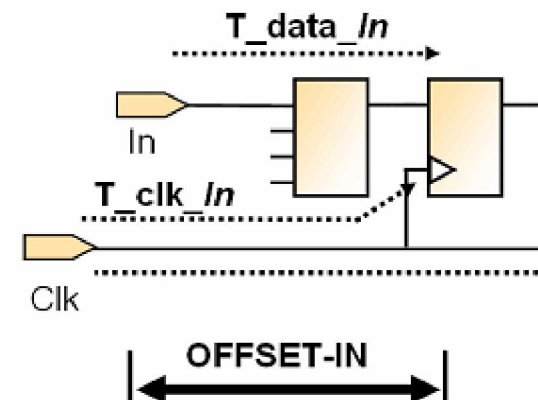
where,

$T_{\text{Data}}$  = Total Data path delay from the Flip Flop

$T_{\text{Setup}}$  = Intrinsic Flip Flop setup time

$T_{\text{Clock}}$  = Total Clock path delay to the Flip Flop

$T_{\text{Offset\_IN\_BEFORE}}$  = Overall Setup Requirement



**Clock to Pad** is the path starting at the Q output of a flip-flop or latch and ending at Output Port of the design. It includes the Clock-to-Q delay of the flip-flop and path delay from that flip-flop to FPGA output.

The Clock to Pad constraint defines the maximum time required for the data to leave the source flip-flop, travel through logic and routing, and arrive at output pin of FPGA.

The clock-to-pad path time is the maximum time required for the data to leave the source flip-flop, travel through logic and routing, and leave the chip.

# Specifying User Constraints – Timing Constraints

[Source : Xilinx Timing Constraint User Guide]

*Clock to Pad Delay Constraint can be calculated as follows :-*

$$T_Q + T_{\text{Data2Out}} + T_{\text{Clock}} \leq T_{\text{Offset\_OUT\_AFTER}}$$

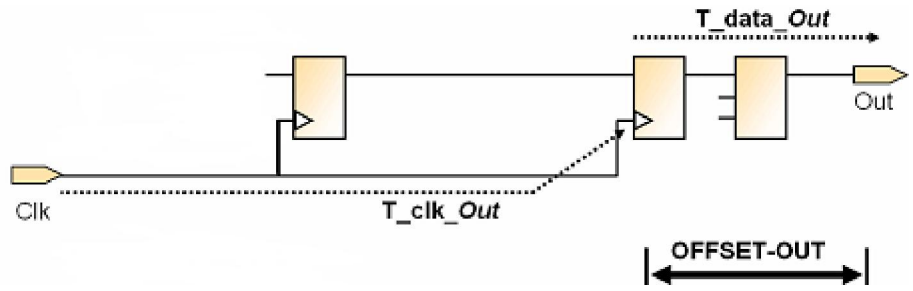
where,

$T_Q$  = Intrinsic Flip Flop Clock to Out

$T_{\text{Clock}}$  = Total Clock path delay to the Flip Flop

$T_{\text{Data2Out}}$  = Total Data path delay from the Flip Flop

$T_{\text{Offset\_OUT\_AFTER}}$  = Overall Clock to Out Requirement

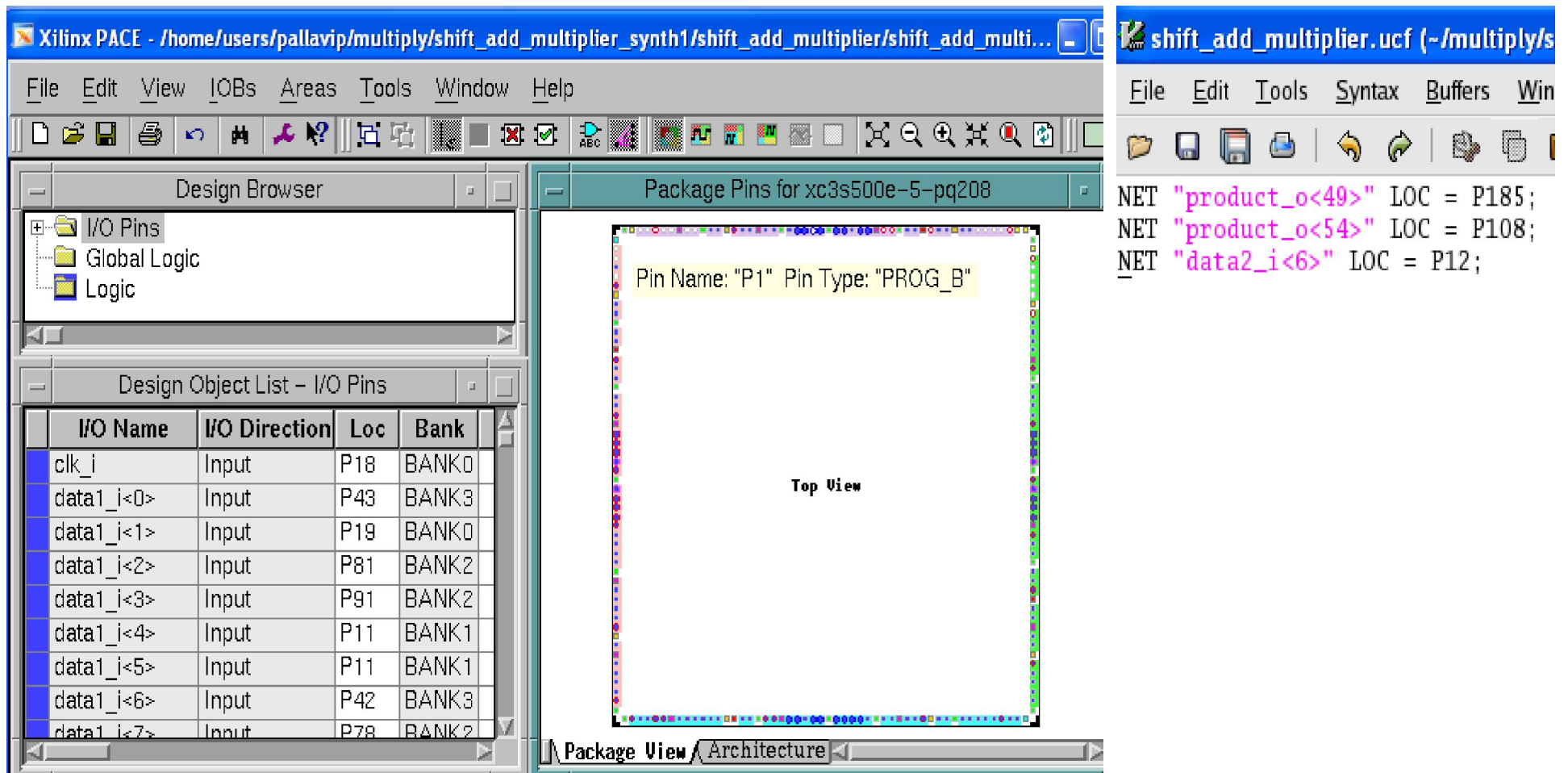


**Pad to Pad** constraint constrains combinatorial asynchronous paths having start and endpoints as Pads of the design.

# Specifying User Constraints – Assigning Package Pins

## (ii) Assigning Package Pins :

Package Pins can be assigned to Design's Input/Output Ports through **LOC** constraint in **.ucf** file or through **Assign Package Pin** GUI Window.



The screenshot displays the Xilinx PACE software interface for assigning package pins. The Design Object List - I/O Pins table is shown below:

I/O Name	I/O Direction	Loc	Bank
clk_i	Input	P18	BANK0
data1_i<0>	Input	P43	BANK3
data1_i<1>	Input	P19	BANK0
data1_i<2>	Input	P81	BANK2
data1_i<3>	Input	P91	BANK2
data1_i<4>	Input	P11	BANK1
data1_i<5>	Input	P11	BANK1
data1_i<6>	Input	P42	BANK3
data1_i<7>	Input	P78	BANK2

The Package Pins for xc3s500e-5-pq208 window shows the top view of the package with a pin labeled "P1" of type "PROG\_B".

The .ucf file content is shown in the shift\_add\_multiplier.ucf window:

```
NET "product_o<49>" LOC = P185;  
NET "product_o<54>" LOC = P108;  
NET "data2_i<6>" LOC = P12;
```

# Translating Design

## Translate Process Functionality :

During translation, the NGDBuild program performs the following functions :-

- Converts input design netlists and writes results to a single merged NGD netlist. The merged netlist describes the logic in the design as well as any location and timing constraints.
- Performs timing specification and logical design rule checks.
- Adds constraints from the User Constraints File (UCF) to the merged netlist.

## Steps to Translate the Design :

- Translate Process gets automatically executed while opening Constraint Editor GUI Window (or) User can right-click *Translate* option in *Processes* Window and select *Run* option.
- To set *Translate Properties*, right-click *Translate* in the *Processes* window; select *Properties* to display the Process Properties dialog box.
- Synthesis Report file is stored with extension <project\_name>.**blb** in ISE Project Directory.

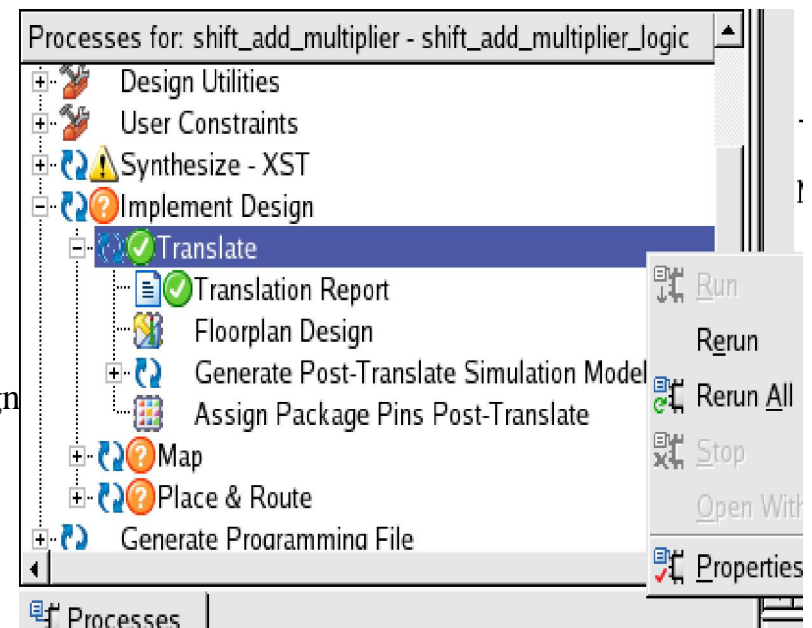
## Translate Process File Types :

Translate Process uses following files as input :-

- § NGC netlist file from Synthesis Process.
- § UCF constraint file containing timing and layout constraints.

Translate Process creates following files as output :-

- § NGD file, containing logical description of the design, expressed in terms of lower level Xilinx Primitives, with constraint applied to design
- § BLD Report file shows following error in design or UCF file :-
  - Missing or untranslatable hierarchical blocks
  - Invalid or incomplete timing constraints
  - Output contention, loadless outputs, and sourceless inputs





# Translating Design – Understanding Translate Options

[Source : Xilinx Development System Reference Guide]

**Use LOC Constraints** :- Deselecting this option allows to ignore *Location* constraint in UCF file, when user may require to migrate to a different device or architecture, because location in one architecture may not match location in another.

**Create I/O Pads from Ports** :- Adds a PAD symbol to every signal that is connected to a port on the root-level cell.

**Allow Unexpanded Blocks** :- Translate Process generates an error if a block in the design cannot be expanded to NGD primitives. If *Allow Unexpanded Blocks* option is selected, only warning is generated instead of an error, and NGD file is still written, containing the unexpanded block.

This option is used to perform preliminary mapping, placement and routing, timing analysis, or simulation on the design, even though the design is not complete.

**Allow Unmatched LOC Constraints** :- Translate Process generates error if the constraints specified for pin, net, or instance names in the UCF file cannot be found in the design. If this error occurs, an NGD file is not written. If *Allow Unmatched LOC* option is selected, Translate Process generates a warning instead of an error for LOC constraints, and still writes an NGD file.

This option is useful if User Constraints File includes location constraints for pin, net, or instance names that have not yet been defined in the HDL or schematic. This allows user to maintain single version of User Constraints File for both partially complete and final designs.

# Mapping the Design

## MAP Process Functionality :

- Allocates CLB and IOB resources for all basic logic elements in the design.
- Processes all location and timing constraints, performs target device optimizations, and runs a design rule check on the resulting mapped netlist.

## Steps to Map the Design :

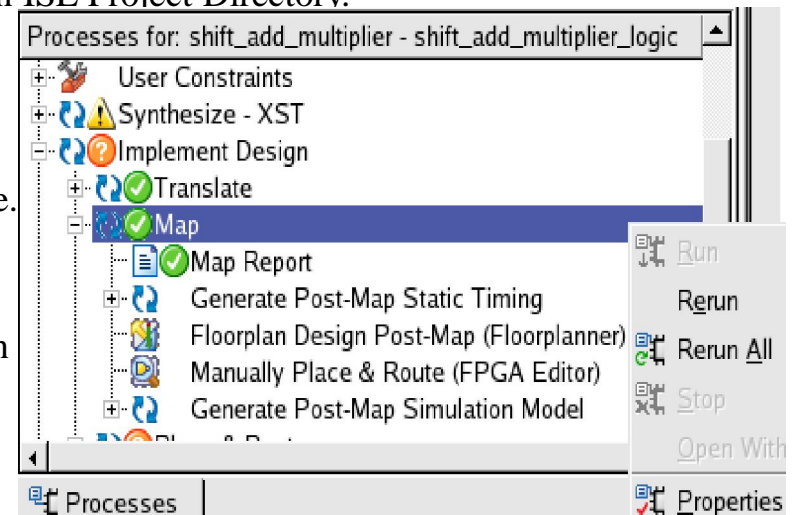
- To set Map Process Properties, right-click *Map* in the *Processes* window; select *Properties* to display the Process Properties dialog box.
- Right-click *Map* label in *Processes* Window and select *Run* option.
- Synthesis Report file is stored with extension <project\_name>.mrp in ISE Project Directory.

## Map Process File Types :

MAP Process uses NGD file, created during Translate Process, as Input file.

MAP Process creates following files as output :-

- NCD (Native Circuit Description) file containing physical description of design in terms of the components in the target Xilinx device.
- PCF (Physical Constraints File) contains constraints specified during design entry expressed in terms of physical elements.
- MRP (MAP Report File) confirms the resources used within the device; and describes trimmed and merged logic. Detailed Report also describes exactly where each portion of the design is located in the device.



# Mapping Design – Understanding MAP Options

[Source : Xilinx Development System Reference]

- **Perform Timing driven packing & Placement** :- directs MAP to give priority to timing critical paths during packing, then places the design. Timing-driven packing and placement is recommended to improve design performance, timing, and packing for highly utilized designs.
- **Map Effort Level** :- specifies the level of effort MAP uses to pack the design.
- **Extra Effort** :- *Continue on Impossible* allows to direct MAP to continue and improve packing, until little or no improvement can be made.
- **Combinatorial Logic Optimization** :- Invokes post-placement logic restructuring for improved timing and design performance.
- **Register Duplication** :- option duplicates registers to improve timing when running timing-driven packing.
- **Replicate Logic to Allow Logic level Reduction** :- Logic replication is an optimization method in which MAP operates on a single driver that is driving multiple loads and maps it as multiple components, each driving a single load which makes it easier to meet timing requirements, since some delays can be eliminated on critical nets.
- **Allow Logic Optimization across Hierarchy** :- specifies whether *Area/Speed/Balanced* criteria has to be used during the *cover* phase of MAP, during which, MAP assigns the logic to CLB function generators (LUTs).
- **Use RLOC Constraints** :- Unchecking this option allows to ignore the RLOC constraint that cannot be met.
- **Disable Register Ordering** :- By default, MAP looks at the register bit names for similarities and tries to map register bits in an ordered manner. Specify this option, register bit names are ignored when registers are mapped, and the bits are not mapped in any special order.
- **CLB Pack Factor Percentage** :- determines the degree to which CLBs are packed when the design is mapped.
- **MAP Slice Logic into Unused Block RAMs** :- When block RAM mapping is enabled, MAP attempts to place LUTs and FFs into single-output, single-port block RAMs.
- **Power Reduction** :- Specifies that placement is optimized to reduce the power consumed by a design during timing-driven packing and placement.

# Mapping Design – Analyzing MAP Report

Following is the sectional details of MAP Report (.mrp) file :-

- Design Summary - Summarizes the mapper run, showing the number of errors and warnings, and how many of the resources in the target device are used by the mapped design.

## Design Summary

```
-----
Number of errors:      0
Number of warnings:    0
Logic Utilization:
  Number of Slice Flip Flops:      163 out of  9,312    1%
  Number of 4 input LUTs:         379 out of  9,312    4%
Logic Distribution:
  Number of occupied Slices:              202 out of  4,656    4%
  Number of Slices containing only related logic:  202 out of    202  100%
  Number of Slices containing unrelated logic:    0 out of    202    0%
  *See NOTES below for an explanation of the effects of unrelated logic
Total Number of 4 input LUTs:      379 out of  9,312    4%
  Number of bonded IOBs:          133 out of   158   84%
  IOB Flip Flops:                   2
  Number of GCLKs:                  1 out of    24    4%
Total equivalent gate count for design:  4,377
Additional JTAG gate count for IOBs:    6,384
```

Section 4 - Removed Logic Summary

---

2 block(s) optimized away

Section 5 - Removed Logic

---

Optimized Block(s):

TYPE	BLOCK
GND	XST_GND
VCC	XST_VCC

- Removed Logic - Describes in detail all logic (design components and nets) removed for the following reasons, from the input NGD file when the design is mapped :-
  - The design uses only part of the logic in a library macro.
  - The design has been mapped even though it is not yet complete.
  - The mapper has optimized the design logic.
  - Unused logic has been created in error during schematic entry.

# Mapping Design – Analyzing MAP Report

- IOB Properties - Lists each IOB to which the user has supplied constraints along with the applicable constraints.

## Section 6 - IOB Properties

IOB Name	IOB Type	Direction	IO Standard	Drive Strength	Slew Rate	Reg (s)	Resistor	IBUF/IFD Delay
clk_i	IBUF	INPUT	LVC MOS25					0 / 0
data1_i<0>	IBUF	INPUT	LVC MOS25					0 / 0
data1_i<1>	IBUF	INPUT	LVC MOS25					0 / 0
data1_i<2>	IBUF	INPUT	LVC MOS25					0 / 0

- Timing Report - This section, produced with *Perform Timing driven packing & Placement* option, shows information on timing constraints considered during the MAP run.

## Section 11 - Timing Report

INFO:Timing:3284 - This timing report was generated using estimated delay information. For accurate numbers, please refer to the post Place and Route timing report.

Asterisk (\*) preceding a constraint indicates it was not met.  
This may be due to a setup or hold violation.

Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
* TS_clk_i = PERIOD TIMEGRP "clk_i" 7.5 ns HIGH 50%	SETUP HOLD	-0.091ns 1.263ns	7.591ns	1 0	91 0

1 constraint not met.

# Placing and Routing the Design

## Place And Route Process Functionality :-

- During placement, PAR places components into sites based on factors such as constraints, the length of connections, and the available routing resources.
- After placing the design, the router performs a converging procedure for a solution that routes the design to completion and meets timing constraints.

## Steps to Place and Route the Design :-

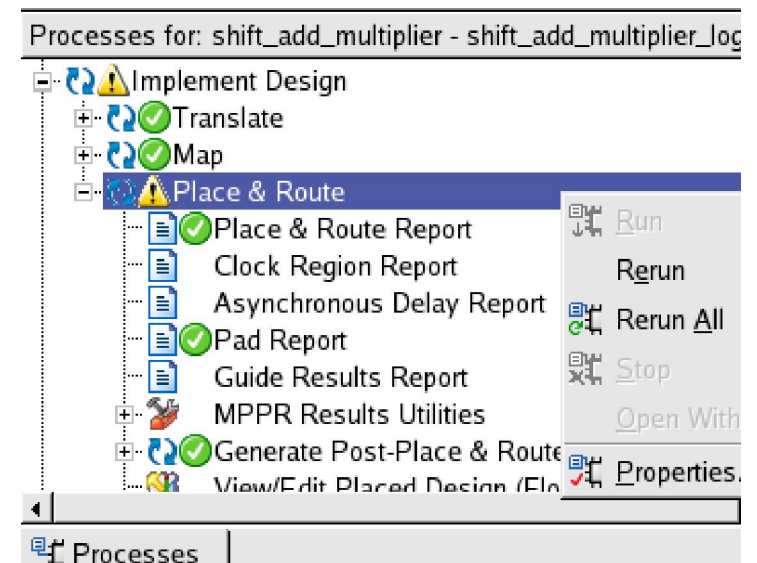
- To execute PAR in the *Processes* tab, right-click *Place & Route* under the *Implement Design* process group, and select *Run* option.
- To set *Place and Route* Properties, right-click *Place & Route* in the *Processes* window; select *Properties* to display the *Process Properties* dialog box.

## PAR Process File Types :-

PAR Process uses Mapped Design (NCD) File and Physical Constraint (PCF) file created during MAP Process, as Input File.

PAR Process creates following files as output :-

- Placed and Routed NCD Design File
- PAR Report File, including summary information of all placement and routing iterations.



# Placing and Routing Design – *Understanding PAR Options*

[Source : Xilinx Development System Reference Guide]

- **Place and Route Effort Level** :- specifies the level of effort PAR uses to place and route design to completion and to achieve timing constraints.
- **Extra effort Level** :- option *Continue on Impossible* allows user to direct PAR to continue routing, even if PAR determines the timing constraints cannot be met. PAR , then, continues to attempt to route and improve timing until little or no timing improvement can be made.
- **Use Timing Constraints** :- On deselecting this option, all timing constraints are ignored and the implementation tools do not use any timing information to place and route the design.
- **Power Reduction** :- option optimizes the capacitance of non-timing driven design signals



# Placing and Routing Design – Analyzing PAR Report

Following is the sectional details of PAR Report (. *par*) file :-

- **Design Summary** - Provides a breakdown of the resources in the design and includes the Device Utilization Summary

Design Summary Report:

Number of External IOBs	133 out of 158	84%
Number of External Input IOBs	67	
Number of External Input IBUFs	67	
Number of LOCed External Input IBUFs	67 out of 67	100%
Number of External Output IOBs	66	
Number of External Output IOBs	66	
Number of LOCed External Output IOBs	66 out of 66	100%
Number of External Bidir IOBs	0	
Number of BUFGMUXs	1 out of 24	4%
Number of Slices	202 out of 4656	4%
Number of SLICEMs	0 out of 2328	0%

- **Clock Report** - Lists all clocks in the design and provides information on the routing resources, number of fanout, maximum net skew for each clock, and the maximum delay. The locked column in the clock table indicates whether the clock driver (BUFGMUX) is assigned to a particular site or left floating..

Clock Net	Resource	Locked	Fanout	Net Skew(ns)	Max Delay(ns)
clk_i_BUFGP	BUFGMUX_X2Y11	No	153	0.055	0.159

# Placing and Routing Design – Analyzing PAR Report

- **Delay Summary Report** - Summarizes the connection and pin delays for the design.

The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is: 0

The AVERAGE CONNECTION DELAY for this design is: 1.211

The MAXIMUM PIN DELAY IS: 4.125

The AVERAGE CONNECTION DELAY on the 10 WORST NETS is: 3.290

Listing Pin Delays by value: (nsec)

d < 1.00	< d < 2.00	< d < 3.00	< d < 4.00	< d < 5.00	d >= 5.00
838	648	286	46	3	0

- **Timing Score** - Lists information on timing constraints contained in the input PCF, including how many timing constraints were met.

Timing Score: 0

Asterisk (\*) preceding a constraint indicates it was not met.

This may be due to a setup or hold violation.

Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
TS_clk_i = PERIOD TIMEGRP "clk_i" 7.5 ns HIGH 50%	SETUP HOLD	0.021ns 1.148ns	7.479ns	0 0	0 0

All constraints were met.

**Slack** value indicates the difference between the constraint and the analyzed value, with negative slack indicating an error condition.

- **Setup Slack** informs about the amount of setup violation seen for a path, and is calculated as :

$$\text{Setup Slack} = \text{Constraint\_requirement} - T_{\text{clock\_skew}} - T_{\text{data\_path}} - T_{\text{su}}$$

- **Hold Slack** :- Hold/Race checks are performed on register-to-register paths by taking the data path ( $T_{\text{ckQ}} + T_{\text{route\_total}} + T_{\text{logic\_total}}$ ) and subtracting the clock skew ( $T_{\text{dest\_clk}} - T_{\text{src\_clk}}$ ) and the register hold delay ( $T_{\text{h}}$ ) i.e.

$$\text{Hold Slack} = T_{\text{data}} - T_{\text{skew}} - T_{\text{hold}}$$

# Placing and Routing Design – Asynchronous Delay Report

- Asynchronous Delay Report is concerned with Worst Path logic and routing delays in the design

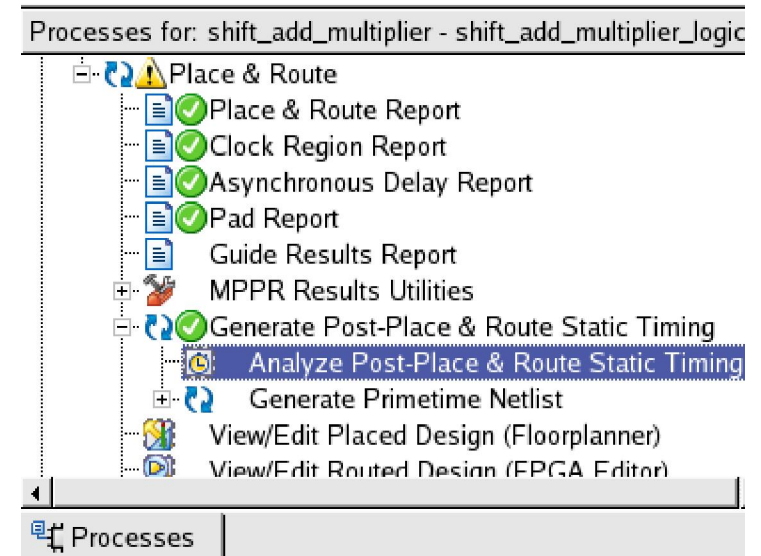
For e.g. Following is the data from Shift Add Multiplier Design Asynchronous Delay Report :-

```
File: shift_add_multiplier.dly

The 20 worst nets by delay are:
+-----+
| Max Delay      | Netname      |
+-----+
    4.125        start_i_IBUF
    3.520        data1_i_4_IBUF
    3.471        data1_i_25_IBUF
    3.438        product_d1<10>
    3.325        cur_state<0>
    3.057        product_d1<43>
    3.050        product_d1<54>
    3.013        data2_i_8_IBUF
    2.975        product_d1<41>
    2.927        product_d1<22>
    2.841        product_d1<29>
    2.821        product_d1<24>
    2.785        product_d1<34>
    2.690        cur_state<2>
    2.656        product_d1<33>
    2.648        data1_i_12_IBUF
    2.617        data1_i_21_IBUF
    2.617        product_d1<37>
    2.611        mul2_shift_reg<0>
    2.604        cur_state<1>
```

# Placing and Routing Design - *Post PAR Static Timing Analysis*

- Post PAR Static Timing report evaluates the logical block delays and the routing delays.
- To display this report, run *Analyze Post-Place & Route Static Timing* process in the *Processes* view under *Implement Design > Place & Route > Generate Post-Place & Route Static Timing* label.



*Advanced Design Analysis* report can be generated by selecting *Perform Advanced Analysis* in *Post Place and Route Static Timing Report Properties* (accessed by right-clicking on *Generate Post-Place and Route Static Timing* label and selecting *Properties* option)

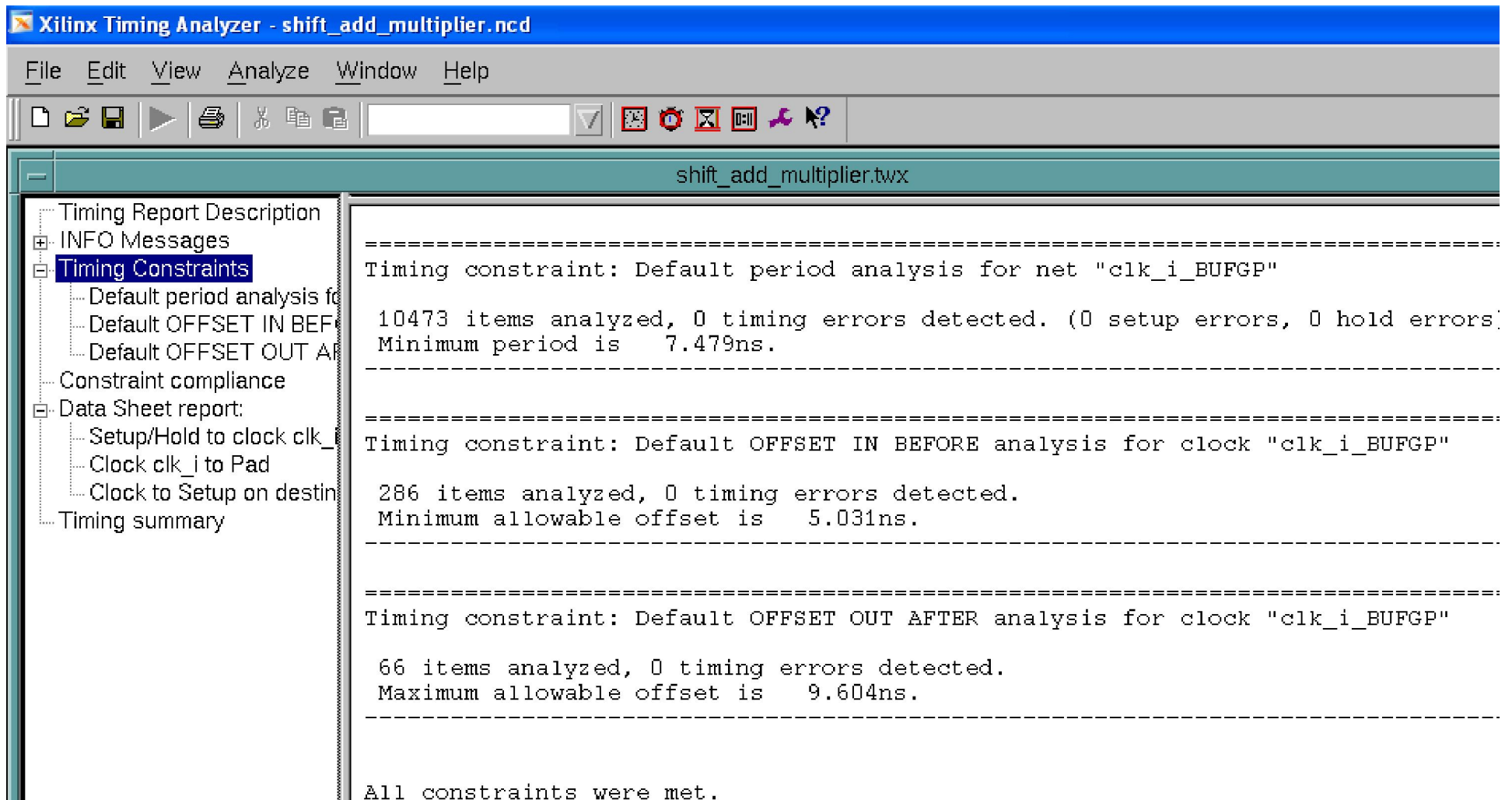
Post-Place & Route Static Timing Report Properties	
Property Name	Value
Report Type	Error Report
Number of Items in Error/Verbose Report (0 - 2 Billion)	3
Perform Advanced Analysis	<input checked="" type="checkbox"/>

The Timing Report will open in *Timing Analyzer* window.

# Placing and Routing Design - *Post PAR Static Timing Analysis*

Following is the sectional details of *Analyze Post Place and Route Static Timing* GUI window options and Static Timing Report (. *twr*) text file :-

(i) *Timing Constraints Analysis* report compares the design's performance to the timing constraints.



# Placing and Routing Design - Post PAR Static Timing Analysis

- (ii) *Data Sheet Report* includes the source and destination PAD names, and either the propagation delay between the source and destination or the setup and hold requirements for the source relative to the destination.

Timing Constraints	Data Sheet report:
Default period analysis for	-----
Default OFFSET IN BEFORE	All values displayed in nanoseconds (ns)
Default OFFSET OUT AFTER	
Constraint compliance	Setup/Hold to clock clk_i
<b>Data Sheet report:</b>	-----+-----+-----+-----+-----+
Setup/Hold to clock clk_i	Source   Setup to   Hold to   Internal Clock(s)   Clock
Clock clk_i to Pad	clk (edge)   clk (edge)       Phase
Clock to Setup on destination	-----+-----+-----+-----+-----+
Timing summary	data1_i<0>   2.211(R)   -0.645(R)   clk_i_BUFGP   0.000
	data1_i<1>   1.958(R)   -0.427(R)   clk_i_BUFGP   0.000
	data1_i<2>   1.819(R)   -0.304(R)   clk_i_BUFGP   0.000
	data1_i<3>   2.253(R)   -0.675(R)   clk_i_BUFGP   0.000
	-----+-----+-----+-----+-----+
Default OFFSET IN BEFORE	Clock clk_i to Pad
Default OFFSET OUT AFTER	-----+-----+-----+-----+-----+
Constraint compliance	Destination   clk (edge)   Internal Clock(s)   Clock
<b>Data Sheet report:</b>	to PAD       Phase
Setup/Hold to clock clk_i	-----+-----+-----+-----+-----+
<b>Clock clk_i to Pad</b>	mul_done_o   5.611(R)   clk_i_BUFGP   0.000
Clock to Setup on destination	mul_rdy_o   5.584(R)   clk_i_BUFGP   0.000
Timing summary	product_o<0>   8.517(R)   clk_i_BUFGP   0.000
	product_o<1>   7.722(R)   clk_i_BUFGP   0.000
	-----+-----+-----+-----+-----+
Constraint compliance	Clock to Setup on destination clock clk_i
<b>Data Sheet report:</b>	-----+-----+-----+-----+-----+
Setup/Hold to clock clk_i	Source Clock   Src:Rise   Src:Fall   Src:Rise   Src:Fall
Clock clk_i to Pad	Dest:Rise   Dest:Rise   Dest:Fall   Dest:Fall
<b>Clock to Setup on destination</b>	-----+-----+-----+-----+-----+
Timing summary	clk_i   7.479
	-----+-----+-----+-----+-----+

Negative Hold Time indicates that data pin of the flip-flop can change ahead of the clock pin and still meet Hold Time check, due to internal data path-delay of Flip-Flop.

# Placing & Routing Design-Generating Post PAR Simulation Model

## Requirement & Generation of Post-Place & Route Simulation Model :

Timing Simulation is required to verify whether the correct functionality of the design is retained, after the netlist is back-annotated with logic and routing delay information.

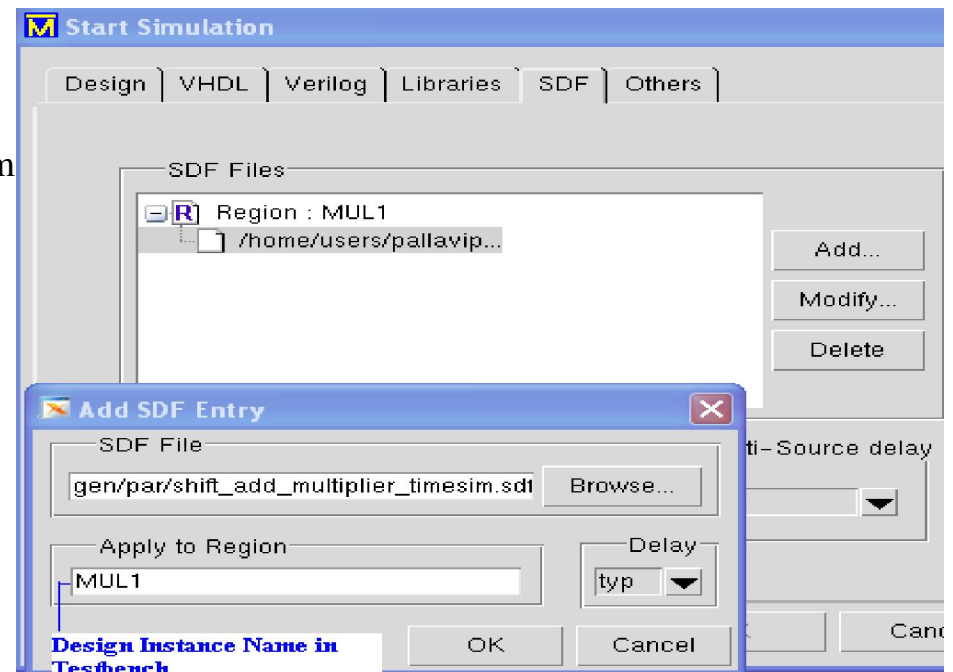
Equivalent simulation model for Placed & Routed Netlist can be generated, by clicking on **Generate Post-Place & Route Simulation Model** option within **Place & Route** process list. (Netlist simulation model is generated in **netgen/par** directory)

## Timing Simulation using Modelsim :

Post-PAR Simulation Model can be compiled & simulated using the same HDL testbench, that was used for HDL behavioral code verification.

i) In Modelsim, after creating a new project for Post Synthesis Simulation, include Netlist (**\_synthesis.vhd**) from **<ise\_project>/netgen/par** directory. This netlist file is compiled along with testbench, instead of HDL behavioral Code being compiled.

ii) After compiling Netlist & Testbench, add SDF file (containing Netlist Logic & Routing Delay information) from **<ise\_project>/ netgen/ par** directory, through *Simulate* Menu -> *Start Simulation* Window -> *SDF* tab.

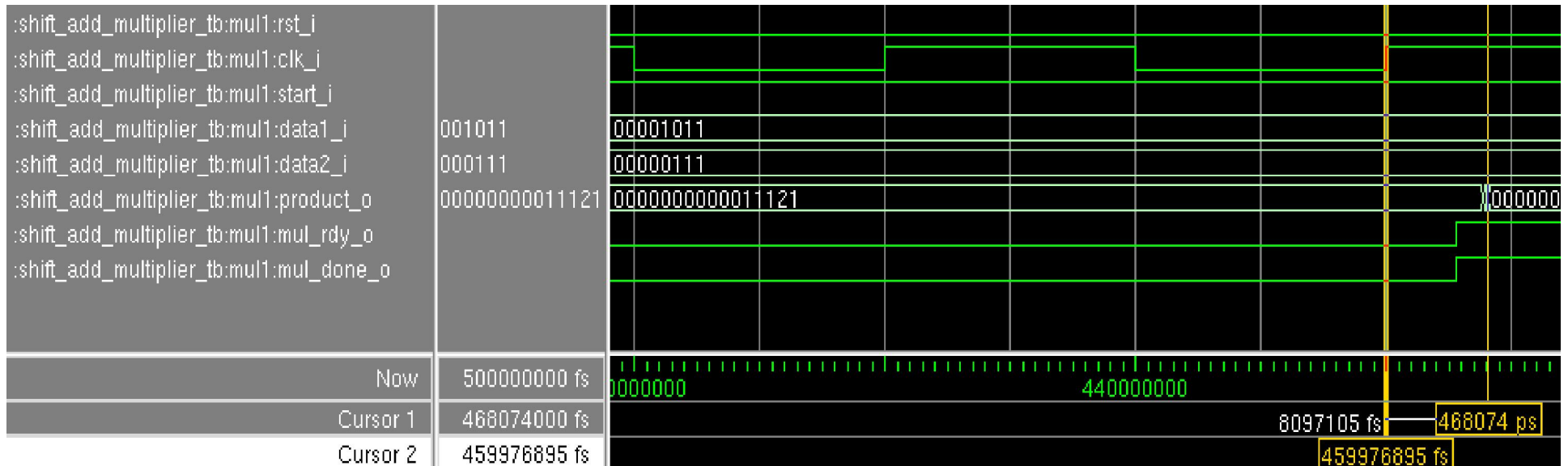




# Placing & Routing Design-Generating Post PAR Simulation Model

## **Post-PAR Simulation Result :**

Since, in this Timing Simulation, SDF file containing logic/routing delay information is back-annotated to the netlist, simulation waveform result shows delay in updating output signal ports, after input clock and data is applied to the design.



For e.g., In case of Shift-Add Multiplier Design, as indicated in Static Timing Report (.twr), Clock to Pad Output Path Delay is around 8ns for *product\_dly\_o* data bus, which is also observed in Post PAR Simulation Waveform, wherein Registered Product data output appears after 8ns relative to Clock rising edge.

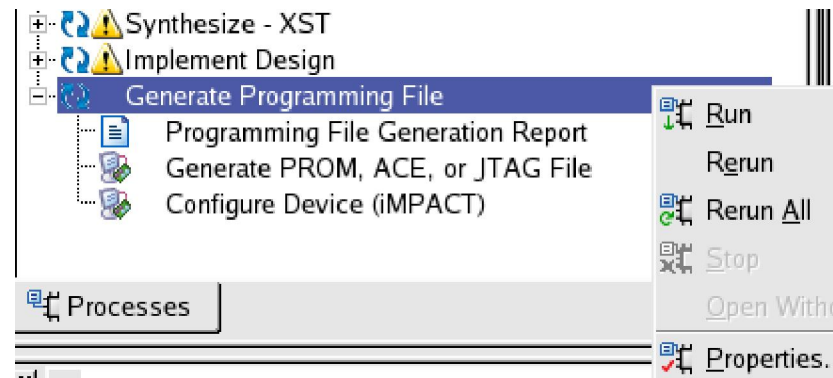
# Generating BitMap Programming File

## Purpose of Generating Programming File :

After design has been routed, it is required to generate the binary data which can be used to program the physical device. The Programming BIT File for FPGA Device should contain all the configuration information, defining the internal logic and interconnections of the FPGA, plus device-specific information from other files associated with the target device. The binary data in the BIT file can then be downloaded into the FPGA's memory cells or it can be used to create a PROM file.

## Step to Generate Programming File :

To create BitMap file, right-click *Generate Programming File* label in *Processes* window & click *Run* option.



## BitGen Process File Types :

Xilinx uses *BitGen* process for generating Bitstream program. *BitGen* takes a fully routed NCD file, generated during PAR process as its input, and produces a configuration bitstream - a binary file with a **.bit** extension.