XAPP918 (v1.0) June 7, 2007

Incremental Design Reuse with Partitions

Author: Chris Zeh

Summary

This Application Note discusses the use of Partitions in the Incremental Design Flow. It is recommended that module instances with high logic density, timing critical paths, or timing critical module instances be designated Partitions. Designating a module instance as a Partition instructs the synthesis and implementation tools to use the previous implementation results for that module instance, if it has not been modified. The advantages of Partitions are that they can reduce implementation tool runtime, eliminate re-verification of unchanged module instances, and help achieve timing closure. Partitions can be manipulated via Project Navigator or the Tcl interface, and enable fine-grained control over how implementation results are reused.

What is a Partition?

Partitions preserve unchanged portions of the design that have been previously implemented. Partitions can optimize the implementation process of a design. If a Partition's HDL, timing and physical constraints, and implementation options are unchanged, the implementation tools will use a “copy-and-paste” process to guarantee that the implementation data for that Partition is preserved.

By preserving implementation results Partitions enable the modified portions of the design to be implemented without affecting the rest of the design or the Partitions that are preserved. The preservation of portions of the design helps to improve the implementation runtime by not implementing the entire design. In general, the runtime improvement is proportional to the amount of logic preserved: it is faster to preserve Partitions than to implement the entire design. The creation of a Partition on a logical module instance will direct the Xilinx ISE™ design tools to reuse the previous implementation results when possible.

The preservation technique used by Partitions enables the following advantages:

- Improves synthesis and implementation runtime (and turns per day).
- Enables 100% preservation of a functional block.
  - If a functional block is designated as a Partition, once it is implemented it can be verified once and then preserved 100% for future iterations. An unchanged Partition need not be re-verified.
- Allows locking down of a functional block in the design.
  - For example, if most of the specification of the design is stable, except for one or a few functional blocks, Partitions allow you to lock down unchanged functional blocks and decrease the impact of last minute design changes.
  - For example, any functional block that is not changing, such as previously implemented IP cores, is a good candidate to designate a Partition.
- Your design has major functional blocks that are broken up by natural design hierarchy (e.g., large designs, team-based designs, EDK designs, and DSP designs). In general, results are best if the functional blocks have registered boundaries. This maximizes the benefits of Partition insulation.

- Improves stability of timing critical portions of the design.
  - Your design has timing problems that move around unpredictably in successive implementations and make timing closure difficult. Partitions enable a divide-and-conquer technique to scope, isolate, and fix timing problems. Note that the timing for up-to-date Partitions is equivalent to the previous implementation so timing problems will not reoccur in a Partition that is preserved.
- Reports resource utilization statistics per hierarchical block.
- Isolates the implementation changes to a few (preferably one) Partition, even when the changes are “large”. If the design is highly utilized, this might require changing the preservation levels of unchanged Partitions from routing to placement or synthesis.
- Isolates the implementation of cores that do not need to be optimized with the rest of the design.
- Speeds up implementation when there are small changes to the design that are contained by a Partition.
- Enables a fine level of control over the preservation of each Partition in the design.
  - Note: SmartGuide™ is either on or off, and applies to the entire design. The preservation level of control is on a per-Partition basis. Cannot use SmartGuide and Partitions at the same time.
- Enhances designs that cannot use SmartGuide.

The use of Partitions can improve the ability to meet the timing objectives or physical constraints (AREA_GROUPs, etc.) of the design. Once the timing objectives are met, the timing of the Partition will not change if it is exactly preserved. Partitions can reduce the variability in timing results when attempting to meet timing requirements. This occurs when you change one portion of the design to meet timing and another (unrelated) portion of the design now fails timing.

To use Partitions most effectively:

- Place the registers that drive output pins at the top level of the design or in the same Partition as the output buffers.
- Place the registers that are driven by input pins at the top level of the design or in the same Partition as the input buffers.
- Place ODDRs and IDDRs at the top level of the design or in the same Partition as the pins associated with the ODDRs or IDDRs.
- Place registers at the input and output of the Partition module.
  - This proves maximum flexibility and optimization when packing at the Partition boundary.
- For maximum runtime improvement, the Partitions in a design should contain roughly the same amount of synchronous elements and logic.
  - Runtime improvements are negligible for Partitions with a small amount of logic that is not timing critical.
- Place Partitions on blocks that have difficulty meeting timing so that once timing is met, they will be preserved.
- To set a Partition on an IP core or EDK design, the ISE 9.1 release requires that it is instantiated by an HDL wrapper. The Partition is then set on the HDL wrapper of the IP core or EDK design.
- If the implementation has difficulty fully routing the design, change the “preserve” trait on the top level Partition from “routing” to “placement.” All child Partitions “inherit” the preservation level of the top level Partition.
- Synplify Pro 8.8.1 supports Partitions with the Compile Point feature.

- A lower level netlist must be instantiated by a top level HDL wrapper when performing bottom-up synthesis methods with Partitions. The Partition is then set on the wrapper of the netlist.

A Partition can be placed on any level of hierarchy for these types:

- HDL (Verilog or VHDL) source file
- Schematic source
- EDIF source
  - Created on the HDL wrapper
  - Created with Compile Points in Synplify Pro

Partitions can be nested throughout the hierarchy. A module with multiple instances throughout the hierarchy can have multiple Partitions or one Partition for each module instance. The top module of the design is automatically designated as a Partition when a lower level instance has a Partition defined. There is no restriction on the amount of logic in the top module. A lower level instance of the design is defined by the module instance name in Verilog and the entity architecture or component instance name in VHDL.

Partitions can be created or deleted at anytime. However, creating or deleting a Partition and the parent Partition will require the synthesis and implementation tools to re-implement that portion of the design. As a result, it is most efficient to review the instances of the design prior to the first implementation cycle to identify which ones should be designated a Partition. It is recommended to establish Partitions initially rather than using an incremental process of adding Partitions later in the design cycle.

Because of XST limitations, do not use Partitions with the following synthesis constructs when synthesizing with XST:

- Generate statements
- Included statements

The following constructs and optimizations should be avoided when using Partitions:

- TBUFs crossing Partition boundaries
- Asynchronous timing critical networks that cross Partition hierarchy boundaries
- Global optimization settings such as “map -global_opt”
- Designs with high SLICE utilization, plus adding Partitions, results in more SLICEs utilized.
  - In some designs, Partitions will decrease the SLICE utilization and in other designs the SLICE count will increase.
- The ChipScope™ CORE Generator™ is fully supported by Partitions. However, the ChipScope Core Inserter is not supported by Partitions.
- Partitions are not supported by Project Navigator when using integrated 3rd party synthesis tools. For example, if Synplify Pro is used for synthesis and implement the design with Partitions in the Synplify Pro environment, not Project Navigator, Partitions are not supported. An HDL design with Partitions can be implemented with Project Navigator and the XST synthesis tool, or implemented with Project Navigator in an EDIF flow and independent Synplify Pro synthesis tool.
- IDDR, ODDR, IFD, or OFD registers on different module levels than the pins
- Bi-Directional Ports in a lower level module
- Bottom up synthesis flow
- Dangling or Unconnected ports of a component
- Cores or hierarchical modules are in the same source file

- Constants that need to propagate across hierarchical blocks or across Partition boundaries.

The following Architectures support Partitions:

- Spartan™-3
- Spartan-3E
- Spartan-3L
- Spartan-3A
- Spartan-3A DSP
- Virtex™-II
- Virtex-II Pro
- Virtex-II Pro X
- Virtex-4
- Virtex-5

Choosing Between Partitions and SmartGuide

SmartGuide is a preservation technique where the previous implementation is compared to the current implementation and unchanged portions of the design are preserved as much as possible. A preserved portion of the design will be re-implemented if it is necessary to meet the timing goals and generate a successful implementation.

ISE 9.1i supports either Partitions or SmartGuide for a particular design. They are not supported together in the same design. The choice of Partitions or SmartGuide is highly specific to the design situation, so there is no specific rule that can always be followed. SmartGuide applies to the entire design and operates on the lowest level of physical elements in the FPGA. The following types of design scenarios work well with Partitions:

- Cannot use SmartGuide (e.g., you do not want to use map -timing)
- Design changes result in large changes in synthesis results
- Improve implementation runtime of synthesis and all the Xilinx implementation tools, such as NGDBuild, MAP, and PAR. SmartGuide will improve implementation runtimes for MAP and PAR.
- Reduce verification: Partitions that are preserved to not require re-verification because they are exactly the same implementation.
- Finer level of control over preservation: SmartGuide is either on or off and applies to the whole design. Partitions support per-Partition preservation control of synthesis, placement, or routing.

The following changes to the design will work well with SmartGuide:

- Small logical changes (less than 10%) in one or two modules that do not affect the rest of the design.
- Moving a Pad location.
- Change an attribute on a component.
- Change a timing constraint.
  - In ISE 9.x, changing a timing constraint will force all Partitions to be out-of-date.

How to Use Partitions

Prior to synthesis, Partitions need to be placed on the levels of hierarchy where design reuse is required. The synthesis tools will not optimize across the Partition interface. If an asynchronous timing critical path crosses Partition boundaries, logic optimizations will not occur across the Partition boundary. To mitigate this issue, add a register to the asynchronous signal at the Partition boundary.

Partitions do not require floorplanning, such as ranges for implementation reuse. Partitions support any amount of logic at the top level of the design. Project Navigator displays the design hierarchy in the Sources tab. To set a Partition on a level of hierarchy, right-click the instance and select “New Partition”, as shown in Figure 1.

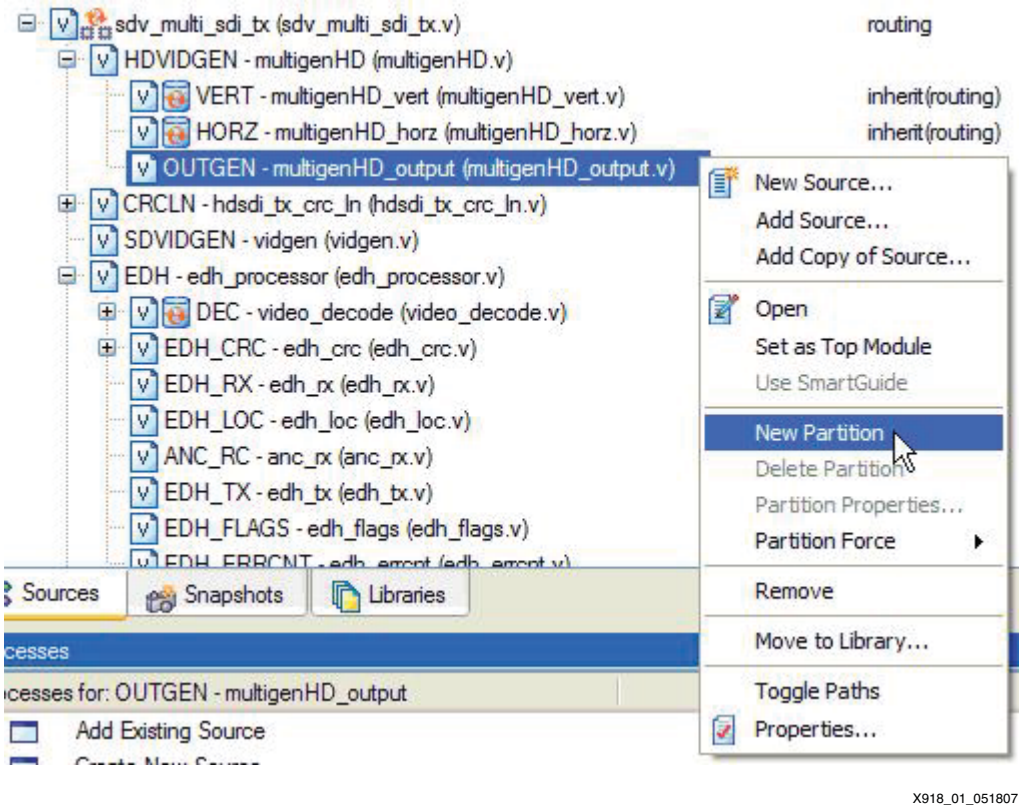

Figure 1: **Setting a Partition on a Level of Hierarchy**

There are several icons for the status of the Partitions. The icons provide information on the state of the Partition and necessary action required for each Partition. There are two sets of icons for Partitions:

- A set for the top-level Partition
- A set for the instance-level Partition

The icons for the state of the top-level Partition:


No Partitions in the design from previous implementation cycle.


Partition data will be preserved during the next implementation cycle.


Partition will be implemented during the next implementation cycle.


The entire Partition or portion of the Partition may be preserved during the next implementation cycle, but is dependant on other Partitions that will be re-implemented and the preservation level.

The icons for the state of an instance-level Partition:


Partition data will be preserved during the next implementation cycle.


Partition will be implemented during the next implementation cycle.


The entire Partition or a portion of the Partition may be preserved during the next implementation cycle, but is dependant on other Partitions that will be re-implemented and the preservation level.

In ISE 9.1i, Partitions do not support the following implementation options:

- Multi-Pass Place and Route (MPPR) options (-s, -n, etc.)
- Logic Optimization (-logic_opt)
- Global Optimization (-global_opt)
- SmartGuide (-smartguide)

When the resource utilization is high, subdivide the parent Partition into smaller child Partitions. To improve tool runtime, subdivide large Partitions into several smaller Partitions. If subdividing large Partitions is not possible, changing the preservation levels of the remaining Partitions from “routing” to “placement” may help reduce runtime.

After about 15-20 implementation cycles, Xilinx recommends implementing with preservation set to “placement” or “synthesis”, to optimize the entire design. This allows placement and routing to be optimized for new or modified logic. This can potentially improve the overall implementation results.

Using Partitions with Synplify 8.8.1

The use of the Compile Points constraint gives Synplify Pro and Synplify Premier tools the ability to create Partitions. The define_compile_point command defines a compile point in a top-level constraint file (SDC). Use one define_compile_point command for each compile point or module to define as a Partition. You can create this command in Synplify Pro by editing the SDC file with the Scope tool. The SDC file will be an additional tab with several sub-tabs. One of the sub-tabs is “Compile Points” and the selection of which module to define as the compile point. Once a module is selected, the type must be specified as “locked, partition”. This will add the following constraint to the SDC file:

- define_compile_point {v:work.multigenHD_vert} -type {locked, partition} -cpfile {}
- define_compile_point {v:work.multigenHD_horz} -type {locked, partition} -cpfile {}

- Once the compile point is added to the SDC file, Synplify will add the “PARTITION” string and time stamp to the EDIF file for each hierarchical node. When the ISE project is created, the EDIF will be added as the design source. When ISE parses the EDIF file, it will create a Partition for each hierarchical node in the EDIF with the “PARTITION” string.

The EDIF that is created by Synplify Pro will be used to create a project in ISE Project Navigator to ensure the Partitions are recognized and preserved during implementation.

Using Partitions with Tcl

The creation of partitions can also be done through the Xilinx Tcl interface. The Tcl interface is described in the Tcl chapter of the Development System Reference Guide. The Xilinx “partition” Tcl command contains a set of subcommands for working with Partitions. Example scripts below show how to create and manipulate Partitions in a design.

Sample Tcl script to create Partitions:

```
puts "Create new project dev_ccir_top.ise\n"
project new dve_ccir_top.ise

puts "Set the device\n"
project set family Virtex4
project set device xc4vlx15
project set speed -11
project set package sf363

puts "Add the HDL source files\n"
xfile add Hdl/dve_ccir_aps.v
xfile add Hdl/dve_ccir_dds.v
xfile add Hdl/dve_ccir_dph.v
xfile add Hdl/dve_ccir_fir.v
xfile add Hdl/dve_ccir_lut.v
xfile add Hdl/dve_ccir_mlt8x9.v
xfile add Hdl/dve_ccir_top.v
xfile add dve_ccir_top.ucf
xfile add Hdl/dve_ccir_vtg.v

puts "Define the partitions\n"
partition new /dve_ccir_top/DATAPATH
partition new /dve_ccir_top/DATAPATH/CHROMA_FIR
partition new /dve_ccir_top/GENERATOR

puts "set the implementation tool options\n"
# set batch application options :
#XST options
# 1. set synthesis optimization goal to speed
project set "Optimization Goal" Speed
#translate options
# 2. ignore any LOCs in NGDBuild
project set "Use LOC Constraints" FALSE
#map options
# 3. perform timing-driven packing
project set "Perform Timing-Driven Packing and Placement" TRUE
#par options
# 4. use the highest par effort level
project set "Place & Route Effort Level (Overall)" High
# 5. set the par extra effort level
project set "Extra Effort (Highest (PAR level only)"  "Continue on Impossible"
# 6. set verbose report type
project set "Report Type" Verbose
# 7. pass "-instyle xflow" to the par command-line
project set "Other Place & Route Command Line Options" "-intsyle xflow"
# 8. generate a verbose report from trce
project set "Report Type" "Verbose Report"
# 9. create the IEEE 1532 file during bitgen
project set "Create IEEE 1532 Configuration File" TRUE

puts "Run implementation tools\n"
if {[catch {process run "Implement Design"}]}{
   puts "Caught an Error executing process or time commands"
   exit 1
}
process "Generate Post-Place & Route"

puts "Close project\n"
project close
```

Sample Tcl script to get and modify the properties of the Partitions:

```
#open project file
puts "open project file"
project open dve_ccir_top.ise

puts "Getting Partition Properties\n"
# Returns the preservation level for this Partition
partition get /dve_ccir_top/DATAPATH/CHROMA_FIR preserve
# Returns status of Implementation results of this Partition (true or false)
partition get /dve_ccir_top/DATAPATH/CHROMA_FIR up_to_date_implementation

puts "Modifying the Partition Properties\n"
# Forces the Partition to rerun Synthesis
partition rerun /dve_ccir_top/DATAPATH/CHROMA_FIR synthesis
# Sets the preservation level of this Partition to Placement
partitions set /dve_ccir_top/DATAPATH/CHROMA_FIR preserve placement

puts "Run implementation tools\n"
if {[catch {process run "Implement Design"}]}{
   puts "Caught an Error executing process or time commands"
   exit 1
}
process "Generate Post-Place & Route"
# close the project
puts "close the project"
project close
```

Preservation Level of a Partition

The preservation level gives the implementation tools directives on how to operate on each copied Partition. The preservation level can be set for each Partition in the design. By default, the “Routing” preservation level is set on the top-level module, and the lower-level Partitions have the preservation level set to “Inherit”. The default preservation will preserve all the implementation data of the design from synthesis through routing. The preservation level for each Partition can be set by right-clicking the Partition in the source view and select “Partition Properties”, as shown in Figure 2.

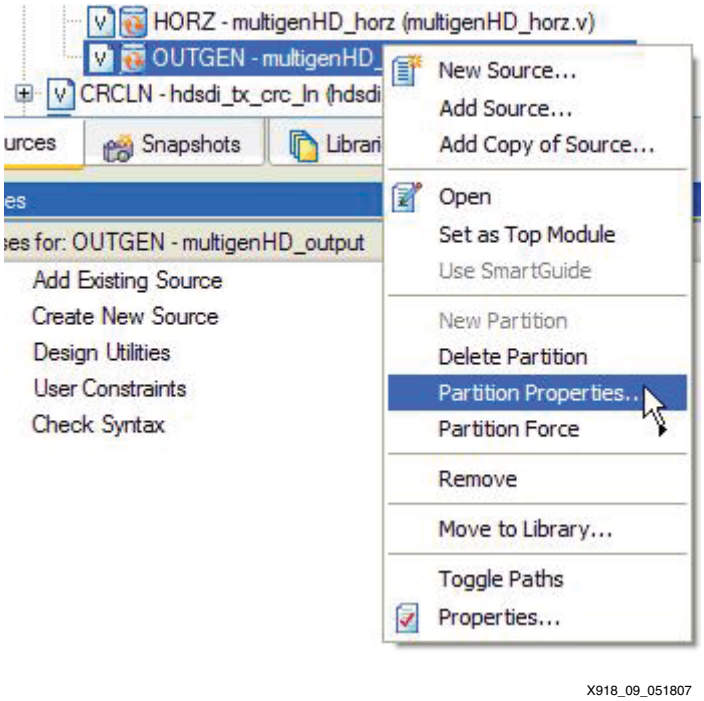

Figure 2: **Setting the Preservation Level for Each Partition**

The preservation level can be changed from the default level of Routing for each Partition. The amount of design data preserved will decrease as the preservation level is changed from “routing” to “placement” to “synthesis”. If the design is not meeting the timing constraint objectives or cannot route or fit the design, relax or lower the preservation level for the Partition. The preservation level can be set to one of the following:

- Routing
- Placement
- Synthesis
- Inherit

The **Routing** preservation level preserves the data of the Partition through routing. This level of preservation gives the implementation tools the least amount of flexibility to meet the timing or implementation objectives, but provides the highest degree of preservation. The following implementation data is preserved when “preserve” is set to “routing”:

- Synthesized netlist and information
- Placement information
- Routing information

The **Placement** preservation level preserves the data of the Partition through placement. If a re-implemented Partition requires the routing resources from another Partition, then, the routing resources of a preserved Partition with “preserve” set to “placement” may be modified by the re-implemented Partition. Partitions always attempt to preserve routing. So, even with “preserve” set to “placement”, the routing will be preserved unless it must be modified to allow another Partition to be implemented successfully. The Partition summary in the PAR report file (design.par) informs the user if the routing for the Partition has been modified. The following implementation data is always preserved when “preserve” is set to “placement”:

- Synthesized netlist and information
- Placement information

- Some Routing information, possibly all Routing information

The **Synthesis** preservation level only preserves the synthesis netlist of the design. If a re-implemented Partition requires the placement and routing resources from another Partition, then, the resources of a preserved Partition with “preserve” set to “synthesis” may be modified during implementation. If the resources of the Partition with “preserve” set to “synthesis” were not modified, then the Partition will be exactly preserved. The following implementation data is preserved when “preserve” is set to “synthesis”:

- Synthesized netlist and information
- Some Placement information, possibly all Placement information
- Some Routing information, possibly all Routing information.

The **Inherit** preservation level sets the “preserve” attribute to the same level as the parent Partition. The default setting for all child Partitions is Inherit. The default setting for the top-level Partition is Routing. Therefore, the initial condition for all designs is that all child Partitions have “preserve” set to “inherit” with the value resolving to “routing” from the top level Partition.

Partition Preservation

The Partitions that are not modified are reused on subsequent implementation cycles. Modifying the HDL source code or changing physical constraints will cause a Partition to be re-implemented. Partitions automatically detect when an input source file has changed, forcing that Partition to be re-implemented. Some types of source changes will force all Partitions to be re-implemented. The types of changes include:

- Changes in Timing Constraints
- Changes in Process Properties (MAP/PAR optimizations such as effort level, etc.)
- Changes in Project Properties (device, speed grades, etc.)
- Removing implementation files, such as NGD or NCDs from project directory

A Partition can be manually forced to be re-implemented with the “force” command. This is independent of the HDL source files being modified or the preservation level. Forcing a Partition to be re-implemented only affects that Partition; it will not force the entire design to be re-implemented. To force the change in status for the Partition, right-click the Partition and select “Partition Force -> Force Synthesis Out-of-date” or “Partition Force -> Force Implementation Design Out-of-date”, as shown in Figure 3.

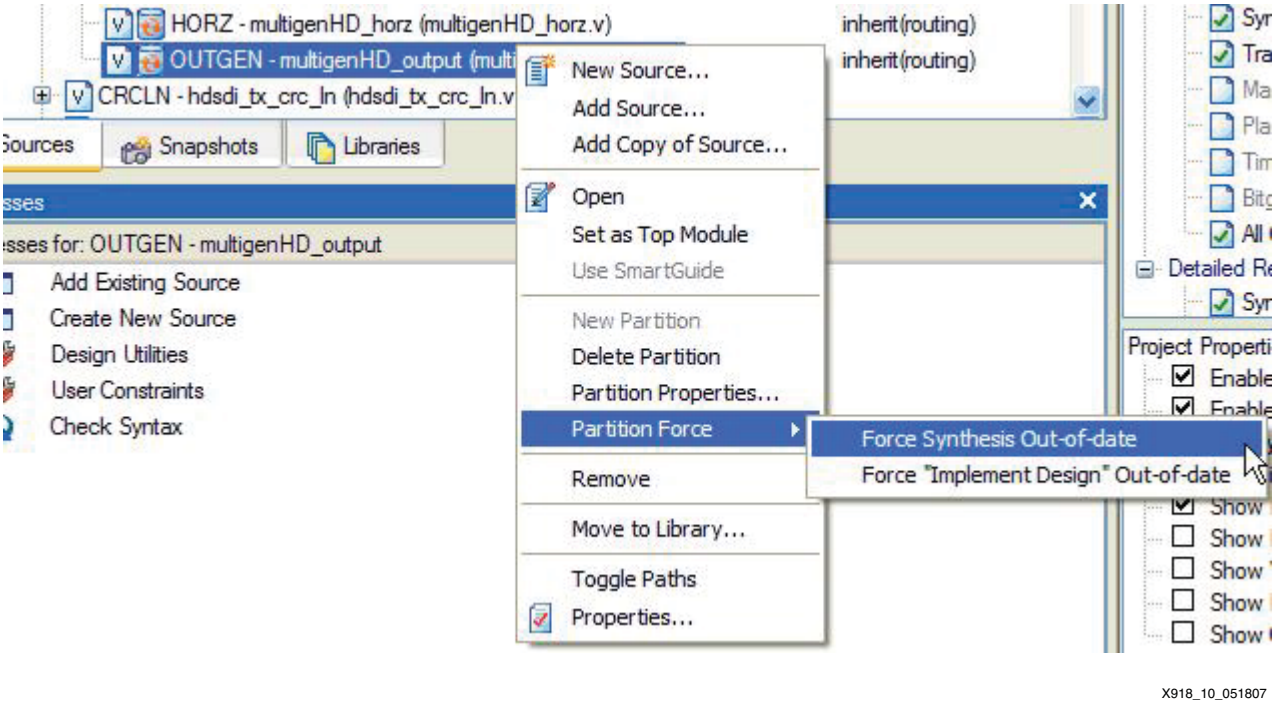

Figure 3: **Forcing the Change in Status of the Partition**

In the Partition Force menu, the following options are available:

- Force Synthesis Out-of-date - Causes the Partition to be re-synthesized and re-implemented
- Force Implement Design Out-of-date - Causes the Partition to be re-implemented, the synthesized netlist will be preserved

Once the Partition Force command is used, the Partition will be re-implemented regardless of any previous implementation data. After the next implementation cycle, the Partition will be preserved based upon the Partition preservation level.

Floorplanning a Partition

Partition does not require an Area Group constraint or placement constraints such as range. The option to Floorplan the logic associated with Partitions is available to solve timing closure issues. Xilinx does not recommend floorplanning the logic associated with Partitions unless the Partitions have difficultly meeting the timing objectives. A complete set of well-constructed timing constraints are the preferred method for timing closure. Floorplanning is not a replacement for timing constraints. Floorplanned Partitions can improve the performance of the design by constraining the logic associated with Partition to a specific region of the device. The area_group range constraint is not set on the Partition; it is set on the logic or instance associated with the Partition.

To explore the placement of the design elements, see the valid locations, and to create the area_group constraints, use the Floorplan Editor and Floorplanner. When the Partition logic is floorplanned, the associated instance is given a range of SLICE constraints in the User Constraints File (UCF), similar to “SLICE_X46Y73:SLICE_X20Y100”. The range of area_group constraints defines where the logic in the Partition will be placed in the device. All of the logic elements in a hierarchical node will be included in the location constraint.

In Floorplan Editor, the instance or logic associated with the Partition can be placed into a region of the device by a drag-and-drop process. The size of the rectangular region is automatically estimated to fit the amount of logic in the instance. Once the instance is placed in

a rectangular region, the size and shape can be modified as needed. The Design Rules Check (DRC) of Floorplan Editor will ensure that the placed instance's rectangular region is legal for the implementation tools. The area_group range constraint informs the implementation tools that the corresponding logic must be contained within the boundaries of that rectangular region.

Reporting of Partitions

After implementing the design with Partitions, use the Design Summary and the various implementation reports to review the resources used by each Partition. If the Partition is too large or has difficult timing constraints, the implementation of that Partition may take longer than expected. These Partitions should be subdivided into smaller Partitions if possible.

Synplify Pro, XST, NGDBuild, MAP, and PAR generated reports contain information about the Partitions such as preservation status and resources used during implementation. The Guide Report File (GRF) provides details about the reimplemented components, new components, and nets in the design that are associated with SmartGuide. The per-Partition resources reported in the XST synthesis report and MAP report are reflected in the Sources tab in Project Navigator, next to the corresponding Partitions (as shown in Figure 4), and Design Summary of Project Navigator (as shown in Figure 5).

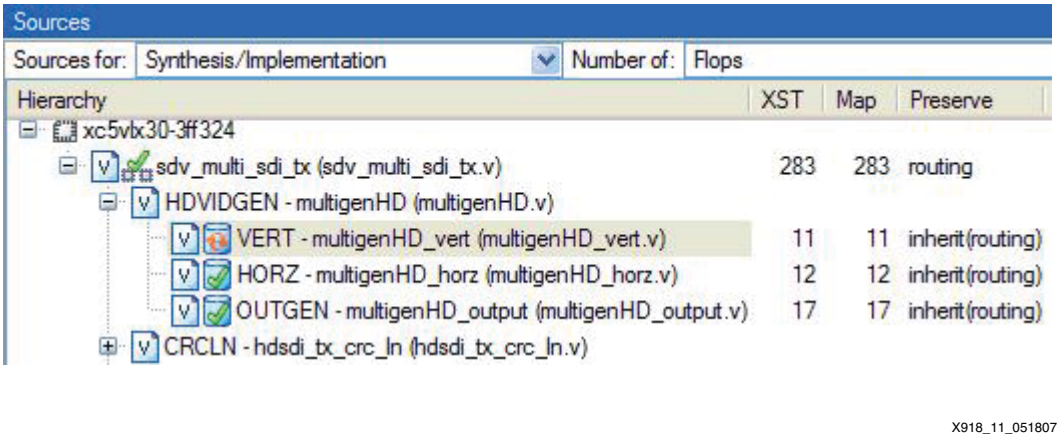

Figure 4: **Resource Estimates from XST and MAP per Partition**

LAB1 Partition Summary

Partition Name	Synthesis Status	Placement Status	Routing Status
/sdv_multi_sdi_tx	Preserved	Preserved	Preserved
/sdv_multi_sdi_tx/EDH/DEC	Preserved	Preserved	Preserved
/sdv_multi_sdi_tx/HDVIDGEN/HORZ	Preserved	Preserved	Preserved
/sdv_multi_sdi_tx/HDVIDGEN/OUTGEN	Preserved	Preserved	Preserved
/sdv_multi_sdi_tx/HDVIDGEN/VERT	Implemented	Implemented	Implemented

X918_12_051807

Figure 5: **Status Summary of each Partition**

The Synplify Pro synthesis report contains the following sub-header for the Partitions:

- Summary of Compile Points - List of preserved Partitions

```
Name                 Status        Reason
-------------------------------------------------
multigenHD_vert      Remapped      Design changed
multigenHD_horz      Remapped      Design changed
sdv_multi_sdi_tx     Unchanged     -
```

The XST synthesis report contains the following sub-headers for the Partitions under Partition Implementation Status:

- Preserved Partitions - List of preserved Partitions
  - Example: `Partition "/sdv_multi_sdi_tx/EDH/DEC"`
- Implemented Partitions - List of implemented Partitions and reason why they were implemented
  - Example: `Partition "/sdv_multi_sdi_tx/HDVIDGEN/VERT":`
    - `HDL source file(s) were modified.`
- Partition NGC Files - List of all Partitions and associated NGC files
  - Example: `Partition "/sdv_multi_sdi_tx/HDVIDGEN/HORZ":`
    - `NGC File: HDVIDGEN_HORZ#multigenHD_horz.ngc`

The NGDBuild report contains the following sub-headers for the Partitions in the design under Partition Implementation Status:

- Preserved Partitions - List of preserved Partitions
  - Example: `Partition "/sdv_multi_sdi_tx/HDVIDGEN/HORZ"`
    - `Implemented Partitions - List of implemented Partitions and reason why they were implemented`
  - Example: `Partition "/sdv_multi_sdi_tx/HDVIDGEN/VERT":`
    - `Synthesis modified the Partition.`

The MAP report contains the following sub-headers for the Partitions in the design under Section 9 - Area Group & Partition Summary -> Partition Implementation Status:

- Preserved Partitions - List of preserved Partitions
  - Example: `Partition "/sdv_multi_sdi_tx/HDVIDGEN/HORZ"`
- Implemented Partitions - List of implemented Partitions and reason why they were implemented
  - Example: `Partition "/sdv_multi_sdi_tx/HDVIDGEN/VERT":`
    - `An upstream application reimplementation of the Partition.`
- Partition Resource Summary - List of all Partitions and resources used
  - Example: `Partition "/sdv_multi_sdi_tx/EDH/DEC":`
    - `Slice Logic Utilization:`
    - `Number of Slice Registers:221`
    - `Number of Slice LUTs:263`
    - `Number used as logic:263`

- Slice Logic Distribution:
  - Number of occupied Slices:136
  - Number of LUT Flip Flop pairs used:357
  - Number with an unused Flip Flop:136 out of 357 38%
  - Number with an unused LUT:93 out of 357 26%
  - Number of fully used LUT-FF pairs: 128 out of 357 35%

- Area Group Information - List of SLICE location and usage patterns
  - Example: Area Group "AG_HDVIDGEN/HORZ"
    - No COMPRESSION specified for Area Group "AG_HDVIDGEN/HORZ"
  - RANGE: SLICE_X7Y49:SLICE_X2Y46
  - Slice Logic Utilization:
    - Number of Slice Registers:12 out of _12%
    - Number of Slice LUTs:33 out of _34%
    - Number used as logic:33
  - Slice Logic Distribution:
    - Number of occupied Slices:11 out of 24 45%
    - Number of LUT Flip Flop pairs used:33
    - Number with an unused Flip Flop:21 out of 33 63%
    - Number with an unused LUT:0 out of 33 0%
    - Number of fully used LUT-FF pairs:12 out of 33 36%
      - Note: Percentages are based upon an AREA GROUP for this Partition.
  - Number of Block RAM/FIFOs:1

The PAR report contains the following sub-headers for the Partitions in the design under Partition Implementation Status:

- Preserved Partitions - List of preserved Partitions
  - Example: Partition "/sdv_multi_sdi_tx/HDVIDGEN/HORZ"
- Implemented Partitions - List of implemented Partitions and reason why they were implemented
  - Example: Partition "/sdv_multi_sdi_tx/HDVIDGEN/VERT":
    - An upstream application reimplementation of the Partition.

Removing Partitions

If the use of Partitions is not improving the runtime or the preservation of the design, the removal of one or several Partitions might be necessary. The logic associated with the removed Partition will now be a part of its parent Partition, which could be the top level Partition. The Partition commands and properties will no longer apply to the logic that was previously in the Partition. In Project Navigator, to remove a Partition from the Partition hierarchy of the design, right-click the Partition and select “Delete Partition” (as shown in Figure 6).

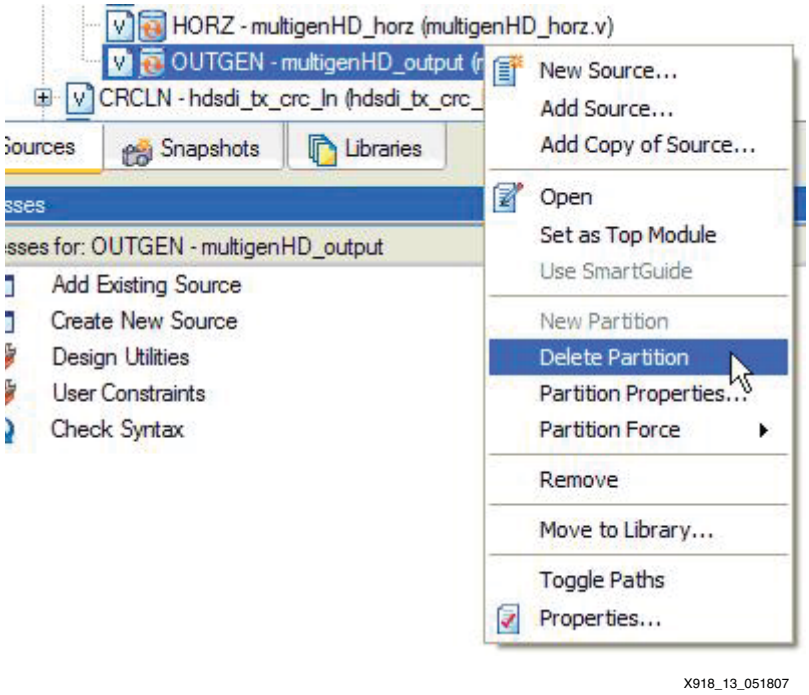

Figure 6: **Removing a Partition from the Partition Hierarchy of the Design**

Partitions and Old Incremental Design Techniques

ISE 8.1i, and previous versions, did not support the Partition method of design preservation. Incremental guide and incremental synthesis were used by these versions of ISE for design preservation. The following incremental design functions are not compatible with a design using Partitions:

- XST Incremental Synthesis constraint in the XCF file
- MAP guide mode “-gm incremental”
- PAR guide mode “-gm incremental”
- Area_group range (this is not required for Partitions, but may be used for floorplanning)

It is not recommended that the incremental guide and incremental synthesis techniques are used in ISE 8.2i and later versions. Partitions should be used for design preservation. If an older design using incremental design techniques is migrated to ISE 8.2i or later, these constraints should be removed and replaced with Partitions.

Conclusion

It is recommended to place Partitions on major or timing critical modules. This allows the synthesis and implementation tools to use the previous version of the design for reuse and synthesize, and to implement the changed Partition. This will help to reduce runtime of the implementation tools. The creation of Partitions can be done through Project Navigator or the Tcl interface. The Tcl interface is described in the Development System Reference Guide under Tcl.

It is also recommended to use “map -timing” to help improve packing and density of the design. If the design has high utilization, then it is recommended using a “pack factor” of 80 to 90% to create space to avoid placement and routing issues.

In most designs, it is faster to preserve the design data than to reimplement the entire design. The time to implement is decreased by using Partitions or SmartGuide. The average runtime improvement seen in an extensive design suite used to test Partitions and SmartGuide is about 2.5 times faster than the initial implementation. In the best-case scenarios, the runtime improvement was up to 6 times better for design interactions using Partitions than for equivalent design changes without the use of Partitions.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
6/7/07	1.0	Initial Release