Tutorial on Exporting and Importing Partitions with Repeated Modules ($paramDocNum)

Achronix Data Acceleration

V1.1 – February 28, 2025 Application Note

Introduction

ACE has the capability to save and restore the placement and routing data for a portion of a design after an ACE compile. Each portion of the design is referred to as a *partition*, and this capability of saving and restoring a placed-and-routed portion of a design is referred to as the exporting and importing of partitions.

An imported partition will be placed by default at the same location as it was exported from. However, ACE also has the capability to *move* an imported partition to a different location after it is imported. The only limitation is that the old and new locations must have the same alignment to a grid defined by repeating elements of the fabric called *fabric clusters*. This functionality is analogous to *copying* and *pasting* a module at different locations across the FPGA fabric. This tutorial demonstrates this copy/paste capability on a design with many repeated copies of a single module.

Partition Definitions

- **Partition** – A single instance of a module in an HDL design which has been marked as a compile point in Synplify Pro or a partition in ACE.
  - Synthesis and re-synthesis optimizations in Synplify Pro and ACE will not modify or optimize across partition boundaries.
  - ACE will keep track of the module boundary during flattening so that the netlist, placement-and routing data for the partition can be exported and imported.
  - The word *partition*, depending on context, can refer to the database image of a partition in ACE, or to the exported database file for a partition on disk.
  - Any module in a user design can be defined as a partition *except* the top-level module. This restriction is because there are no instances of the top-level module.
  - A module instance is defined as a partition using the `define_compile_point` constraint in Synplify Pro, or the `set_partition_info` constraint in ACE. See the section "Using Incremental Compilation (Partitions)" in the ACE User Guide (UG070) .
- **Partition Export Flow** – a synthesis, placement, and routing flow with the goal of generating and exporting one or more placed-and-routed partitions.
- **Partition Import Flow** – a synthesis, placement, and routing flow with the goal of importing one or more previously-exported partitions.

Partition Filename Conventions

- `.fdc` file – FPGA design constraint file in Synplify Pro which, among many other types of constraints, can contain define_compile_point commands to mark module instances as compile points. See the *FPGA Synthesis User Guide* from Synopsys for more information.

- `.prt` file – Place-and-route constraint file in ACE which creates partitions using the `set_partion_info` Tcl command. In the export flow these are created by Synplify Pro from the compile point constraints. In the import flow they must be written by the user.
- `.epdb` file – Exported partition database file containing the netlist, placement and routing data for an exported partition.
- `bb.v` file – Black-box Verilog file created by ACE, along with the `.epdb` files, during partition export. They contain only the module interface declaration and resource count pragmas for a partition, and they can be used in Synplify Pro and ACE during the import flow to represent the partition modules. Because the modules are empty, the black-box files save runtime during RTL synthesis, but their use is optional.

Export/Import Use Models

There are at least three use models for exporting and importing partitions. The flow used depends on the type of design:

- A simple flow in which all of the partitions are exported from a single design in ACE, and then imported into another run of the same design.
- A more advanced use model in which partitions are exported from multiple designs, then imported into a unique top-level design.
- The most advanced use model in which a single partition can be exported and later imported multiple times into a design with repeated module instances similar to copying and pasting.

This tutorial explores the third use model and applies it to a machine learning (ML) style design. Partitions are especially useful for machine learning designs, such as convolutional neural networks (CNNs), because ML operators can be highly parallelized. ML designs often consist of many copies of identical functional blocks that are repeated in a regular pattern across the FPGA. Rather than attempting to place and route the entire design in a flat layout, a single block can be placed and routed as a partition in a separate ACE session, and then imported multiple times for each repeated module in the new top-level design.

Using partitions in this way has two advantages:

- Top-level timing can be closed more easily because each individual partition is optimized separately in its own ACE export session. It is not necessary for all partitions to close timing concurrently in a single flattened hierarchy. Instead, timing on an individual modules can be closed before exporting it as a partition. All partitions that have closed timing can then be imported into a top-level ACE import session.
- Compile time can improve by placing and routing a single partition and later importing the partition for each repeated block. Since the partitions are already placed and routed, the compile time for a project that uses imported partitions can be less than the same design that uses a flattened hierarchy. Placement and routing information for a partition is reused by the tools when mapping it to the equivalent hardware resources in the target design.

Tutorial Overview

This tutorial, consisting of three phases, uses a bottom-up design flow to synthesize, place, and route a representative machine learning design consisting of 40 repeated 2D convolution (conv2d) modules:

1. Compile a single conv2d module and export the netlist, placement and routing data into an `epdb` database file and a `bb.v` black-box Verilog file.
2. Import the conv2d partition multiple times into the top-level design named `partition_import_export_ref_design_top`. For each repeated module in the top-level design, use placement constraints to place each repeated partition in a new non-overlapping location.
3. Run a top-level synthesis, placement, and routing on the top-level design.

The `mlp_conv2d` module used in this tutorial is a derivative of the MLP 2D Convolution reference design. In the design used for this tutorial, however, the two output enable signals, `o_conv_done_oe` and `o_error_oe`, are not ported to the fabric design because they are not connected to a NAP and would, therefore, require top-level routing. Another notable difference is that the highest level module in the tutorial design, `mlp_conv2d_top`, is simply a wrapper around the `mlp_conv2d` module. The reference design file, `mlp_conv2d_top`, and the tutorial design file, `mlp_conv2d` are nearly identical. The RTL module hierarchy of the lower level module in the export design, as well as the higher-level module in the import design, are shown in the figure below.

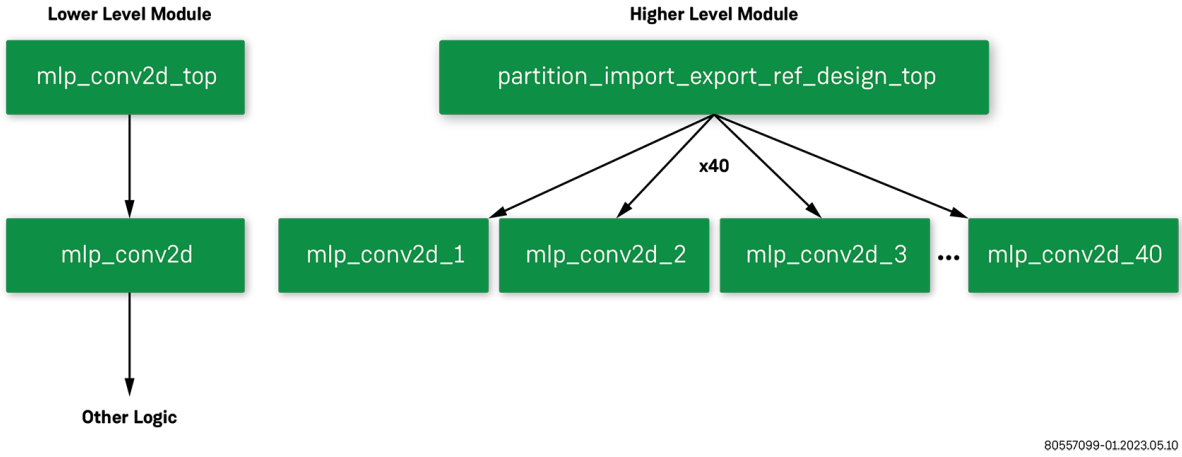

Figure 1 • mlp_conv2d and partition_import_export_ref_design_top Module Hierarchy Diagram

> **Note**
>
> In this example, all of the partitions are independent of each other. All top-level IO connections are made through network access points (NAPs) on the 2D network on chip (NoC). There are no connections between partitions, and no top-level routing other than clocks and resets. Therefore, all partitions function as standalone designs. This important and powerful property allows the partitions to be placed and routed independently without having to worry about top-level route planning or pin assignment.

The export flow demonstrated in this tutorial can also be used to generate a library of placed-and-routed soft IP blocks. Additionally, the same flow can be used to implement a bottom-up design, where multiple teams design their blocks in parallel, and then combine into one top-level design at the end. Those flows are covered in other tutorials.

Partition Flow Limitations

Version Control

Exported partitions in Synplify Pro are represented as a black-box module, not as compile point constraints, which rely on timestamps for version control. For this reason, version control management of partition databases falls to the designer.

Incremental Compilation

ACE does not support mixing incremental compilation and partition export/import in the same run. If incremental compilation is enabled, partitions cannot be exported or imported, and vice versa.

Tutorial Setup: Obtain the Files

All of the tutorial designs are available for download from the archive at our download site. These files are available in the sub-folder "Tutorial Design" in the archive **Demonstration and Reference Designs**. To gain access to this archive, please file a document request ticket with the subject, "Speedster7t and Speedcore Reference Designs". More info can be found in the support article, **How Do I Gain Access to Confidential Documents?**

Once in the archive, search for the file,
`Speedster7t_partitions_export_import_tutorial_AN026_vx.y` package. After downloading the design package, ensure that the files are unzipped into a suitable working directory.

The top-level directory contains the following sub-directories for the two parts of this tutorial:

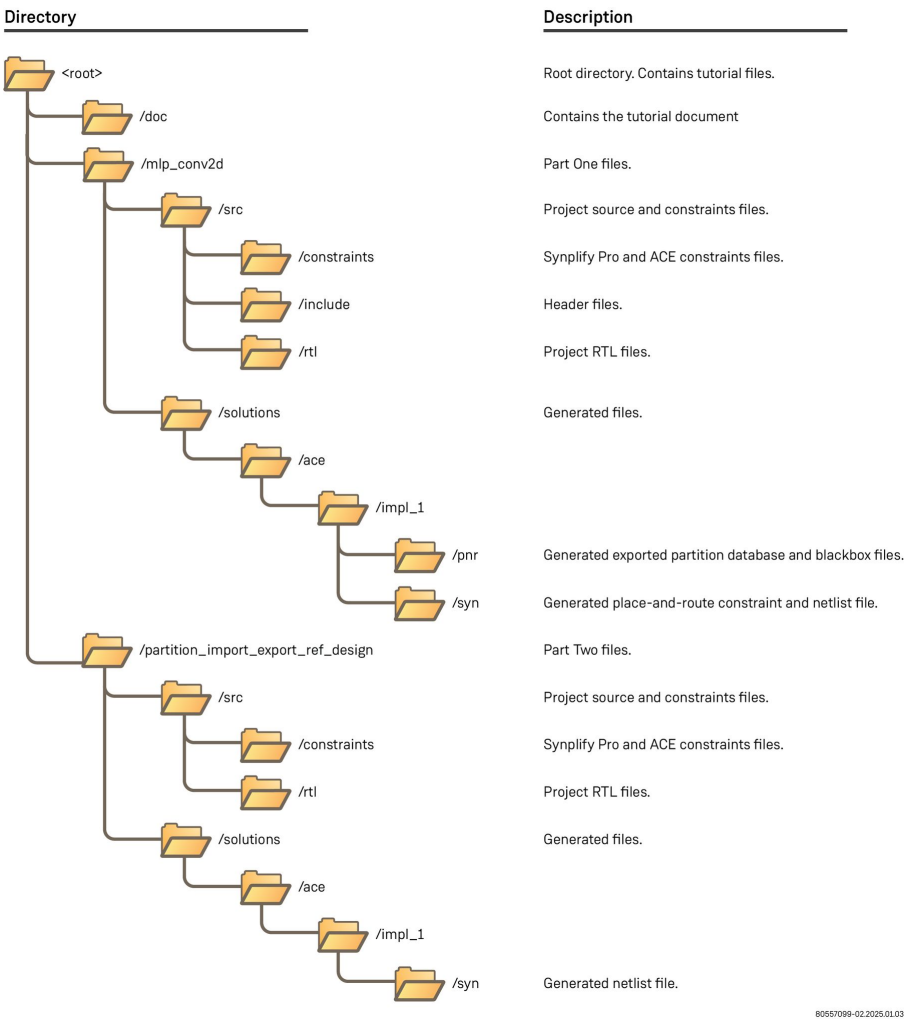

Figure 2 • Tutorial Directory Structure

The `solutions` directories contain files that are either user-generated or generated by ACE or Synplify Pro to be used in future steps of the tutorial. These solutions files can be used in situations where a file, incompatible with future steps of the tutorial, is inadvertently user-generated.

High-Level Design Flow

A bottom-up design flow must be used for this tutorial, which involves synthesizing, placing, and routing a copy of the mlp_conv2d sub-module, exporting sub-module as a partition, importing that partition forty times into a top-level design, and then placing and routing the top-level design.

In this tutorial, the ACE-driven integrated synthesis flow will be used. Refer to the *ACE-Driven Integrated Synthesis* section in the ***Synthesis User Guide*** (UG018) for environment setup and more information on the ACE-driven integrated synthesis flow.

Please follow this bottom-up design flow carefully:

1. Part One: Export a Single Partition for the mlp_conv2d_top Sub Module
   a. Set Up ACE Project for the mlp_conv2d_top Module Synthesis
   b. Synthesize the mlp_conv2d_top Design
   c. Set Up ACE Project for mlp_conv2d_top that Exports the mlp_conv2d_top Partition
   d. Place and Route the mlp_conv2d_top Module in ACE for the mlp_conv2d Block
2. Part Two: Import the mlp_conv2d Partition and Replicate 40 Times into a Top-Level Module
   a. Set Up ACE Project for partition_import_export_ref_design_top Synthesis
   b. Synthesize the partition_import_export_ref_design_top Design
   c. Set Up ACE Project for the partition_import_export_ref_design_top Place and Route
   d. Place and Route the partition_import_export_ref_design_top Design

Part One: Export a Single Partition for the mlp_conv2d_top Sub Module

The `mlp_conv2d` module is a 2D convolution unit that is able to convolve between a 227 × 227 × 3 input image and an 11 × 11 × 3 kernel image. These input sizes can be changed in the RTL via parameter values which can be found in the `mlp_conv2d/src/rtl/mlp_conv2d.sv` file:

```
// Tensor flow parameters for 2D conv
parameter   BATCH               = 4,    // Batch size during training
parameter   IN_HEIGHT           = 227,  // Input image height
parameter   IN_WIDTH            = 227,  // Input image width
parameter   IN_CHANNELS         = 3,    // Number of input channels
parameter   FILTER_HEIGHT       = 11,   // Filter image height
parameter   FILTER_WIDTH        = 11,   // Filter image width
parameter   OUT_CHANNELS        = 1,    // Number of output channels
parameter   INF_DATA_WIDTH      = 144   // Input FIFO data width
```

Each `mlp_conv2d` module uses the 2D network on chip (2D NoC) to connect to memory from which the module reads input and kernel data. Specifically, the `mlp_conv2d` module connects to a 2D NoC access point (NAP) in the design and sends read commands via the 2D NoC to memory.

Tutorial part one covers the following:

- Synthesize, place, and route one copy of the design named `mlp_conv2d_top`.
- Add a placement region constraint as a floor planning directive to specify where the design should be placed in the FPGA fabric.
- Export the netlist, placement, and routing of the block as a partition into `.epdb` and `bb.v` files

Set Up ACE Project for the mlp_conv2d_top Design Synthesis

1. Create a new empty directory `mlp_conv2d/src/ace`.
2. Move into directory `mlp_conv2d/src/ace` and start ACE.
3. In the Projects perspective, remove any existing project in the Projects tab by right-clicking the name and selecting **Remove**.
4. Create a new project by selecting **File** → **Create Project**.
5. Click **Browse** to navigate through the file system and ensure that the project is created under the `ace` sub-directory where ACE was launched.
6. Use `mlp_conv2d` for the Project name, and `impl_1` for the Implementation name, and click **Finish**.
7. Select **Options** → **Project Options** → **Target Device** and set to "AC7t1500".
8. Select **Options** → **Project Options** → **Package** and set to "F53".
9. Select **Options** → **Project Options** → **Speed Grade** and set to "C2".
10. Select **Options** → **Project Options** → **Flow Mode** and set to "Evaluation".
11. In the **Options** → **Project Options** menu, select the **Export All Partitions** implementation option so that the box is checked.
12. In the **Options** → **Project Options** menu, deselect the **Auto-Select Top Module** option and set **Top Module Name** to `mlp_conv2d_top`.
13. In the **Options** → **Project Options** → **HDL Include Path** menu, add the `../include` path to include the `mlp_conv2d/src/include` directory. The path used is relative to the ACE project file being used in `mlp_conv2d/src/ace`. Alternatively, the absolute path to the `mlp_conv2d/src/include` directory could be used.

    > **Note**
    >
    > If the "Can't open file speedster7t/speedster7t_user_macros.v" compilation error appears, double-check that the libraries directory is pointing to the correct location in the include path order.

14. Select **File** → **Add Project Source Files** → **RTL Files** to add all files in the `mlp_conv2d/src/rtl` directory.
15. Select **File** → **Add Project Source Files** → **Synthesis Constraint Files** to add all the following files from the `mlp_conv2d/src/constraints` directory.
    - `mlp_conv2d_synplify.sdc`
    - `mlp_conv2d.fdc`

> **Note**
>
> The `mlp_conv2d.fdc` file contains a compile point constraint for the `mlp_conv2d` module as shown below. Compile points for partitions must be type=locked.
>
> ```
> ###==== BEGIN "Compile Points" - (Populated from tab in SCOPE, do not
> edit)
> define_compile_point  {v:work.mlp_conv2d} -type {locked}
> ###==== END "Compile Points"
> ```

16. In the **Projects** view, under the `mlp_conv2d/Source/RTL` folder, ensure that the `mlp_conv2d_top.sv` filename appears last, preceded by `mlp_conv2d.sv`. Drag and drop file names, if necessary, to change the order.
17. When the above steps have been completed, the synthesis files in the Projects view should be similar to the following:

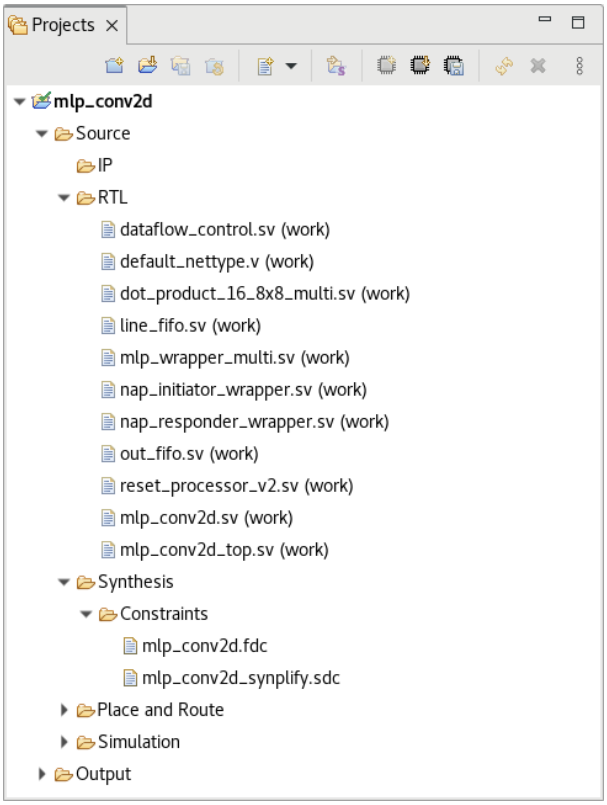

Figure 3 • ACE Projects View Synthesis Files Example

Synthesize the mlp_conv2d_top Design

1. Run the synthesis flow step. To run just the Run Synthesis flow step, perform one of the following:
   - Double-click on the **Run Synthesis** flow step
   - Right-click on the Run Synthesis flow step and select **Run Selected Flow Step**
   - Call `run -step run_synthesis` from the Tcl console

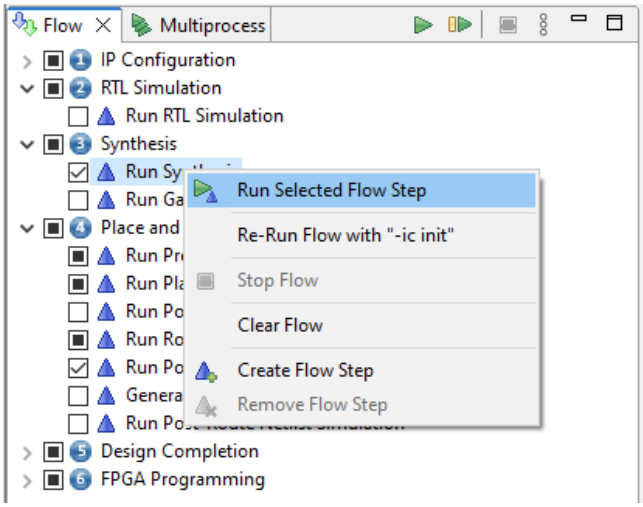

Figure 4 • Running the Run Synthesis Flow Step

> **Note**
>
> The generated `.vm` netlist file will be added to the project after synthesis is complete.

2. Keep ACE open for the steps in the next section.

Set Up ACE Project for mlp_conv2d_top Design Placement, Routing, and Partition Export

1. Select **File** → **Add Project Source Files** → **Place and Route Constraint files** to add all the following files in the `mlp_conv2d/src/constraints` directory:
   - `mlp_conv2d_1.pdc`
   - `mlp_conv2d_top_ioring.sdc`
   - `mlp_conv2d_top_ioring_util.xml`

2. Select **File** → **Add Project Source Files** → **Place and Route Constraint files** to add the `mlp_conv2d_impl_1_partition.prt` file from the `mlp_conv2d/src/ace/impl_1/syn/rev_acx` directory. The file was generated by Synplify Pro in the previous synthesis section.

   The ACE Projects Perspective should now resemble the following:

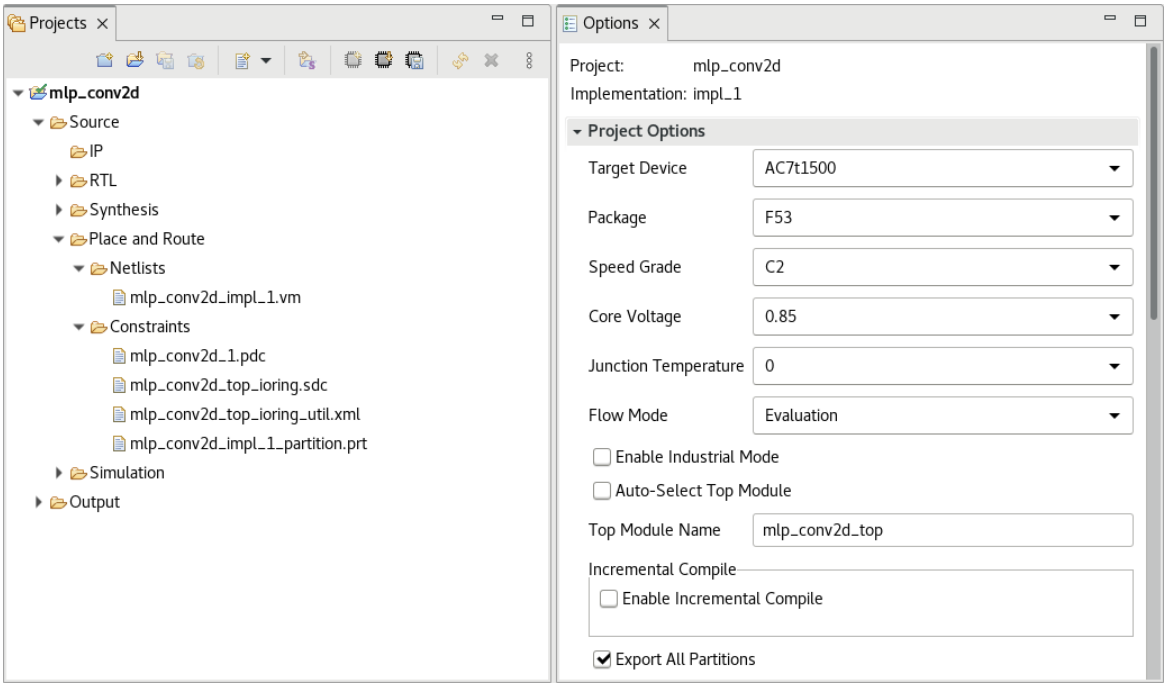

Figure 5 • ACE Projects View Place and Route Files Example

3. Select **File** → **Save Project** to save the project if not already saved.

Place and Route the mlp_conv2d_top Module in ACE for the mlp_conv2d Block

1. Run the place and route flow steps. To run the ACE Prepare, Place and Route flow steps, perform one of the following:
   - Double-click on the Place and Route flow step
   - Right-click on the Place and Route flow step and select **Run Selected Flow Step**
   - Call `run -step place_and_route` from the Tcl console

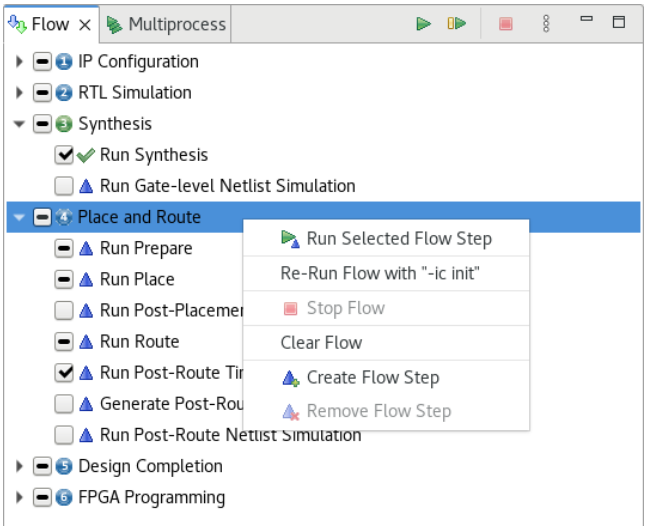

Figure 6 • Running the Place and Route Flow Step

2. When the flow has completed, switch to the Floorplanner perspective.
3. The view should resemble the following image. This particular view enables visibility of the Sites, Instances and Clock Nets. These options may be selected in the fly-out palette shown. Refer to the *ACE User Guide (UG070)* for details.

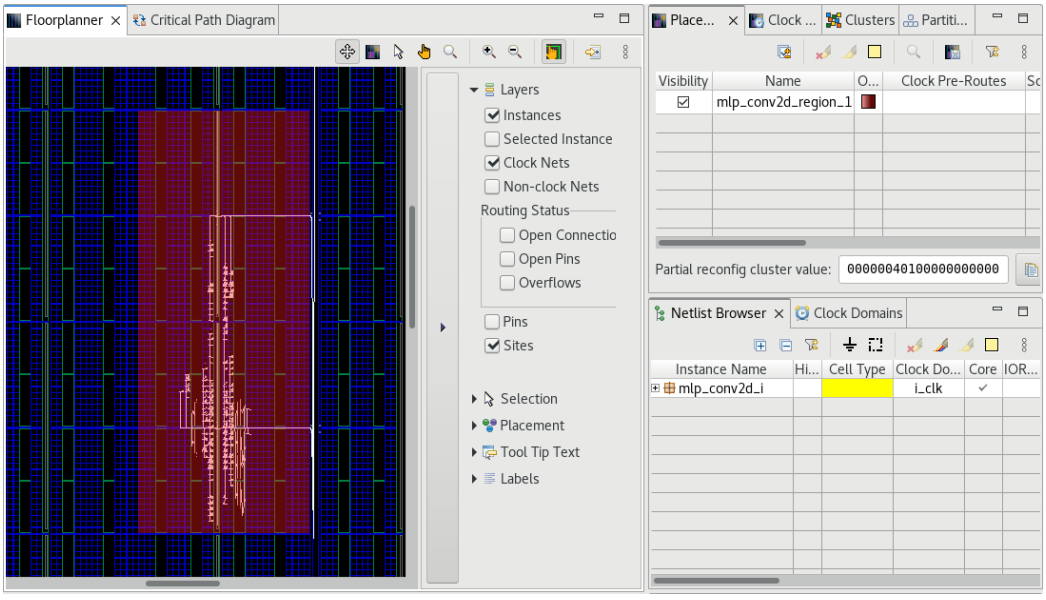

Figure 7 • ACE Floorplanner Perspective Example

The placement region appears as a highlighted box surrounding all of the instances as well as the non-clock and non-reset routes. The region is named `mlp_conv2d_region_1`, and is defined in the `mlp_conv2d_1.pdc` file as follows:

```
create_region {mlp_conv2d_region_1} {117 100 121 153} -snap fabric_clusters
-include_routing
add_region_find_insts {mlp_conv2d_region_1} {find {mlp_conv2d_top_i.*} -insts}
```

With these constraints, all of the instances under the `mlp_conv2d_top` module in the RTL are assigned to the region named `mlp_conv2d_region_1`, and are, therefore, constrained to be placed within the rectangular boundary of that region. The four integer values in the second argument of the `create_region` command specify the coordinates for that region in the format {x1, y1, x2, y2}. The `-snap fabric_clusters` option causes the region coordinates to be snapped into aligned with, if they are not already, the *fabric cluster* grid.

> **Note**
>
> Achronix fabrics are constructed as a 2D grid of identical building blocks called fabric clusters, so the fabric clusters represent the smallest unit of placement-and-routing data that can be relocated using the copy/paste technique demonstrated in this tutorial. The `-include_routing` option will constrain routing within the region.

At the end of the run_route flow step, the following messages should appear in the Tcl console prior to the report_timing_routed step. These messages can also be found in the `impl_1/pnr/log/impl.log` file.

```
INFO: Exporting partition "/mlp_conv2d_top/mlp_conv2d_i" to file
"<PATH_TO_TUTORIAL_DOWNLOAD>/mlp_conv2d/src/ace/impl_1/pnr/output/partitions/
mlp_conv2d_top.mlp_conv2d_i.epdb"
INFO: Writing blackbox netlist for partition "/mlp_conv2d_top/mlp_conv2d_i" to file
"<PATH_TO_TUTORIAL_DOWNLOAD>/mlp_conv2d/src/ace/impl_1/pnr/output/blackboxes/
mlp_conv2d_bb.v"
```

If the **Export All Partitions** implementation option was not enabled during project setup, the `export_all_partitions` Tcl command can be manually executed from the console to export the files at any time after routing.

The `mlp_conv2d_top.mlp_conv2d_i.epdb` file is a binary database containing the partition netlist, placement, and routing data. The `mlp_conv2d_bb.v` file is a black-box Verilog file to be included in the Synplify Pro and ACE projects for the top level design during the partition import flow.

4. In part two of this tutorial, the partition that was just exported will be imported once for *each* of the 40 repeated `mlp_conv2d` blocks in the top-level design. Immediately after import, all instances and routing wires in all 40 copies of the partition will be overlapping. The imported partitions must, therefore, be moved

into non-overlapping locations. This step shows how to find all legal locations, called *compatible* placement locations, for the exported partition, and how those legal locations are expressed for an entire partition. Since a partition is potentially composed of many instances, the placement for an entire partition is represented by the placement of a single representative instance called an *anchor instance*. The placement of the partition can be viewed as being anchored by one instance, with all other instances placed relative to the anchor instance. Any instance can be chosen as the anchor instance, but it is usually convenient to chose the largest instance in the partition.

Achronix library cells in a Speedster7t FPGA can be ranked by size, from largest to smallest, as follows:

Table 1 • Achronix Component IP Library Cell Size Ranking

Rank	Library Cell
1	NAP_AXI_SLAVE / NAP_AXI_MASTER
2	BRAM72K / MLP72
3	ALU8
4	LUT6
5	DFFs

Referring to the Floorplanner view, the largest instance in this design is the ACX_NAP_AXI_SLAVE instance highlighted in yellow as shown in the figure below. The physical design constraints in part two of this tutorial use this instance as the anchor instance.

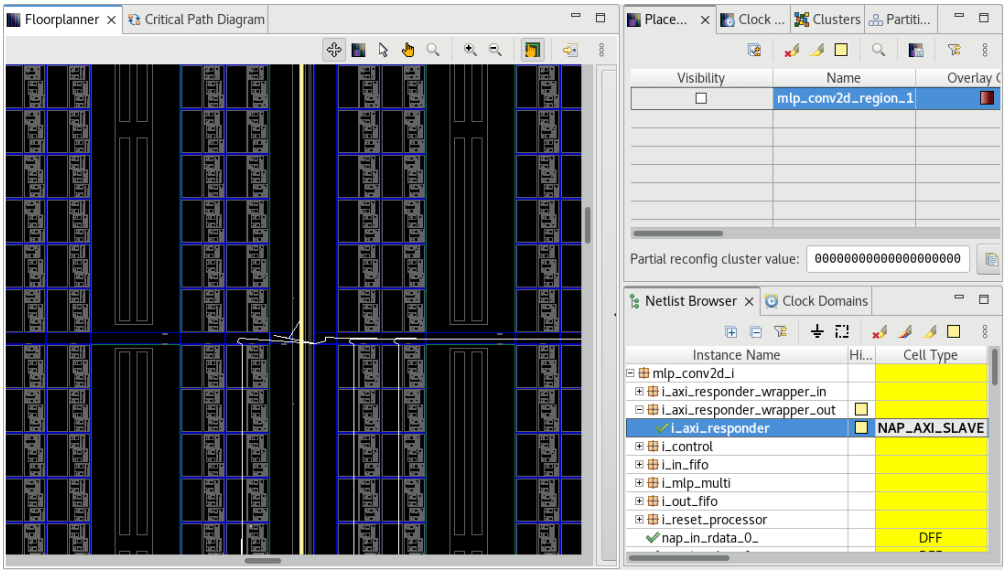

Figure 8 • ACE Floorplanner View ACX_NAP_AXI_SLAVE Cell Example

5. The partition import `.pdc` file refers to the anchor instance by name. To find the name of this instance, hover over it in the GUI and view the tool-tip text, or right-click the instance and select **Add Instance To Selection**. The name of the instance will be returned in the Tcl Console. Here, the name of this instance is `mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder`.
6. The Tcl command `get_compatible_placements` can then be used to find all compatible (fabric cluster grid-aligned) sites on which the anchor instance may be legally placed. In the Tcl Console, run the command shown below. The name of the partition can be found in the `mlp_conv2d/src/ace/impl_1/syn/rev_acx/mlp_conv2d_impl_1_partition.prt` file generated by Synplify Pro, after the `-name` argument. The `-anchor` argument is optional. ACE will select a suitable anchor instance if it is omitted. The `-outputfile` argument is also optional. When omitted, the list of compatible placement sites are returned as a Tcl list, and echoed to the console. However, exporting the list of compatible placement site names to a file is useful, as it can be used as a starting point when writing the `.pdc` file for the import flow.

```
#Tcl Command
#get_compatible_placements {partition_name} -anchor {instance_name}

get_compatible_placements {/mlp_conv2d_top/mlp_conv2d_i} -anchor
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} -outputfile
{compatible_placements_list}
```

The content of the output file is shown below.

Contents of the compatible_placements_list Output File

```
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[1][8].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[2][8].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[3][8].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[4][8].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[5][8].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[6][8].logic.
noc.nap_s}
```

```
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[7][8].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[8][8].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[9][8].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[10][8].logic
.noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[1][6].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[2][6].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[3][6].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[4][6].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[5][6].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[6][6].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[7][6].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[8][6].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[9][6].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[10][6].logic
.noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[1][4].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[2][4].logic.
noc.nap_s}
```

```
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[3][4].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[4][4].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[5][4].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[6][4].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[7][4].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[8][4].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[9][4].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[10][4].logic
.noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[1][2].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[2][2].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[3][2].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[4][2].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[5][2].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[6][2].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[7][2].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[8][2].logic.
noc.nap_s}
```

```
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[9][2].logic.
noc.nap_s}
set_placement -batch -partition
{mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_responder} {x_core.NOC[10][2].logic
.noc.nap_s}
```

This concludes part one of the tutorial, exporting a partition. Exit ACE, and continue to the next section to import the generated partition, place it in repeated locations, and compile in a new top-level design.

Part Two: Import the mlp_conv2d Partition and Replicate 40 Times into a Top-Level Module

In Part two of the tutorial, The `partition_import_export_ref_design_top` design is used to import forty (40) copies of the `mlp_conv2d` partition that was created in part one of the tutorial. The imported partitions are then moved into non-overlapping compatible locations, and the top-level design is then placed and routed.

Set Up ACE Project for partition_import_export_ref_design_top Design Synthesis

1. Create a new empty directory `partition_import_export_ref_design_top/src/ace`.
2. Move into the `partition_import_export_ref_design_top/src/ace` directory and start ACE.
3. In the Projects perspective, if there is an existing project in the Projects tab, right click the project name and select **Remove**.
4. Select **File** → **Create Project**.
5. Click **Browse** to navigate through the file system to open the `partition_import_export_ref_design_top/src/` `ace` directory where ACE was started in a previous step.
6. Use `partition_import_export_ref_design_top` for the project name, and `impl_1` for the implementation name, and click **Finish**.
7. Select **Options** → **Project Options** → **Target Device** and set to "AC7t1500".
8. Select **Options** → **Project Options** → **Package** and set to "F53".
9. Select **Options** → **Project Options** → **Speed Grade** and set to "C2"
10. In the **Options** → **Project Options** menu, deselect the **Auto-Select Top Module** option, and set Top Module Name to `partition_import_export_ref_design_top`.
11. Since this is the top-level design, the black-box models for the sub-block partition created in part one of this tutorial must also be added to the list of RTL files. Select **File** → **Add Project Source Files** → **RTL Files** again,

navigate to the `mlp_conv2d/src/ace/impl_1/pnr/output/blackboxes` directory, and add the `mlp_conv2d_bb.v` file to the project.

12. Select **File** → **Add Project Source Files** → **RTL Files** to add the `partition_import_export_ref_design_top/src/rtl/partition_import_export_ref_design_top.sv` file.
13. In the Projects view, under the `partition_import_export_ref_design_top/Source/RTL` folder, ensure that the `partition_import_export_ref_design_top.sv` filename appears last. Drag and drop the filename, if necessary, to change the order.
14. Select **File** → **Add Project Source Files** → **Synthesis Constraint Files** to add all the following files from the `partition_import_export_ref_design/src/constraints` directory.
    - `partition_import_export_ref_design.sdc`
    - `partition_import_export_ref_design.fdc`
15. When the above steps have been completed, the synthesis files in the Projects view should be similar to the following:

Figure 9 • ACE Projects View Synthesis Source Files Example

Synthesize the partition_import_export_ref_design_top Design

1. Run the synthesis flow step. To run just the Run Synthesis flow step, perform one of the following:
    - Double-click on the Run Synthesis flow step
    - Right-click on the Run Synthesis flow step and select **Run Selected Flow Step**
    - Call `run -step run_synthesis` from the Tcl console

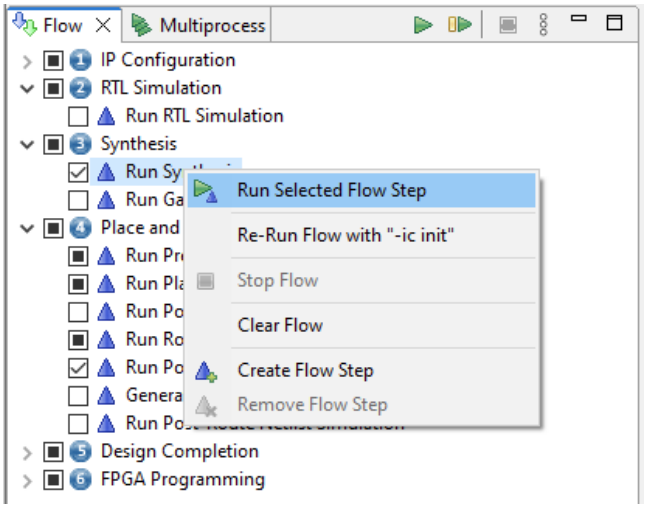

Figure 10 • Running the Run Synthesis Flow Step

> **Note**
>
> The generated `.vm` netlist file will be added to the project after synthesis is complete.

If the black-box files were not added, Synplify Pro issues an error message similar to the following when compiling:

```
Reference to undefined module mlp_conv2d
```

Since this is the top-level design, there is no need to define any synthesis compile point constraints as in part one of the tutorial. The `mlp_conv2d_top` module is represented as a black-boxes in Synplify Pro, and creating a compile point on black-boxes is not permitted.

2. Verify that Synplify Pro used the black-box Verilog models for `mlp_conv2d`, rather than the original RTL implementation, by looking at the Resource Usage report in the `partition_import_export_ref_design/src/ace/impl_1/syn/rev_acx/synlog/partition_import_export_ref_design_top_impl_1_fpga_mapper.srr` log file as shown below. The `mlp_conv2d` cell should be listed as a resource primitive as shown in the following example. The resource usage inside the black boxes is obtained from syn_resources attributes on the mlp_conv2d module definition in `mlp_conv2d_bb.v.`

```
Resource Usage Report for partition_import_export_ref_design_top

Mapping to part: ac7t1500f53c2
```

```
Cell usage:
mlp_conv2d      40 uses

Resource usage inside Blackboxes:
Block RAM : 240 uses
Logic RAM : 0 use
DSP64 : 0 use
ALU8 : 1720 uses
LUT6 : 10840 uses
DFF : 39240 uses
```

3. Select**File** → **Save Project** to save the project.
4. Keep ACE open for the steps in the next section.

Set Up ACE Project for partition_import_export_ref_design_top Design Placement, Routing, and Partition Export

1. Ensure that the **Options** → **Project Options** → **Enable Incremental Compile** option is *not* checked, as partitions cannot be imported into a project that is being run in incremental compile mode.
2. Ensure that the **Options** → **Design Preparation** → **Export All Partitions** is *not* checked since no partitions are being creating for export.
3. Select **File** → **Add Project Source Files** → **Add Place and Route Netlist Files** and add the `mlp_conv2d/src/ace/impl_1/pnr/output/blackboxes/mlp_conv2d_bb.v` file to the project. The black-box models for the sub-block partitions created in part one of this tutorial must be included since they are not defined in `partition_import_export_ref_design_top_impl_1.vm.` Black-box models can be used because the imported partitions include the partition netlist in addition to the placement and routing data.
4. In the Projects tab, ensure that the `partition_import_export_ref_design_top/Place and Route/Netlists` folder has the `partition_import_export_ref_design_top.vm` file listed last since it is the top-level module.
5. Select **File** → **Add Project Source Files** → **Add Place and Route Constraint Files** to add all the following files from the `partition_import_export_ref_design/src/constraints` directory:
   a. `partition_import_export_ref_design.sdc`
   b. `partition_import_export_ref_design.pdc` – In the `partition_import_export_ref_design.pdc` file, the `set_placement -partition` commands are used to move each partition into a legal and non-overlapping location. When called with the `-partition` option, the `instance_name` argument is the name of the partition's anchor instance. The `site_name` argument on which the anchor instance is placed must be a compatible site as listed by the `get_compatible_placements` command shown in part one of the tutorial. All other

instances, as well as all routing wires, in the partition will be automatically placed relative to this anchor instance.

A portion of the `partition_import_export_ref_design.pdc` file is shown in the following example. Each command in the file places one of the imported partition's anchor instances, which in this case is the axi_responder NAP, at a unique location in the fabric.

```
set_placement -batch -partition
{chip_col_0__chip_row_0__mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_respond
er} {x_core.NOC[1][8].logic.noc.nap_s}
set_placement -batch -partition
{chip_col_0__chip_row_1__mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_respond
er} {x_core.NOC[2][8].logic.noc.nap_s}
set_placement -batch -partition
{chip_col_0__chip_row_2__mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_respond
er} {x_core.NOC[3][8].logic.noc.nap_s}
set_placement -batch -partition
{chip_col_0__chip_row_3__mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_respond
er} {x_core.NOC[4][8].logic.noc.nap_s}
set_placement -batch -partition
{chip_col_1__chip_row_0__mlp_conv2d_i.i_axi_responder_wrapper_out.i_axi_respond
er} {x_core.NOC[5][8].logic.noc.nap_s}
```

Alternatively, in the ACE GUI, the partitions can be manually placed by drag-and-drop from the Partitions View to the fabric location in the Floorplanner View. The partition drag-and-drop is available after the Prepare flow step has completed. For more information on partition drag-and-drop, refer to the *Partitions View* section of the ***ACE User Guide*** (UG070). As the partition is dropped, the equivalent `set_placement -partition` Tcl command will be shown in the Tcl console. That Tcl command can then be copied and pasted into a `.pdc` file for future batch-mode runs of the flow.

c. `partition_import_export_ref_design.prt` – In the `partition_import_export_ref_design.prt` file, the `set_partition_info` command is repeated in a loop to import 40 copies of the `mlp_conv2d` partition from the `.epdb` file that was exported in part one of the tutorial.

Caution!

In the `.prt` file for part two of the tutorial, the `set_partition_info -import` argument must reference the `.epdb` file through a path *relative to where ACE was started*. Therefore, if ACE is not started in the `partition_import_export_ref_design/src/ace` directory, the argument shown must be modified. On Windows, the file path specified must be *relative to where the ACE executable file is stored.*

```
for {set col 0} {$col <= 9} {incr col} {
    for {set row 0} {$row <= 3} {incr row} {
        set_partition_info -name "/
partition_import_export_ref_design_top/chip_col_${col}__chip_row_${row}
__mlp_conv2d_i" -view "mlp_conv2d" -cp_type "locked" -import ../../../
mlp_conv2d/src/ace/impl_1/pnr/output/partitions/
mlp_conv2d_top.mlp_conv2d_i.epdb
    }
}
```

6. The Project perspective should now resemble the following example:

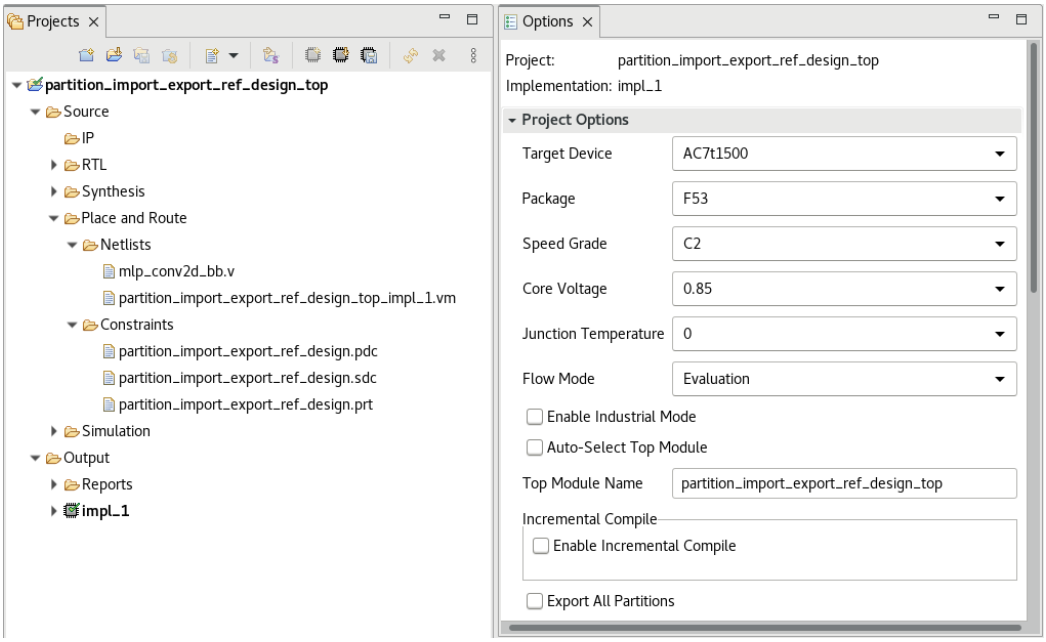

Figure 11 • ACE Project Perspective Constraints Tab Example

7. Select **File** → **Save Project** to save the project if not previously done.

Place and Route the partition_import_export_ref_design_top Design

1. Run the place and route flow steps. To run the ACE Prepare, Place and Route flow steps, perform one of the following:
   - Double-click on the Place and Route flow step
   - Right-click on the Place and Route flow step and select **Run Selected Flow Step**
   - Call `run -step place_and_route` from the Tcl console

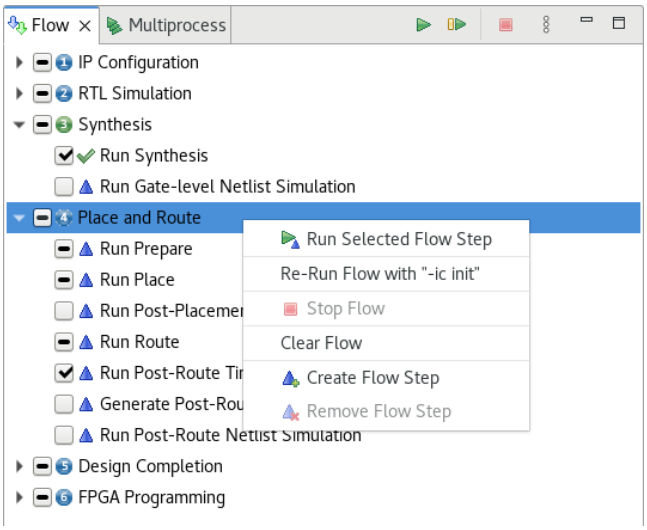

Figure 12 • Running the Place and Route Flow Step

The following messages appear in the console as the `mlp_conv2d` partition database is imported for each of the 40 repeated blocks by the `set_partition_info -import` commands in the `partition_import_export_ref_design.prt` file:

```
Importing partition /partition_import_export_ref_design_top/
chip_col_0__chip_row_0__mlp_conv2d_i from ../../../mlp_conv2d/src/ace/impl_1/
pnr/output/partitions/mlp_conv2d_top.mlp_conv2d_i.epdb ( dbpart->View:
mlp_conv2d, verpart->View(): mlp_conv2d )
Importing partition /partition_import_export_ref_design_top/
chip_col_0__chip_row_1__mlp_conv2d_i from ../../../mlp_conv2d/src/ace/impl_1/
pnr/output/partitions/mlp_conv2d_top.mlp_conv2d_i.epdb ( dbpart->View:
mlp_conv2d, verpart->View(): mlp_conv2d )
Importing partition /partition_import_export_ref_design_top/
chip_col_0__chip_row_2__mlp_conv2d_i from ../../../mlp_conv2d/src/ace/impl_1/
pnr/output/partitions/mlp_conv2d_top.mlp_conv2d_i.epdb ( dbpart->View:
mlp_conv2d, verpart->View(): mlp_conv2d )
```

```
Importing partition /partition_import_export_ref_design_top/
chip_col_0__chip_row_3__mlp_conv2d_i from ../../../mlp_conv2d/src/ace/impl_1/
pnr/output/partitions/mlp_conv2d_top.mlp_conv2d_i.epdb ( dbpart->View:
mlp_conv2d, verpart->View(): mlp_conv2d )
Importing partition /partition_import_export_ref_design_top/
chip_col_1__chip_row_0__mlp_conv2d_i from ../../../mlp_conv2d/src/ace/impl_1/
pnr/output/partitions/mlp_conv2d_top.mlp_conv2d_i.epdb ( dbpart->View:
mlp_conv2d, verpart->View(): mlp_conv2d )
Importing partition /partition_import_export_ref_design_top/
chip_col_1__chip_row_1__mlp_conv2d_i from ../../../mlp_conv2d/src/ace/impl_1/
pnr/output/partitions/mlp_conv2d_top.mlp_conv2d_i.epdb ( dbpart->View:
mlp_conv2d, verpart->View(): mlp_conv2d )
```

The following messages also appear in the console as each partition is moved to its new location by the `set_placement -partition` commands in the `partition_import_export_ref_design/src/constraints/partition_import_export_ref_design.pdc` file:

```
Moved 1471 instances in partition '/partition_import_export_ref_design_top/
chip_col_0__chip_row_0__mlp_conv2d_i' from NAP_AXI_SLAVE_T anchor site
'x_core.NOC[5][6].logic.noc.nap_s' to anchor site 'x_core.NOC[1]
[8].logic.noc.nap_s'
Moved 1471 instances in partition '/partition_import_export_ref_design_top/
chip_col_0__chip_row_1__mlp_conv2d_i' from NAP_AXI_SLAVE_T anchor site
'x_core.NOC[5][6].logic.noc.nap_s' to anchor site 'x_core.NOC[2]
[8].logic.noc.nap_s'
Moved 1471 instances in partition '/partition_import_export_ref_design_top/
chip_col_0__chip_row_2__mlp_conv2d_i' from NAP_AXI_SLAVE_T anchor site
'x_core.NOC[5][6].logic.noc.nap_s' to anchor site 'x_core.NOC[3]
[8].logic.noc.nap_s'
Moved 1471 instances in partition '/partition_import_export_ref_design_top/
chip_col_0__chip_row_3__mlp_conv2d_i' from NAP_AXI_SLAVE_T anchor site
'x_core.NOC[5][6].logic.noc.nap_s' to anchor site 'x_core.NOC[4]
[8].logic.noc.nap_s'
Moved 1471 instances in partition '/partition_import_export_ref_design_top/
chip_col_1__chip_row_0__mlp_conv2d_i' from NAP_AXI_SLAVE_T anchor site
'x_core.NOC[5][6].logic.noc.nap_s' to anchor site 'x_core.NOC[5]
[8].logic.noc.nap_s'
Moved 1471 instances in partition '/partition_import_export_ref_design_top/
chip_col_1__chip_row_1__mlp_conv2d_i' from NAP_AXI_SLAVE_T anchor site
'x_core.NOC[5][6].logic.noc.nap_s' to anchor site 'x_core.NOC[6]
[8].logic.noc.nap_s'
Moved 1471 instances in partition '/partition_import_export_ref_design_top/
chip_col_1__chip_row_2__mlp_conv2d_i' from NAP_AXI_SLAVE_T anchor site
'x_core.NOC[5][6].logic.noc.nap_s' to anchor site 'x_core.NOC[7]
[8].logic.noc.nap_s'
```

```
Moved 1471 instances in partition '/partition_import_export_ref_design_top/
chip_col_1__chip_row_3__mlp_conv2d_i' from NAP_AXI_SLAVE_T anchor site
'x_core.NOC[5][6].logic.noc.nap_s' to anchor site 'x_core.NOC[8]
[8].logic.noc.nap_s'
```

2. When the process completes, switch to the Floorplanner perspective.
3. In the following image, clock and non-clock nets are unselected in the Layers flyout panel to more clearly show where the instances in the partitions are placed. In the **Partitions** tab of the Floorplanner perspective, click the **Auto-Highlight Partitions** button, which assigns a color to each instance based on its parent module in the original RTL hierarchy. The Partitions tab also shows the color that correspond to each partition:

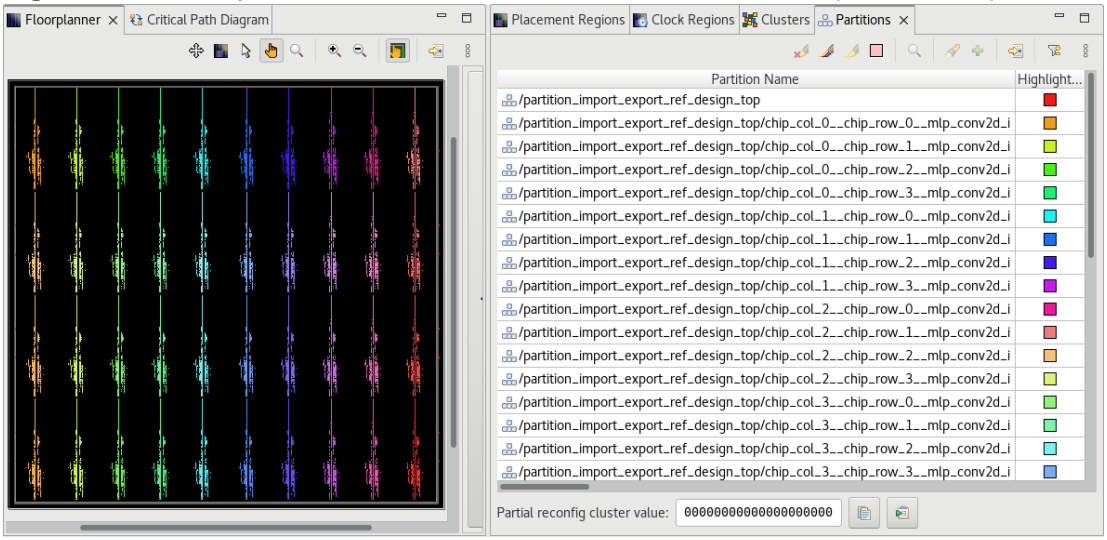

Figure 13 • ACE Floorplanner Perspective Example

This concludes part two of the tutorial, importing partitions for a repeated module into ACE.

Summary

In this tutorial, two designs (`mlp_conv2d_top` and `partition_import_export_ref_design_top`) have been taken through the complete Achronix tool suite synthesis, placement, and routing flow.

- The `mlp_conv2d_top` design consists of a wrapper module that instantiates the `mlp_conv2d` sub-block, which will be exported as a partition.
- A compile Point is defined in Synplify Pro on the `mlp_conv2d` sub-block, which ensures that logic synthesis and optimization does not modify the hierarchical boundary of the `mlp_conv2d module` .
- Synplify Pro exports the compile point definition as a partition definition for ACE.
- ACE also locks the hierarchical boundary of the `mlp_conv2d` partition module boundary during netlist optimization, placement and routing.

- ACE exports the netlist, placement, and routing information for the `mlp_conv2d` partition into a `.epdb` binary database file.
- ACE also exports a `bb.v` black-box Verilog definition file for the partition module.
- The `partition_import_export_ref_design_top` design consists of a top-level netlist that instantiates forty copies of the `mlp_conv2d` sub-block.
- The `mlp_conv2d` sub-block is represented during top-level synthesis in Synplify Pro as a black box module.
- Each of the 40 `mlp_conv2d` modules in the `partition_import_export_ref_design_top` design are instantiated by:
  - Importing the `mlp_conv2d` partition using the `set_partition_info -import` Tcl command.
  - Moving the partition to a compatible non-overlapping location using the `set_placement -partition` Tcl command.
- During top-level placement and routing, the partition instances are locked during placement, and all routing wires are pre-routed.
- In most cases, ACE maintains the pre-placement and pre-routing from an imported partition. However, in rare cases, the ACE placer may need to move a pre-placed instance in order to obtain a legal placement, and the ACE router may need to re-route a pre-routed net in order to solve a top-level congestion problem. Generally speaking, however, timing closure within an imported partition is maintained, and ACE requires less run time for placement and routing compared to using a flat design.
- Runtime improvements can be seen in designs that consist of many repeated blocks using this partition feature rather than placing-and-routing the entire top-level design using a flat methodology

The flow outlined in this tutorial can be used to form the core of a bottom-up hierarchical design methodology, or a module re-use strategy. However, all file management and revision control in such advanced flows must be provided by the user.

Revision History

Version	Date	Description
1.0	07 Aug 2023	• Initial Achronix Release.
1.1	28 Feb 2025	• Updated to use ACE-drive synthesis flow

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

Website: www.achronix.com
E-mail : info@achronix.com

Notice of Disclaimer

The information given in this document is believed to be accurate and reliable. However, Achronix Semiconductor Corporation does not give any representations or warranties as to the completeness or accuracy of such information and shall have no liability for the use of the information contained herein. Achronix Semiconductor Corporation reserves the right to make changes to this document and the information contained herein at any time and without notice. All Achronix trademarks, registered trademarks, disclaimers and patents are listed at www.achronix.com/legal.