Vivado Subsite Routing Tool

Contributors:
Thomas Townsend, Brad White, and Brent Nelson

NSF Center for High Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering
Brigham Young University
Provo, UT 84602

Revised: 1/10/17

Contents

Introduction

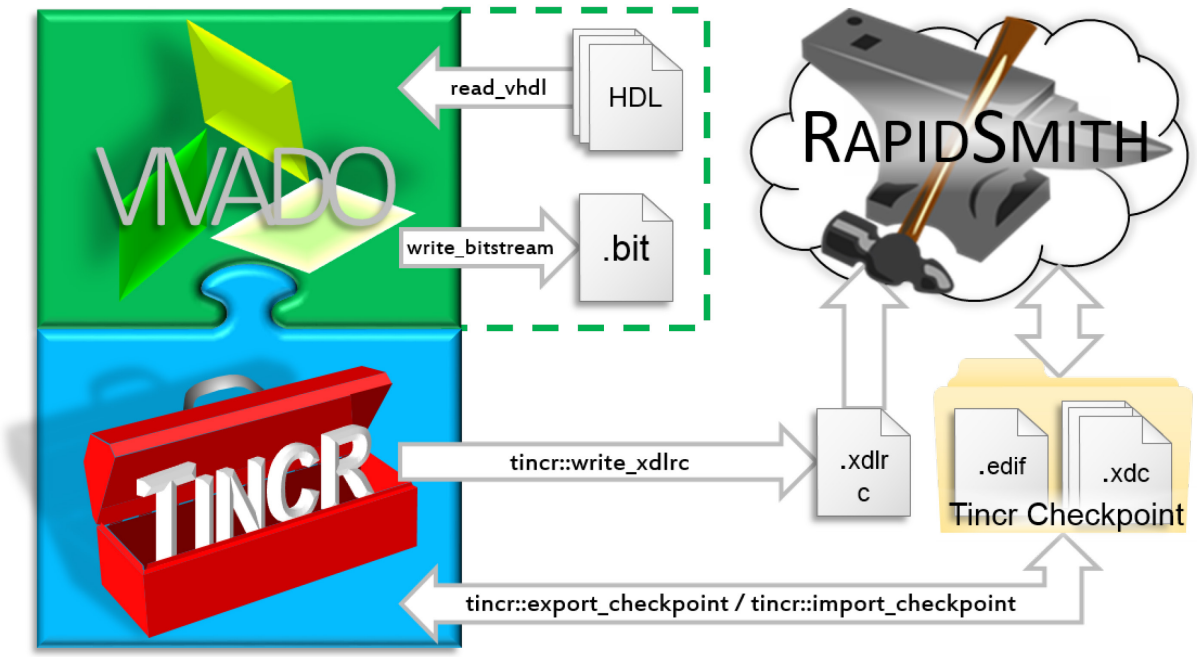
Custom CAD tools for FPGA development have always been in high demand by researchers who desire to test new ideas outside the confines of vendor tools. Existing frameworks such as RapidSmith (Learn more about RapidSmith here) have seen success in creating these tools for Xilinx FPGAs by leveraging the XDL interface offered in Xilinx's old design suite, ISE. However, with the introduction of Xilinx's new design suite, Vivado, tools based on the XDL interface will be rendered obsolete. This means that tools like RapidSmith, which rely on XDL and XDLRC will no longer work. BYU research, specifically the work of Brad White, has shown that it is possible to extract *almost* all of the needed information from Vivado in order to create XDLRC files for newer devices. Namely, everything can be extracted from Vivado's TCL interface besides how components at the primitive site level are connected. The Vivado Subsite Routing Tool, in conjunction with TINCR, is able to generate this missing information, allowing the prolonging of successful CAD tool frameworks that rely on XDL and XDLRC file formats.

RapidSmith 2.0

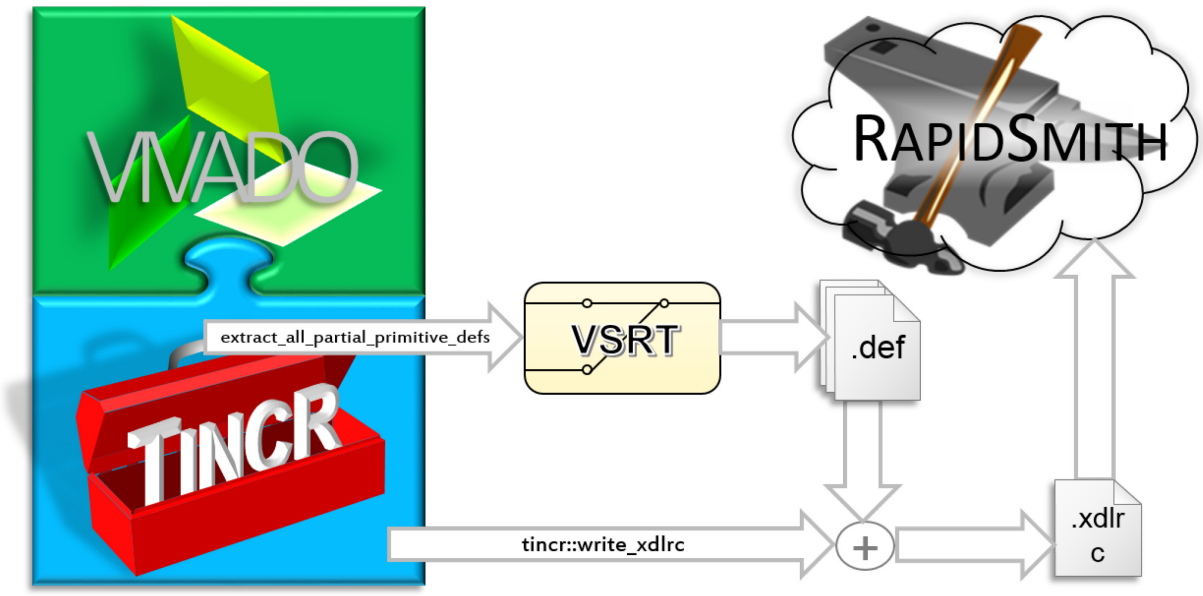
Since XDL is no longer being supported by Xilinx, the RapidSmith tool suite is being revamped to integrate with Vivado. This involves two major tasks.

(1) Extracting design information out of Vivado's TCL interface into the RapidSmith workspace.
(2) Creating the XDLRC-like device files that RapidSmith uses to access all available resources on an FPGA.

In order to achieve these two tasks, a set of TCL libraries, known as TINCR, was created by BYU researchers to allow users to easily gain access to several different parts of a design or a device through Vivado's TCL interface. The three TCL functions that are particularly integral to the success of RapidSmith 2.0 are tincr::write_xdlrc, tincr::write_rscp, and tincr::read_tcp. Write_xdlrc generates the device files that RapidSmith uses, while write_rscp/read_tcp allows the flow of a design to and from RapidSmith. A diagram of the overall flow can be seen below.



The Vivado Sub-site Routing Tool is focused on task (2) listed above. Specifically, VSRT is used to extract the routing information between site-level components. We need this information to produce complete and useful device files for RapidSmith. VSRT facilitates the gathering of this information in two steps. Through TINCR, all available information for each primitive site (site pins, site pips, and bels) are found and written to a dedicated primitive definition file (.def file extension). Each of these primitive definition files are imported into VSRT, and a user manually draws the connections between components within the primitive site. This is done while viewing the same primitive site in the Vivado GUI. A diagram depicting the flow of creating an XDLRC file from Vivado can be seen below.



Primitive Definition Files

The type of files that VSRT produces are primitive definition files which are modeled after the primitive definitions found in XDLRC files. They textually describe the components found within a primitive site and how they are connected. Bels, site pins, site pips, configuration options, and route-throughs are the elements that are most commonly found within these files. An example of a complete primitive definition file of type BUFHCE can be seen below.

```
(primitive_def BUFHCE 3 7
        (pin CE CE input)
        (pin I I input)
        (pin O O output)
        (element CE 1
                (pin CE output)
                (conn CE CE ==> CEINV CE)
                (conn CE CE ==> CEINV CE_B)
        )
        (element I 1
                (pin I output)
                (conn I I ==> BUFHCE I)
        )
        (element O 1
                (pin O input)
                (conn O O <== BUFHCE O)
        )
        (element CEINV 3
                (pin OUT output)
                (pin CE input)
                (pin CE_B input)
                (pin CE_B input)
                (cfg CE CE_B)
                (conn CEINV OUT ==> BUFHCE CE)
                (conn CEINV CE <== CE CE)
                (conn CEINV CE_B <== CE CE)
        )
        (element BUFHCE 3 # BEL
                (pin CE input)
                (pin I input)
                (pin O output)
                (conn BUFHCE CE <== CEINV OUT)
                (conn BUFHCE I <== I I)
                (conn BUFHCE O ==> O O)
        )
        (element CE_TYPE 0
                (cfg SYNC ASYNC)
        )
        (element INIT_OUT 0
                (cfg 0 1)
        )
)
```

TINCR CAD Tool

Creating XDLRC device files for RapidSmith is only one use of the TINCR CAD suite. It, in of itself, is also used as a CAD tool framework where users can write TCL scripts to manipulate designs within the confines of the Vivado. This was largely motivated by the fact that exporting and importing design information out of and into Vivado can be time-consuming. Currently, Vivado offers several different TCL classes that represent various parts of an FPGA. Such classes include the Bel, Pin, and Site Pip class. Using TCL commands “get_bels”, “get_site_pins”, and “get_site_pips”, a user can get a list of bels, site pins, and site pips that exist within a specified primitive site. A Site Wire class, which is not offered by Vivado, could offer the same functionality for wires within a primitive site. The idea is that a user could write a simple TCL command along the lines of “get_site_wires –of $bel” to gain access to all of the wires connected to that bel. This would in turn tell us what other elements that bel is connected to. The complete primitive definitions that are generated from VSRT will be used to create a Site Wire class for each Xilinx part.

Installation

Tool Facts

- Written in Java
- Built on the QtJambi GUI framework - version 4.8.7
- Developed within Java 1.8.0_101
- Supported Operating Systems:
    - Linux 32 and 64 bit
    - Windows 32 and 64 bit
    - MacOSX
- Tested on:
    - Fedora 20
    - Windows 7
    - Windows 8.1

Requirements

- Eclipse IDE installed (Optional)
- Vivado Tool Suite
- Clone the RapidSmith2 project from GitHub (https://github.com/byuccl/RapidSmith2) and follow the installation instructions.
- Clone the TINCR project from GitHub (https://github.com/byuccl/tincr) and follow the installation instructions.
- JDK 1.8 (earlier versions may work, but have not been tested).

Steps

The VSRT tool can be run on the command line or directly from an IDE (like eclipse). To run it in an IDE, simply navigate to src/main/java/edu/byu/ece/rapidSmith/device/vsrt/gui/VSRTool.java and click the IDE's run button. To run from the command line, follow these instructions:

Command Line:

1. Make sure Vivado and the JDK are on your PATH
2. If you haven't already, create a new environment variable called RAPIDSMITH_PATH, and set it to your RapidSmith2 local repository.
3. Run the gradle build in your RapidSmith2 repository ("gradlew build" for unix or "gradlew.bat build" for windows).
4. After the build completes, go to build->distributions. There you will see a tar and zip file with all needed RapidSmith jar files. Extract either the tar file or zip file to the directory of your choice. There, you should see a folder called "lib" and a RapidSmith2 jar file.
5. Include the RapidSmith jar file and all of the jars in the "lib" directory onto your CLASSPATH.
6. Test your installation by running the following command. If you get a usage statement, you have installed everything correctly!

```
[ttown523@] java edu.byu.ece.rapidSmith.device.vsrt.gui.VSRTool
```

Vivado Terminology

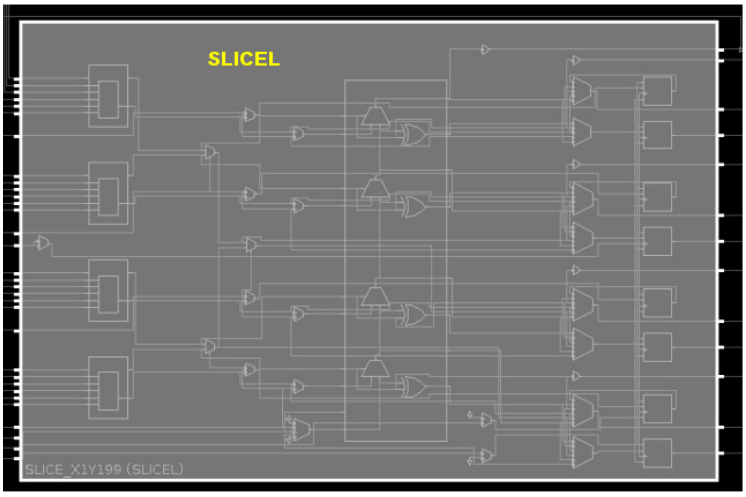
This section is intended to give a brief introduction to the Vivado FPGA terminology that is used within VSRT. If you are already familiar with Vivado terminology, then you can skip to the user guide section.

Tiles

An FPGA can conceptually be thought of as a two dimensional array of Tiles. A tile is essentially a part within an FPGA that performs a specific function. For example, the image below shows three types of template tiles. The tiles in the left and rightmost column are used as routing resources, while the tiles in the middle are used to perform combinational and sequential digital logic. Exact copies of these tiles are stamped throughout the FPGA.

Primitive Sites

Each tile can contain one or more primitive sites. Primitive sites are where the actual hardware components to perform the tile's overall function reside. The tile simply routes signals to and from these primitive sites. If we zoom in on the CLBLL of the picture above, we can see that it contains two primitive sites. One of these sites is of type SLICEM and the other is SLICEL, which is shown in the image below. There are three main components to primitive sites: site pins, site pips, and bels. The VSR-Tool is used to generate the connections between these.



Elements

Elements are all parts of a primitive definition that can be found within a primitive site. They can either be a site pin, site pip, or bel

Site Pins

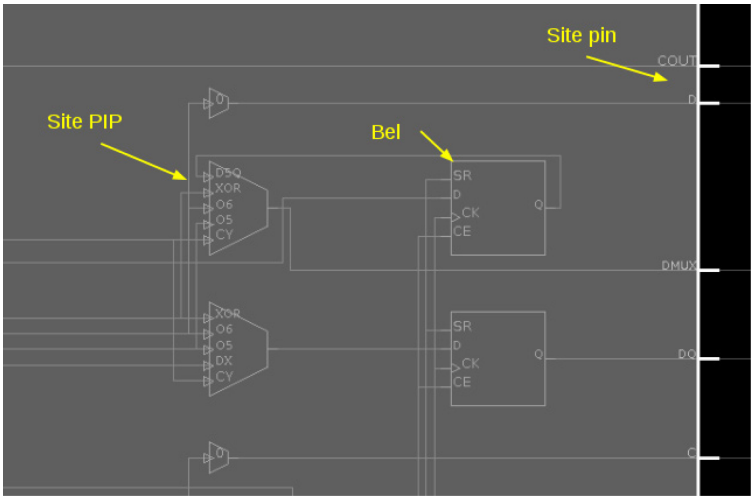
These are the inputs and outputs of a primitive site.

Site Pips

These are the routing resources for a primitive site. They are essentially muxes, multiple inputs with a selected output.

Bels

These are hardware components within a primitive site that perform some sort of logical operation. For example, LUT's and Flip Flops are bels.



User Guide

Once you have installed all of the necessary software, you are now ready to begin using the tool.

Getting Started

To begin, open a command line terminal and open Vivado in TCL mode. You should see something similar to the shell output below if everything is correctly configured on your machine. If Vivado is not recognized as an external command or the TINCR libraries were not loaded correctly, refer back to the needed tools section to make sure you installed everything correctly.

```
[ttown523@CB461-EE09968:~] vivado -mode tcl

****** Vivado v2014.2 (64-bit)
  **** SW Build 928826 on Thu Jun  5 17:55:10 MDT 2014
  **** IP Build 924643 on Fri May 30 09:20:16 MDT 2014
    ** Copyright 1986-2014 Xilinx, Inc. All Rights Reserved.

Vivado%
```

The first step in the process is to extract all of the information that we can from Vivado for each primitive site, and store that information into its own primitive definition file. To do this, run the TCL script shown below and substitute the directory "examplePD/" with wherever you want to store these files.

```
Vivado% tclapp::byu::tincr::extract_all_partial_primitive_defs examplePD/
```

The directory does not have to exist prior to running the script; it will be created if it does not. This script can take up to 30 minutes to run, so take a break or do something else in the meantime. Once completed, you should see something similar in your terminal as the output below: This means that you have successfully extracted all of the partial primitive definitions from Vivado!

```
examplePD/zynq/IDELAYE2_FINEDELAY_ALTERNATE.def
examplePD/zynq/IPAD_ALTERNATE.def
examplePD/zynq/RAMB18E1_ALTERNATE.def
examplePD/zynq/IOB18M_ALTERNATE.def
examplePD/zynq/IOB18S_ALTERNATE.def
Successfully created all .def files in directory examplePD/
Vivado%
```

If you navigate to the directory you passed as an argument to the script, you should see the following directory structure.

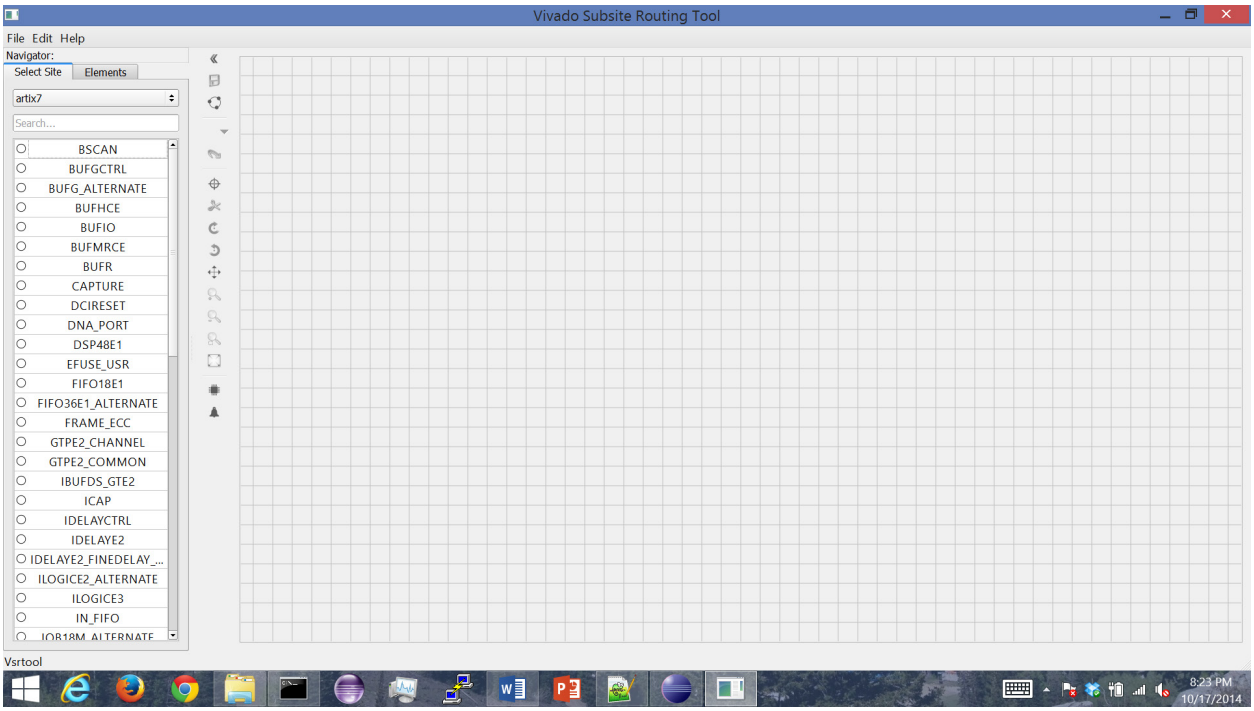
```
[ttown523@CB461-EE09968:examplePD] ls
artix7  kintex7  kintexu  log.txt  virtex7  virtexu  zynq
```

There is a folder for each architecture that the current version of Vivado is currently supporting. Take a look and see which primitive sites are found in each. A log file has also been generated to easily find a part within an architecture where a primitive site can be found, and valid placements for alternate sites.

Now you are ready to start using VSRT. Open a new command prompt and type the following command.

```
[ttown523@CB461-EE09968:~] java
edu.byu.ece.rapidSmith.device.vsrt.gui.VSRTool examplePD/
```

The examplePD/ directory is an argument to the program, and it should point to the directory you created in the previous step. Alternatively, you can run the program directly from your IDE (this is the easiest route, but don't forget to add the argument). The tool should look similar to the screen shot below.

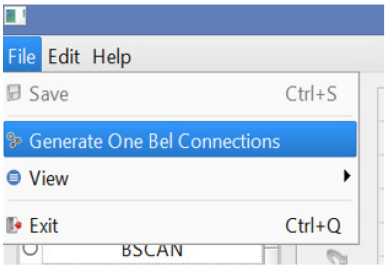


Primitive Definitions with One Bel

Before you do anything else, make sure to automatically generate the sub-site connections for all primitive sites with only one bel. The reason the connections within these sites can bypass the GUI and be generated automatically is they all have the same format. Specifically, site pin connections within these sites can be classified in three ways:

1.) They are directly connected to a bel pin of the same name
2.) They are connected to a site pip inverter of the same name followed by _INV. Take a pin named CLK for example. The inverter would be called CLK_INV, the input pins would be called CLK and CLK_B (the inverted version), and the output pin would be called OUT.
3.) They are unconnected. If this is the case, then the site pins are trimmed from the final primitive definition file.

Knowing this information makes it easy to create a script that can take a primitive site with one bel, and generate the complete definition file for it. To do this for all one-bel primitive sites, click File->Generate One Bel Connections. This should be done before doing any other primitive sites by hand.



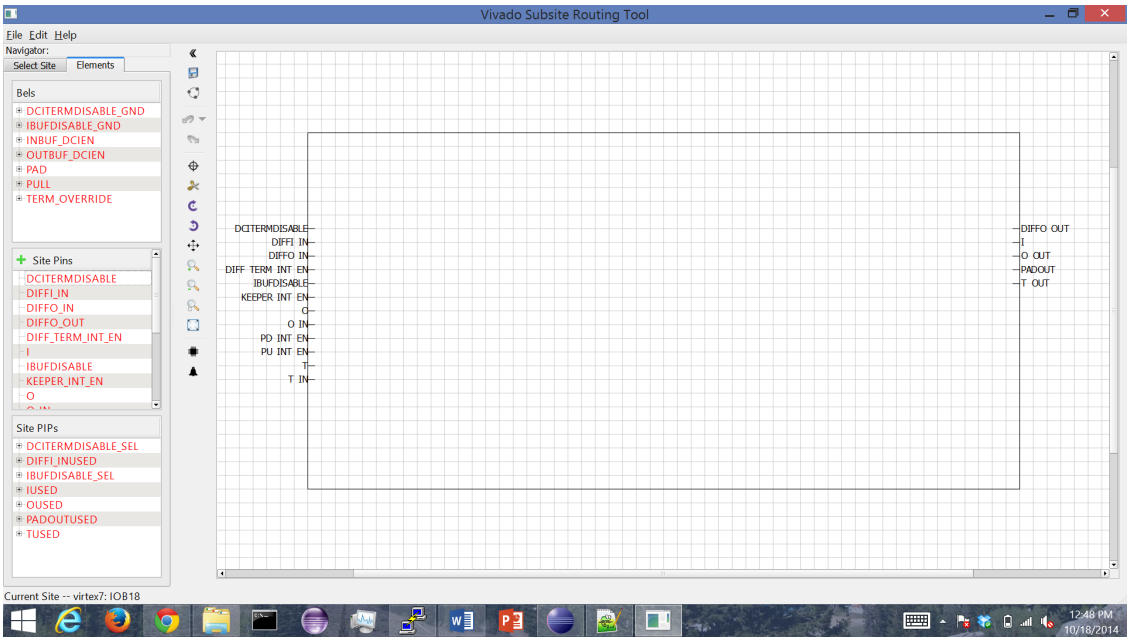
A progress bar will appear showing the state of the operation. It should be done within a minute. After the operation is complete, you should see a significant number of primitive sites that have check marks next to their name. This signifies that their sub-site connections have been successfully generated.

If want to view these newly created primitive definitions, navigate back to your primitive definitions directory. You will find a new directory called *CompletePrimitiveDefinitions* where all of your completed primitive definitions will be stored.

```
[ttown523@CB461-EE09968:examplePD] ls
artix7              CompletePrimitiveDefinitions  kintexu  virtex7  zynq
completedSites.txt  kintex7                       log.txt  virtexu
```

Primitive Site Example

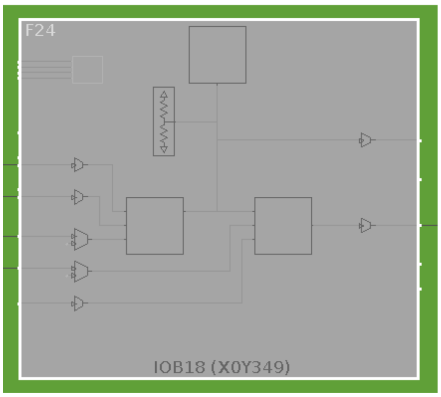
Now that you have generated all of the primitive definitions that you can automatically, the only thing left to do is finish the rest of them by hand. Let's demonstrate the general process by creating a complete primitive definition for an IOB18 primitive site. Start by choosing the virtex7 architecture in the navigator dropdown menu. Type "IOB" into the search menu and the IOB18 primitive site should be highlighted at the top of the table view. Double click the primitive site to open it. You should see something similar to the following. The red text tells the user that there are one or more unconnected pins that need connecting.



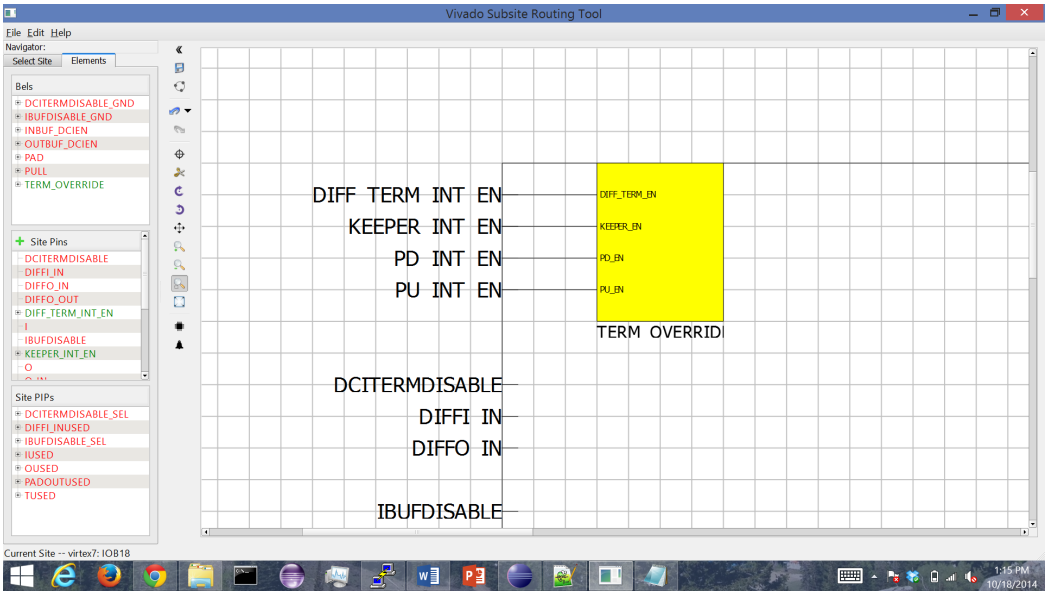
In order to generate the sub-site connections, we have to know what these connections are. Lets' view this same site in Vivado's GUI. To open an IOB18 in Vivado, start by opening Vivado in TCL mode again. Use the following commands to link an open design in Vivado for a Virtex7 part and view an IOB18 primitive site in the GUI. You will have to run the last command in the GUI command line

```
Vivado% link_design -part xc7vx485tffg1157-1
Vivado% set site [lindex [get_sites -filter {SITE_TYPE==IOB18} ] 0]
Vivado% start_gui
Select $site
```

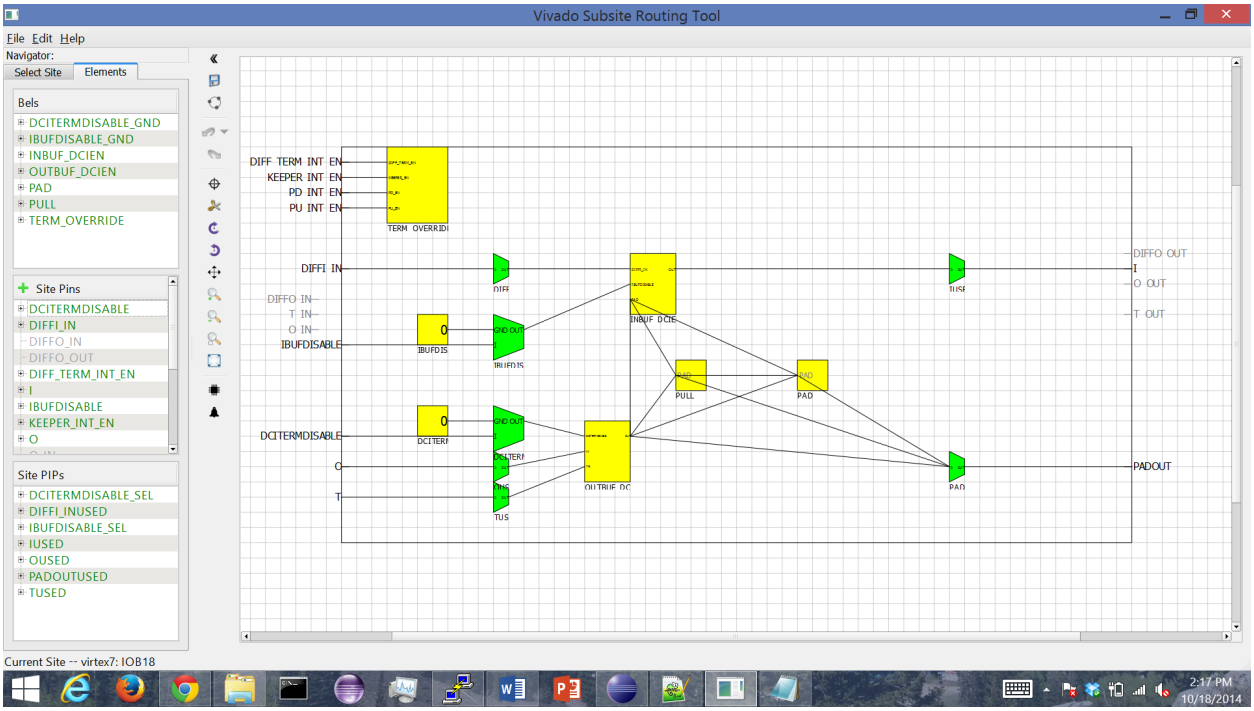
You should now see an instance of an IOB18 primitive site in Vivado's GUI interface:



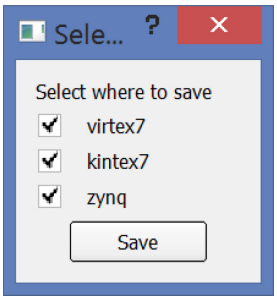
We now have everything we need to make the connections for an IOB18 and create a primitive definition file! Start by double clicking the TERM_OVERRIDE bel to add it to the scene. Place it in the top-left-hand corner of the primitive site. From Vivado's GUI, we can see that the site pins DIFF_TERM_INT_EN, KEEPER_INT_EN, PD_INT_EN, and PU_INT_EN are connected to this bel. So, drag and drop each of the site pins, place them so that they are across their associated bel pins, and connect each of them to the bel. It should look something like:



If you look in the navigator view, you can now see that TERM_OVERRIDE and all of the site pins that you moved have GREEN text. This means that all of the pins are connected and they are good to go! Expand the tree view and you can see the connections for each bel and site pin. You should now be able to connect the rest of the connections on your own. Once finished, the site may look something like:



To write out the completed primitive definitions, click File->Save. A popup menu will appear listing all of the architectures with a primitive site of the same name.



In this case, the IOB18 primitive is identical for both the kintex7 and Zynq architectures. Check these architectures and click save. The complete primitive definition for the IOB18 will now be written to each of these architectures. Woo Hoo! You have just created your first complete primitive definition from Vivado.

Saving

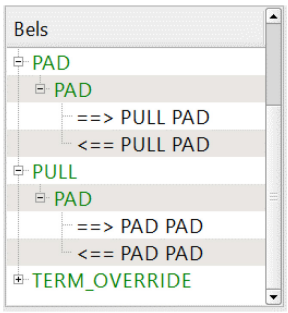
To save the progress of a primitive site click File->Close Site. A prompt will display asking you if you want to save your progress. Click Yes. This will write your progress out to an XML file and bring you back to the default primitive site navigator view. To continue working on the site, all you have to do is open it once again. The primitive site will look identical to the way you left it.

In-out Pins

There are many bel pins in Vivado whose direction are INOUT. In VSRT, the color of these site pins are gray rather than black. INOUT pins can connect to INPUT, OUTPUT, or other INOUT pins, and each case is handled differently. The chart below shows the connections that will be generated for each combination.

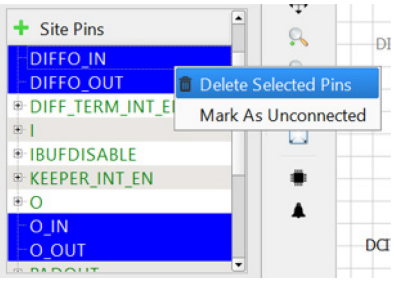
Pin 1 Type	Pin 2 Type	Connection(s) Generated
INOUT	INPUT	Pin 2 → Pin 1
INOUT	OUTPUT	Pin 1 → Pin 2
INOUT	INOUT	Pin 1 → Pin 2 Pin 1 ← Pin 2

You don't have to worry about correctly generating connections involving INOUT pin types, we take care of that for you! To test this out, connect PAD and PULL bel, and view the connections in the tree view. You will see that two connections have been generated for each bel.



Manipulating Site Pins

If you look closely at the Vivado GUI, you can see that there are some site pins that aren't connected to anything. Mark these site pins as unconnected by right clicking on them in the tree view, and selecting "Mark As Unconnected." They will now appear gray in the graphics scene. Also, if there is a site pin that shows up in VSRT but not in the Vivado's GUI, you can delete it from the primitive definition by right clicking on the pin and selecting "Delete Selected Pins".



Other Features

 Hide Navigator View – If you have already placed all of the bels and site pips, click this to hide the navigator view and give yourself a little extra room to draw connections.

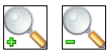
 Save Primitive Site – Once you are confident that you have generated all of the connections for a given primitive site, click this button to write out a complete primitive definition file. Alternatively, you can also click File->Save.

 Undo/Redo – Shortcuts: "Ctrl + Z" and "Ctrl + Y"

 Draw Wire – Click this button to draw wires between bels, site pins, and site pips. Shortcut: "w"

 Remove Elements – Click this button to remove elements from the scene. Drag and drop a rectangular region, and all elements within that region will be removed (except site pins).

 Rotate Elements – If you want to rotate an element to make drawing wires easier, click these buttons. Shortcuts: "Ctrl + R" and "Ctrl + Shift + R".

 Zoom in/out – To zoom in and out of the scene, click these buttons. Shortcuts: "Ctrl + scroll".

 Zoom to rectangle – To zoom to a particular rectangular region, use this button.

 Zoom to best fit – Zoom the scene so that the entire primitive site is available.

 Add site pin – If a site pin shows in the Vivado GUI, but not in the site pin tree, use can use this to button to add the site pin.