

grittyengineer.com

Creating Vivado IP the Smart Tcl Way - Gritty Engineer

Christopher Hogstrom

9–12 minutes

Recently at work I checked out a co-workers Vivado project that was under review. Instead of putting all of the project files in SVN they only put the source files, constraints, IP files, and a Tcl script that re-created the project. Overall it worked pretty well. However, there was just one catch, Vivado reported that one of the IP files under source control was generated using a different version of Vivado than the one currently being used. Oddly enough, the version numbers matched but I still had to regenerate the IP before the warning would go away.

I later learned that the team using this project spent two weeks trying to find a bug that was only present on hardware. The culprit, Vivado IP being out-of-date. I wish I could say this was an isolated case but I've experienced this first hand on other projects. Somehow the IP files are corrupted under source control or are not properly updated causing difficult to find bugs. So, like a gritty engineer, I decided to find a better solution that would allow for consistent IP setup.

Two Approaches

Vivado supports two methods that yield superior results and both rely on Tcl. The first approach allows you to create Vivado IP from scratch using

Tcl. This means that there is no .xci file to store under source control, instead, it is a Tcl script. There are a few of advantages:

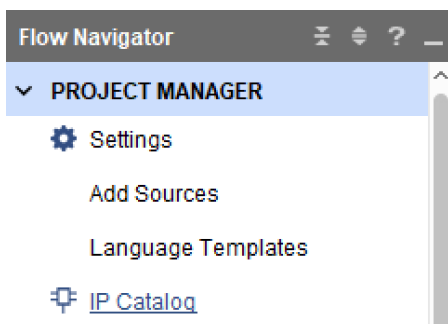
- It is simple to see changes between versions in source control
- IP is created using the current version of Vivado
- If the IP already exists then the Tcl script skips generating the IP

The second method requires the .xci file to be stored under source control. A Tcl script checks if the IP is locked and if a new version of the IP is available. If the IP is locked and a new version is available then the Tcl script updates the .xci file. Once the IP is determined up-to-date , the script produces the IP output products. The advantages to this approach are:

- IP is not regenerated if not needed
- The scripting is slightly easier than generating the IP from scratch
- Updating IP is slightly faster than creating IP from scratch

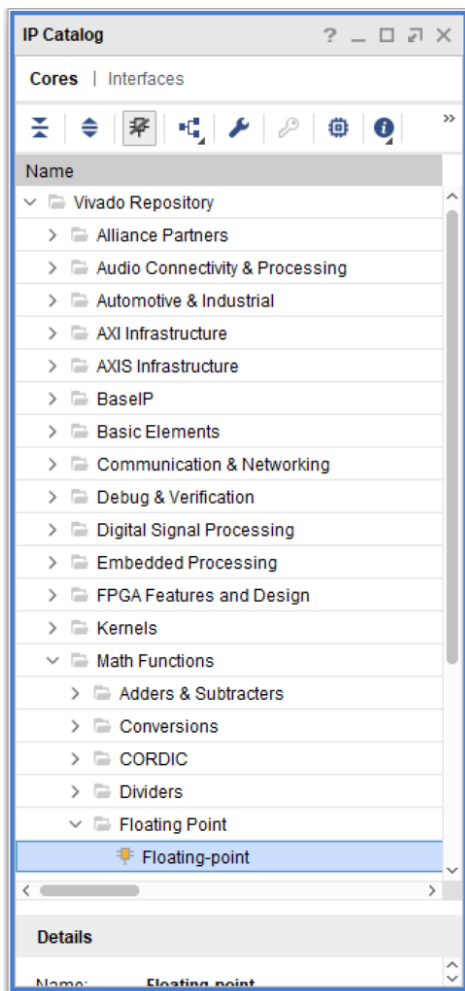
The main disadvantage to this approach is that you can't easily see changes made to the .xci files. If this is a mature project that is unlikely to experience changes in the IP then it's not an issue. However, many projects are in flux and occasionally engineers must make updates to the IP. When this is the case, using Tcl scripts to build the IP from scratch is advantageous.

Get Vivado IP Tcl commands from the GUI



Opening Vivado IP catalog

As with most things there's an easy way and a hard way to create IP. I prefer the easy way unless there's a strong incentive otherwise. The easy way is to create the IP using the Vivado GUI and then capture the Tcl commands generated in the journal file (.jou). For this example, we'll create a fixed point to floating point IP block. In the Flow Navigator under Project Manager select the IP catalog.

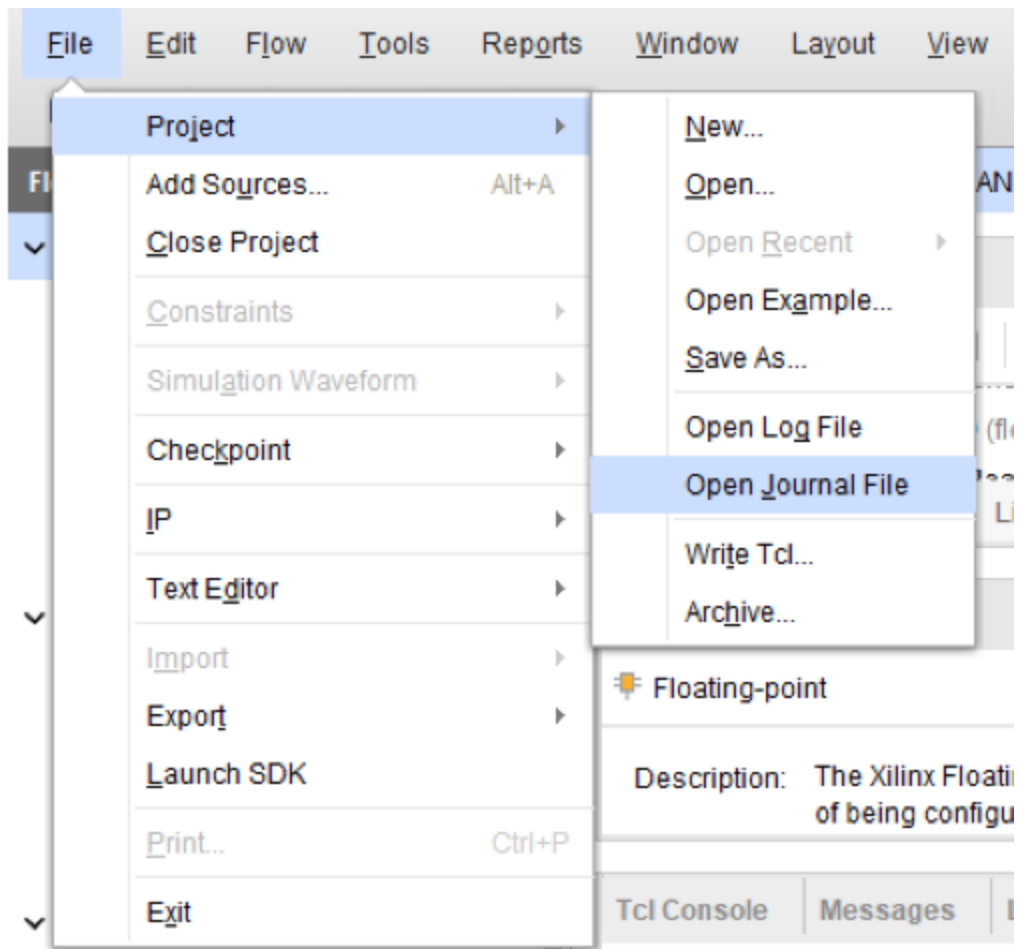


In the IP catalog window select Cores and then Vivado Repository>Math Functions>Floating Point>Floating-point.

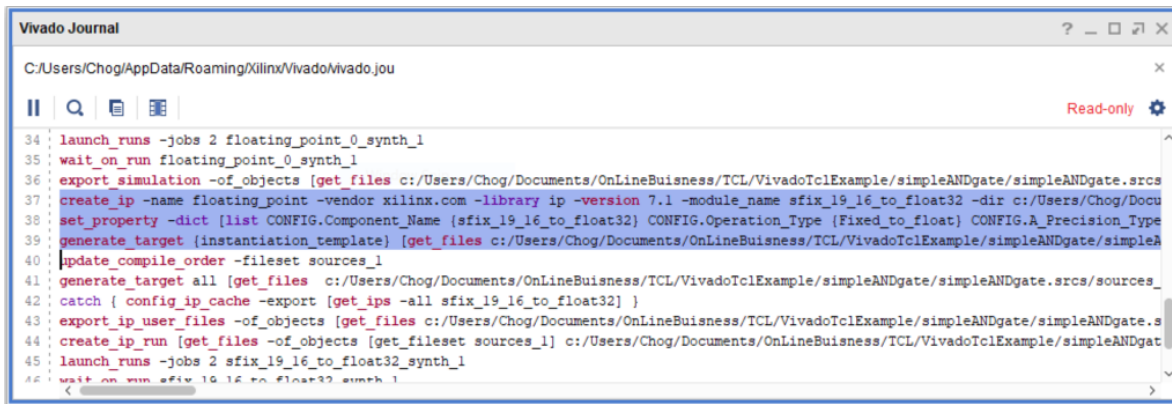
A Customize IP window appears and you can specify a component name in the "Component Name" box. Here I have given the component name `sfix_19_16_to_float32`. Under the "Operation Selection" tab select "Fixed-to-float". Select the "Precision of Inputs" tab and select whatever

input precision suites your fancy. I opted for a custom input precision with an integer width of 19 bits and fractional width of 16 bits. Check each of the remaining tabs to make sure the IP is generated with the correct settings. Click “OK”. Finally, in the Generate Output Products window select “Generate”.

Now that we have created the Vivado IP it's time to capture the corresponding Tcl commands. Open the journal file by going to File>Project>Open Journal File. In the Journal file, you'll see a number of Tcl commands but we're interested in three. The commands start with **create_ip**, **set_property**, and **generate_target**. If done correctly, these three commands are executed sequentially. The **create_ip** command has an argument called **-module_name <userDefinedName>**. The **userDefinedName** should match the name you gave the component.



Open Journal file



The screenshot shows the Vivado Journal window with a Tcl script. The script includes commands for launching runs, waiting for completion, exporting simulation objects, creating an IP block, setting properties, generating a target, updating the compile order, generating a target, catching IP cache, exporting IP user files, creating an IP run, and launching runs. The script is read-only and has a settings icon.

```
34 launch_runs -jobs 2 floating_point_0_synth_1
35 wait_on_run floating_point_0_synth_1
36 export_simulation -of_objects [get_files c:/Users/Chog/Documents/OnLineBuisness/TCL/VivadoTclExample/simpleANDgate/simpleANDgate.srcs
37 create_ip -name floating_point -vendor xilinx.com -library ip -version 7.1 -module_name sfix_l9_l6_to_float32 -dir c:/Users/Chog/Docu
38 set_property -dict [list CONFIG.Component_Name {sfix_l9_l6_to_float32} CONFIG.Operation_Type {Fixed_to_float} CONFIG.A_Precision_Type
39 generate_target (instantiation_template) [get_files c:/Users/Chog/Documents/OnLineBuisness/TCL/VivadoTclExample/simpleANDgate/simpleA
40 update_compile_order -fileset sources_1
41 generate_target all [get_files c:/Users/Chog/Documents/OnLineBuisness/TCL/VivadoTclExample/simpleANDgate/simpleANDgate.srcs/sources_
42 catch { config_ip_cache -export [get_ips -all sfix_l9_l6_to_float32] }
43 export_ip_user_files -of_objects [get_files c:/Users/Chog/Documents/OnLineBuisness/TCL/VivadoTclExample/simpleANDgate/simpleANDgate.s
44 create_ip_run [get_files -of_objects [get_fileset sources_1] c:/Users/Chog/Documents/OnLineBuisness/TCL/VivadoTclExample/simpleANDgat
45 launch_runs -jobs 2 sfix_l9_l6_to_float32_synth_1
46 wait_on_run sfix_l9_l6_to_float32_synth_1
```

Capture Vivado IP Tcl commands in Journal file

Approach one: Adding Vivado IP Tcl commands to Non-Project mode script

Before we go further we need to establish the assumed directory structure. A common directory structure is shown below. The source control tool contains the parent folder Root. It also contains the sub-folders: Constraints, RTL, Sim, and TclScripts. The IP folder is not stored in source control but is created by the Tcl script. The IP sub-folder stores the .xci files and all output products of the .xci files.

```
Root
- Constraints
- RTL
- Sim
- TclScripts
- IP
```

Now it's time to put the three Vivado IP Tcl commands into our Tcl scripts. For the purposes of this article, I'm assuming you are using Vivado in [Non-Project](#) mode. If not, check out the tutorial [here](#). In a Non-Project mode Tcl script there is a command called **read_ip**. Typically, this

command comes after reading the source and constraints files. Instead of reading in the .xci file using the **read_ip** command we're going to use a simple if statement to check whether or not the IP folder exists (for tutorials on Tcl if statements and what the brackets are for click [here](#) and [here](#)). The reason is that when the design is first checked out from source control there is no IP folder and no .xci files so we want to create them for the first time. However, once created we don't want to go through the process of recreating the folder and .xci files every time the script is executed.

```
# Check IP
if { [file isdirectory ../"IP"] } {
    # if the IP files exist, we already generated the IP,
    so we can just
    # read the ip definition (.xci)
    read_ip ../IP/sfix_19_16_to_float32.xci
} else {
    # IP folder does not exist. Create IP folder
    file mkdir ../IP

    # create_ip requires that a project is open in memory.
    Create project
    # but don't do anything with it
    create_project -in_memory

    # paste commands from Journal file to recreate IP
    create_ip -name floating_point -vendor xilinx.com -
library ip \
    -version 7.1 -module_name sfix_19_16_to_float32 \
    -dir ../IP
```

```
set_property -dict [list CONFIG.Component_Name
{sfix_19_16_to_float32}\
    CONFIG.Operation_Type {Fixed_to_float}
CONFIG.A_Precision_Type {Custom} \
    CONFIG.C_A_Exponent_Width {19}
CONFIG.C_A_Fraction_Width {16}\
    CONFIG.Maximum_Latency {false}
CONFIG.C_A_Fraction_Width {16}\
    CONFIG.Result_Precision_Type {Single}
CONFIG.C_Result_Exponent_Width {8}
CONFIG.C_Result_Fraction_Width {24}\
    CONFIG.C_Accum_Msb {32} CONFIG.C_Accum_Lsb {-31}\
    CONFIG.C_Accum_Input_Msb {32} CONFIG.C_Mult_Usage
{No_Usage}\
    CONFIG.C_Latency {8} CONFIG.C_Rate {1}] [get_ips
sfix_19_16_to_float32]

generate_target all [get_ips]

# Synthesize all the IP
synth_ip [get_ips]
}
```

For the purposes of this tutorial, I wrapped the commands across multiple lines by using the escape operator, `\`, on the carriage return. This is solely for readability and not necessary. Note: you must change where the IP is stored when using the **create_ip** command. I modified the **-dir** argument to point to the newly created IP folder.

Approach two: Store xci files and check for updates

For the second approach, we will use the IP fixed-to-float conversion IP created earlier. We assume the directory structure below. In this case, the only folder not under source control is BuildFiles.

```
Root
- Constraints
- RTL
- Sim
- TclScripts
- IP
  - sfix_19_16_to_float32.xci
- BuildFiles
  - sfix_19_16_to_float32
    - sfix_19_16_to_float32.xci
```

To make things cleaner we avoid storing output files/build files inside of directories under source control. For that reason, inside of TclScripts we store a script that creates the BuildFiles directory and copies each of the .xci files stored in IP into their own subfolder within BuildFiles. The .xci files are copied into their own subfolders because if a single folder contains all of the .xci files then problems occur generating the IP output products. Vivado stores the IP output files in the same location as the .xci files and existing IP output products may be overwritten.

There are five steps when reading in .xci files for the first time.

1. Read in IP using the **read_ip** command
2. Check to see if the IP is locked and store the result in a Tcl variable
3. Check to see if the IP contains an upgrade and store the result in a Tcl Variable

4. If the IP is locked and an upgrade is an available upgrade the IP
5. Generate the IP output products

The script below shows the proper Vivado IP Tcl commands.

```
#Step 1: Read in IP
read_ip ../IP/sfix_19_16_to_float32/
sfix_19_16_to_float32.xci

#Step 2: Check if IP is locked
set locked [get_property IS_LOCKED [get_ips
sfix_19_16_to_float32]]

#Step 3: Check if upgrade is available
set upgrade [get_property UPGRADE_VERSIONS [get_ips
sfix_19_16_to_float32]]

#Step 4: Upgrade IP if required
if {$locked && $upgrade != ""} {
upgrade_ip [get_ips sfix_19_16_to_float32]}

#Step 5: Produce IP output products
generate_target all [get_ips sfix_19_16_to_float32]
```

For more information check out the Vivado guide [UG939](#).

Conclusion

How do you generate and store your Vivado IP? Let us know in the comments section. All comments are read and appreciated. As a bonus, we've included our Tcl script that creates the BuildFiles directory and then copies your .xci files from your IP folder to the newly created

BuildFiles directory within their own subfolders. Simply click the button below and sign up.