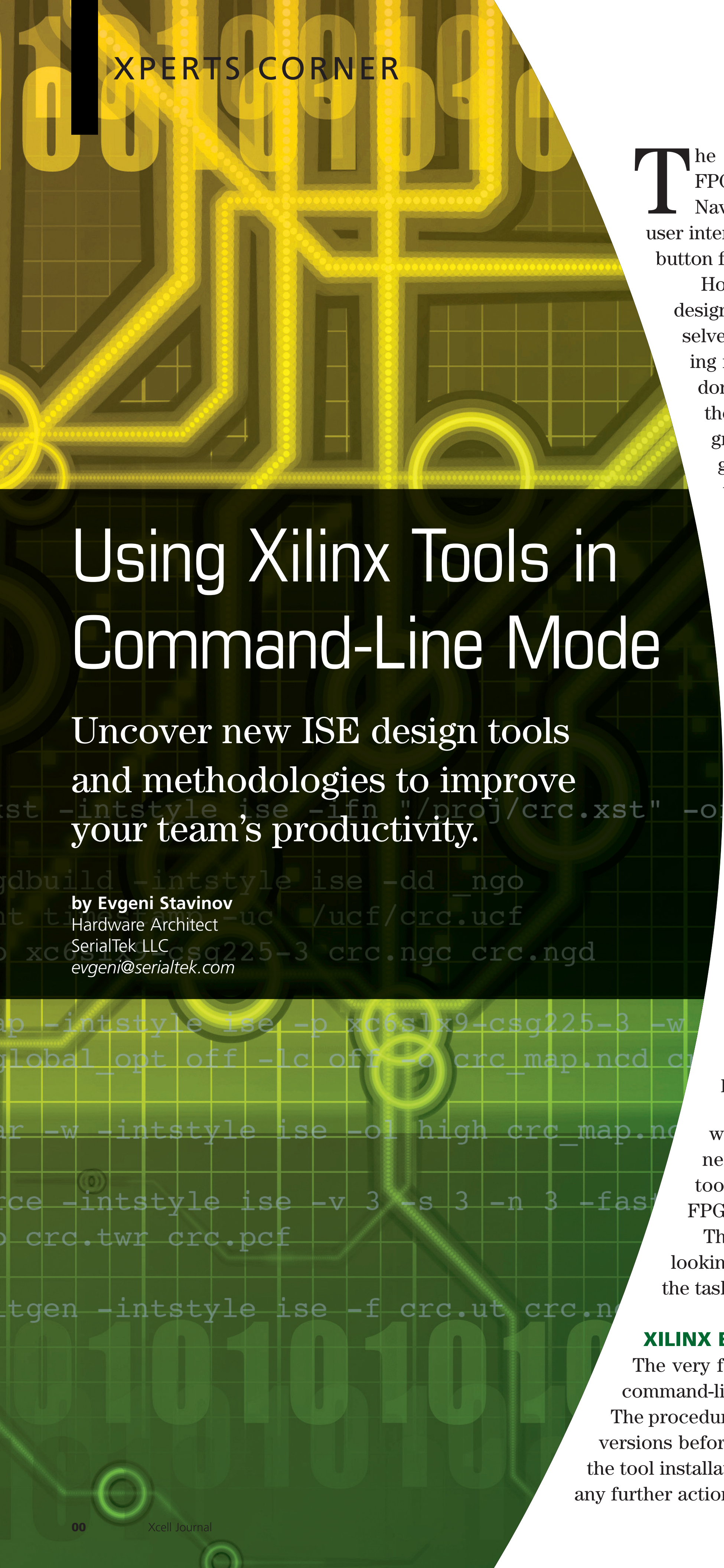

st -intstyle ise -ifn "/proj/crc.xst" -o
gdbuild -intstyle ise -dd _ngo
nt -uc /ucf/crc.ucf
p xc6s csg225-3 crc.ngc crc.ngd
ap -intstyle ise -p xc6slx9-csg225-3 -w
global_opt off -lc off -o crc_map.ncd c
ar -w -intstyle ise -ol high crc_map.nc
ce -intstyle ise -v 3 -s 3 -n 3 -fas
crc.twr crc.pcf
tgen -intstyle ise -f crc.ut crc.n

Using Xilinx Tools in Command-Line Mode

Uncover new ISE design tools and methodologies to improve your team's productivity.

by Evgeni Stavinov
Hardware Architect
SerialTek LLC
evgeni@serialtek.com

The majority of designers working with Xilinx® FPGAs use the primary design tools—ISE® Project Navigator and PlanAhead™ software—in graphical user interface mode. The GUI approach provides a push-button flow, which is convenient for small projects.

However, as FPGAs become larger, so do the designs built around them and the design teams themselves. In many cases GUI tools can become a limiting factor that hinders team productivity. GUI tools don't provide sufficient flexibility and control over the build process, and they don't allow easy integration with other tools. A good example is integration with popular build-management and continuous-integration solutions, such as TeamCity, Hudson CI and CruiseControl, which many design teams use ubiquitously for automated software builds.

Nor do GUI tools provide good support for a distributed computing environment. It can take several hours or even a day to build a large FPGA design. To improve the runtime, users do builds on dedicated servers, typically 64-bit multicore Linux machines that have a lot of memory but in many cases lack a GUI. Third-party job-scheduling solutions exist, such as Platform LSF, to provide flexible build scheduling, load balancing and fine-grained control.

A less obvious reason to avoid GUI tools is memory and CPU resource utilization. ISE Project Navigator and PlanAhead are memory-hungry applications—each GUI tool instance uses more than 100 Mbytes of RAM. On a typical workstation that has 2 Gbytes of memory, that's 5 percent—a substantial amount of a commodity that can be put to a better use.

Finally, many Xilinx users come to FPGAs with an ASIC design background. These engineers are accustomed to using command-line tool flows, and want to use similar flows in their FPGA designs.

These are some of the reasons why designers are looking to switch to command-line mode for some of the tasks in a design cycle.

XILINX ENVIRONMENT VARIABLES

The very first task a user encounters while working with command-line tools is setting up environment variables. The procedure varies depending on the Xilinx ISE version. In versions before 12.x, all environment variables are set during the tool installation. Users can run command-line tools without any further action.

That changed in ISE 12.x. Users now need to set all the environment variables every time before running the tools. The main reason for the change is to allow multiple ISE versions installed on the same machine to coexist by limiting the scope of the environmental variables to the local shell or command-line terminal, depending on the operating system.

There are two ways of setting up environment variables in ISE 12.x. The simpler method is to call a script settings32.{bat,sh,csh} or settings64.{bat,sh,csh}, depending on the platform. The default location on Windows machines is C:\Xilinx\12.1\ISE_DS\; on Linux, it's /opt/Xilinx/12.1/ISE_DS/.

Here is a Linux bash example of calling the script:

```
$ source /opt/Xilinx/12.1/ISE_DS/settings64.sh
```

The other option is to set the environment variables directly, as shown in the following chart:

Windows	`>set XILINX= C:\Xilinx\12.1\ISE_DS\ISE`
	`>set XILINX_DSP=%XILINX%`
	`>set PATH=%XILINX%\bin\nt;%XILINX%\lib\nt;%PATH%`

Linux bash	`$ export XILINX=/opt/Xilinx/12.1/ISE_DS/ISE`
	`$export XILINX_DSP=$XILINX`
	`$ export PATH=${XILINX}/bin/lin64:${XILINX}/sysgen/util:${PATH}`

Linux csh	`% setenv XILINX/opt/Xilinx/12.1/ISE_DS/ISE`
	`% setenv XILINX_DSP    $XILINX`
	`% setenv PATH  ${XILINX}/bin/lin64:${XILINX}/sysgen/util:${PATH}`

Xilinx command-line tools can run on both Windows and Linux operating systems, and can be invoked using a broad range of scripting languages, as Table 1 illustrates. Aside from these choices, other scripts known to run Xilinx tools are Ruby and Python.

MANY CHOICES IN SCRIPTING LANGUAGES

Perl	Perl is a popular scripting language used by a wide range of EDA and other tools. Xilinx ISE installation contains a customized Perl distribution. Users can start the Perl shell by running xilperl command: `$ xilperl –v # display Perl version`
TCL	Tool Command Language (TCL) is a de facto standard scripting language of ASIC and FPGA design tools. TCL is very different syntactically from other scripting languages, and many developers find it difficult to get used to. This might be one of the reasons TCL is less popular than Perl. TCL is good for what it's designed for—namely, writing tool command scripts. TCL is widely available, has excellent documentation and enjoys good community support. Xilinx ISE installation comes with a customized TCL distribution. To start the TCL shell use xtclsh command: `$ xtclsh –v # display TCL version`
Unix bash and csh	There are several Linux and Unix shell flavors. The most popular are bash and csh. Unlike other Unix/Linux shells, csh boasts a C-like scripting language. However, bash scripting language offers more features, such as shell functions, command-line editing, signal traps and process handling. Bash is a default shell in most modern Linux distributions, and is gradually replacing csh.
Windows batch and PowerShell	Windows users have two scripting-language choices: DOS command line and the more flexible PowerShell. An example of the XST invocation from DOS command line is: `>xst -intstyle ise -ifn "crc.xst" –ofn "crc.syr"`

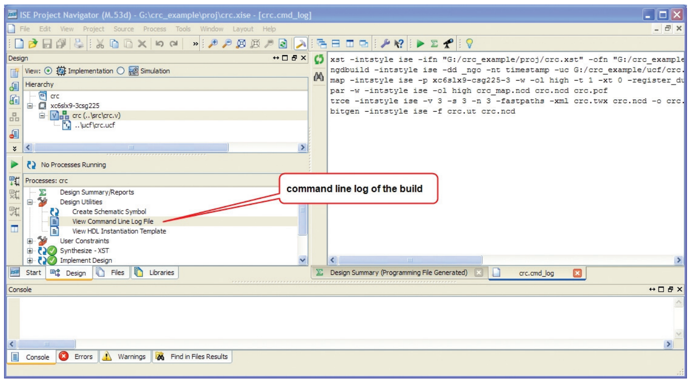
Table 1 – The most popular scripting languages are Perl, TCL, Unix/Linux and Windows batch and PowerShell.

BUILD FLOWS

Xilinx provides several options to build a design using command-line tools. Let's take a closer look at the four most popular ones: direct invocation, xflow, xtclsh and PlanAhead. We'll use a simple CRC generator project as an example.

The direct-invocation method calls tools in the following sequence: XST (or other synthesis tool), NGDBuild, map, place-and-route (PAR), TRACE (optional) and BitGen.

You can autogenerate the sequence script from the ISE Project Navigator by choosing the **Design -> View command line log file**, as shown in Figure 1.



```
xst -intstyle ise -ifn "G:/crc_example/proj/crc.xst" -ofn "G:/crc_example
ngdbuild -intstyle ise -dd _ngo -nt timestamp -uc G:/crc_example/ucf/crc.
map -intstyle ise -p xc6slx9-csg225-3 -w -ol high -t 1 -xt 0 -register_du
par -w -intstyle ise -ol high crc_map.ncd crc.ncd crc.pcf
trce -intstyle ise -v 3 -s 3 -n 3 -fastpaths -xml crc.twx crc.ncd -o crc.
bitgen -intstyle ise -f crc.ut crc.ncd
```

ISE Project Navigator (M.53d) - G:\crc_example\proj\crc.xise - [crc.cmd_log]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
crc
xc6slx9-3csg225
crc (..\src\crc.v)
..\ucf\crc.ucf
No Processes Running
Processes: crc
Design Summary/Reports
Design Utilities
Create Schematic Symbol
View Command Line Log File
View HDL Instantiation Template
User Constraints
Synthesize - XST
Implement Design
command line log of the build
Start
Design
Files
Libraries
Design Summary (Programming File Generated)
crc.cmd_log
Console
Errors
Warnings
Find in Files Results

Figure 1 – Direct tool invocation sequence

The following script is an example of building a CRC generator project:

```
xst -intstyle ise -ifn "/proj/crc.xst" -ofn
"/proj/crc.syr"

ngdbuild -intstyle ise -dd _ngo
-nt timestamp -uc  /ucf/crc.ucf
-p xc6slx9-csg225-3 crc.ngc crc.ngd

map -intstyle ise -p xc6slx9-csg225-3 -w -ol high
-t 1 -xt 0
-global_opt off -lc off -o crc_map.ncd crc.ngd
crc.pcf

par -w -intstyle ise -ol high crc_map.ncd crc.ncd
crc.pcf

trce -intstyle ise -v 3 -s 3 -n 3 -fastpaths -xml
crc.twx crc.ncd
-o crc.twr crc.pcf

bitgen -intstyle ise -f crc.ut crc.ncd
```

EASIER TO USE

Xilinx's XFLOW utility provides another way to build a design. It's more integrated than direct invocation, doesn't require as much tool knowledge and is easier to use. For example, XFLOW does not require exit-code checking to determine a pass/fail condition.

The XFLOW utility accepts a script that describes build options. Depending on those options, XFLOW can run synthesis, implementation, BitGen or a combination of all three.

Synthesis only:

```
xflow -p xc6slx9-csg225-3 -synth synth.opt
../src/crc.v
```

Implementation:

```
xflow -p xc6slx9-csg225-3 -implement impl.opt
../crc.ngc
```

Implementation and BitGen:

```
xflow -p xc6slx9-csg225-3 -implement impl.opt -config
bitgen.opt ../crc.ngc
```

You can generate the XFLOW script (see Figure 2) manually or by using one of the templates located in the ISE installation at the following location: $XILINX\ISE_DS\ISE\xilinx\data\. The latest ISE software doesn't provide an option to autogenerate XFLOW scripts.

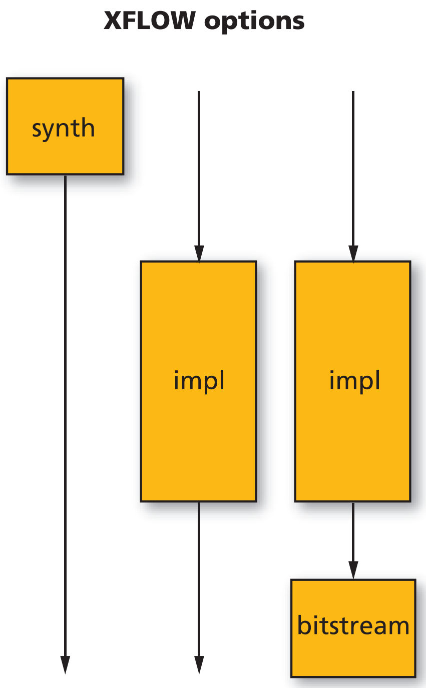


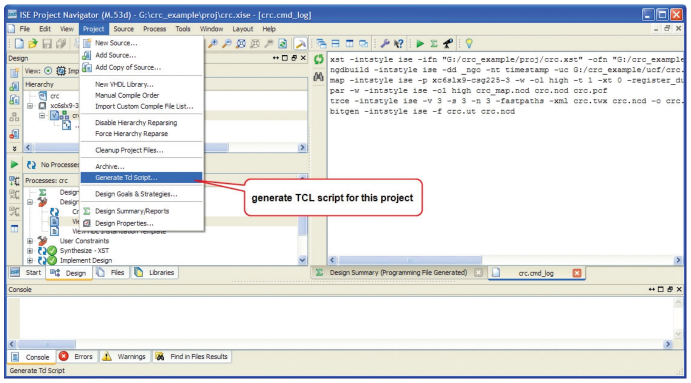
Figure 2 – The XFLOW utility is more integrated than direct invocation.

HOW TO USE XTCLSH

You can also build a design using TCL script invoked from the Xilinx xtclsh, as follows:

```
xtclsh crc.tcl rebuild_project
```

You can either write the TCL script, which passes as a parameter to xtclsh, manually or autogenerate it from the ISE ProjectNavigator by choosing **Project->Generate TCL Script** (see Figure 3).


ISE Project Navigator (M.53d) - G:\crc_example\proj\crc.xise - [crc.cmd_log]
File Edit View Project Source Process Tools Window Layout Help
New Source…
Add Source…
Add Copy of Source…
New VHDL Library…
Manual Compile Order
Import Custom Compile File List…
Disable Hierarchy Reparsing
Force Hierarchy Reparse
Cleanup Project Files…
Archive…
Generate Tcl Script…
Design Goals & Strategies…
Design Summary/Reports
Design Properties…
generate TCL script for this project
Design
Hierarchy
Processes: crc
User Constraints
Synthesize - XST
Implement Design
Start Design Files Libraries
Console Errors Warnings Find in Files Results
Generate Tcl Script

```
xst -intstyle ise -ifn "G:/crc_example/proj/crc.xst" -ofn "G:/crc_example
ngdbuild -intstyle ise -dd _ngo -nt timestamp -uc G:/crc_example/ucf/crc.
map -intstyle ise -p xc6slx9-csg225-3 -w -ol high -t 1 -xt 0 -register_du
par -w -intstyle ise -ol high crc_map.ncd crc.ncd crc.pcf
trce -intstyle ise -v 3 -s 3 -n 3 -fastpaths -xml crc.twx crc.ncd -o crc.
bitgen -intstyle ise -f crc.ut crc.ncd
```

Figure 3 – TCL script generation

Xtclsh is the only build flow that accepts the original ISE project in .xise format as an input. All other flows require projects and file lists, such as .xst and .prj, derived from the original .xise project.

Each of the XST, NGDBuild, map, PAR, TRACE and BitGen tool options has its TCL-equivalent property. For example, the XST command-line equivalent of the Boolean “Add I/O Buffers” is –iobuf, while –fsm_style is the command-line version of “FSM Style” (list).

Each tool is invoked using the TCL “process run” command, as shown in Figure 4.

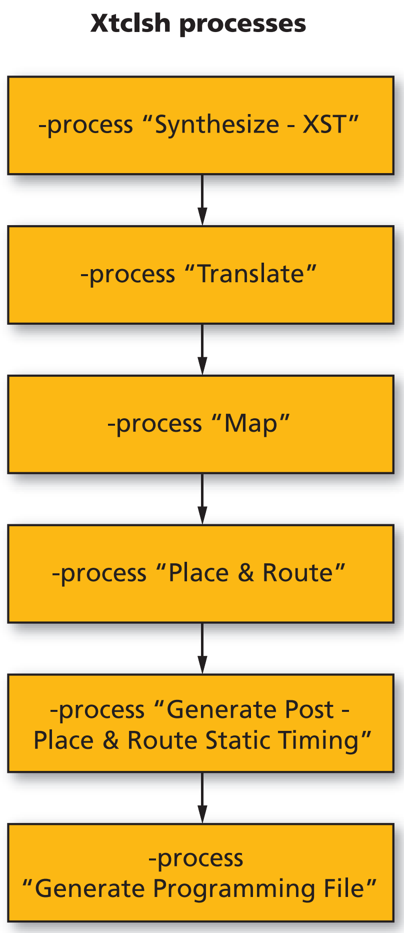

Xtclsh processes
-process “Synthesize - XST”
-process “Translate”
-process “Map”
-process “Place & Route”
-process “Generate Post - Place & Route Static Timing”
-process “Generate Programming File”

Figure 4 – Xtclsh is the only flow that accepts .xise input.

THE PLANAHEAD ADVANTAGE

An increasing number of Xilinx users are migrating from ISE Project Navigator and adopting PlanAhead as a main design tool. PlanAhead offers more build-related features, such as scheduling multiple concurrent builds, along with more flexible build options and project manipulation.

PlanAhead uses TCL as its main scripting language. TCL closely follows Synopsys' SDC semantics for tool-specific commands.

PlanAhead has two command-line modes: interactive shell and batch mode. To enter an interactive shell, type the following command:

```
PlanAhead –mode tcl
```

To run the entire TCL script in batch mode, source the script as shown below:

```
PlanAhead –mode tcl –source <script_name.tcl>
```

The PlanAhead build flow (or run) consists of three steps: synthesis, implementation and bitstream generation, as illustrated in Figure 5.

The PlanAhead software maintains its log of the operations in the *PlanAhead.jou* file. The file is in TCL format, and located in C:\Documents and Settings\<user name>\Application Data\HDI\ on Windows, and ~/.HDI/ on Linux machines.

Users can run the build in GUI mode first, and then copy some of the log commands in their command-line build scripts.

Below is a PlanAhead TCL script used to build an example CRC project.

```
create_project pa_proj {crc_example/pa_proj} -part
xc6slx9csg225-3
```

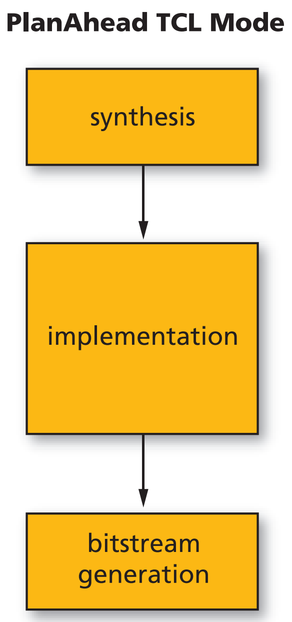


Figure 5 – PlanAhead offers many build-related features.

```
set_property design_mode RTL
  [get_property srcset
  [current_run]]

add_files -norecurse
  {crc_example/proj/.. /src/crc.v}

set_property library work
  [get_files -of_objects
  [get_property srcset [current_run]]
  {src/crc.v}]

add_files -fileset [get_property
  constrset [current_run]]
  -norecurse {ucf/crc.ucf}

set_property top crc [get_property
  srcset [current_run]]
  set_property verilog_2001 true
  [get_property srcset
  [current_run]]

launch_runs -runs synth_1 -jobs 1
  -scripts_only -dir
  {crc_example/pa_proj/pa_proj.runs}

launch_runs -runs synth_1 -jobs 1

launch_runs -runs impl_1 -jobs 1

set_property add_step Bitgen [get_runs impl_1]

launch_runs -runs impl_1 -jobs 1 -dir
  {crc_example/pa_proj/pa_proj.runs}
```

In addition to providing standard TCL scripting capabilities, PlanAhead offers other powerful features. It allows query and manipulation of the design database, settings and states, all from a TCL script. This simple script illustrates some of the advanced PlanAhead features that you can use in a command-line mode:

```
set my_port [get_ports rst]
report_property $my_port
get_property PULLUP $my_port
```

THE XILINX FPGA BUILD PROCESS

Before delving into the details of command-line flows, it's useful to briefly review the Xilinx FPGA build process (see figure), or the sequence of steps involved in building an FPGA design, from RTL to bitstream.

Xilinx provides two GUI tools with which to do a build: ISE Project Navigator and PlanAhead. There are several third-party tools that can do Xilinx FPGA builds as well, for example, Synopsys' Synplify product.

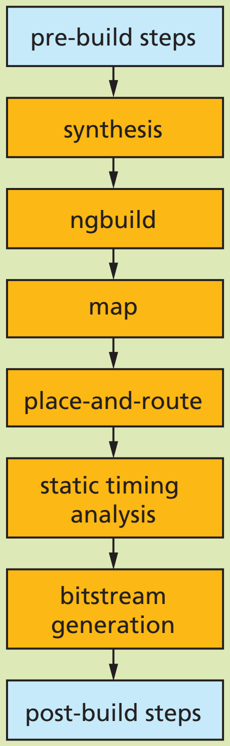
The exact build sequence will differ depending on the tool used. However, any Xilinx FPGA build will contain eight fundamental steps: pre-build, synthesis, NGDBuild, map, place-and-route, static timing analysis, BitGen and post-build.

The main pre-build tasks are getting the latest code from a source control repository, assembling project file lists, setting all tool environment variables, acquiring tool licenses, incrementing a build number and preprocessing the RTL.

During the synthesis stage, designers may use the Xilinx XST tool or a third-party offering. The major third-party synthesis tools for general-purpose FPGA design are Synopsys' Synplify and Mentor Graphics' Precision.

The NGDBuild phase involves netlist translation using the Xilinx NGDBuild tool, while the next step, map, involves mapping the netlist into FPGA-specific resources, such as slices, RAMs and I/Os, using the Xilinx MAP tool. Xilinx's PAR tool handles placement and routing, and the TRACE tool performs static timing analysis. Finally, the BitGen stage is the point in the design cycle at which FPGA bitstreams are generated.

Wrapping up the cycle, the post-build period involves tasks such as parsing the build reports, processing and archiving build results and sending e-mail notifications to users that own the build. – *Evgeni Stavinov*



The Xilinx FPGA build process involves a number of sequenced steps.

```
set_property PULLUP 1 $my_port
get_property PULLUP $my_port
```

The simple script adds a pull-up resistor to one of the design ports. Of course, you can do the same by changing UCF constraints and rebuilding the project, or by using the FPGA Editor. But PlanAhead can do it with just three lines of code.

Finally, many developers prefer using the *make* utility when doing FPGA builds. The two most important advantages of *make* are built-in checking of the return code and automatic dependency tracking. Any of the flows described above can work with *make*.

LESSER-KNOWN COMMAND-LINE TOOLS

The Xilinx ISE installation contains several lesser-known command-line tools that FPGA designers might find useful. ISE Project Navigator and PlanAhead invoke these tools behind the scenes as part of the build flow. The tools are either poorly documented or not documented at all, which limits their exposure to users.

Ten of these tools especially are worth investigating.

- **data2mem:** This utility, which is used to initialize the contents of a BRAM, doesn't require rerunning Xilinx implementation. It inserts the new BRAM data directly into a bitstream. Another data2mem usage is to split the initialization data of a large RAM consisting of several BRAM primitives into individual BRAMs.

- **fpga_edline:** A command-line version of the FPGA Editor, fpga_edline can be useful for applying changes to the post-PAR .ngc file from a script as part of the build process. Some of the use cases include adding ChipScope™ probes, minor routing changes, changing LUT or IOB properties.

- **mem_edit:** This is not a tool per se, but a script that opens Java applications. You can use it for simple memory content editing.

- **netgen:** Here is a tool you can use in many situations. It gets a Xilinx design file in .ncd format as an input, and produces a Xilinx-independent netlist for use in simulation, equivalence checking and static timing analysis, depending on the command-line options.

- **ngcbuild:** This is a tool that consolidates several design netlist files into one. It's mainly used for the convenience of working with a single design file rather than multiple designs and IP cores scattered around different directories.

- **obngc:** This is a utility that obfuscates .ngc files in order to hide confidential design features, and prevent design analysis and reverse-engineering. You can explore NGC files in FPGA Editor.

- **pin2ucf:** You can use this utility to generate pin-locking constraints from an NCD file in UCF format.

- **XDL:** An essential tool for power users, XDL has three fundamental modes: report device resource information, convert NCD to XDL and convert XDL to NCD. XDL is a text format that third-party tools can process. This is the only way to access post-place-and-route designs.

- **xreport:** This is a utility that lets you view build reports outside of the ISE Project Navigator software. It has table views of design summary, resource usage, hyperlinked warnings and errors, and message filtering.

- **xinfo:** A utility that reports system information, xinfo also details installed Xilinx software and IP cores, licenses, ISE preferences, relevant environment variables and other useful information.

The tools are installed in $XILINX>/ISE/bin/{nt, nt64, lin, lin64} and $XILINX>/common/bin/{nt, nt64, lin, lin64} directories. More inquisitive readers might want to further explore those directories to discover other interesting tools that can boost design productivity.

ASSESSING THE OPTIONS

In this article we've discussed advantages of using Xilinx tools in command-line mode, explored several Xilinx build flows, examined different scripting languages and taken a closer look at lesser-known tools and utilities. The goal was not to identify the best build flow or script language, but rather to discuss the available options and parse the unique advantages and drawbacks of each to help Xilinx FPGA designers make an informed decision.

You can base your selection process on a number of criteria, such as existing project settings, design team experience and familiarity with the tools and scripts, ease of use, flexibility, expected level of customization and integration with other tools, among others. Users can find all scripts and example projects used in this article on the author's website: *http://outputlogic.com/xcell_using_xilinx_tools/*.

Evgeni Stavinov is a longtime Xilinx user with more than 10 years of diverse design experience. Before becoming a hardware architect at SerialTek LLC, he held different engineering positions at Xilinx, LeCroy and CATC. Evgeni is a creator of OutputLogic.com, an online productivity-tools portal.