Unleashing the Power of the Command-Line Interface

Jeremy W. Webb
Senior Design Engineer

Centellax, Inc.
Santa Rosa, CA USA

www.centellax.com

ABSTRACT

The development of complex ASIC or FPGA designs involving multiple teams and loosely integrated tools is an arduous process. There is an inherent challenge in maintaining coherency and separation of source and generated files throughout the build process, particularly through different tool versions and vendors. These aspects of the development process make results hard to reproduce, reuse, and share. This paper highlights the benefits of a command-line-based build environment as an alternative to using graphical user interfaces (GUIs) for RTL development. A well-reasoned directory structure for projects is proposed, as well as a template for command-line integration of ASIC or FPGA development tools.

Table of Contents

Table of Figures

Table of Listings

1. Introduction

Developing complex ASIC or FPGA designs involving multiple teams and loosely integrated tools is a complex endeavor. While each tool used in the ASIC or FPGA build process typically has an integrated development environment (IDE) intended to tie front-end and back-end tools together, they can be difficult to set up. Back-end tools attempt to integrate the front-end tools into their flow, and vice-versa. Fortunately, these tools provide a method of controlling the flow using a command-line interface (CLI). Using custom Makefiles and scripts in a well-reasoned directory structure allows for the designer to leverage the strengths of each tool via the CLI used in the build process. Employing a hierarchical RTL design can further improve the design efficiency and fosters a team design flow. Listing 1.1 shows an example synthesis flow initiated from the command-line interface using a Makefile.

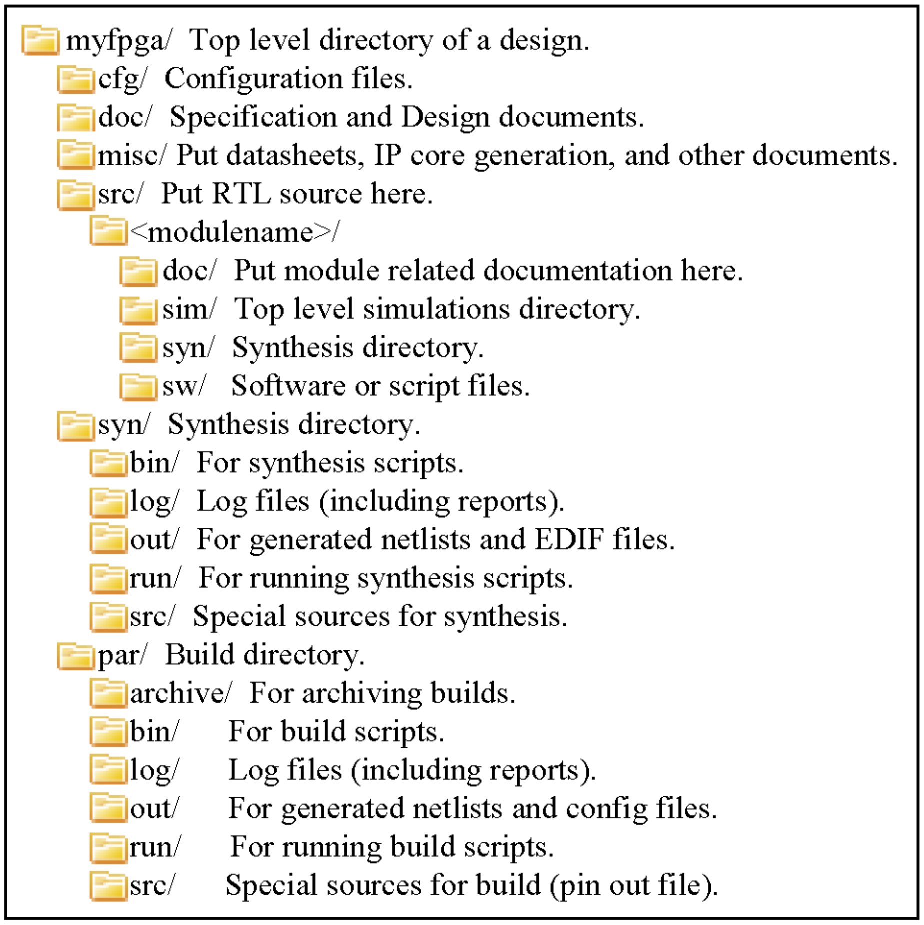
Listing 1.1 Example Synthesis Command-Line Flow

```
[jwwebb@darthbane ~]
$ cd ~/snug/git/myfpga/par/bin/
[jwwebb@darthbane ../git/myfpga/par/bin]
$ make setup
Executing: make setup
 [jwwebb@darthbane ../git/myfpga/par/bin]
$ make synthesize
Executing: make synthesize

***********************************************************************
   Launch Synplify Pro
***********************************************************************
../../src/myfpga/myfpga.sv ../../src/in_buf/in_buf.sv ../../src/out_buf/out_buf.sv ../../src/reg_if/reg_if.sv ../../src/sys_rst/sys_rst.sv
../../src/sync_2stage/sync_2stage.sv ../../src/sync_rst/sync_rst.sv ../../src/sys_dcm/sys_dcm.sv  ../../src/adc_ctrl/adc_ctrl.sv
../../src/mem_block/mem_block.sv ../../src/ad7928_adc/ad7928_adc.sv ../run/reg_defines.h
***********************************************************************
../bin/outarch.sh myfpga ../log ../out ../run
***************************************************************
* myfpga.edf and myfpga.ncf do not exist.
***************************************************************
* Clean up log directories...
***************************************************************
synplify_pro -batch ../bin/myfpga.tcl
***************************************************************
Loading ../bin/synhooks.tcl
Starting:    /opt/synopsys/fpga_f201109/linux_a_64/mbin/synbatch
Version:     F-2011.09
Arguments:   -product synplify_pro -batch ../bin/myfpga.tcl
ProductType: synplify_pro

Running proj_1|log
Job flow Compile Process completed on proj_1|log
Running Premap on proj_1|log
Job flow Compile completed on proj_1|log
Running Map on proj_1|log
Job flow Map completed on proj_1|log
Running Place and Route Post-Synthesis on proj_1|log
Job flow Logic Synthesis completed on proj_1|log
Job flow proj_1|rev_1 completed on proj_1|rev_1

TCL script complete: "../bin/myfpga.tcl"

exit status=0

***********************************************************************
   Synplify Pro completed.
***********************************************************************
```

2. Design Hierarchy

Whether creating a new design or leveraging an existing one, maintaining coherency and separation of source and generated files in a flat or semi-hierarchical directory structure can be difficult. Employing a hierarchical directory structure with a clearly defined location for source files, simulation projects, synthesis builds, place and route builds, and other miscellaneous files can improve the efficiency of the design process. Figure 1 shows a recommended directory hierarchy for either an FPGA or ASIC RTL design. The creation of the directory hierarchy can be automated with Perl or bash shell scripts.

Figure 1 Hierarchal Directory Structure



```
myfpga/  Top level directory of a design.
    cfg/  Configuration files.
    doc/  Specification and Design documents.
    misc/ Put datasheets, IP core generation, and other documents.
    src/  Put RTL source here.
        <modulename>/
            doc/  Put module related documentation here.
            sim/  Top level simulations directory.
            syn/  Synthesis directory.
            sw/   Software or script files.
    syn/  Synthesis directory.
        bin/  For synthesis scripts.
        log/  Log files (including reports).
        out/  For generated netlists and EDIF files.
        run/  For running synthesis scripts.
        src/  Special sources for synthesis.
    par/  Build directory.
        archive/  For archiving builds.
        bin/      For build scripts.
        log/      Log files (including reports).
        out/      For generated netlists and config files.
        run/      For running build scripts.
        src/      Special sources for build (pin out file).
```

2.1. Configuration Directory

The configuration (cfg) directory is intended to store the FPGA configuration files generated by the back-end FPGA place and route tools. For example, a printed circuit (PC) board design employing a Xilinx FPGA and a serial peripheral interface (SPI) electrically erasable programmable read-only memory (EEPROM) would require both a bit file and an mcs file to configure each device. These files would be stored in the cfg directory.

2.2. Documentation Directory

The documentation (doc) directory is intended to store design specification documents, block diagrams, control scripts, and other important design documents.

2.3. Miscellaneous Directory

The miscellaneous (misc) directory is intended to store IP core generation projects, peripheral manufacturer's data sheets, and back-end planning tool projects. For example, when defining constraints for a Xilinx FPGA design, a PlanAhead™ project can be created and stored in the misc directory.

2.4. RTL Source Directory

The RTL (src) directory is intended to store all of the design source files. Ideally, each RTL module or sub-circuit would reside in its own directory. A typical module directory contains the following:

- doc - module specific documentation
- sim - module simulation project
- sw - module software or scripts
- syn - module synthesis project

In the case of simple RTL modules, the sub-directories within the module directory can be omitted. An RTL module containing only I/O buffer instantiations, for example, is simple enough to not require documentation, simulation, software or synthesis sub-directories.

Module Document Directory

The module documentation (doc) directory is intended to store RTL module specific documentation. The data sheet or user guide of a module generated using an FPGA IP core generation tool could be stored in the doc directory.

Module Simulation Directory

The module simulation (sim) directory is intended to store the RTL and gate level simulation projects. ASIC designs will contain both an RTL and gate-level simulation directory, whereas an FPGA design will typically only contain a gate-level simulation directory for the top-level module. The FPGA design equivalent of an ASIC gate-level simulation would be a post-place and route simulation, which uses a top-level RTL module generated by the FPGA back-end tools.

Module Software Directory

The module software (sw) directory is intended to store software files, whether they are Perl scripts, Bash scripts, Matlab programs, or C programs that aid in the design and verification of the RTL module.

Module Synthesis Directory

The module synthesis (syn) directory is intended to store a synthesis project for evaluating the gate count and power performance of an RTL module. In addition, it can provide a means of evaluating the equivalent gate-level schematic of the RTL module. For example, Synopsys FPGA can be used to synthesize an RTL module and evaluate both the gate level and FPGA device specific schematics.

2.5. Synthesis Directory

The top-level synthesis (syn) directory is intended to store a synthesis project for generating an electronic design interchange format (EDIF) net list file for the RTL design, which can be used by either the ASIC or FPGA back-end tools. Additionally, it can provide a means of evaluating the equivalent gate-level schematic of the RTL design. The synthesis directory contains the following:

- bin - synthesis scripts
- log - log and report files
- out - generated net lists and EDIF files
- run - synthesis project directory
- src - synthesis constraint files

2.6. Place and Route Directory

The top-level place and route (par) directory is intended to store an ASIC or FPGA back-end project. In the case of FPGA designs, this directory would store a Makefile driven tool flow for translating the EDIF net list file, mapping, place and route, and generating configuration files. The synthesis directory contains the following:

- archive - archives of past builds
- bin - Makefiles and scripts
- log - log and report files
- out - generated configuration files
- run - par build directory
- src - par constraint files

3. Synthesis Flow

The Synopsys advanced FPGA synthesis tools, Synplify Pro™ and Synplify Premier™, provide a Tcl scripting interface which allows the tools to be controlled via the command-line interface. The synthesis build is facilitated by three scripts which are stored in the ../myfpga/syn/bin/ directory:

- Makefile
- parsetcl.sh
- outarch.sh

The synthesis build is initiated by executing the "synthesize" target of the Makefile shown in Listing 3.2. Typically, Make is provided with a list of source files which are used during the build flow. Maintaining the list of source files in both the Makefile and the project Tcl file can be troublesome and prone to error. This problem can be solved by using a bash shell script to automate the file list generation. An example bash shell script called parsetcl.sh, shown in Listing 3.1, is used by the Makefile to determine which source files in the directory hierarchy have changed and when a synthesis build can be performed. The parsetcl.sh bash shell script searches the project Tcl file (myfpga.tcl shown in Listing 3.3) for the add_file keyword, and concatenates the source file paths into a continuous string which is assigned to the SRCS variable in the Makefile. The first step of the synthesis build is to execute the outarch.sh bash shell script, which archives and cleans the log, out, and run directories. The next step is to launch the synthesis tool in batch mode using the project Tcl file.

The actual synthesis takes place within the ../myfpga/syn/run/ directory, and the report and log files are stored in the ../myfpga/syn/log/ directory. Upon completion of synthesis, the EDIF and NCF files are stored in both the ../myfpga/syn/out/ and ../myfpga/par/run/ directories using callback functions stored in the ../myfpga/syn/bin/synhooks.tcl file. In order to use callback functions in a synthesis flow, the SYN_TCL_HOOKS environment variable must be assigned the name and location of the synhooks.tcl file.

In addition to the project Tcl file, a constraints file can be used in the batch synthesis flow to provide a variety of design related constraints, including clock, I/O delay, register, attributes, and

I/O standards. The design constraint file can be created either manually in a text editor, or by using the synthesis constraint optimization environment (SCOPE) tool. When used with the RTL view of the HDL Analyst tool, the SCOPE tool allows specific nets to be dragged and dropped into a spreadsheet-like interface. When starting a new design, the constraint file can be omitted from the project Tcl file during the first synthesis build; the constraint file can be created after the build is completed by opening the Synplify GUI from within the ../myfpga/syn/run/ directory and using the SCOPE and HDL Analyst tools as described above. Once the constraints file has been created, it can be added to the project Tcl file as shown in Listing 3.3.

Listing 3.1 Synthesis Tcl File Parser Script

```
#!/bin/sh

#*********************************************************************
#
# parsetcl.sh module
#
#*********************************************************************
# Source/Include File TCL Parser
#
#  This utility is intended to parse a Synplify Pro TCL file and extract the source file
#  names with file paths for use by the Makefile. This will allow the Makefile to only
#  re-generate the EDIF if the source files have changed.
#
#*********************************************************************
PROJECTNAME=$1

# Copy the Synplify Pro project TCL file:
cp $PROJECTNAME.tcl foo

# Extract the source files from the Synplify Pro project TCL file:
grep "add_file -verilog" foo > tmp
grep "add_file -_include" foo >> tmp

# Remove all but source file paths.
sed -e "s/^#.*$//" \
    -e "s/\"$//" \
    -e "s/^.*unisim.v$//" \
    -e "/^$/d" \
    -e "s/add_file -verilog.*\"//" \
    -e "s/add_file -_include \"//" tmp > bar

cat -s bar
rm -f foo tmp bar
```

Listing 3.2 Synthesis Top-Level Makefile

```
#************************************************************************
#
# Synthesis Makefile
#
#************************************************************************
# <Design Name>
#
# This is the top-level synthesis Makefile, which synthesizes the RTL source code for
# the FPGA design. This Makefile references the RTL source directory for extracting the
# files during synthesis. This Makefile must be run from within the bin directory: "./bin".
# All results from the build are stored in the out directory.
#
#************************************************************************

# Project Name:
PROJNAME  := myfpga

# Directory Variables:
PROJDIR   := ..
SRCDIR    := $(PROJDIR)/src
BINDIR    := $(PROJDIR)/bin
LOGDIR    := $(PROJDIR)/log
OUTDIR    := $(PROJDIR)/out
RUNDIR    := $(PROJDIR)/run
CODEDIR   := ../../src

# Synplify Pro Variables:
SYNPLIFY  := synplify_pro
TCLFILE   := $(BINDIR)/$(PROJNAME).tcl
SYNTHFILE := $(OUTDIR)/$(PROJNAME).edf
NCFFILE   := $(OUTDIR)/$(PROJNAME).ncf

# Source Code:
SRCS      :=$(shell $(BINDIR)/parsetcl.sh $(PROJNAME))
INDEX=$(SRCS)

# Environment Variables:
export SYN_TCL_HOOKS=../bin/synhooks.tcl

default:
    @echo "** Synthesis ******************************************************"
    @echo "targets:"
    @echo "  make synthesize      - synthesize chip"
    @echo "  make clean           - clean current build folder"
    @echo "*********************************************************************"

synthesize : $(SYNTHFILE)

$(SYNTHFILE) : $(INDEX)
    @echo "*********************************************************************"
    @echo "    Launch Synplify Pro"
    @echo "*********************************************************************"
    @echo "$(INDEX)"
    @echo "*********************************************************************"
    $(BINDIR)/outarch.sh $(PROJNAME) $(LOGDIR) $(OUTDIR) $(RUNDIR)
    $(SYNPLIFY) -batch $(TCLFILE)
    @echo "*********************************************************************"
    @echo "    Synplify Pro completed."
    @echo "*********************************************************************"

clean :
    @echo "*********************************************************************"
    @echo "    Clean up synthesis directories"
    @echo "*********************************************************************"
    $(BINDIR)/outarch.sh $(PROJNAME) $(LOGDIR) $(OUTDIR) $(RUNDIR)
```

Listing 3.3 Synthesis Tcl File

```
#******************************************************************
#
# myfpga.tcl module
#
#******************************************************************
# <Design Name>
#
# This TCL file sets up SynplifyPro, and synthesizes the FPGA design. This script
# generates an EDIF file named: myfpga.edf.
#
#******************************************************************
project –new
set TECHNOLOGY “Virtex6”
set PART “XC6VLX195T”
set PKG “FF784”

#******************************************************************
#** Xilinx Library
#******************************************************************
add_file -verilog "$LIB/xilinx/unisim.v"
#******************************************************************
#** Top-Level Module
#******************************************************************
add_file -verilog "../../src/myfpga/myfpga.sv"
#******************************************************************
#** Sub-Module(s)
#******************************************************************
#******************************************************************
#** Input/Output Buffers
#******************************************************************
add_file -verilog "../../src/in_buf/in_buf.sv"
add_file -verilog "../../src/out_buf/out_buf.sv"
#******************************************************************
#** Constraints
#******************************************************************
add_file -constraint "../src/myfpga.sdc"

#implementation: "rev_1"
impl -add "../log"

#implementation attributes
set_option -vlog_std sysv
set_option -project_relative_includes 1

#par_1 attributes
set_option -job par_1 -add par
set_option -job par_1 -option run_backannotation 0
impl -active "log"

#device options
set_option -technology $TECHNOLOGY
set_option -part $PART
set_option -package $PKG
set_option -speed_grade -2
set_option -part_companion ""

#compilation/mapping options
set_option -use_fsm_explorer 0
set_option -top_module "myfpga"

# mapper_options
set_option -frequency auto
set_option -write_verilog 0
set_option -write_vhdl 0
```

```
# Xilinx Virtex6
set_option -run_prop_extract 1
set_option -maxfan 10000
set_option -disable_io_insertion 0
set_option -pipe 1
set_option -update_models_cp 0
set_option -retiming 1
set_option -no_sequential_opt 0
set_option -fixgatedclocks 3
set_option -fixgeneratedclocks 3
set_option -enable_prepacking 1

# NFilter
set_option -popfeed 0
set_option -constprop 0
set_option -createhierarchy 0

# sequential_optimization_options
set_option -symbolic_fsm_compiler 1

# Compiler Options
set_option -compiler_compatible 0
set_option -resource_sharing 0

#VIF options
set_option -write_vif 1

#automatic place and route (vendor) options
set_option -write_apr_constraint 1

#set result format/file last
project -result_file "myfpga.edf"
project -log_file "../log/myfpga.srr"
project -save "../run/myfpga.prj"
project -run
```

4. FPGA Implementation Flow

The Xilinx ISE Design Suite™ is an integrated development environment which is made up of multiple command-line programs. The FPGA implementation flow typically consists of the following steps:

- Netlist Translate
- Mapping
- Place and route
- Timing Analysis
- Configuration File Generation

Each of these programs generate a considerable number of files. The designer must then sift through the files in the design directory to determine which files are report files and which are output files required by the next stage in the FPGA implementation flow. This process can be simplified by using Makefiles which automatically place report and log files in a log directory and leave intermediate files in the run directory as the design progresses through the implementation process. The place and route build is facilitated by three scripts which are stored in the ../myfpga/par/bin/ directory:

- Makefile
- par.xilinx.mk
- outarch.sh

The place and route build for the Xilinx FPGA design "myfpga" is initiated by executing the "setup" and "all" targets of the Makefile, shown in Listing 4.1, located in the ../myfpga/par/bin/ directory. The "setup" Make target copies the par.xilinx.mk file, shown in Listing 4.2, to the ../myfpga/par/run/ directory and renames the file to Makefile. Other than the "archive" Make target, all remaining Make targets indirectly execute targets of the same name in the Makefile located in the ../myfpga/par/run directory. The "synthesize" target of the par.xilinx.mk Makefile indirectly executes the "synthesize" target of the Makefile stored in the ../myfpga/syn/bin/ directory. Upon completion of the synthesis flow, the EDIF and NCF files are copied into the ../myfpga/par/run/ directory, and the remaining Make targets operate on these files. The role of the par.xilinx.mk Makefile is to organize the files generated during the place and route build flow. After each Make target is completed, the associated log and output files are stored in the directories defined in Section 2.6. Using the set of Makefiles shown in Listing 4.1 and Listing 4.2, the steps of the place and route build flow can either be automated or executed one at a time, as necessary. For example, when using the Xilinx ChipScope Pro Inserter™, the place and route build flow must be continued from the netlist translate stage.

In addition to the script files, a user constraints file (UCF) must be used in the place and route build flow in order to achieve the desired performance and functionality. The UCF file contains a variety of design related constraints, including clock, I/O delay, pin assignments, and I/O standards. The design constraint file can be created either manually in a text editor, or by using the Xilinx PlanAhead™ tool.

Listing 4.1 Place and Route Top-Level Makefile

```
#*************************************************************************
#
# Place and Route FPGA Makefile
#
#*************************************************************************
# <Design Name>
#
# This is the top-level place and route Makefile, which builds the FPGA design using
# back-end tools. This Makefile must be run from within the bin directory: "../par/bin".
# All results from the build are stored in the log, out, and run directories.  This script is
# dedicated to the Xilinx ISE toolset.
#
#*************************************************************************

# Project Name:
PROJNAME  := myfpga

# Directory Variables:
PROJDIR   := ..
BINDIR    := $(PROJDIR)/bin
LOGDIR    := $(PROJDIR)/log
OUTDIR    := $(PROJDIR)/out
RUNDIR    := $(PROJDIR)/run
SRCDIR    := $(PROJDIR)/src
ARCHDIR   := $(PROJDIR)/archive

default:
	@echo "** Place and Route Build *******************************************"
	@echo "targets:"
	@echo "  make archive         - archive current build"
	@echo "  make synthesize      - synthesize chip"
	@echo "  make setup           - setup build"
	@echo "  make translate       - translate chip"
	@echo "  make map             - map chip"
	@echo "  make par             - par chip"
	@echo "  make bit             - generate bit file"
	@echo "  make prom            - generate prom file"
	@echo "  make trace           - run timing analyzer"
	@echo "  make sdf             - generate post place & route files"
	@echo "  make download        - program entire JTAG chain."
	@echo "  make all             - run all make targets"
	@echo "  make clean           - clean current build folder"
	@echo "*******************************************************************"

archive :
	./outarch.sh ${PROJNAME} ${LOGDIR} ${OUTDIR} ${RUNDIR} ${SRCDIR} ${ARCHDIR}

setup :
	@echo "Executing: make setup"
	cp par.xilinx.mk ../run/Makefile
	chmod 775 ../run/Makefile

synthesize :
	@echo "Executing: make synthesize"
	cd ../run; make synthesize

translate :
	@echo "Executing: make translate"
	cd ../run; make translate

map :
	@echo "Executing: make map"
	cd ../run; make map
par :
	@echo "Executing: make par"
	cd ../run; make par

bit :
	@echo "Executing: make bit"
	cd ../run; make bit

prom :
	@echo "Executing: make prom"
	cd ../run; make prom

trace :
	@echo "Executing: make trace"
	cd ../run; make trace

sdf :
	@echo "Executing: make sdf"
	cd ../run; make sdf

download :
	@echo "Executing: make download"
	cd ../run; make download

all :
	@echo "Executing: make all"
	cd ../run; make all

clean :
	@echo "Executing: make clean"
	cd ../run; make clean
```

Listing 4.2 Place and Route Xilinx Makefile

```
#***********************************************************************
#
# Xilinx Place and Route FPGA Makefile
#
#***********************************************************************
# <Design Name>
#
# This is the Xilinx place and route Makefile. The synthesis build is executed in a different
# level of the design hierarchy. The resulting EDIF files are copied to the ../par/run/ directory
# and used with the Xilinx back-end tools. This Makefile is copied from the ../par/bin/ directory
# to the ../par/run/ directory and executed from a top-level place and route Makefile. All results
# from the build are stored in the log, out, and run directories.  This script is dedicated to the
# Xilinx ISE toolset.
#
#***********************************************************************

# Project Name:
PROJNAME  := myfpga
CHIP := XC6VLX195T-FF784-2

# Directory Variables:
PROJDIR   := ..
COMMONDIR := ../../../common/bin
SRCDIR    := $(PROJDIR)/src
BINDIR    := $(PROJDIR)/bin
LOGDIR    := $(PROJDIR)/log
OUTDIR    := $(PROJDIR)/out
RUNDIR    := $(PROJDIR)/run
SYNDIR    := $(PROJDIR)/../syn
CODEDIR   := ../../src
CFGDIR    := ../../cfg

# Translate File Variables:
UCF       := $(SRCDIR)/$(PROJNAME).ucf
SYNTHFILE := $(RUNDIR)/$(PROJNAME).edf
TRANOPTS  := -p $(CHIP) -intstyle silent -uc $(UCF) -sd "<path_to_edif_or_ngd_files>/" -sd "../run/" $(SYNTHFILE) $(PROJNAME).ngd

# MAP File Variables:
```

```
MAPOPTS   := -p $(CHIP) –w -logic_opt off -ol high -t 1 -xt 0 -register_duplication off -r 4 -global_opt off -mt 2 –detail \
-ir off -pr b -lc off -power off -o map.ncd

# PAR File Variables:
PAROPTS   := -w -ol high -mt 2

# Bitgen File Variables:
BITOPTS   := -w -m -g DebugBitstream:No -g Binary:no -g CRC:Enable -g ConfigRate:2 -g CclkPin:PullUp -g M0Pin:PullNone \
-g M1Pin:PullNone -g M2Pin:PullNone -g ProgPin:PullUp -g InitPin:Pullnone -g CsPin:Pullnone -g DinPin:Pullnone -g BusyPin:Pullnone \
-g RdWrPin:Pullnone -g HswapenPin:PullUp -g TckPin:PullNone -g TdiPin:PullNone -g TdoPin:PullNone -g TmsPin:PullNone \
-g Disable_JTAG:No -g UnusedPin:PullNone -g UserID:0xDEADBEEF -g ConfigFallback:Enable -g BPI_page_size:1 \
-g OverTempPowerDown:Disable -g next_config_addr:None -g JTAG_SysMon:Enable -g DCIUpdateMode:Quiet -g StartUpClk:CClk \
-g DONE_cycle:4 -g GTS_cycle:5 -g GWE_cycle:6 -g Match_cycle:NoWait -g Security:Level2 -g DonePipe:No -g DriveDone:Yes \
-g Encrypt:No

# PromGen File Variables:
PROMOPTS  := -u 0x0 $(PROJNAME).bit -p mcs -o $(PROJNAME).mcs -s 16384 -spi -w

# Trace Variables:
TRACEOPTS := -e 3 -l 3 $(PROJNAME).ncd $(PROJNAME).pcf -xml $(LOGDIR)/$(PROJNAME) -o $(PROJNAME).twr

# SDF Variables:
SDFOPTS   := -w -ofmt verilog -aka -fn -pcf $(PROJNAME).pcf -s 2 -sim -tb -ism -ne -sdf_anno true $(PROJNAME).ncd

# Environment Variables:
export SYN_TCL_HOOKS=../bin/synhooks.tcl
export XIL_PAR_DESIGN_CHECK_VERBOSE=1

default:
    @echo "** Xilinx Place and Route Build ************************************"
    @echo "targets:"
    @echo "  make synthesize   - synthesize chip"
    @echo "  make translate    - translate chip"
    @echo "  make map          - map chip"
    @echo "  make par          - par chip"
    @echo "  make bit          - generate bit file"
    @echo "  make prom         - generate prom file"
    @echo "  make trace        - run timing analyzer"
    @echo "  make sdf          - generate post place & route files"
    @echo "  make download     - program entire JTAG chain."
    @echo "  make all          - run all make targets"
    @echo "  make clean        - clean current build folder"
    @echo "*******************************************************************"

synthesize:
        @echo "*******************************************************************"
        @echo "    Launch Synthesizer"
        @echo "*******************************************************************"
        cd $(SYNDIR)/bin; make synthesize;

translate : $(PROJNAME).ngd

$(PROJNAME).ngd : $(SYNTHFILE) $(UCF)
        @echo "*******************************************************************"
        @echo "    Launch NGDBUILD"
        @echo "*******************************************************************"
        ngdbuild -f $(TRANOPTS) $(SYNTHFILE) $(PROJNAME).ngd
        mv $(RUNDIR)/$(PROJNAME).bld $(LOGDIR)/
        mv $(RUNDIR)/netlist.lst $(LOGDIR)/
        cp -f $(RUNDIR)/$(PROJNAME).log $(LOGDIR)/

map : map.ncd

map.ncd : $(PROJNAME).ngd
        @echo "*******************************************************************"
        @echo "    Launch MAP"
        @echo "*******************************************************************"
        map  -f $(MAPOPTS) $(PROJNAME).ngd $(PROJNAME).pcf
        mv $(RUNDIR)/map.mrp $(LOGDIR)/
        mv $(RUNDIR)/map.map $(LOGDIR)/
```

```
	mv $(RUNDIR)/$(PROJNAME)*.xml $(LOGDIR)/
	cp -f $(RUNDIR)/$(PROJNAME).log $(LOGDIR)/

par : $(PROJNAME).ncd

$(PROJNAME).ncd : map.ncd
	@echo "*********************************************************************"
	@echo "    Launch PAR"
	@echo "*********************************************************************"
	par -f $(PAROPTS) map.ncd $(PROJNAME).ncd $(PROJNAME).pcf
	mv $(RUNDIR)/$(PROJNAME).unroutes $(LOGDIR)/
	mv $(RUNDIR)/$(PROJNAME).pad $(LOGDIR)/
	mv $(RUNDIR)/$(PROJNAME).par $(LOGDIR)/
	mv $(RUNDIR)/$(PROJNAME).xpi $(LOGDIR)/
	mv $(RUNDIR)/$(PROJNAME)_pad.csv $(LOGDIR)/
	mv $(RUNDIR)/$(PROJNAME)_pad.txt $(LOGDIR)/
	cp -f $(RUNDIR)/$(PROJNAME).log $(LOGDIR)/

bit : $(PROJNAME).bit

$(PROJNAME).bit : $(INDEX) $(PROJNAME).ncd
	@echo "*********************************************************************"
	@echo "    Launch BITGEN"
	@echo "*********************************************************************"
	bitgen $(PROJNAME).ncd -f $(BITOPTS)
	cp $(PROJNAME).bit $(OUTDIR)/$(PROJNAME).bit
	cp $(PROJNAME).msk $(OUTDIR)/$(PROJNAME).msk
	cp $(PROJNAME).bit $(CFGDIR)/
	cp $(PROJNAME).msk $(CFGDIR)/
	mv $(RUNDIR)/$(PROJNAME).bgn $(LOGDIR)/
	mv $(RUNDIR)/$(PROJNAME).drc $(LOGDIR)/
	mv $(RUNDIR)/$(PROJNAME)*.xml $(LOGDIR)/

prom : $(PROJNAME).mcs

$(PROJNAME).mcs : $(PROJNAME).bit
	@echo "*********************************************************************"
	@echo "    Launch PROMGEN for SPI PROM"
	@echo "*********************************************************************"
	promgen -f $(PROMOPTS)
	cp $(PROJNAME).mcs $(OUTDIR)/$(PROJNAME).mcs
	cp $(PROJNAME).mcs $(CFGDIR)/$(PROJNAME).mcs
	cp $(PROJNAME).cfi $(CFGDIR)/$(PROJNAME).cfi
	mv $(PROJNAME).prm $(LOGDIR)/$(PROJNAME)_SPI.prm

trace : $(PROJNAME).twr

$(PROJNAME).twr : $(PROJNAME).ncd
	@echo "*********************************************************************"
	@echo "    Launch TRACE"
	@echo "*********************************************************************"
	trce -f $(TRACEOPTS) $(PROJNAME).ncd $(PROJNAME).pcf -o $(PROJNAME).twr
	cp $(PROJNAME).twr $(LOGDIR)/$(PROJNAME).twr

sdf : $(PROJNAME).sdf

$(PROJNAME).sdf : $(PROJNAME).ncd
	@echo "*********************************************************************"
	@echo "    Launch NETGEN"
	@echo "*********************************************************************"
	netgen -f $(SDFOPTS) $(PROJNAME).ncd
	cp $(PROJNAME).v $(CODEDIR)/$(PROJNAME)/sim/gatesim/bench/$(PROJNAME).v
	cp $(PROJNAME).tv $(CODEDIR)/$(PROJNAME)/sim/gatesim/bench/$(PROJNAME).tv
	cp $(PROJNAME).sdf $(CODEDIR)/$(PROJNAME)/sim/gatesim/bench/$(PROJNAME).sdf
	mv $(PROJNAME).v $(OUTDIR)/
	mv $(PROJNAME).tv $(OUTDIR)/
	mv $(PROJNAME).nlf $(LOGDIR)/

download: $(PROJNAME).bit
	@echo "*********************************************************************"
```

```
        @echo "    Download MCS File to PROM and Bitfile to FPGA"
        @echo "*********************************************************************"
        impact -batch $(BINDIR)/download.cmd

all : synthesize translate map par trace bit prom
        @echo "*********************************************************************"
        @echo "    This build has finished."
        @echo "*********************************************************************"

clean :
    rm -f $(PROJNAME)*
    rm -f map.mrp map.ncd map.ngm netlist.lst
    rm -f *.log
```

5. Team Design

The design hierarchy described in Chapter 2 fosters a team design flow through its use of unique sub-directories for each HDL module. The design of the HDL modules can be distributed amongst team members to be designed, simulated, and synthesized, then committed to a revision control system, such as Git. This allows each member of the team to receive updates as modules are completed. Care must be taken to avoid committing intermediate generated files to the revision control system. Git provides a means of avoiding such a condition by use of a .gitignore file placed at the root of the repository. Listing 5.1 shows an example .gitignore file used with the “myfpga” design. Any files located in the directories shown in Listing 5.1, which were not previously committed to the repository, will be ignored by Git when staging or adding files to the repository.

Listing 5.1 Git .gitignore File

```
#*******************************************************************
#
# .gitignore file
#
#*******************************************************************
#
# This is the .gitignore file for the <myfpga> git repository.
# It defines which files will be ignored by Git.
#
#*******************************************************************

#-----------------------------------------------------------------
# Auto-generated Files
#-----------------------------------------------------------------
.DS_Store
.*.swp

#-----------------------------------------------------------------
# Files generated by FPGA tools
#-----------------------------------------------------------------
myfpga/par/run/
myfpga/par/log/
myfpga/par/out/
myfpga/syn/bin/*.log
myfpga/syn/run/
myfpga/syn/log/
myfpga/syn/out/
```

6. Build Environment

The synthesis flow and place and route build flow described in this paper are supported on both a Windows and a Linux operating system. On Windows, the Makefiles and scripts can be executed in a Cygwin environment along with GNU Make. The Makefiles and scripts are natively supported on a Linux operating system. The back-end and front-end tools are supported on both operating systems.

7. Conclusion

The suggested directory structure and use of command-line interface scripts and Makefiles can improve the FPGA or ASIC design efficiency and promotes a team design flow. The design flow is controlled such that all files generated by both the design team and the tools are stored in a known location. In turn, this will make debugging a design, either during the build or when hardware arrives, much more straightforward. The techniques employed in the FPGA implementation flow can be easily leveraged to an ASIC implementation flow.

8. References

[1] Synopsys FPGA Synthesis User Guide, November 2011.

[2] Xilinx ISE Design Suite 13.3, October 2011.