Streamlining your FPGA synthesis process

The idea was to port a number of existing designs to the Xilinx Spartan 3E and 3AN starter kit boards: a few of my own designs, either targeting other boards/devices or never been yet tested at the board level, or designs from other people. Mike Field's hamsterworks website is an excellent source of FPGA designs with varying degree of complexity! Most of them are for Spartan 6/Artix 7, so I had to backport some of his ideas to my less contemporary devices :)

One thing with porting or backporting a number of designs effectively is that I had to streamline the synthesis process; all had to be done from the command line. Borrowing some ideas from Evgeni Stavinov's Using Xilinx Tools in Command-Line Mode I was able to synthesize, generate bitstream and finally program the FPGA device via my download cable, without any GUI interaction.

I will use as our vehicle maybe the simplest possible design. Let's call it bstest, as a stand-in for "buttons and switches tester". It is a very simple design for testing the four push buttons, four slide switches and eight discrete LEDs available on the Spartan-3E Starter Kit board by Digilent.

The design associates push button and slide switch actions to specific LEDs. The code (bstest.vhd) is really simple:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity bstest is
  port (
    sldsw  : in  std_logic_vector(3 downto 0);
    button : in  std_logic_vector(3 downto 0); -- N-E-W-S
    led    : out std_logic_vector(7 downto 0)
  );
end bstest;

architecture dataflow of bstest is
begin
  --
  led(7 downto 4) <= sldsw;
  led(3 downto 0) <= button;
--  led <= sldsw & button; -- we could also do it this way
  --
end dataflow;
```

The UCF (User Constraints File) associates a "net" (input or output port of our top-level design) to a "loc" (location), referring to a specific FPGA pin. The specific FPGA

device available on the Spartan-3E Starter Kit is the XC3S500E-FG320-4 and the UCF (bstest.ucf) in this case should be as follows:

```
NET "sldsw"   LOC = "N17";
NET "sldsw"   LOC = "H18";
NET "sldsw"   LOC = "L14";
NET "sldsw"   LOC = "L13";

NET "button"  LOC = "V4";   # NORTH
NET "button"  LOC = "H13";  # EAST
NET "button"  LOC = "D18";  # WEST
NET "button"  LOC = "K17";  # SOUTH

NET "led"     LOC = "F9"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "led"     LOC = "E9"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "led"     LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "led"     LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "led"     LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "led"     LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "led"     LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "led"     LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
```

The top-level HDL design file and the UCF map is all we would need, if we intended to use the Xilinx ISE/XST GUI (I have version 14.6 installed). However, we can be much more productive, if we resort to a few scripts for automating this part of the process.

First, a Makefile (xst.mk) based on Evgeni Stavinov's article can be used for automating all the way to bitfile generation:

```
all: $(PROJECT).bit

floorplan: $(PROJECT).ngd $(PROJECT).par.ncd
	$(FLOORPLAN) $^

report:
	cat *.srp

clean::
	rm -f *.work *.xst
	rm -f *.ngc *.ngd *.bld *.srp *.lso *.prj
	rm -f *.map.mrp *.map.ncd *.map.ngm *.mcs *.par.ncd *.par.pad
	rm -f *.pcf *.prm *.bgn *.drc
	rm -f *.par_pad.csv *.par_pad.txt *.par.par *.par.xpi
	rm -f *.bit
	rm -f *.vcd *.vvp
	rm -f verilog.dump verilog.log
	rm -rf _ngo/
	rm -rf xst/

################################################################################
# Xilinx tools and wine
```

```
##############################################################################

XST_DEFAULT_OPT_MODE = Speed
XST_DEFAULT_OPT_LEVEL = 1
DEFAULT_ARCH = spartan3
DEFAULT_PART = xc3s700an-fgg484-4

XBIN=$(XDIR)/bin/nt64
XST=$(XBIN)/xst
NGDBUILD=$(XBIN)/ngdbuild
MAP=$(XBIN)/map
PAR=$(XBIN)/par
TRCE=$(XBIN)/trce
BITGEN=$(XBIN)/bitgen
PROMGEN=$(XBIN)/promgen
FLOORPLAN=$(XBIN)/floorplanner

XSTWORK   = $(PROJECT).work
XSTSCRIPT = $(PROJECT).xst

IMPACT_OPTIONS_FILE   ?=_impact.cmd

ifndef XST_OPT_MODE
XST_OPT_MODE = $(XST_DEFAULT_OPT_MODE)
endif
ifndef XST_OPT_LEVEL
XST_OPT_LEVEL = $(XST_DEFAULT_OPT_LEVEL)
endif
ifndef ARCH
ARCH = $(DEFAULT_ARCH)
endif
ifndef PART
PART = $(DEFAULT_PART)
endif

$(XSTWORK): $(SOURCES)
        > $@
        for a in $(SOURCES); do echo "vhdl work $$a" >> $@; done

$(XSTSCRIPT): $(XSTWORK)
        > $@echo -n "run -ifn $(XSTWORK) -ifmt mixed -top $(TOP) -ofn $(PRO
        >> $@echo " -ofmt NGC -p $(PART) -iobuf yes -opt_mode $(XST_OPT_MOD

$(PROJECT).bit: $(XSTSCRIPT)
        $(XST) -intstyle ise -ifn $(PROJECT).xst -ofn $(PROJECT).syr
        $(NGDBUILD) -intstyle ise -dd _ngo -nt timestamp -uc $(PROJECT).ucf
        $(MAP) -intstyle ise -p $(PART) -w -ol high -t 1 -global_opt off -o
        $(PAR) -w -intstyle ise -ol high $(PROJECT).map.ncd $(PROJECT).ncd
        $(TRCE) -intstyle ise -v 4 -s 4 -n 4 -fastpaths -xml $(PROJECT).twx
        $(BITGEN) -intstyle ise $(PROJECT).ncd
```

```
$(PROJECT).bin: $(PROJECT).bit
        $(PROMGEN) -w -p bin -o $(PROJECT).bin -u 0 $(PROJECT).bin
```

So what happens here? The synthesis procedure invokes several Xilinx ISE command-line tools for logic synthesis as described in the corresponding Makefile, found in the bstest main directory.

Typically, the process includes the following:

- Generation of the `*.xst` synthesis script file.
- Generation of the `*.ngc` gate-level netlist file in NGC format.
- Building the corresponding `*.ngd` file.
- Performing mapping using map which generates the corresponding `*.ncd` file.
- Place-and-routing using par which updates the corresponding `*.ncd` file.
- Tracing critical paths using trce for reoptimizing the `*.ncd` file.
- Bitstream generation (`*.bit`) using bitgen, however with unused pins.

As a result of this process, the `bstest.bit` bitstream file is produced.

Then, the shell script invokes the Xilinx IMPACT tool by a Windows batch file named impact_s3esk.bat, automatically passing a series of commands that are necessary for configuring the target FPGA device:

```
setMode -bs
setCable -p auto
identify
assignFile -p 1 -file bstest.bit
program -p 1
exit
```

Each line provides a specific command to the IMPACT tool:

1. Set mode to binary scan.
2. Set cable port detection to auto (tests various ports).
3. Identify parts and their order in the scan chain.
4. Assign the bitstream to the first part in the scan chain.
5. Program the selected device.
6. Exit IMPACT.

If using the Spartan-3AN starter kit board, the "program" command should be used with `-onlyFPGA` command-line option, in order to only program the FPGA device.

Finally, a Bash shell script can be used as our entry point (and only needed access point) to the process. The corresponding synthesis script (bstest-syn.sh) can be edited in order to specify the following for adapting to the user's setup:

- XDIR: the path to the /bin subdirectory of the Xilinx ISE/XST installation where the `xst.exe` executable is placed

- arch: specific FPGA architecture (device family) to be used for synthesis
- part: specific FPGA part (device) to be used for synthesis

```
#!/bin/bash

# Change XDIR according to your host configuration.
export XDIR=/c/XilinxISE/14.6/ISE_DS/ISE
make -f xst.mk clean
make -f xst.mk PROJECT="bstest" \
  SOURCES="bstest.vhd" \
  TOPDIR="./log" TOP="bstest" \
  ARCH="spartan3" PART="xc3s500e-fg320-4"

# Invoke impact.exe for manual download of the generated bitstream to a
# hardware platform.
${XDIR}/bin/nt64/impact.exe -batch impact_s3esk.bat
```

And that's about it!

I have used this script in MinGW on Windows 7/64-bit. My setup is for an ISE 14.6 installation. There might be a catch here; on some systems, there is a board driver problem regarding ISE. To fix this issue, open a command prompt and uninstall, then reinstall the driver, as follows:

```
cd c:\XilinxISE\14.6\ISE_DS\ISE\bin\nt64
wdreg -compat -inf %cd%/windrvr6.inf uninstall
wdreg -compat -inf %cd%/windrvr6.inf install
```

Following these steps, you will be able to synthesize the bstest design. You can download either the Spartan-3E version (bstest-s3esk.zip) or the Spartan-3AN one (bstest-s3ansk.zip) to start experimenting!

BTW in case your antivirus program goes crybaby you can safely ignore it, `impact_s3esk.bat` is the Windows batch file for passing commands to IMPACT, nothing special about it.