FPGAs at the Command Line

By Bob Smith

Introduction

Wire-wrap and low cost TTL parts has made the puzzle-solving fun of digital logic design accessible to hobbyists and engineers everywhere. Although TTL works fine for small designs, it becomes expensive and time consuming for larger designs. Field Programmable Gate Arrays (FPGAs) contain thousands or millions of uncommitted gates and are almost ideal for large designs. The problem with FPGAs is that you have to learn a design language (Verilog or VHDL) and that in order to write even simple Verilog programs you need to set up a fairly complex development environment. It is the latter problem, the development environment, that this article addresses.

In this article you'll see how to install the Xilinx FPGA design tools, how to use the Xilinx command line tools to compile a Verilog or VHDL design, how to download the compiled code to an FPGA board, and how to automate the whole process using a Makefile. The Xilinx command line tools are the same for both Linux and Windows, making this article useful for almost anyone with a computer (including Mac users using a Linux virtual machine).

The sample design for this article is a 28 bit counter which has the most significant 8 bits visible on LEDs. It is counting transitions on a 12.5 MHz clock giving the least significant LED a flash rate of about 6 Hertz. The circuit has a counter reset line tied to a button.

The hardware used for this article is the Demand Peripherals Baseboard-4 which has a Xilinx Spartan-3E, LEDs, buttons, and a USB interface for downloads and user data. Since the Baseboard-4 is downloaded through a USB serial port it doesn't require the expense and complexity of a separate JTAG dongle, or the hassle of installing JTAG drivers. Not requiring JTAG drivers is particularly nice for Linux developers. (I helped design the Baseboard-4 and one of our major design goals was to allow USB serial downloads.)

Installing the Xilinx WebPACK Toolkit

Xilinx provides a set of free design tools, the WebPACK, that runs on both Windows and on Linux. Xilinx supports the tools on Windows XP Professional and on recent versions of Red Hat Enterprise Linux and SUSE Linux. While not officially supported, WebPACK runs fine on XP Home Edition, Vista, and Ubuntu.

To get the tools working on your system you have to go through about a dozen web pages to start the WebPACK download, wait for a 2.2 GB download, and then go through another dozen or so screens for the installation. It is tedious, but straightforward.

Start by going to the Xilinx download site at: http://www.xilinx.com/support/download/index.htm. Skip the Search button and scroll directly down to click on the "Download ISE WebPACK" link. This will take you to a login page where you can select "Create Account" (since you probably don't already have a Xilinx account). You will receive an email message with a web link where you can go to finish the registration and to login. After logging in for the first time you're asked to provide more information. The Next button will take you to a screen that displays what Xilinx packages you are entitled to download. Select ISE WebPack10.1 and click on the Next button.

The next page, the Product Registration and Download page, has your registration ID. You might want to cut and paste this ID into a file for use in the installation, and in case you want reinstall the package. This page also has several links for documentation and two links for downloading WebPACK. It is probably best not to click the large Download button as this tries to install software on you PC. Instead, select the "Download Files Individually" link, and then select the Download link which appears next to the file size. The 2.2 GB download is going take awhile even on a relatively fast Internet link.

The WebPACK download file is in a "tar" format, which, while common on Linux, is not well known on Windows. Windows users may need to install a package such as Winzip (http://www.winzip.com) or WinRAR (http://www.winrar.com), both of which can handle tar files.

Install the software by double-clicking setup.exe in the ZIP file or, for Linux, by untarring the download file and running the "setup" script in the top level directory. Windows users will need administrator privileges to install the software. The default installation directory on Windows is C:\Xilinx\10.1 and on Linux it is /opt/Xilinx/10.1. If installing on Linux as a non-root user, you might want to create /opt/Xilinx/10.1 beforehand and give yourself write permission on it.

The installation will ask if you want to do an immediate update and whether or not to install the cable drivers. Saying "yes" to an immediate update is a good idea but will trigger another 600 MB download. For this tutorial you do not need the update and can safely postpone it. You can also say "no" to the "Install Cable Drivers" option since the board we're using does not need JTAG drivers. The installation takes ten or twenty minutes once all the licenses are accepted and the installation options are set.

Test the installation on Windows by opening a Command Window and entering the command `C:\Xilinx\10.1\ISE\bin\nt\xst -h`. On Linux the command is `/opt/Xilinx/10.1/ISE/bin/lin/xst -h`. If everything is installed correctly you should see a display of the usage of the xst command. From this point on I won't give the full path to the command. You should either prepend the full path to the command or modify your shell's execution path variable. A convenient way to do this on Linux is to source /opt/Xilinx/10.1/ISE/settings32.sh.

Creating a Simple Counter in Verilog

The Xilinx tools can compile either Verilog or VHDL and I've chosen Verilog for the test program. Whether it VHDL or Verilog, you have to tell the compiler about the hardware on your FPGA board. The Xilinx "User Constraints File" (.ucf) relates FPGA pin numbers or locations to logical names for use in the Verilog code. Consider, for example, the partial schematic of the Demand Peripherals Baseboard-4 shown in Figure 1. The constraints file for this hardware might look like that shown in Listing 1. The NET field is the pin name as it appears in your Verilog program. The LOC field is the pin location on the physical FPGA. Xilinx specifies pin locations as the letter P followed by the pin number quad-flat-packs, and by the grid location for parts in a ball grid array package.

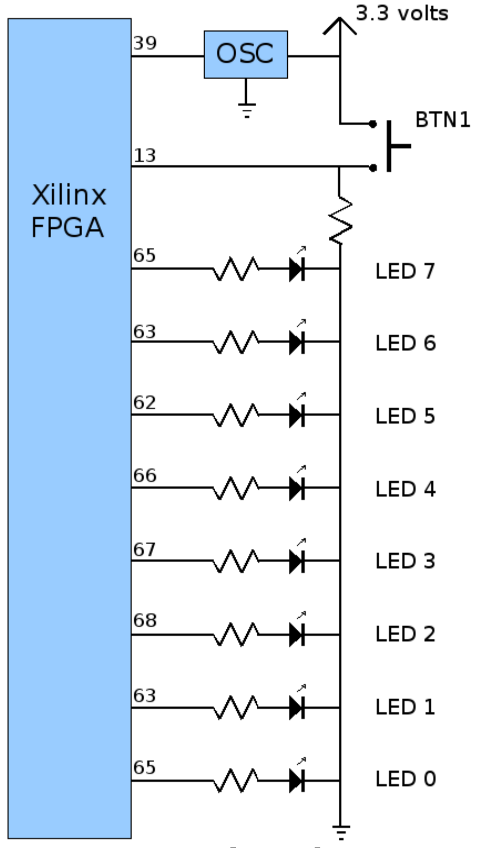

Figure 1: Baseboard-4 Schematic

```
  NET "CLEAR"    LOC = "P13"  ;    # Button 1
  NET "CK12"     LOC = "P39"  ;    # 12.5 MHz clock
  NET "LED<0>"   LOC = "P70"  ;    # LED 0
  NET "LED<1>"   LOC = "P71"  ;    # LED 1
  NET "LED<2>"   LOC = "P62"  ;    # LED 2
  NET "LED<3>"   LOC = "P66"  ;    # LED 3
  NET "LED<4>"   LOC = "P67"  ;    # LED 4
  NET "LED<5>"   LOC = "P68"  ;    # LED 5
  NET "LED<6>"   LOC = "P63"  ;    # LED 6
  NET "LED<7>"   LOC = "P65"  ;    # LED 7
```

Listing 1: A User Constraints File for the Baseboard-4

The user constraints file can do much more than just relate Verilog names to pin numbers. It can also add pull-up or pull-down resistors, set output current limits, set timing constraints, and set output slew rate. Details about the user constraints file can be found on the Xilinx web site by searching for "user constraints file". In particular, additional information is available in this document: http://toolbox.xilinx.com/docsan/data/alliance/dev/dev3.htm.

Use Notepad, vi, or your favorite editor to create a text file called counter.ucf and copy Listing 1 into it. (All three files for this article are in a ZIP file in the download section of the Demand Peripherals web site: http://www.demandperipherals.com)

The Verilog program for this tutorial implements a simple counter that counts an input clock and displays the high 8 bits on the LEDs. A clear input can force the count to zero. Figure 2 illustrates the circuit for the Verilog design shown in Listing 2.

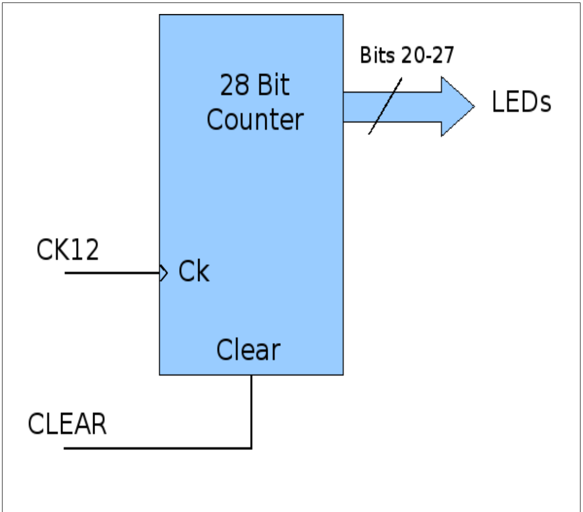

Figure 2: A 28 Bit Counter

```
module counter(CLEAR, CK12, LED);
  input CLEAR;         // Set counter=0
  input CK12;          // Clock source
  output [7:0] LED;    // Output display

  reg [27:0] count;    // a 28 bit counter
  reg metaclear;       // bring CLEAR into
                       // our clock domain

  always @(posedge CK12)
  begin
    metaclear <= CLEAR;
    if (metaclear)
      count <= 0;
    else
      count <= count + 1;
  end

  assign LED = count[27:20]; // Set display

endmodule
```

Listing 2: counter.v, A 28 Bit Counter in Verilog

Copy the program shown in Listing 2 into a file called counter.v in your working directory.

Compiling Your Verilog Counter for Xilinx FPGAs

Xilinx provides a graphical development environment called ISE. You'll be issuing the same commands that ISE issues behind the GUI, but you'll be doing it from the command line. It is the use of command line tools that makes it easy automate the build process in a batch file or a makefile. ISE, the Xilinx GUI tool, leaves its configuration in several binary files in your working directory. Using just command line tools lets you clean the directory leaving only your original text files. Having just text files for your design makes it easier to a user version control and to extract meaningful differences between two versions of a design.

There is insufficient space in this tutorial to give detailed descriptions of the commands but your download of the Xilinx tools includes comprehensive manuals for the Xilinx command line tools. Look in ISE/doc/usenglish/books/docs/

The first command, xst, synthesizes the Verilog file into a hardware design that is saved as a netlist file with an .ngc extension. Xilinx's xst program is actually a command line interpreter and it expects input from standard-in. Use an echo command and a pipe operator to give xst input from standard-in if you want to keep all of your build information in a Makefile. The only difference between Windows and Linux for this article is in the echo command below. Linux requires the quotes and Windows does not.

```
  echo "run -ifn counter.v -ifmt Verilog -ofn counter -p xc3s100e-4-vq100 \
  -opt_mode Speed -opt_level 1" | xst
```

You have to specify the input file, the input file format, the name of the output file and the exact type of FPGA. If your goal is to learn VHDL and not Verilog, you can change the input format in the above command from "Verilog" to "VHDL", replace the 28 bit Verilog counter with its VHDL equivalent, and continue reading the article as if it were about VHDL. Xst generates several report files and directories, but the real output is a netlist file with an .ngc extension that is required for the next command. You can examine the output files and reports to better understand the how the synthesis works and an appendix in the xst manual describes the output files and reports in detail.

The ngdbuild command further decomposes the design into FPGA native elements such as flip-flops, gates, and RAM blocks.

```
 ngdbuild  -p xc3s100e-4-vq100 -uc counter.ucf  counter.ngc
```

It is the ngdbuild command that first considers the pin location, loading, and timing requirements specified in the user constraints file. Like the other Xilinx commands, ngdbuild produces several reports but its real output is a "Native Generic Database" stored in a (.ngd) file.

The Xilinx map command converts the generic elements from the step above to the elements specific to the target FPGA. It also performs a design rules check on the overall design. The map command produces two files, a Physical Constraints File file and a Native Circuit Description file, that are used in subsequent commands.

```
 map -detail -pr b counter.ngd
```

The map command produces quite a few reports. As you gain experience with FPGA design you may come to rely on these report to help identify design and timing problems.

The place and route command (par) uses the Physical Constraints File and the

Native Circuit Description to produce another Native Circuit Description file which contains the fully routed FPGA design.

```
 par -w counter.ncd parout.ncd counter.pcf
```

Output processing starts with the bitgen program which converts the fully routed FPGA design into the pattern of bits found in the FPGA after download.

```
 bitgen -w -g StartUpClk:CClk -g CRC:Enable parout.ncd counter.bit counter.pcf
```

The bitgen program lets you specify which clock pin to use during initialization and whether or not to generate a CRC checksum on the download image. Files which contain a raw FPGA download pattern are called bitstream files and traditionally has a .bit file extension. Bitstream files are good for downloads using JTAG but since we're downloading over a USB serial connection one more command is required to convert the bitstream file into a download file.

```
 promgen -w -p bin -o counter.bin -u 0 counter.bit
```

The promgen program is a utility that converts bitstream files into various PROM file formats. The format for the Baseboard-4 is called bin so the promgen command uses the -p bin option. The output of promgen, counter.bin, is what is downloaded to the Baseboard-4.

All of the commands described above, including xst, ngdbuild, map, par, bitgen, and promgen have excellent PDF manuals in either the ISE/doc/usenglish/books/docs/xst directory or the ISE/doc/usgnglish/de/dev directory of your WebPACK installation.

Listing 3 is a Makefile that captures the commands and dependencies described above. Verilog does not lend itself to incremental compilation so just copying the above commands into a .bat file or a shell file is practically as good as using a Makefile.

Downloading Your Counter to an FPGA Board

Now that you've compiled your Verilog program you're ready to download it to the Baseboard-4. Since the Baseboard-4 is powered by a USB cable it should be plugged directly into a USB port on your computer and not through an (unpowered) hub. The Baseboard-4 uses an FTDI245 USB serial interface and you'll need the appropriate drivers for you operating system. Figure 3 shows the Baseboard-4 powered from USB. You can clearly see the Xilinx FPGA in the center of the board. The IC to the left of the FPGA is the FTDI FT245 and the smaller IC to the lower left is a Xilinx CPLD that coordinates the download between the FPGA and the USB interface.

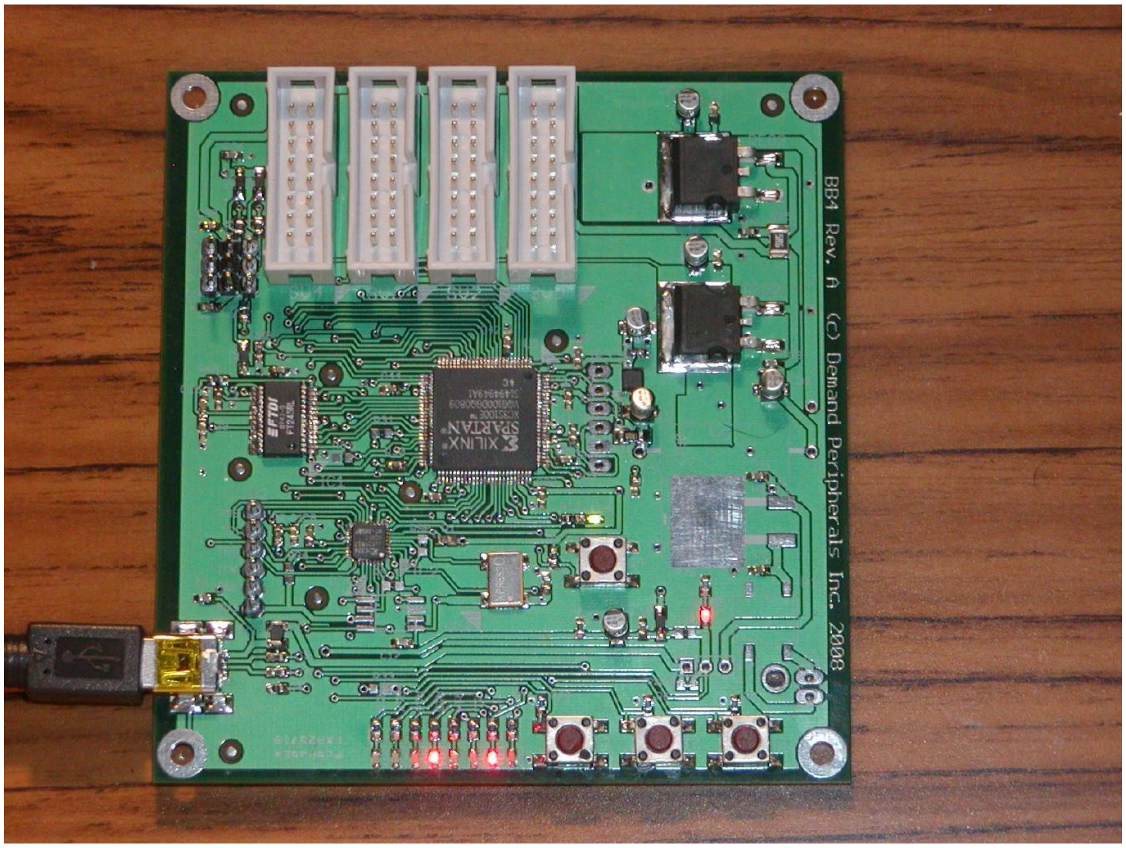

Figure 3: A Baseboard-4 Downloaded from and Powered by USB

On Windows you can plug the board in and you'll be prompted with the usual "New Hardware Found" message. After loading the Windows driver for the FTDI245 you may be asked to load a driver for a USB serial port. Looking at System folder in the Control Panel will show which COM port was assigned to the USB serial port. On my system it was assigned to COM5 so I'll use that for the example. The Windows command to download the compiled Verilog code is just:

```
  copy counter.bin /B COM5: /B
```

Linux ships with a driver for the FTDI245 and it will be loaded automatically when you connect the Baseboard-4 to your system. Ubuntu users should watch for a bug in Ubuntu in which the device driver for a USB Braille reader interferes with all USB serial ports. On my system the FTDI USB serial port is assigned to /dev/ttyUSB0. Linux does what is called post processing on data sent to a serial port and you need to turn this off to prevent it from corrupting your download file. The two commands needed to download the compiled Verilog code on Linux are

```
  stty --file=/dev/ttyUSB0 -opost  # No post processing
  cat counter.bin > /dev/ttyUSB0
```

Are the LEDs incrementing? If so, congratulations; you're on your way into the world of FPGA programming with Verilog!

Most FPGAs lose their programming when powered-down and most boards, including the Baseboard-4 are ready for download immediately at power up. You can also press the reset button to get the board ready for another download. While not a requirement, it is considered polite to disable a USB device under Window before you unplug it.

Dealing with JTAG

The Demand Peripherals Baseboard-4 and the XESS XSA-3S1000 are two examples of FPGA boards that do not require JTAG for code download. Dealing with JTAG at the command line is possible, but a common problem, especially for Linux users, is first finding and installing the device driver that matches the JTAG dongle that you buy. JTAG drivers are more standard and easier to install on Windows, and for this reason many would-be Linux FPGA developers move to Windows for FPGA development. I know one Linux user who does all of his development on Linux and then copies his files to Windows for the JTAG downloads.

The Xilinx command to manage JTAG devices and downloads is `impact`. It is a command line interpreter similar to xst, and its internal commands and options are described in full in Appendix C of the iMPACT User Guide at http://toolbox.xilinx.com/docsan/xilinx4/data/docs/pac/preface.html. Xilinx compatible JTAG dongles are available from several commercial sources, and Appendix B of the iMPACT User Guide even has a schematic of at simple JTAG dongle that attaches to a legacy parallel port.

The sub-commands to use inside iMPACT can vary a great deal depending on the type of FPGA board that you are using. I've found that the easiest way to deal with impact is to run ISE and invoke the GUI interface to impact the first time I download to a board, and to then extract the impact sub-commands from from the _impact.log file. Using this technique on the SparkFun Spartan 3E Breakout and Development Board gives an iMPACT command of `impact -batch impact.bat` where the batch file is:

```
      setMode -ss
      setMode -sm
      setMode -hw140
      setMode -spi
      setMode -acecf
      setMode -acempm
      setMode -pff
      setMode -bs
      setCable -p auto
      addDevice -p 1 -file counter.bit
      Program -p 1 -defaultVersion 0
      quit
```

Note the `addDevice -p 1` sub-command in the above batch file. JTAG devices are usually arranged as a string of devices and the addDevice sub-command specifies which device in the chain of devices to program or examine. Some manufacturers, such as Digilent, prefer PROMs that can be programmed directly from JTAG. Other manufacturers, such as SparkFun, have you use JTAG to load an FPGA program that then reads from a serial port, burning the received bytes into the PROM. The Demand Peripherals Baseboard-4 is meant to be connected to a PC, and since a download to the Baseboard-4 takes less than 100 milliseconds, it hardly seemed necessary to add a PROM and force the user into the cost and driver issues associated with JTAG.

Conclusions and Next Steps

In this article I focused on two things, expressing your design as text files and using just a few standard commands to compile and download your design. Having your design in text files will make it easier for you to track changes and will make it easier for you to add version control for your projects. Using command line tools can be faster and is a nice way to better understand the steps to convert a design from Verilog to a bitstream file. Even if you switch to a GUI based approach later, at least you now have more appreciation for what is actually going on.

You may have noticed another purpose in this article. It is an attempt to get a working Verilog program is as few steps as possible. I meant this article to be something of a "Hello, World!" program for Verilog.

The next steps are, of course, up to you. This article just got the tools working; we didn't even scratch the surface on real Verilog design. My advice is for you to select and buy two are three Verilog (or VHDL) books and read them cover-to-cover.

Before leaving this project completely let's see if you can make a few simple modifications to the program. Say Button 2 is on Pin 30 and that Button 3 is on Pin 69. What would have to do to the .ucf file to add Button 2 as a "Hold" button and Button 3 as a "Direction" button? What would you have to do to the Verilog file to make the count freeze while Hold is being pressed? What would it take to make the counter count down while Direction is being pressed?

```
# Makefile to compile and download a simple Verilog program

DEVICE=xc3s100e-4-vq100
default: counter.bin

counter.ngc: counter.v
  echo "run -ifn counter.v -ifmt Verilog -ofn counter -p \
   $(DEVICE) -opt_mode Speed -opt_level 1" | xst

counter.ngd: counter.ngc counter.ucf
  ngdbuild -p $(DEVICE) -uc counter.ucf counter.ngc

counter.ncd: counter.ngd
  map -k 6 -detail -pr b counter.ngd

counter.pcf: counter.ngd
  map -k 6 -detail -pr b counter.ngd

parout.ncd: counter.ncd
  par counter.ncd parout.ncd counter.pcf

counter.bit: parout.ncd
  bitgen -g CRC:Enable -g StartUpClk:CClk -g Compress \
   parout.ncd counter.bit counter.pcf

counter.bin: counter.bit
  promgen -w -p bin -o counter.bin -u 0 counter.bit

install: counter.bin
  stty --file=/dev/ttyUSB0 -opost  # We want raw output
  cat counter.bin > /dev/ttyUSB0

clean:
  rm -rf counter.bgn counter.bin counter.bit counter.bld \
   counter.drc counter.map counter_map.xrpt counter.mrp \
   counter.ncd counter.ngc counter.ngd counter_ngdbuild.xrpt \
   counter.ngm counter_par.xrpt counter.pcf counter.prm \
   counter_summary.xml counter_usage.xml counter_xst.xrpt \
   netlist.lst parout.ncd parout.pad parout_pad.csv \
   parout_pad.txt parout.par parout.ptwx parout.unroutes \
   parout.xpi xlnx_auto_0.ise xlnx_auto_0_xdb xst
```

Listing 3: A Makefile for Xilinx Command Line Tools.