Lattice iCE40-HX8K Breakout Board Setup Guide

October 23, 2019

1 Generic

1.1 Links

Board Main Website:
https://www.latticesemi.com/Products/DevelopmentBoardsAndKits/iCE40HX8KBreakoutBoard.aspx
Board Documentation:
https://www.latticesemi.com/view_document?document_id=50373
Main Open Source Toolchain to be used, use yosys+nextpnr, as you can see from the Makefile in the example project:
https://github.com/FPGAwars/toolchain-yosys/releases
and:
https://github.com/FPGAwars/toolchain-ice40/releases
(so far not required) iCEcube2 vendor-specific proprietary software, might be required in later experiments:
http://www.latticesemi.com/iCEcube2

1.2 Tools and Project Overview

We supply you a basic Makefile-based project in `basic_example_vlog_foss`, please read that Makefile to get an idea about it. Here we explain the basic steps and in the following subsections explain the details.

After getting the respective tools from the *toolchain-ice40* and *toolchain-yosys* git, the Makefile will use the respective yosys and nextpnr commands to generate a bitstream for the FPGA — after you added the PATH variable as explained in the next section. If you don't add the PATH variables, you might need to adjust the Makefile for the respective path(s) in which the ice40 and nextpnr toolchains are installed, but we recommend using the PATH variable.

After that, the bitstream file (top_level.bin) can be generated and programmed to the FPGA's SRAM. Then it can be accessed through UART using a terminal emulator.

1.3 PATH variable in Linux

To 'install' the toolchain projects, simply extract their provided .tar.gz to a permanent location (i.e. /home/<username>/Software/toolchain-ice-and-yosys/). You can just extract both archives 'over each other', meaning that their bin-directories will include files from both, but you can also keep them separated. After that, just add the 'bin' subdirectory to your path. Most probably you are using BASH and want to add it to your .bashrc. That means, edit the .bashrc in your home director (/home/<username>/.bashrc). Files starting with a '.' are considered hidden files in UNIX-like systems, so you might want to enable an option to see it. If you don't have a .bashrc, please don't start with an empty file, but copy the one from /etc/skel/.bashrc to your home directory, and then edit that. Simply add the following line to the end of the file (adjusted to where you put the downloaded toolchain):

```
export PATH=${HOME}/Software/toolchain-ice-and-yosys/bin:$PATH
```

If you want to keep nextpnr and yosys separated, you can add both paths, like:

```
export PATH=${HOME}/Software/toolchain-ice40/bin:${HOME}/Software/toolchain-nextpnr/bin:$PATH
```

Please make sure the files are executable by adding +x permissions (in both paths, if you keep them separated):

```
chmod +x ${HOME}/Software/toolchain-ice-and-yosys/bin/*
```

After you have changed the .bashrc, every shell that you open **after** the change will be able to run the respective yosys, nextpnr, iceprog, etc. commands that the *Makefile* is using. That means by entering 'make' in the project subdirectory.

1.4 Compiling the example project with make and UART port permissions

```
# cd FPGA_basics/basic_example_vlog_foss/
# make
```

The output should look like follows:

```
yosys \
-p "hierarchy -top top_level" \
-p "synth_ice40  -json top_level.json" \
top_level.v uart.v

/-----------------------------------------------------------------------------\
|                                                                             |
|  yosys -- Yosys Open SYnthesis Suite                                        |

... and so on ...
```

Ending with:

```
...

// Reading input .asc file..
// Reading 8k chipdb file..
// Creating timing netlist..
// Timing estimate: 7.13 ns (140.30 MHz)
icepack top_level.asc top_level.bin
#
```

Then you can connect and program the FPGA board:

```
# make prog
iceprog -S top_level.bin
init..
cdone: high
reset..
cdone: low
programming..
cdone: high
Bye.
```

If there are any issues in programming, you might want to add a respective udev rules file. Please add a new file as root in /etc/udev/rules.d/ (for example 99-lattice-hx8k.rules), with the following content:

```
ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6010", MODE="660", GROUP="plugdev"
```

Reload rules as root with: `sudo udevadm control --reload` Then reconnect the board, after which the programming with 'make prog' should work.

1.5 Programming the board and experimenting further

After you managed to generate the bitstream file (top_level.bin) and followed the previous setup steps, please program it to the SRAM of the FPGA as shown below in Section 2.1 for Windows/Linux. Be aware of the jumper settings.

I.e. in Linux you just type the following:
`iceprog -S <projectname>/top_level.bin`
Which can also be run from the Makefile: `make prog`

Then, you can try it out by using a terminal emulator (see below in Section 2.2) to send an ascii 's' or 'S'. You will need to set up 115200 as the baud rate and probably select something like /dev/ttyUSB1 (in Linux, Windows would be COM3, COM4, or similar). In Linux you can watch the kernel log with 'dmesg -w'. Run that before you plug-in the board, and then check the output when you plug it in. You should see two ttyUSB# devices, where the second number # is the one you have to use in your terminal emulator.

Now, please take a look into the verilog code in top_level.v and try to understand it. The next steps are to experiment with it to get some basic 'feeling' for how it works. For example, let LEDs blink differently; try making an LED blink every second by counting up 12 million times, based on the 12MHz clock; try answering with more than 1 character messages on the UART; try to go through multiple inputs on the UART (like a 'password entry' after which you light up a LED on success); or whatever comes into your mind. Refer to the verilog explanations from the introductory presentation, if required (FPGA_basics/fpga_practice.pdf) — also feel free to search for verilog examples online. A good idea is to search for state-machines in verilog, since performing logic step-by-step is pretty simple with a state machine.

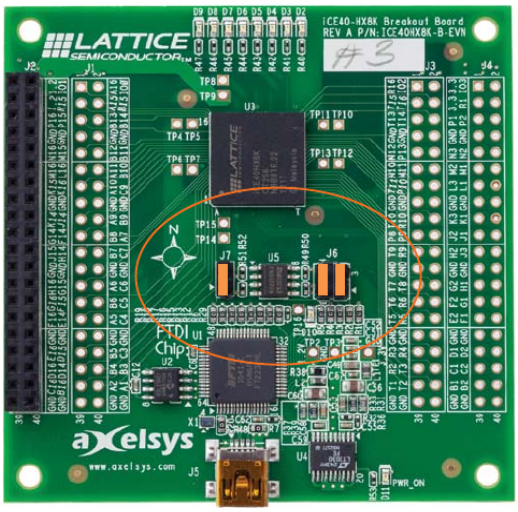
2 Additional Software and Hardware Information

2.1 Programmer software

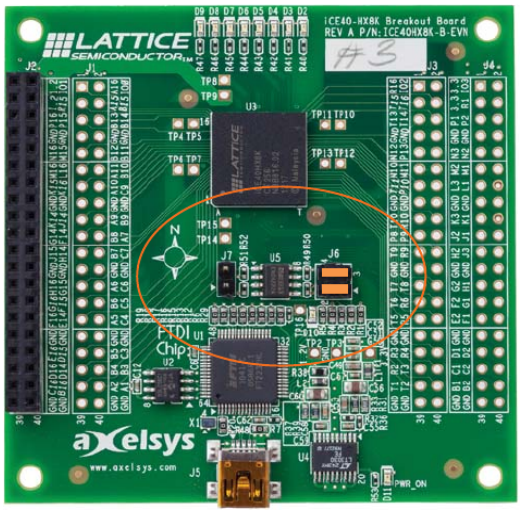
Please first try the open-source software iceprog{.exe}:
`iceprog -S <projectname>/top_level.bin`
If you are using Windows and if there are any problems, you can try the Lattice Diamond Programmer Standalone software from:
http://www.latticesemi.com/latticediamond
Please make sure the board has the respective jumper settings for either programming to the SRAM or Flash (usually we use SRAM, which is faster but will be gone after the FPGA is disconnected from USB). Jumper settings are shown in Figure 1 as explained in the iCE40HX-8K Breakout Board User Guide.

2.2 Terminal emulation

Use any terminal emulator and make sure that you set the baud rate to "115200", without parity (typically the default). That is the default speed our project uses.
In Windows, you can, for example, use hterm or teraterm. Search and Download for them. For example:
https://www.heise.de/download/product/hterm-53283
In Linux, you can use cutecom: `sudo apt-get install cutecom`
Screen also works, for example like this: `screen /dev/ttyUSB1 115200`



(a) Programming the flash memory on the board, which then gets transferred to the FPGA internal SRAM during power-up. (with 'iceprog' command or 'make prog-flash')



(b) Programming the SRAM memory on the FPGA chip directly. (with 'iceprog -S' command or 'make flash')

Figure 1: Jumper Settings to Program the Flash (slower but non-volatile) or SRAM (faster but volatile).