Intel/Altera Hardware Information

Introduction

The DE10-Lite hardware has been donated to Temple University for evaluation. This document outlines the design flow set up for some of this evaluation. There are two parts to this evaluation: a) a normal design flow where a student has access to the hardware, much in the same way as when using the Digilent hardware; and b) a *handsfree* set up, where various boards are connected to Temple lab PC's and students can remotely access them through a simulated interface. The latter is something that is part of a beta development by Intel.

Traditional Design Flow

Quartus is the software design suite provided by Intel/Altera. It was installed in the summer of 2020 on Temple's AWS Cloud9 virtual machine. The standard installation and design flow is GUI based (from a local PC). This GUI design flow was analyzed in order to create a *command line* flow for Cloud9.

As a resource, the design template for the DE10-Lite hardware was downloaded from Terasic. The template for the DE10-Lite created a top level module: ***DE10_LITE_Golden_Top***. I did some edits to this and created a Temple version in SystemVerilog: ***DE10_LITE_Temple_Top.sv***. It has many ***ifdef***'s in the i/o list so that a designer can easily enable i/o signals in the list. The template also included a default ***qsf*** file (see below). This file maps the i/o signal names to the pin numbers of the FPGA. This information was used to create a default ***qsf*** file. More details will be provided below.

Critical Project Files

The critical files to understand and modify for *command line* design flow are:

- *<project_name>*.qsf – *Quartus Settings File* – this is the file that contains most of the information of the design. It contains the FPGA type, i/o connections and the source file names and directories used for the design. Examples of the internals of this file will be given below.
- *< project_name >*.sdc – *Synopsys Design Constraint* file – this file contains the timing constraints of the i/o signals. It may also contain constraint definitions for any internal signals (false paths, etc.) It follows the format of the Synopsys design tools. If the constraint file is not complete, the synthesis operation will likely run with warnings; and the resultant output file will be functional.
- *< project_name >*.qpf – *Quartus Project File* – it's not clear what function this has, other than keeping track of version information. If it is omitted, the tools will create this file.

Shell Scripts and Runner File

The installed path of the Quartus tools is:

- **/data/courses/ece_2612/Intel_Altera/install_directory**

To set up the PATH variable to run the Quartus commands:

```
export QUARTUS_ROOTDIR=/data/courses/ece_2612/Intel_Altera/install_directory/quartus
export PATH=$PATH:$QUARTUS_ROOTDIR/bin
```

To synthesize, the following command is used, where *<project_name>* is connected to the critical files described in the previous section.

quartus_sh --flow compile *<project_name>*

A runner file was set up for the Quartus synthesize operation:

- **/data/courses/ece_2612/cloud9_runners/q_synthesize.run**

Its contents are below. Note that it is called when running a ***qsf*** file. Also the synthesis command defined above is used.

```
// Create a custom Cloud9 runner - similar to the Sublime build system
// For more information see http://docs.aws.amazon.com/console/cloud9/create-run-config
{
    "info" : "Synthesizing with Quartus ... $project_path$file_name",
    "script": [
      // set Quartus environment
      "export QUARTUS_ROOTDIR=/data/courses/ece_2612/Intel_Altera/install_directory/quartus",
      "export PATH=$PATH:$QUARTUS_ROOTDIR/bin",

      // set to exit if any non-zero returned
      "set -e",
      "quartus_sh --flow compile $file_base_name",
      // clean up vcd files
      "rm -rf *.vcd *.lxt2"
    ],
    "env" : {},
    // select any *.qsf extension
    "selector": "^.*\\.(qsf)$"
}
```

Defining and Including Design Hierarchy

In the traditional design flow the Temple version of the top level design is the SystemVerilog file: ***DE10_LITE_Temple_Top.sv***. Looking at an example use of this module:

```
…
//
//  ////////////////////////////////////////////////////////////////////////////

`define ENABLE_SW
`define ENABLE_LED
`define ENABLE_KEY

module DE10_LITE_Temple_Top(
```

```
        //////////// SDRAM: 3.3-V LVTTL //////////
`ifdef ENABLE_SDRAM
   output              [12:0]              dram_addr,
   output               [1:0]              dram_ba,
   output                                  dram_cas_n,
…
```

Note the **`*define*** statements to enable appropriate i/o data sets above the module definition. For this example design the switches (***ENABLE_SW***), LED's (***ENABLE_LED***) and push buttons (***ENABLE_KEY***) are used.

```
…
   // tie unused LED's low
   assign ledr[9:5] = 5'b0;

   // pushbuttons (key signal) are active low
   top_latch u_top_latch (.qa(ledr[0]), .qb(ledr[1]), .q_lvl(ledr[2]),
      .q_latch(ledr[3]), .q_ff(ledr[4]), .s(sw[0]), .r(sw[1]),
      .d(sw[2]), .ck(~key[0]));

endmodule
```

There are two statements inside the module statement: a) an assign statement to tie up the unused LED's and b) the instantiation of the design. In the example above the design is ***top_latch***.

The *Quartus Settings File* (extension ***.qsf***) contains most all of the critical design support information. Slices of an example file are shown here:

```
…
# ---------------------------------------------------------------------- #
# ----------------start of i/o assignments---------------------------------- #
# ----comment or uncomment the i/o signals needed in your top design-------- #
# ---------------------------------------------------------------------- #
# --------------------clocks section-------------------------------------- #
# ---------------------------------------------------------------------- #
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to max10_clk1_50
#set_location_assignment PIN_P11 -to max10_clk1_50
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to max10_clk2_50
#set_location_assignment PIN_N14 -to max10_clk2_50
#set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to adc_clk_10
#set_location_assignment PIN_N5 -to adc_clk_10
# ------------------------push button section---------------------------- #
# ---------------------------------------------------------------------- #
set_instance_assignment -name IO_STANDARD "3.3 V SCHMITT TRIGGER" -to key[0]
set_location_assignment PIN_B8 -to key[0]
set_instance_assignment -name IO_STANDARD "3.3 V SCHMITT TRIGGER" -to key[1]
set_location_assignment PIN_A7 -to key[1]
…
```

The characteristics and pin numbers of the i/o signals are defined in this file. As shown in this example the clock signals ***(*clk**) are not used so they remain commented out. The pushbuttons (***key***) are used, so they are uncommented. For completeness, all of the i/o signals are included

from the template file provided by Terasic, the board designer. The unused signals are commented out.

```
…
# ------------------end of i/o assignments---------------------------------- #
# ------------------------------------------------------------------------- #
# top level entity definition - this should match the name of the top module
set_global_assignment -name TOP_LEVEL_ENTITY DE10_LITE_Temple_Top
#
…
set_global_assignment -name PROJECT_OUTPUT_DIRECTORY output_files
# -------------------list of design files are here---------------------------- #
# ------------------------------------------------------------------------- #
set_global_assignment -name SYSTEMVERILOG_FILE top_latch.sv
…
set_global_assignment -name SYSTEMVERILOG_FILE DE10_LITE_Temple_Top.sv
…
```

After the i/o signals, the top level module name is defined, then the output directory and finally a list of source files for the design.

A *Synopsys design constraint* (***.sdf***) file is also required. I have created a default file from Terasic which defines the clock signals and their frequencies. Other signals are missing, but for the most part will create warnings in the synthesis process. For completion constraints on these signals can be added following the format defined in Synopsys documentation.

Synthesis Steps

The synthesis step is pretty straightforward for the Cloud9 design flow. The user should right click on the ***<project_name>.qsf*** file and select run (similar to the Xilinx/Vivado flow). Many output files are created in the ***output_files*** directory (as shown above in the *qsf* file). The two output files of interest are:

- ***<project_name>.sof*** – SRAM object file – binary design image to load into the configuration (volatile) memory of the FPGA.
- ***<project_name>.pof*** – programmer object file – binary design image to load into the flash (nonvolatile) memory of the FPGA.

These files should be downloaded to the local PC where the hardware will be connected.

Programming the Hardware

You can use either file to program the hardware. The ***sof*** is almost an instantaneous load. The ***pof*** takes ***about one minute to load*** because of the time required to write to the internal flash memory.

The quickest way to program the device on Windows is to use the Quartus software with the programming tool. Connect the hardware to a USB port and select the appropriate object file to load into the device.

The remainder of this section covers alternative programming solutions (command line vs. GUI based) that are more time efficient for loading multiple designs.

With either programming method, the USB bit blaster driver must be properly installed. The steps for this are on the Intel/Altera web pages. I had problems loading the driver on my Windows 10 PC and needed to follow the steps on this link:

https://www.intel.com/content/www/us/en/programmable/support/support-resources/knowledge-base/component/2017/update-driver-software---usb-blaster--windows-encountered-a-prob.html

Once the driver in installed and the hardware connected to the USB port, my installed Quartus tool set had two possible ***JTAG*** chains to follow: a) the local USB bit blaster and b) some remote server. You can see the chains by running the command:

- **%QUARTUS_ROOTDIR%\bin64\quartus_pgm -l**

```
...
Info: Command: quartus_pgm -l
1) USB-Blaster [USB-0]
2) Remote server 10.8.0.3:33533: Unable to connect
Info: Quartus Prime Programmer was successful. 0 errors, 0 warnings
...
```

I chose to remove this remote server. However, it must be done from the programmer GUI with these steps:

- In Programmer, click Hardware.
- In the Hardware Setup dialog box, click the JTAG Settings tab.
- Select the existing remote server and click Remove Server.

Now re-running:

- **%QUARTUS_ROOTDIR%\bin64\quartus_pgm -l**

```
...
Info: Command: quartus_pgm -l
1) USB-Blaster [USB-0]
Info: Quartus Prime Programmer was successful. 0 errors, 0 warnings
...
```

Note the Remote Server is gone and only the USB-Blaster is there. You can test to see if it will identify the device in this chain:

- **%QUARTUS_ROOTDIR%\bin64\quartus_pgm -a**

```
...
```

```
Info: Command: quartus_pgm -a
Info (213045): Using programming cable "USB-Blaster [USB-0]"
1) USB-Blaster [USB-0]
  031050DD   10M50DA(.|ES)/10M50DC
...
```

If you do not want to remove the remote server, you need to add the following option to the ***quartus_pgm*** command line: ***-c "USB-Blaster [USB-0]"***. When there is only one JTAG chain, the command option is not needed and it will default to the only chain available.

Now that everything is set up and tested, this is the command to program the hardware (binary files must be in the executable path or added to the command line).

- **`%QUARTUS_ROOTDIR%\bin64\quartus_pgm -m jtag -o p;`*<sof or pof file>***

The sof load is quick, the pof takes about one minute – so be patient.

Finally, if you are doing a lot of programming, you can associate these file extensions with this command and automatically run the command when you double click or open a ***sof*** or ***pof*** file. Here are the steps for that setup (it will alter your registry):

- Open a command prompt in the administrator mode.
- `assoc .sof=quartus_prog_files`
- `assoc .pof=quartus_prog_files`
- `ftype quartus_prog_files=%QUARTUS_ROOTDIR%\bin64\quartus_pgm -m jtag -o p;"%1"`

The first two commands sets up an association of the file extensions with a file type (my definition). The last command defines a command to run when opening files with this file type.

Close your command prompt and test by opening a ***sof*** or ***pof*** file (with the hardware plugged in).

Handsfree Design Flow

This section covers the design flow from Cloud9 to the *handsfree* systems in the Temple hardware lab. What are the *handsfree* systems? They are basically remote hardware systems. A hardware lab has been set up at Temple linking various PC's to DE10-Lite boards – one to one. This section describes the steps needed to create a hardware design to run on these remote systems. A GUI is created on the remote machine that simulates the i/o interface – while the design is actually running on the connected hardware.

You must have a Temple login to connect to these machines. Contact the IT support group for instructions as how to log into these PC's. The steps below assume that you have access and can log into one of these systems.

Critical Project Files

The critical files are the same as in the Traditional Design Flow described above. I have added a ***hf_*** prefix to the additional files so that both traditional and the handsfree design flow can be supported from the same directory.

Shell Scripts and Runner File

The scripts and runner files are the same as in the Traditional Design Flow described above.

Defining and Including Design Hierarchy

Another level of hierarchy has been added to the *handsfree* flow. Basically the top level design in the traditional flow (***DE10_LITE_Temple_Top.sv***) is instantiated inside a *handsfree* top module (***hf_top.sv***). An example module is shown here:

```
…
module hf_top(input logic max10_clk1_50);

        // Peripheral interconnect signals
        logic [9:0] leds;       // output signal to the LED's
        // output signals to the seven segment digits
        logic [7:0] seg7_0, seg7_1, seg7_2, seg7_3, seg7_4, seg7_5;
        logic [9:0] sws;        // input signals from the switches
        logic [1:0] pbs;        // input signals from the pushbuttons

        // Parameter interconnect signals
        logic [31:0] param1, param2, param3;

        // User instantiates design below
        // instantiate the design
        DE10_LITE_Temple_Top u_design (.ledr(leds), .key(pbs), .sw(sws[2:0]));

        // tie up all of the unused outputs
        assign seg7_0 = 8'b11111111;
        assign seg7_1 = 8'b11111111;
        assign seg7_2 = 8'b11111111;
        assign seg7_3 = 8'b11111111;
        assign seg7_4 = 8'b11111111;
        assign seg7_5 = 8'b11111111;

        // IP to allow simple user design interfacing with development kit
        pin_ip platform_designer_pin_ip (.clock(max10_clk1_50), .leds,
                .seg7_0, .seg7_1, .seg7_2, .seg7_3, .seg7_4, .seg7_5,
                .sws, .pbs, .param1, .param2, .param3);

endmodule
…
```

Note that the only actual hardware input to this device is the system clock. So if one were to observe the remote hardware running a program, all of the designed i/o's are not connected to the real hardware, but rather intercepted by the ***pin_ip*** module. This module has been supplied by Intel. It is a JTAG interface to the USB port that will communicate with a GUI to replicate the behavior of the i/o's from the traditional design (***DE10_LITE_Temple_Top***).

The ***pin_ip*** and supporting modules are in the ***handsfree*** path of the design tree. Thus, they can be easily reused for other labs.

Now let's look at the differences of the *Quartus Settings File* (extension ***.qsf***) for this design flow. Slices of an example file (with the ***hf_*** prefix) are shown here:

```
…
# --------------------------------------------------------------------
#
# ----------------start of i/o assignments---------------------------
#
# ----comment or uncomment the i/o signals needed in your top design--------
#
# --------------------------------------------------------------------
#
# --------------------clocks section-------------------------------------
#
# --------------------------------------------------------------------
#
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to max10_clk1_50
set_location_assignment PIN_P11 -to max10_clk1_50
# --------------------------------------------------------------------
#
# ------------------end of i/o assignments-----------------------------
#
# --------------------------------------------------------------------
#
# top level entity definition - this should match the name of the top module
set_global_assignment -name TOP_LEVEL_ENTITY hf_top
#
…
set_global_assignment -name PROJECT_OUTPUT_DIRECTORY output_files
# -------------------list of design files are here--------------------------
- #
# --------------------------------------------------------------------
#
set_global_assignment -name SYSTEMVERILOG_FILE top_latch.sv
…
set_global_assignment -name SYSTEMVERILOG_FILE DE10_LITE_Temple_Top.sv
set_global_assignment -name SYSTEMVERILOG_FILE ../handsfree/pin_ip.sv
set_global_assignment -name QSYS_FILE ../handsfree/internal_pin_if.qsys
set_global_assignment -name SYSTEMVERILOG_FILE hf_top.sv
…
```

Because this setup will only use the clock signal as an input and all of the other i/o's are simulated, only that signal is in the i/o list. The top level entity has been changed to ***hf_top***. The other design files (***DE10_LITE_Temple_Top*** and below) are still included. However, the *handsfree* path has been added with the ***pin_ip*** information. The last SystemVerilog file in the design file list is the top module, ***hf_top.sv***.

A *Synopsys design constraint* (***hf_*.sdf***) file is also required. This one requires a definition of the clock frequency or period. Here is the necessary definition:

- create_clock -period 20ns [get_ports max10_clk1_50]

Synthesis Steps

The synthesis step is the same as the traditional flow. The user should right click on the ***hf_<project_name>.qsf*** file and select run (similar to the Xilinx/Vivado flow). Many output files are created in the ***output_files*** directory (as shown above in the *qsf* file). The two output files of interest are:

- ***hf_<project_name>.sof*** – SRAM object file – binary design image to load into the configuration (volatile) memory of the FPGA.
- ***hf_<project_name>.pof*** – programmer object file – binary design image to load into the flash (nonvolatile) memory of the FPGA.

These files should be downloaded to the local machine, then copied/moved to a shared drive like OneDrive as described below.

Programming the Hardware

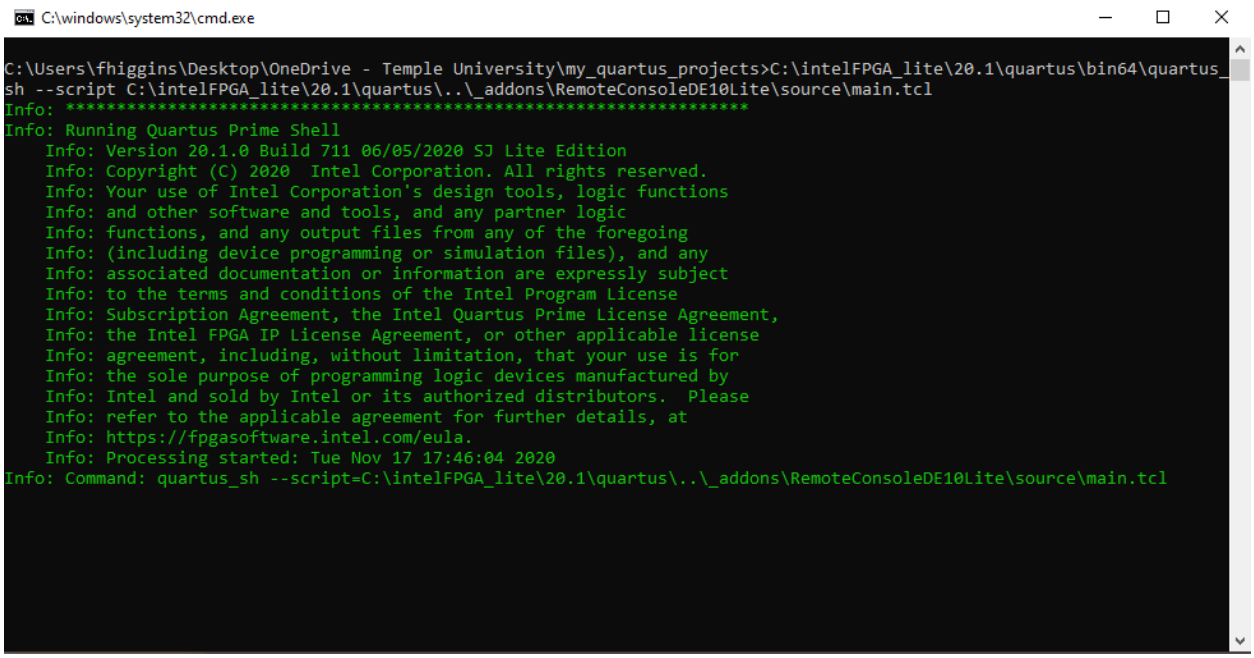
Here are the steps for setting up your ***OneDrive*** shared drive:

- Create a directory to hold your sof and pof design files – I suggest something like ***quartus_projects***.
- Under ***quartus_projects*** create an ***output_files*** directory. Copy or move your ***sof*** and ***pof*** files there.
- Create (or copy) a batch file called ***run_hf.bat***. The contents are (all one line):

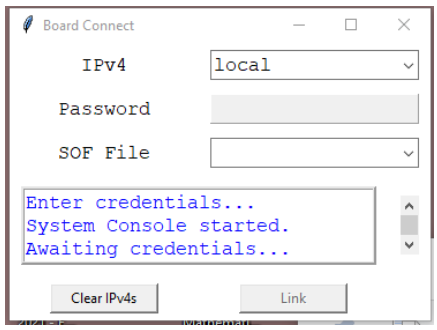
```
%QUARTUS_ROOTDIR%\bin64\quartus_sh --script
%QUARTUS_ROOTDIR%\..\_addons\RemoteConsoleDE10Lite\source\main.tcl
```

Next login to one of the Temple lab PC's that have the DE10-Lite hardware. After logging in follow these steps:

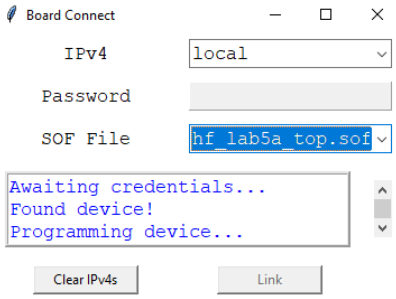
- Connect your ***OneDrive*** to this PC.
- Open the file explorer and go to the ***quartus_projects*** directory.
- Double click on ***run_hf.bat*** batch file. This will run a tcl script and show a command prompt screen as shown here:



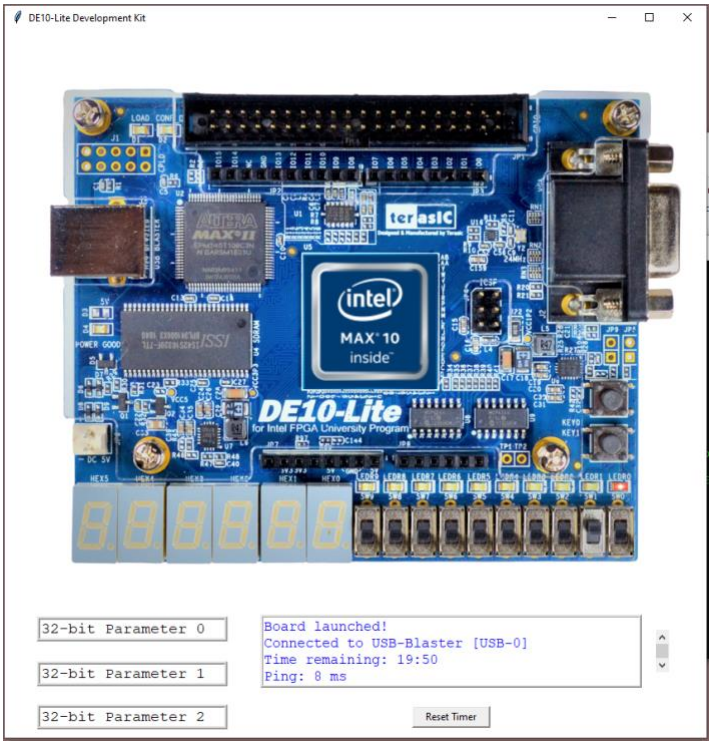
- After a few seconds this should start up a GUI:



- You can ignore the password window. Select the ***SOF File*** dropdown and the list of files in your ***output_files*** directory should be visible. Select the file of interest and select the ***Link*** button. It should start programming the device.



- Then the hardware GUI should appear and your design will be running on the remote hardware.



- When you are finished close this window, and then the command prompt window should close.
- To test another design, restart the batch file and select another ***sof*** file.

You should only use ***sof*** files for the remote hardware, as there is no point in storing you design in the nonvolatile flash memory. Also only load the *handsfree* designs (***hf_*** prefix). If you load a traditional design you will not be able to see it operate remotely.