Xilinx Kintex-7
Emulation System Tutorial
Designing and Debugging

Fengbo Ren (renfengbo@gmail.com)

6/28/2012
Preface

In this tutorial, you will go through the basic design flow of Xilinx Kintex-7 emulation system (KC705) with an example design. You will learn how to:

1. creates or imports your Verilog design;
2. maps your design onto FPGA;
3. selects and connects the resources on the board;
4. utilize the virtual I/O and integrated logic analyzer to debug or control your design during hardware simulation through ChipScope software interface.
1. Designing

**Initial Setup (Kodiak.ee.ucla.edu)**

a. In your working directory, copy over the Verilog code of the example design.

   ```bash
   lnx[0]: mkdir rtl
   lnx[0]: cp /mnt/designs/fren/project/FPGA/Kintex7/example_design/rtl_led/* rtl/
   ```

b. Open “Xilinx ISE Design Suite”.

   ```bash
   lnx[0]: source /opt/Xilinx/14.3/ISE_DS/settings64.sh
   lnx[0]: ise &
   ```

c. Create a “new project” → Name it as “counter_led” → “Next”.

![Image of ISE Project Navigator and New Project Wizard]
d. Set “Evaluation Development Board” to be “Kintex-7 KC705 Evaluation Platform” → “Next” → “Finish”. Now you should see the FPGA part name in the “Hierarchy” panel with no design in it.

![New Project Wizard](image)

**IP Core Generation**

In this tutorial, we start the design entry by first creating all the Xilinx IPs to be used later in the design. First, we add a digital clock management (DCM) module for on-chip clock management. By default, all the generated files will be in the “ipcore_dir” folder under the project directory.

a. Click on the “new source” icon → select source type as “IP (CORE Generator & Architecture Wizard)” → name the module as “clk_gen” → check “Add to project” → “Next”.

![IP Core Generation](image)
b. Select the IP as “FPGA Features and Design” → “Clocking” → “Clocking Wizard” → “Next” → “Finish”.
In the pop-up window, you can configure a variety of parameters of the DCM module to meet your need. Note that the DCM module has no crystal oscillator built-in. So it requires a source clock with fixed frequency as input from the outside of the FPGA. But it can regenerate multiple output clocks with different user specified frequencies. In this tutorial, we choose to use the clock generator integrated on the board as our clock source. It provides a 200MHz differential clock signal.

c. Set “Input Freq (MHz)” to DCM as “200” → select “Source” as “Differential clock capable pin” → “Next”.

d. Check “CLK_OUT2” to allow DCM to create two output clocks (more clocks with different frequencies can be added) → set “Output Freq (MHz)” to “10” and “20”, respectively → “Next” → uncheck “RESET” → “Generate”.

The phase is calculated relative to the active input clock.

<table>
<thead>
<tr>
<th>Output Clock</th>
<th>Output Freq (MHz)</th>
<th>Phase (degrees)</th>
<th>Duty Cycle (%)</th>
<th>Drives</th>
<th>Use Fine Ps</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK_OUT1</td>
<td>10</td>
<td>10.000</td>
<td>50.000</td>
<td>BUFG</td>
<td></td>
</tr>
<tr>
<td>CLK_OUT2</td>
<td>20</td>
<td>20.000</td>
<td>50.000</td>
<td>BUFG</td>
<td></td>
</tr>
<tr>
<td>CLK_OUT3</td>
<td>100.000</td>
<td>0.000</td>
<td>50.000</td>
<td>BUFG</td>
<td></td>
</tr>
<tr>
<td>CLK_OUT4</td>
<td>100.000</td>
<td>0.000</td>
<td>50.000</td>
<td>BUFG</td>
<td></td>
</tr>
<tr>
<td>CLK_OUT5</td>
<td>100.000</td>
<td>0.000</td>
<td>50.000</td>
<td>BUFG</td>
<td></td>
</tr>
<tr>
<td>CLK_OUT6</td>
<td>100.000</td>
<td>0.000</td>
<td>50.000</td>
<td>BUFG</td>
<td></td>
</tr>
<tr>
<td>CLK_OUT7</td>
<td>100.000</td>
<td>0.000</td>
<td>50.000</td>
<td>BUFG</td>
<td></td>
</tr>
</tbody>
</table>

The module generation will take a while. After it is done, you should be able to see a “clk_gen” module added to the project. The instantiation template for it is also available. Note that the the output port “locked” is to indicate the status of the PLL. All output clocks are valid when “locked = 1”.

Next, we are going to create another three modules in the same way for debugging purpose. These three modules are designed to work with the “ChipScope Pro Analyzer” software interface and are very useful in debugging the hardware simulation.
e. Click on the “new source” icon → select source type as “IP (CORE Generator & Architecture Wizard)” → name the module as “vio_async” → check “Add to project” → “Next” → Select the IP as “Debug & Verification” → “ChipScope Pro” → “VIO” → “Next” → “Finish”.

![Virtual Input/Output (VIO) Module](image)

The Virtual Input/Output (VIO) module can be conceptualized as the virtual interface between the chip and the user’s monitor as shown in the above figure. It allows the designer to monitor any signal in the design that is connected to the input of VIO in a “real-time” fashion, as well as to set the value of any signal that is driven by the output of VIO.

f. In the pop-up configuration window, check both “Enable Asynchronous Input Port” and “Enable Asynchronous output Port” (synchronous mode is also supported) → set the port “Width” to be “5” and “4” bits, respectively → “Generate”. The “vio_async” module will be added to the project.
g. Click on the “new source” icon → select source type as “IP (CORE Generator & Architecture Wizard)” → name the module as “ila_dst” → check “Add to project” → “Next” → Select the IP as “Debug & Verification” → “ChipScope Pro” → “ILA” → “Next” → “Finish”.

Different form VIO, the Integrated Logic Analyzer (ILA) module can capture the waveform of any signal that is connected to the “DATA” port of ILA by storing its sample values in block RAMs (BRAMs) for a certain time window during the hardware simulation. The time of capturing is determined by a trigger scheme—the time window becomes transparent whenever the trigger signal connected to the “TRIG” port of ILA matches the trigger pattern pre-defined by the designer in the “ChipScope Pro Analyzer” GUI. This allows the designer to observe the waveform of the signal of interest around any critical time point.

h. In the pop-up configuration window, set “Number of Trigger Ports” to “1” (multiple trigger ports are supported) → set “Sample Data Depth” to “2048” → check “Data Same As Trigger” (in this case, the trigger signal is the signal of interest itself) → “Next”.

ILA (ChipScope Pro - Integrated Logic Analyzer)
i. Set “Trigger Port Width” to “10” bits → “Generate”. The “ila_dst” module will be added to the project.

Note that both VIO and ILA modules have an inout “CONTROL”. This port is designed to work with the Integrated Controller (ICON) module for interfacing with the computer through JTAG connection. Therefore the ICON module is required for every VIO or ILA module instantiated in the design.

j. Click on the “new source” icon → select source type as “IP (CORE Generator & Architecture Wizard)” → name the module as “icon” → check “Add to project” → “Next” → Select the IP as “Debug & Verification” → “ChipScope Pro” → “ICON” → “Next” → “Finish”.

k. In the pop-up configuration window, set “Number of Control Ports” to “3” (In the example design, we will instantiate two “vio_asyn” and onw “ila_dst”. The number of instance of VIO and ILA modules in the design should match the “Number of Control Ports” set here) → “Generate”. The “icon” module will be added to the project.

**Design Input**

a. Click on the “Add Source” icon → select “rtl/top.v” → “Open” → “OK”.

The tool will recognize the hierarchy and add the imported design to the design tree properly. Double-click on “top”, you will be able to review and modify design in the built-in text editor. When design is saved, the tool will automatically check and display any syntax error in the “Error” Tab.
The example design includes two synchronized counters running at different frequencies and instantiates two VIO and one ILA modules as the virtual control and monitor units. It also utilizes the push buttons and LEDs on the board as the other real control and display units.

The block diagram of the example design is shown as follows.
2. Mapping onto FPGA

Constraint File

Note that for general designs, the designers’ input is required for only two types of constraints: timing and I/O constraints. A detail user guide on all the constraints can be found [here](#).

For the timing constraint, any clock signal that is directly from the outside source has to be defined. When DCM is used, the tool will automatically generate the timing constraints for all its output clocks, as long as the input clock source is defined. In addition, input and output delays have to be defined for all the unregistered digital I/Os except for clocks. Since all the I/O boxes in FPGA have dedicated flip-flop, it is highly recommended to register all the inputs and outputs of the top level design for better timing (in this case, constraints on input and output delays are no longer needed).

In the KC705 system, all of the FPGA I/Os have been connected to the different resources on the PCB board. Therefore, to connect the design with specific resources, you need to assign the top level ports of the design to the corresponding I/O pins of the package that connects to the desired resource. This information has to be defined in the I/O constraints. The mapping information between package pins and hardware resources on board can be looked up from the file “HW-K7-KC705”.

Design constraints can be added either from GUI or text editor following the specific formats. For the first time, we recommend to enter all the constraints from the GUI. Later on, you can modify these constraints in text editor following the generated format.

a. Select “top.v” → in the “Processes” panel → under “User Constraints” → double-click on “Create Timing Constraints”. The timing constraint GUI will pop-up after running pre-synthesize for a while. Click “Yes” to the pop-up question so that a constraint file (“top.ucf”) will be added to the project.
b. Double-click on the signal “sys_clk_p” from the “Unconstrained Clocks” table → Set the clock period “Time” as “5 ns” (since our source clock is 200 MHz) → “OK”.

c. Double-click on the signal “sys_clk_n” from the “Unconstrained Clocks” table → select “Relative to other period TIMSPEC” → Set “Reference TIMSPEC” to “TS_sys_clk_p” → Set “Phase” as “Minus 2.5 ns” → “OK”.

Therefore, a differential pair of clock signal is defined. The other “Unconstrained Clocks” listed by the tool can be ignored as long as the source clock of DCM is defined. Save and close the “Timing Constraints” tab. Double-click on “top.ucf”. You should see the timing constraints are added to the constraint file with a certain formats.
d. Under “User Constraints” → double-click on “I/O Pin Planning (PlanAhead)-Pre-Synthesis”. The “PlanAhead” tool will pop-up and automatically read in the design. You should see the bottom-view of the FPGA package showing the pin locations. Also, all the top level ports should be listed in the “I/O Ports” panel. Type the pin name into the column “Site” in order (look it up from “HW-K7-KC705”) and make sure they map to the correct port name that you want to assign to. For the example design, pins (I/Os) are assigned as follows.

<table>
<thead>
<tr>
<th>I/O Ports</th>
<th>Name</th>
<th>Dir</th>
<th>Neg</th>
<th>Diff</th>
<th>Site</th>
<th>Bank</th>
<th>I/O Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>All ports</td>
<td>button[0]</td>
<td>Input</td>
<td></td>
<td></td>
<td>G12</td>
<td>18</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>All ports</td>
<td>button[1]</td>
<td>Input</td>
<td></td>
<td></td>
<td>AA12</td>
<td>33</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>All ports</td>
<td>button[2]</td>
<td>Input</td>
<td></td>
<td></td>
<td>AB12</td>
<td>33</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>All ports</td>
<td>button[3]</td>
<td>Input</td>
<td></td>
<td></td>
<td>AD7</td>
<td>34</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>All ports</td>
<td>button[4]</td>
<td>Input</td>
<td></td>
<td></td>
<td>AC5</td>
<td>34</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>All ports</td>
<td>led[0]</td>
<td>Output</td>
<td></td>
<td></td>
<td>AB8</td>
<td>33</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>All ports</td>
<td>led[1]</td>
<td>Output</td>
<td></td>
<td></td>
<td>AA8</td>
<td>33</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>All ports</td>
<td>led[2]</td>
<td>Output</td>
<td></td>
<td></td>
<td>AC9</td>
<td>33</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>All ports</td>
<td>led[3]</td>
<td>Output</td>
<td></td>
<td></td>
<td>AB9</td>
<td>33</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>All ports</td>
<td>led[4]</td>
<td>Output</td>
<td></td>
<td></td>
<td>AE26</td>
<td>13</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>All ports</td>
<td>led[5]</td>
<td>Output</td>
<td></td>
<td></td>
<td>G19</td>
<td>17</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>All ports</td>
<td>led[6]</td>
<td>Output</td>
<td></td>
<td></td>
<td>F18</td>
<td>17</td>
<td>LVCMOS18</td>
</tr>
<tr>
<td>All ports</td>
<td>led[7]</td>
<td>Output</td>
<td></td>
<td></td>
<td>F16</td>
<td>18</td>
<td>LVCMOS18</td>
</tr>
</tbody>
</table>

Note that the tool automatically recognize the clock signal as a differential pair—once port “sys_clk_p” is assigned to pin “AD12”, port “sys_clk_n” will be automatically assigned to the differential pin “AD11”.

e. Click on “File” → “Save Design” → go back to “ISE 13.4i” and look at “top.ucf” again. I/O constraints should be attached to the file.

**Important note:** You should see both the “LOC” and “IOSTANDARD” attributes added to every top level port in the design. Otherwise, it will fail Mapping. If any of the two attributes of any port is missing in the constraint file, the tool might be buggy. To avoid this, do the following instead: Click on “File” → “Export” → “Export I/O Ports” → uncheck “CVS” and check “UCF” → import the constraints into some other place and copy over all the contents into “top.ucf”. An example of the correct constraint file can be found at:

/mnt/designs/fren/project/FPGA/Kintex7/example_design/rtl_led/top.ucf
Bit File Generation

Bit file contains all the configuration bits and is required for FPGA programming. It is generated after "Synthesis" → "Translate" → "Map" → "Place and Route".

a. Select "top.v" → in the "Processes" panel double-click on "Generate Programming File". The tool will automatically run all the necessary flows to generate the bit file. The generated bit file ("top.bit") will be under the project folder.

If errors occur in any part of the flow, bit file will not be generated. Error messages can be checked in the "Errors and Warnings" session of the "Design Summary" Tab.
Programming FPGA

a. Connect the KC705 board to your local computer (which has Xilinx ChipScop Pro installed) and power it on as shown in the following.
b. Launch “ChipScope Pro Analyzer” → click on the “Open Cable/Search JTAG Chain” icon. The tool will automatically detect the device connected. → click on “OK” to confirm the detection result.

c. Click on “Device” → “DEV 0 MyDevice 0 (XC7K325T)” → “Configure” → in the pop-up window, click on “Select New File” and choose the bit file (“top.bit”) generated by ISE (You might need to access this file through the network disc) → “OK”.

   ![ChipScope Pro Analyzer][1]

The programming should start right away and will take a while. After this is done, the example design has been successfully mapped onto your FPGA board. Try pushing the center switch button (“SW5”) on the board, you should see “LED[0]” flashes correspondingly.

d. Click on “File” → “Save Project”. Save the ChipScope project as “counter_led.cpj” into your design directory so that your settings will be automatically loaded next time.

   ![ChipScope Pro Analyzer][1]
3. Debugging Hardware Simulation

Virtual Control and Monitoring through VIO

After the FPGA is successfully programmed, you should see all the VIO and ILA modules instantiated in the design automatically detected and listed in the “Project” panel of the “ChipScope Pro Analyzer” GUI (two VIO instances and one ILA instance in the example design). Be aware of that the default names of these instances are not passed from the design.

a. Double-click on “VIO Console” under “MyVIO0” and “MyVIO1”, respectively. → Set “JTAG Scan Rate” to “250 ms”, so that any change will be reflected at a rate of 4 Hz.

In the open-up GUI, the input and output signals to the VIO instances are listed in blue and green, respectively. The GUI allows you to monitor the values of blue signals from the chip and set the values of green signals to the chip through the JTAG connection. Again, the default names of these signals are not passed from the design. You could right-click and rename them as above for the convenience. To set signal value in the “Value” field, you could double-click and typing in the value, or right-click on the signal, select “Type” as “Push Button” so that you can toggle the value by a single click.

b. Enable “and_button_north” and “and_button_south” → try pushing the north and south switch buttons on the board, you should see “LED[1]” and “LED[2]” flashing both on the board and in GUI correspondingly.

c. Enable “counter_0_en” and “counter_1_en”, you should see “LED[5:7]” flashing with descending frequencies on the board, as well as “counter_1[22:23]” and “counter_0[22:23]” toggling in GUI.
You may notice the lagging behavior in GUI. This is due to the frequency difference between the actual signals and the JTAG Scan Rate. Note that “clk_locked” is always high, indicating the clock signals out of DCM are valid.

**Waveform Capturing through ILA**

The waveform capturing of the signal of interest can be triggered either manually or by pattern matching of the trigger signal.

a. Double-click on “Trigger Setup” under “MyILA2” → in the “Match” panel, set “Function” to “==”, set “Radix” to “Unsigned” and set “Value” to “1023” → in the “Capture” panel, set “Depth” to “2048” and set “Position” to “1023”.

Since in the example design, the trigger signal is set to be the signal of interest itself, these settings are specified to capture a total 2048 samples of the signal (“counter_0[9:0]”) with the value of “1023” being the 1023 sample in position out of 0 to 2047. In other words, the trigger pattern is “counter_0[9:0]==10’d1023”, and the capturing starts from 1023 samples ahead of the trigger event.

b. Double-click on “Waveform” → in the open-up window, right-clicking on all signals → “Move to Bus” → “New Bus” → right-click on “DataPort” → set “Bus Radix” to “Unsigned” → click on the “Apply Settings and Arm Trigger” icon to start the capturing process. You should see the waveform as what we specified.
b. Double-click on "Bus Plot" → check "DataPort".

Since a 10-bit counter has a period of 1024 clock cycles, you should see exactly 2 periods of the counter signal get captured in the plot (trigger value 1023 is in position 1023).

c. Click on the "Instant Trigger" icon to trigger the capturing process instantly. Notice the waveform will change randomly.