FTUNSHADES 2

UFF 3.7

Beginner's Guide

UFF

Table of Contents

About This Document 1
Project Preparation 2
  Pin File 4
  Ucf File 4
  Bit and LL Files 6
  VCD and Dat Files 9
  Appendix A 14
First Steps on UFF 18
  UFF Workspace 18
  Setting Up The Design 19
  Campaign Runner 23
  Hardware Debugger 24
  Closing The Tool 27
Working with partial bitstreams 28
  Design preparation for partial reconfiguration 28
  Example design 28
  Adding multiple bitstreams to a user project 29
  Working with multiple bitstreams in the debugger 30

About This Document

This manual is a guide about how to use the UFF (User Friendly Framework, is executed as an extension of an HTTP server such as Apache. This enables it to be present "in the cloud" as a service provided by a web server) web server interface. But firstly, the user should to get all files needed to be able to do a campaign (A campaign is an automatic test of an emulated circuit during which a number of errors will be injected in the SEU target FPGA to an user-defined set of registers in an user-defined set of cycles. Then its output will be compared to that of the other where no errors were introduced) using the FTU2 (Fault Tolerant UNiversidad de Sevilla HArdware DEbugging System 2) platform. These files are described in the first chapter called "Project Preparation". For a greater knowledge about how to do an injection, how to debug the design behavior before injection and after injection and also to analyze all the results that the TNT (TNT is the suite of Test aNalysis Tools for the FTU system) system tool can get, see the chapter called "First Steps On UFF".

In this guide some tools implemented in TNT are used to get certain files from the UFF web interface. Therefore, it is recommended to request a user account to access the application. Such petition can be made via the contacts listed on the website: FTU2 SITE.

For users who prefer to work through a console there is the possibility of using TNT Scripting Guide (tnt-book3.7) that is available in the website: FTU2 DOC.

In the second chapter it explains the basic use about the UFF web server interface to be able to run a campaign with Fault Injection (FI) over the users design. From the files obtained in the previous chapter, firstly it is necessary to make a setting up the design and then the campaign is carried out. The user can also debug the design using the hardware debugger that FTU2 offers.

This is the user manual for the 3.7 version of the UFF web interface as it exists at the end of 2017.

Project Preparation

This chapter describes all files that we need for the FTU2 tool use. Firstly, we have to explain our work environment which made this guide. Finally, it appears a list with all file descriptions.

We are going to work using the ISE 14.7 tool from Xilinx to create the example project and we use the ISim (simulator tool) from Xilinx too. Our preferred language to design is VHDL. Designs have to fit in a XC5VFX70T device (Universidad de Sevilla FPGA model available) and it is limited to 512 I/Os. We work under a Linux SO distribution (CentOS 6.6 version).

The goal is to get three files mainly: the bitstream (.bit), the logic location (.ll) and the I/O vectors (.dat), without these files it is not possible to run a campaign. In addition, there are other files that are needed to create some of the files mentioned above.

These files must be written by the user:

name_design.vhd
: It's the design source code (in VHDL language)

tb_name_design.vhd
: The Test Bench file is needed to simulate the design and get the VCD (stimuli file)

name_design.pin
: A pin file is a declarative file for the clock, inputs, bidirectional and output signals of the target design. This file represents the order in which the pins will be implemented in the target FPGA.

These files are generated by a tool:

name_design.ucf
: The User Constraints File is an ASCII file specifying constraints on the logical design. These constraints affect how the logical design is implemented in the target device.

name_design.vcd
: Value Change Dump is an ASCII-based format for dumpfiles. By simulating the previously created design, the user may generate this file with the values of all I/O pins during simulation.

name_design.bit
: A bitstream is a stream of data that contains location information for logic on a FPGA. A bitstream configures the target FPGAs to emulate any circuit.

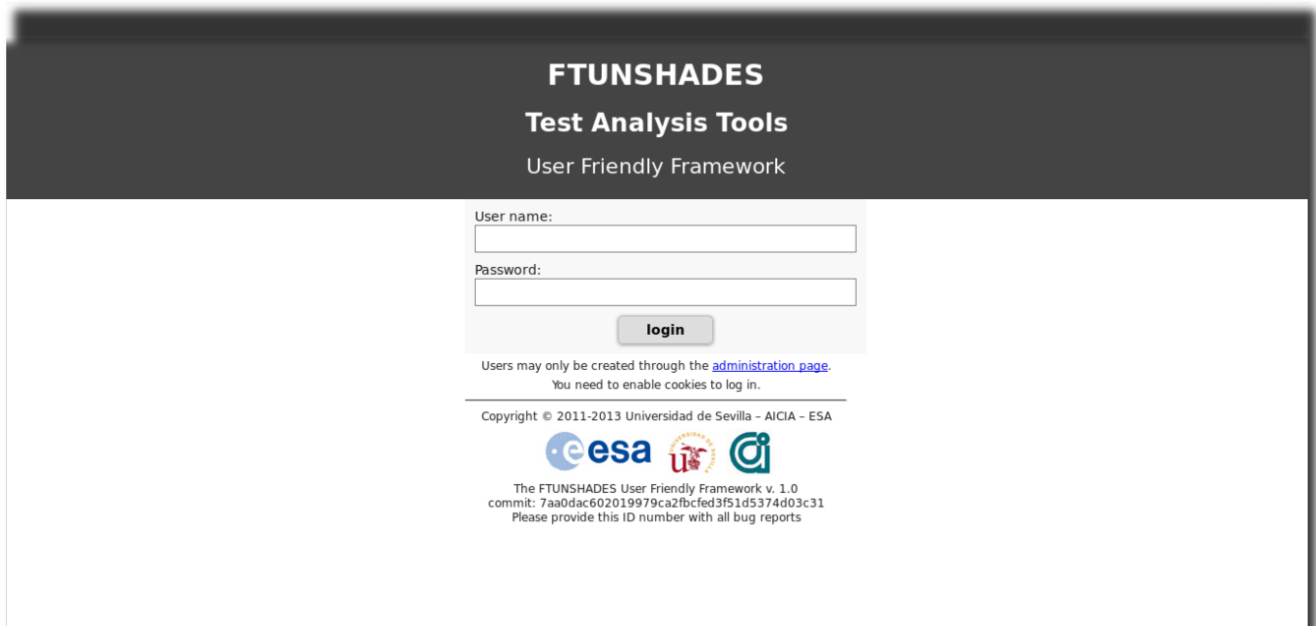
name_design.ll
: The logic location file, which indicates the bitstream position of storage elements such as latches, flip-flops, and IOB inputs and outputs.

name_design.dat
: The VCD file contains the set of stimuli and from this data it's possible to create the FTU2 I/O vectors file.

So let's go to get all previously described files. We are going to use an example design in this

manual that will be used in the next chapter too. It's a simple 8 bit counter and you can see its source code in Appendix A. In this manual, all files will be called “counter8bit.*”.

First, log in into the UFF SITE:



And then, the user must create a project:

Pin File

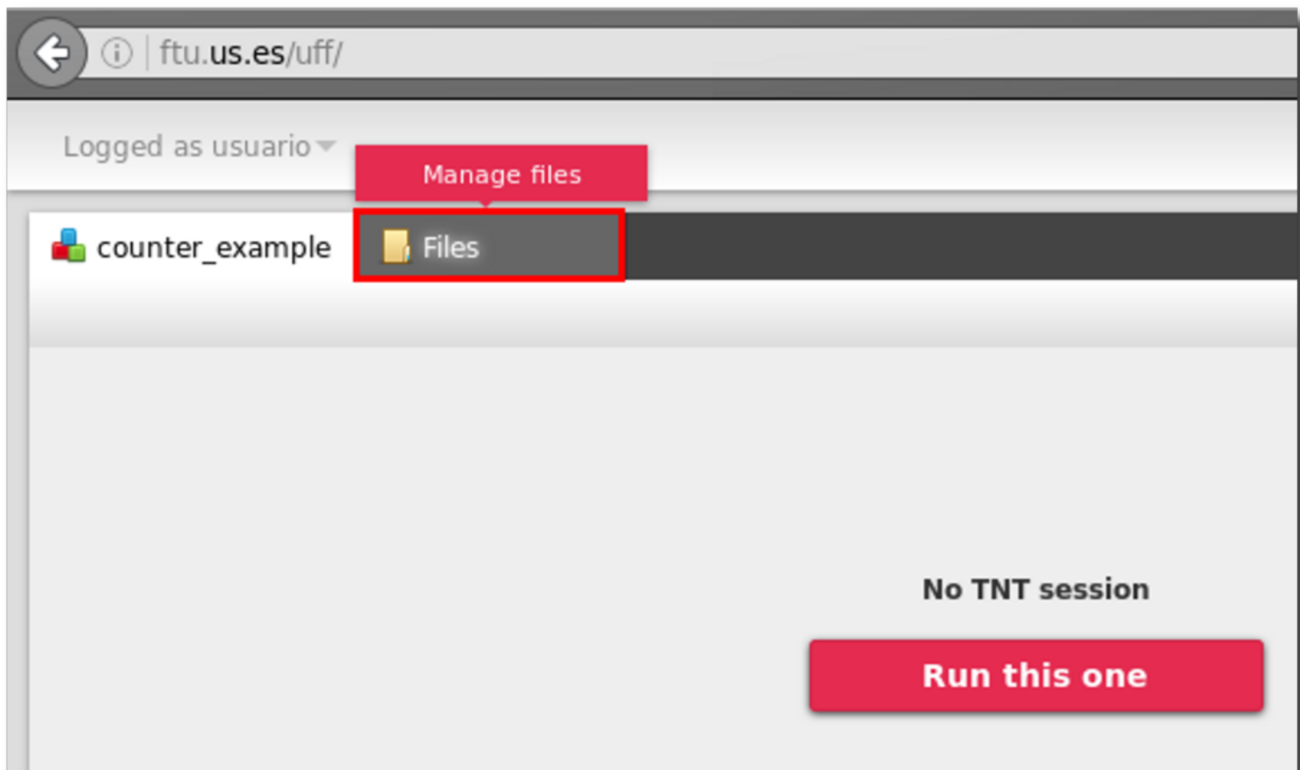
To create the pin file, the user can utilize any text editor to write the inputs, bidirectional and output signals of the target design. For this example, the pin file is shown below:

```
--control
clk
--input
rst_high
enable
--bidir
--output
data_out 7 0
```

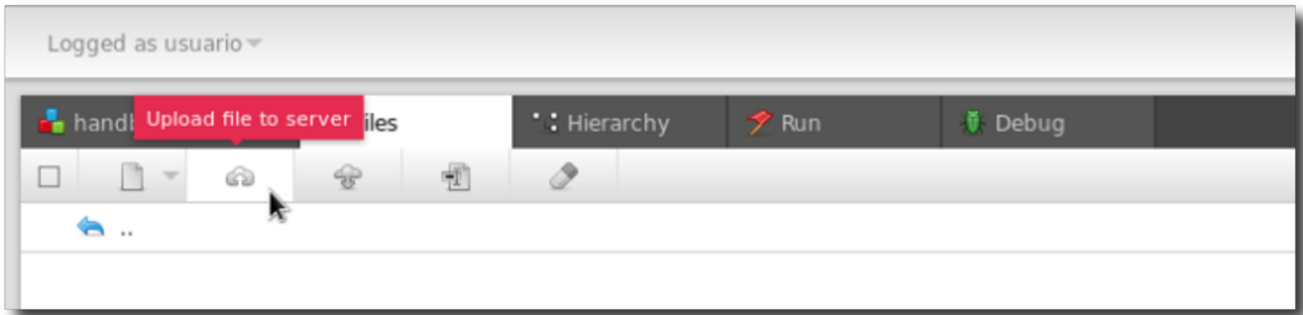
Save the file to get the first file called “counter8bit.pin”.

Ucf File

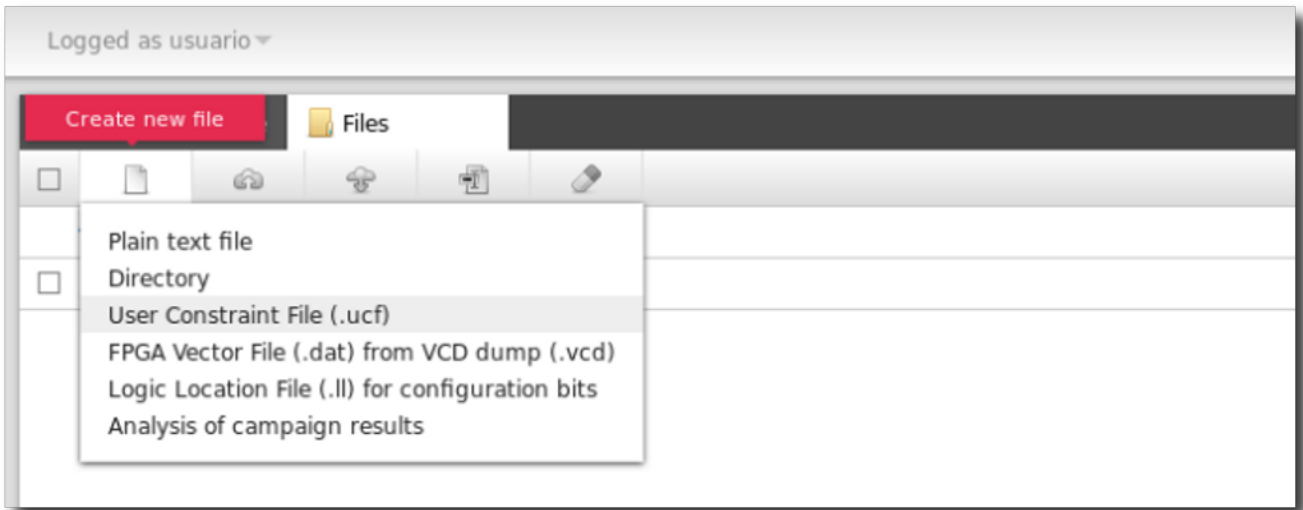
To get the UCF, it is necessary to add the pin file to the project, click over your project and then click over “Files” tab:



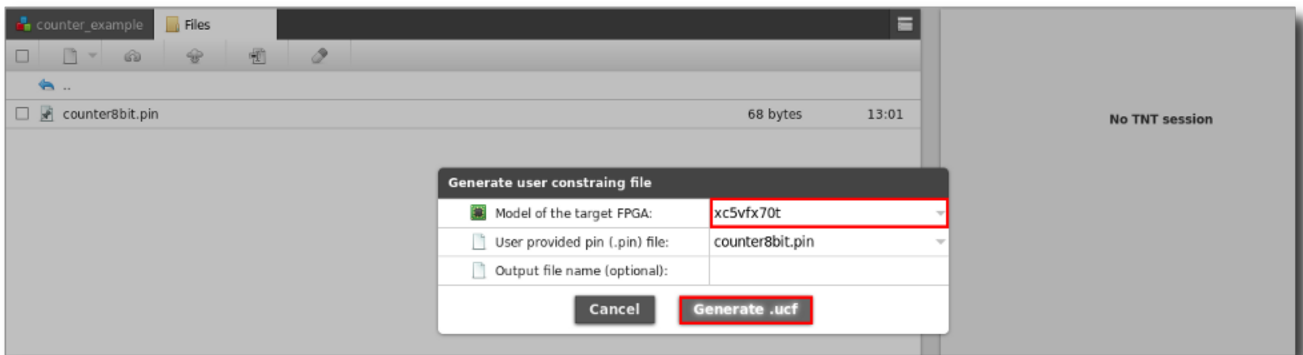
Now, upload the pin file:



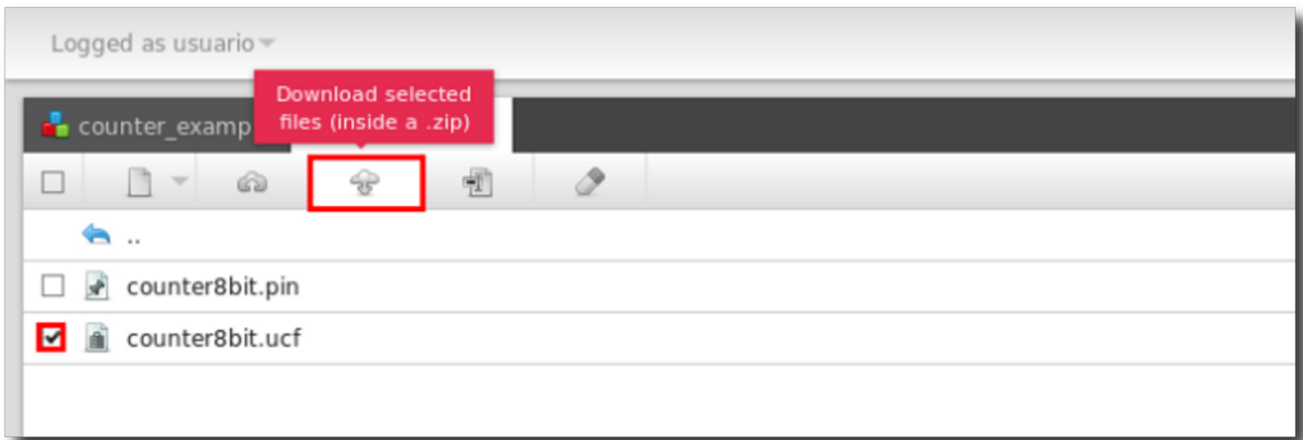
Then click over “Create new file” tab and select the “User Constraint File (.ucf)” option:



Select the correct FPGA model and generate the UCF:



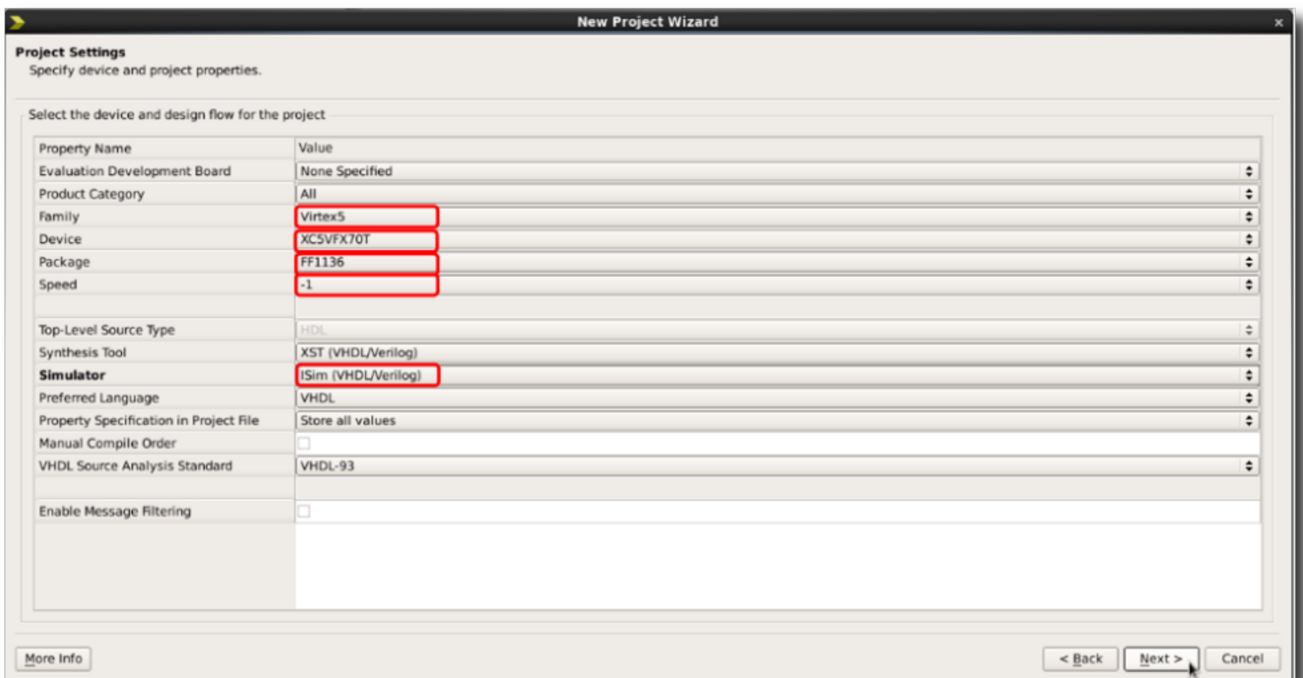
You have already the second file called “counter8bit.ucf”. It is necessary to download the UCF to add it into the ISE project, so select the UCF and click over “Download selected files” tab:



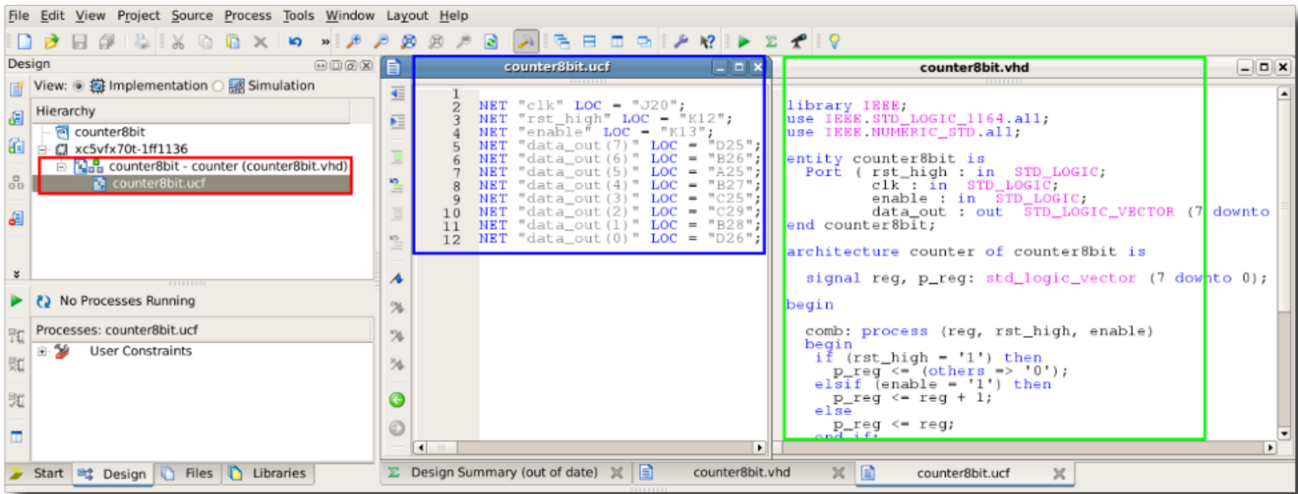
Bit and LL Files

This guide is developed using the ISE 14.7 tool from Xilinx to create the example project and we use the ISim (simulator tool) from Xilinx too. The preferred language to design is VHDL. Designs have to fit in a XC5VFX70T device (Universidad de Sevilla FPGA model available) and it is limited to 512 I/Os. All these tools work under a Linux SO distribution (CentOS 6.6 version).

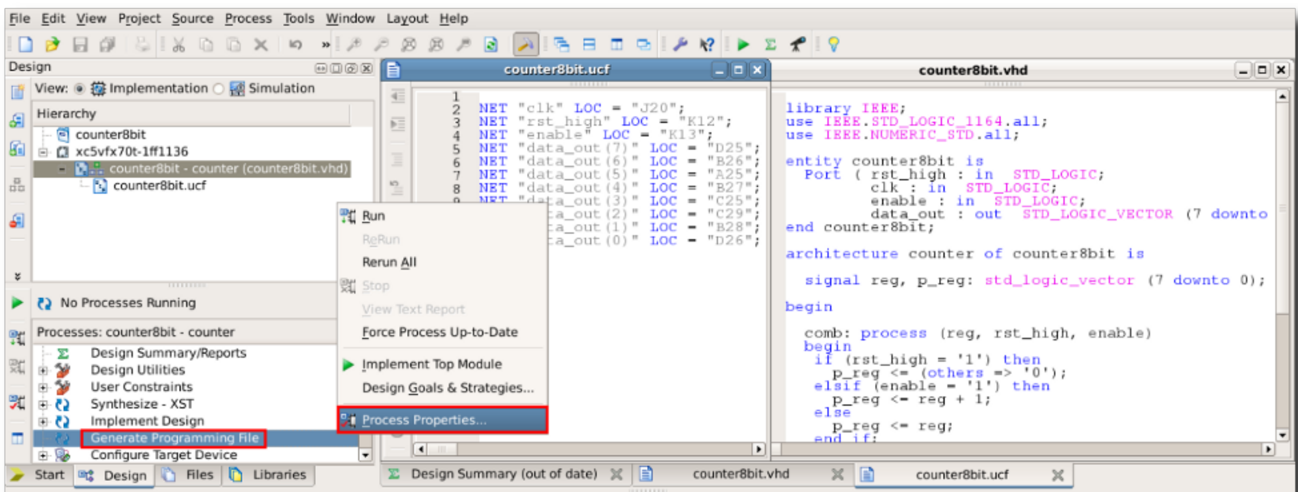
Now it is time to obtain the bitstream. When a new project in ISE is created, select the settings as picture indicates:



Firstly, add the design source code and the UCF into the project:



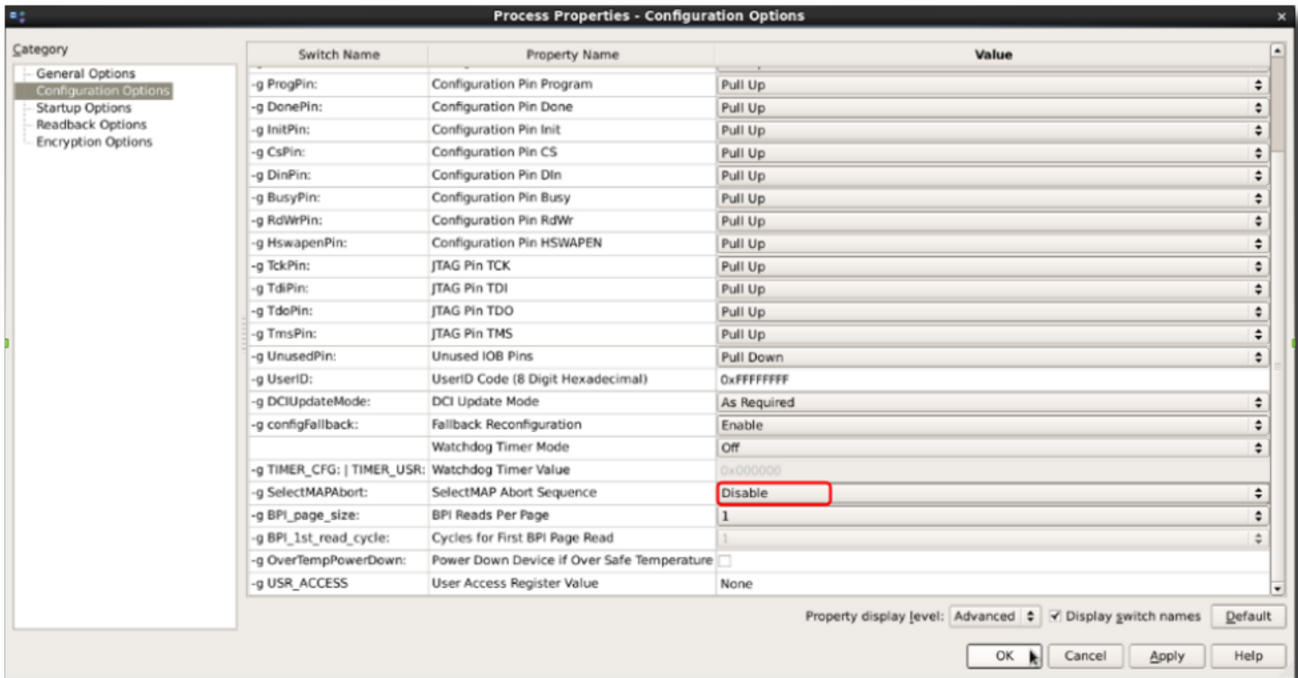
Before getting the bitstream, it's necessary to select some options into the "Process Properties" menu according to using the FTU2 tool. These options are shown in the following pictures:

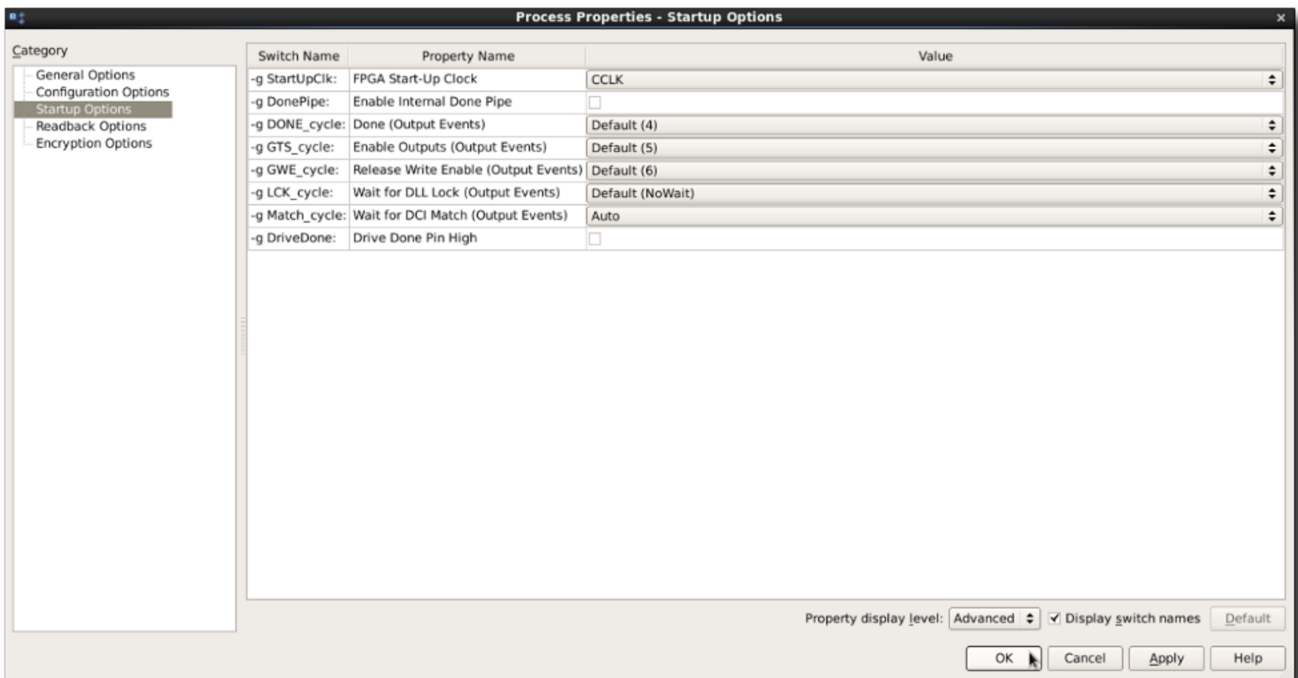


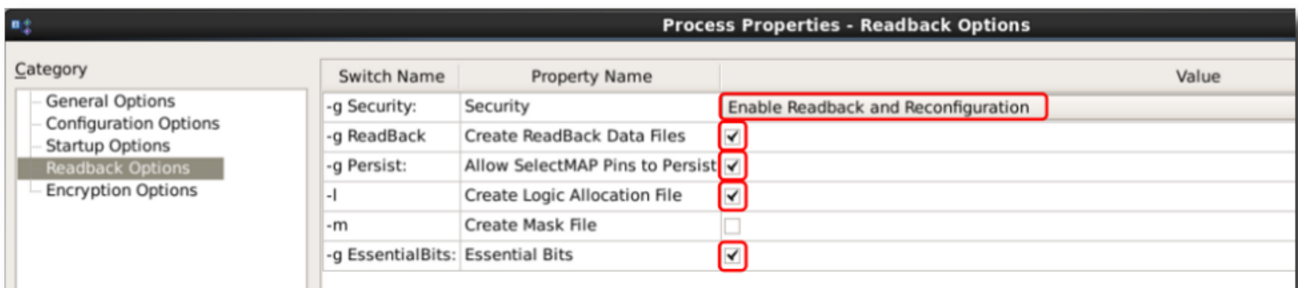
Process Properties - General Options

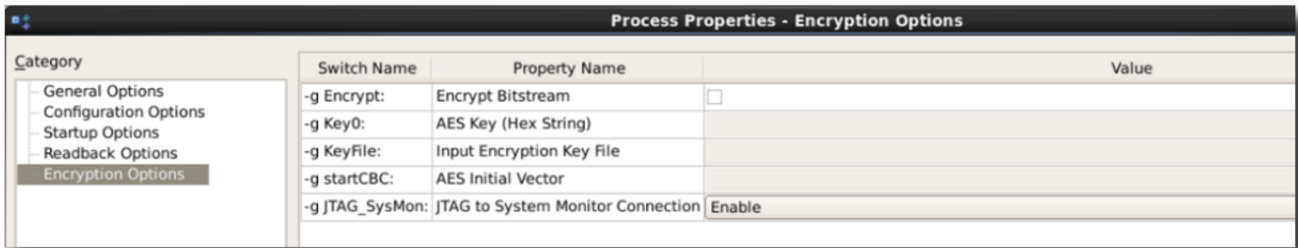
Category
- General Options
- Configuration Options
- Startup Options
- Readback Options
- Encryption Options

Switch Name	Property Name	Value
-d	Run Design Rules Checker (DRC)	☑
-j	Create Bit File	☑
-g Binary:	Create Binary Configuration File	☐
-b	Create ASCII Configuration File	☐
-g IEEE1532:	Create IEEE 1532 Configuration File	☐
-g Compress	Enable BitStream Compression	☐
-g DebugBitstream:	Enable Debugging of Serial Mode BitStream	☐
-g CRC:	Enable Cyclic Redundancy Checking (CRC)	☐
	Other Bitgen Command Line Options	

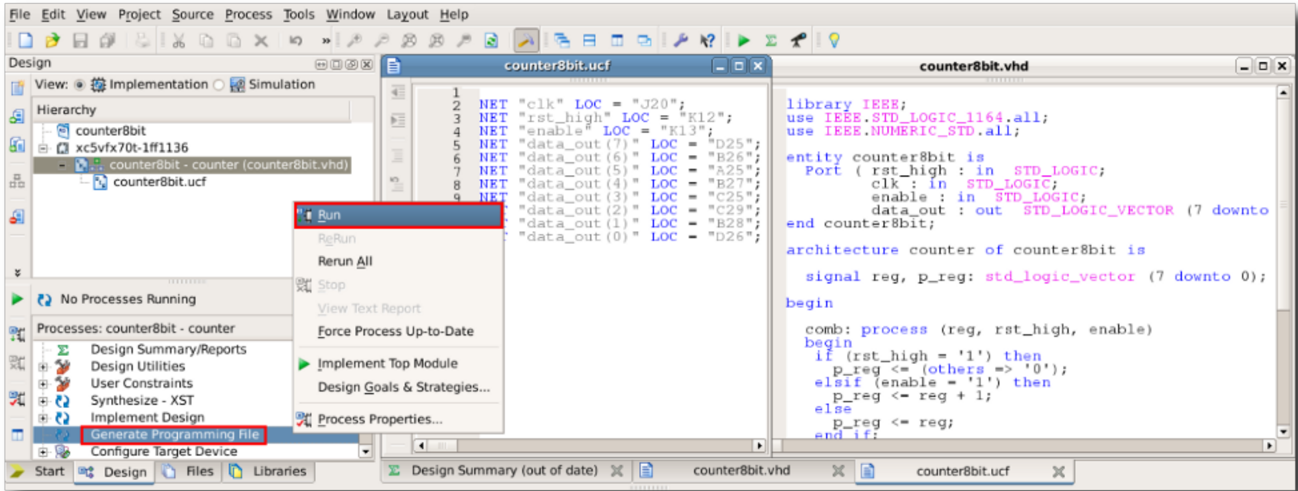

Process Properties - Configuration Options
Category
General Options
Configuration Options
Startup Options
Readback Options
Encryption Options
Switch Name
Property Name
Value
-g ProgPin: Configuration Pin Program Pull Up
-g DonePin: Configuration Pin Done Pull Up
-g InitPin: Configuration Pin Init Pull Up
-g CsPin: Configuration Pin CS Pull Up
-g DinPin: Configuration Pin Din Pull Up
-g BusyPin: Configuration Pin Busy Pull Up
-g RdWrPin: Configuration Pin RdWr Pull Up
-g HswapenPin: Configuration Pin HSWAPEN Pull Up
-g TckPin: JTAG Pin TCK Pull Up
-g TdiPin: JTAG Pin TDI Pull Up
-g TdoPin: JTAG Pin TDO Pull Up
-g TmsPin: JTAG Pin TMS Pull Up
-g UnusedPin: Unused IOB Pins Pull Down
-g UserID: UserID Code (8 Digit Hexadecimal) 0xFFFFFFFF
-g DCIUpdateMode: DCI Update Mode As Required
-g configFallback: Fallback Reconfiguration Enable
Watchdog Timer Mode Off
-g TIMER_CFG: | TIMER_USR: Watchdog Timer Value 0x000000
-g SelectMAPAbort: SelectMAP Abort Sequence Disable
-g BPI_page_size: BPI Reads Per Page 1
-g BPI_1st_read_cycle: Cycles for First BPI Page Read 1
-g OverTempPowerDown: Power Down Device if Over Safe Temperature
-g USR_ACCESS User Access Register Value None
Property display level: Advanced
Display switch names
Default
OK
Cancel
Apply
Help


Process Properties - Startup Options
Category
General Options
Configuration Options
Startup Options
Readback Options
Encryption Options
Switch Name
Property Name
Value
-g StartUpClk: FPGA Start-Up Clock CCLK
-g DonePipe: Enable Internal Done Pipe
-g DONE_cycle: Done (Output Events) Default (4)
-g GTS_cycle: Enable Outputs (Output Events) Default (5)
-g GWE_cycle: Release Write Enable (Output Events) Default (6)
-g LCK_cycle: Wait for DLL Lock (Output Events) Default (NoWait)
-g Match_cycle: Wait for DCI Match (Output Events) Auto
-g DriveDone: Drive Done Pin High
Property display level: Advanced
Display switch names
Default
OK
Cancel
Apply
Help

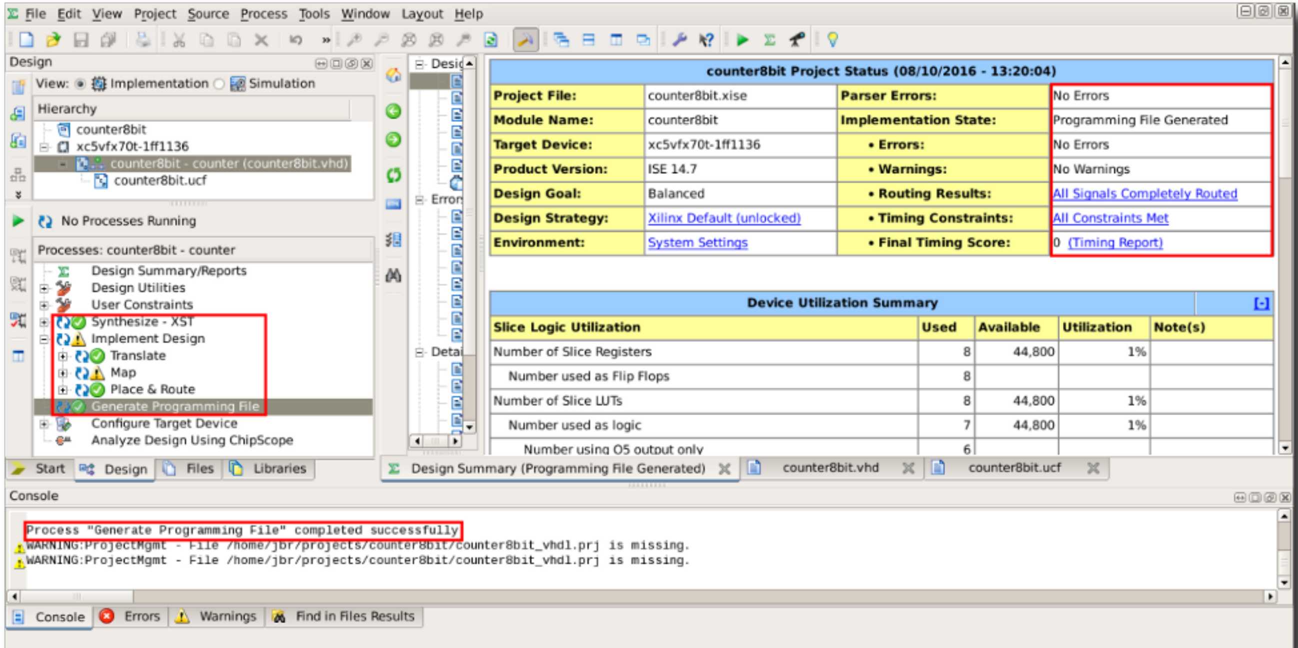

Process Properties - Readback Options
Category
General Options
Configuration Options
Startup Options
Readback Options
Encryption Options
Switch Name
Property Name
Value
-g Security: Security Enable Readback and Reconfiguration
-g ReadBack Create ReadBack Data Files
-g Persist: Allow SelectMAP Pins to Persist
-l Create Logic Allocation File
-m Create Mask File
-g EssentialBits: Essential Bits



Finally, run “Generate Programming File” to get the bitstream:



Check that the process finish without errors:

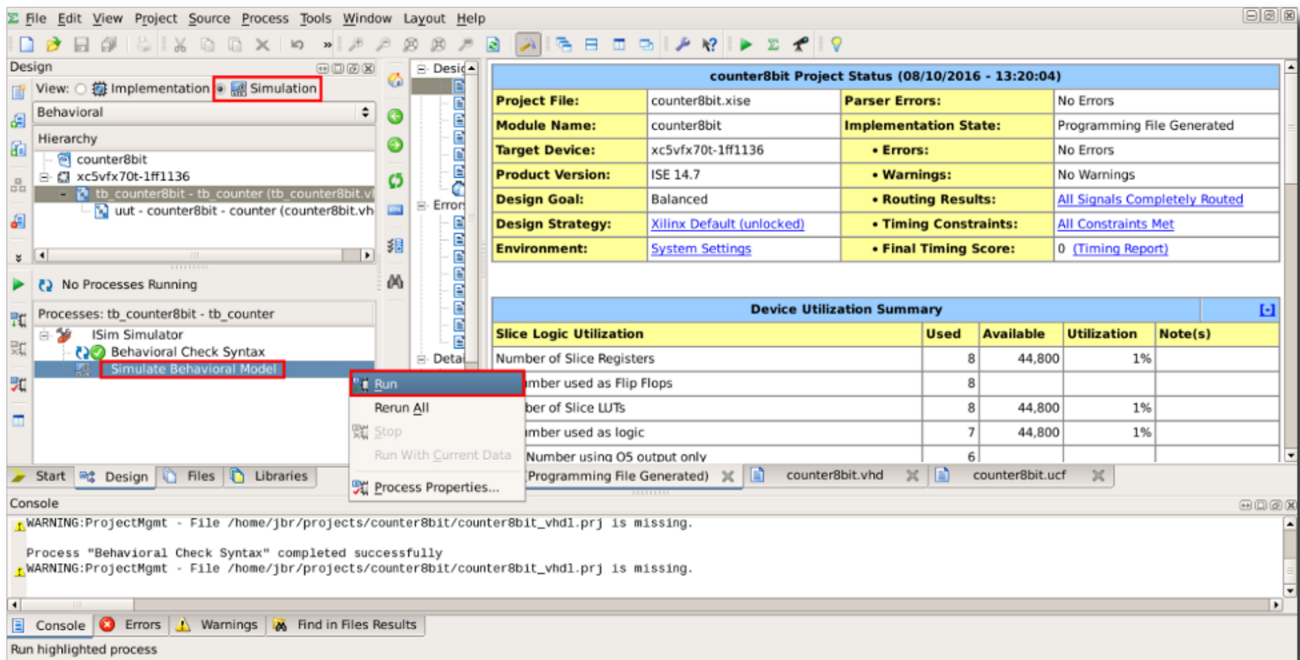


Now there are in the user’s project folder two of the target files: “counter8bit.bit” and “counter8bit.ll”.

VCD and Dat Files

The last target file that is missing is the DAT file but before to get it is necessary to generate the VCD

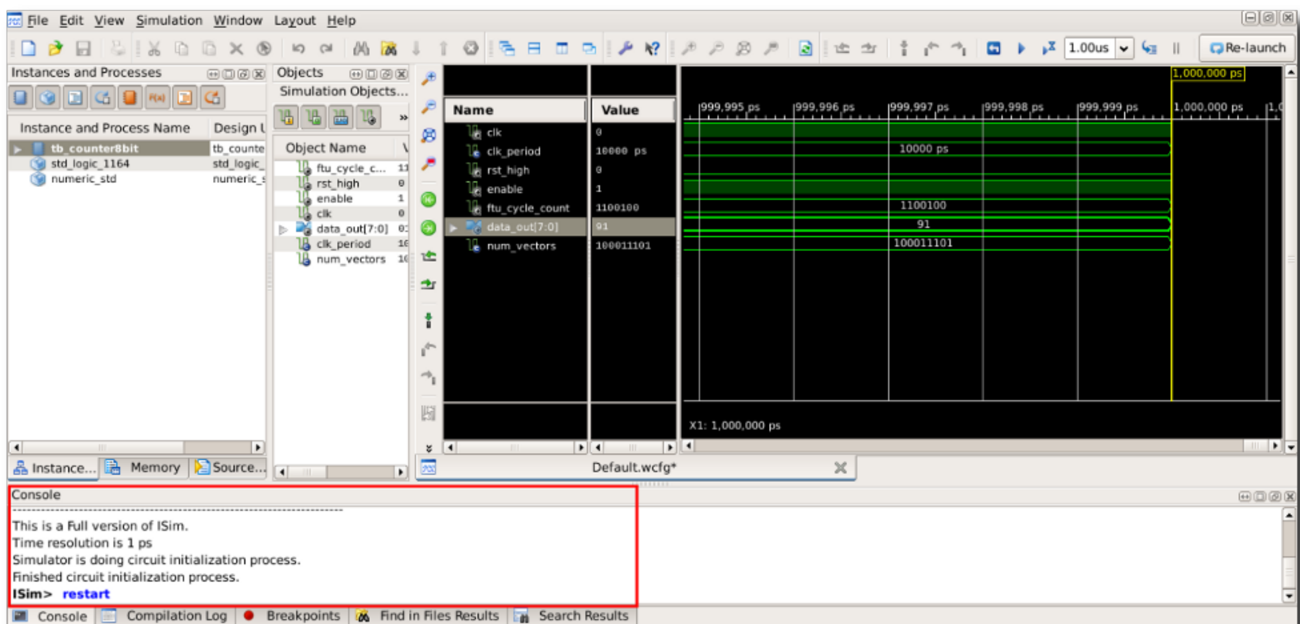
file using the ISim tool. So add the test bench into the project and run the simulator:

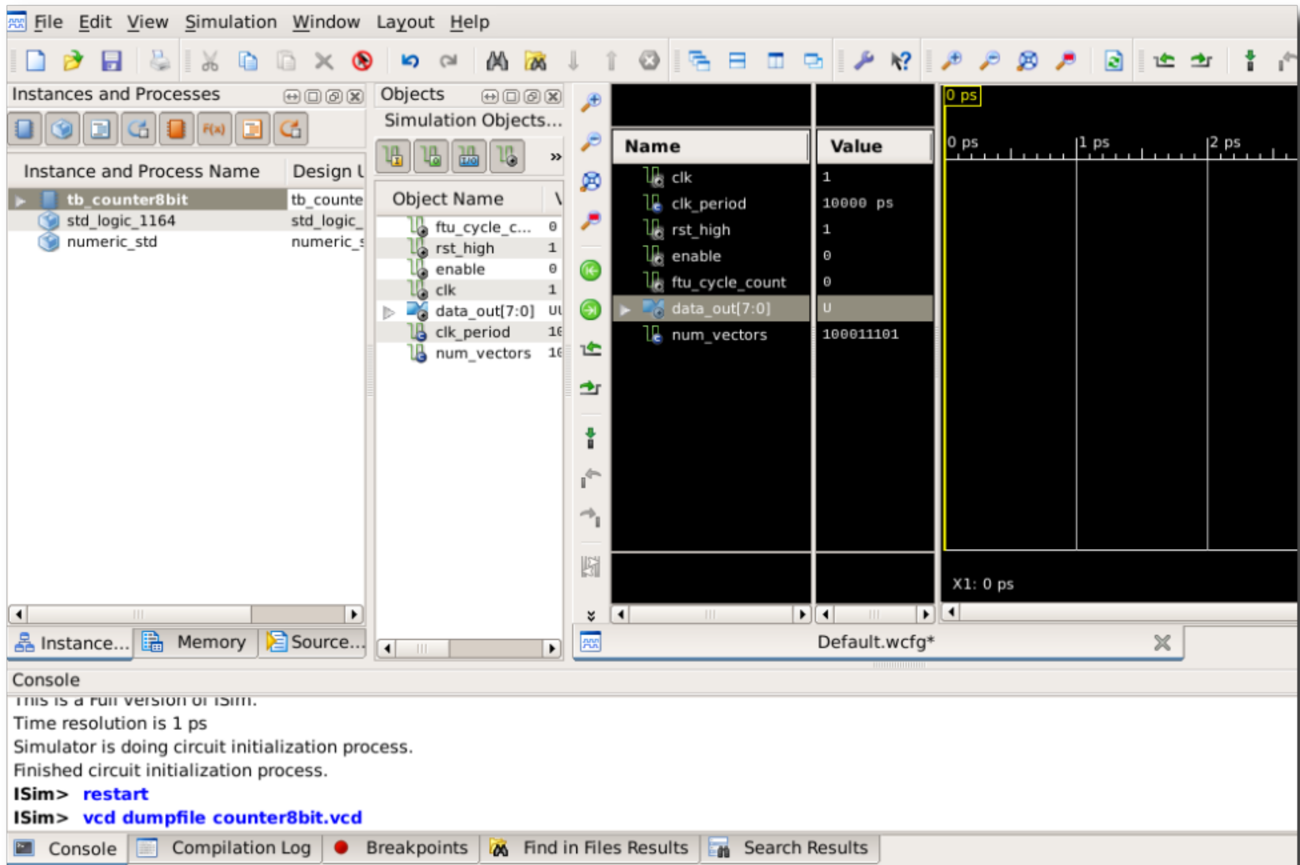


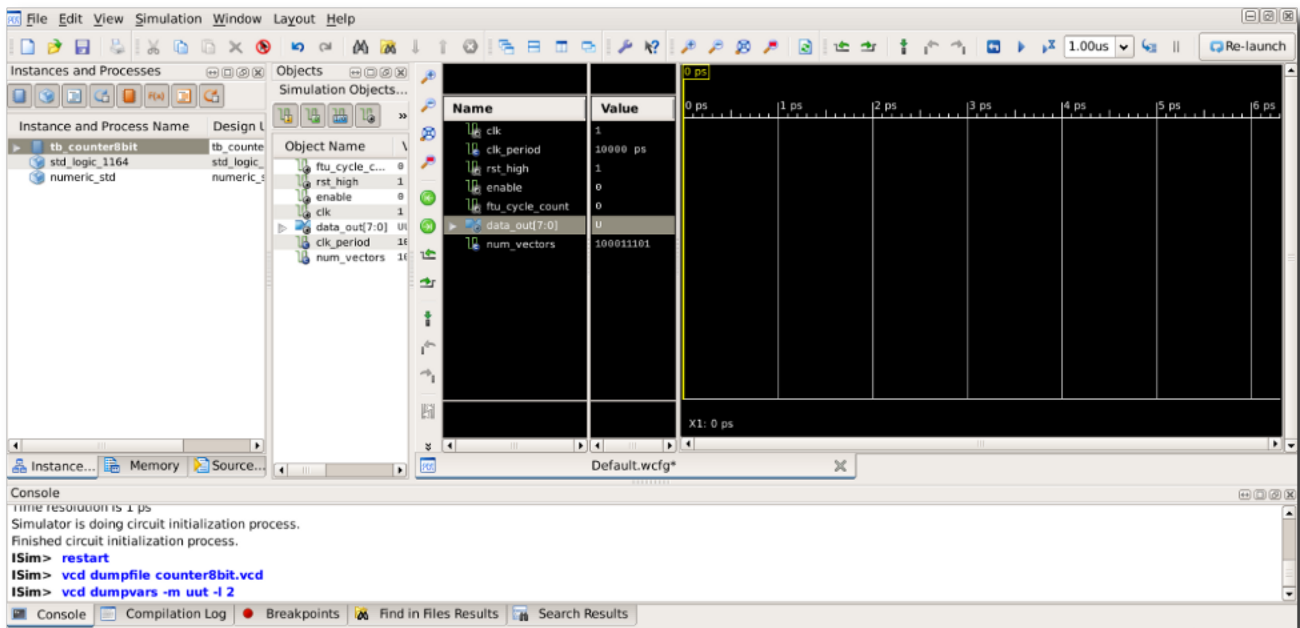
The VCD file is obtained by some commands running on the console of the ISim tool:

```
restart
vcd dumpfile <design>.vcd
vcd dumpvars -m <inst> -l 2
```

Note: <inst> is the name of the instance given for the highest level module. Normally is in the design call in the simulation framework.

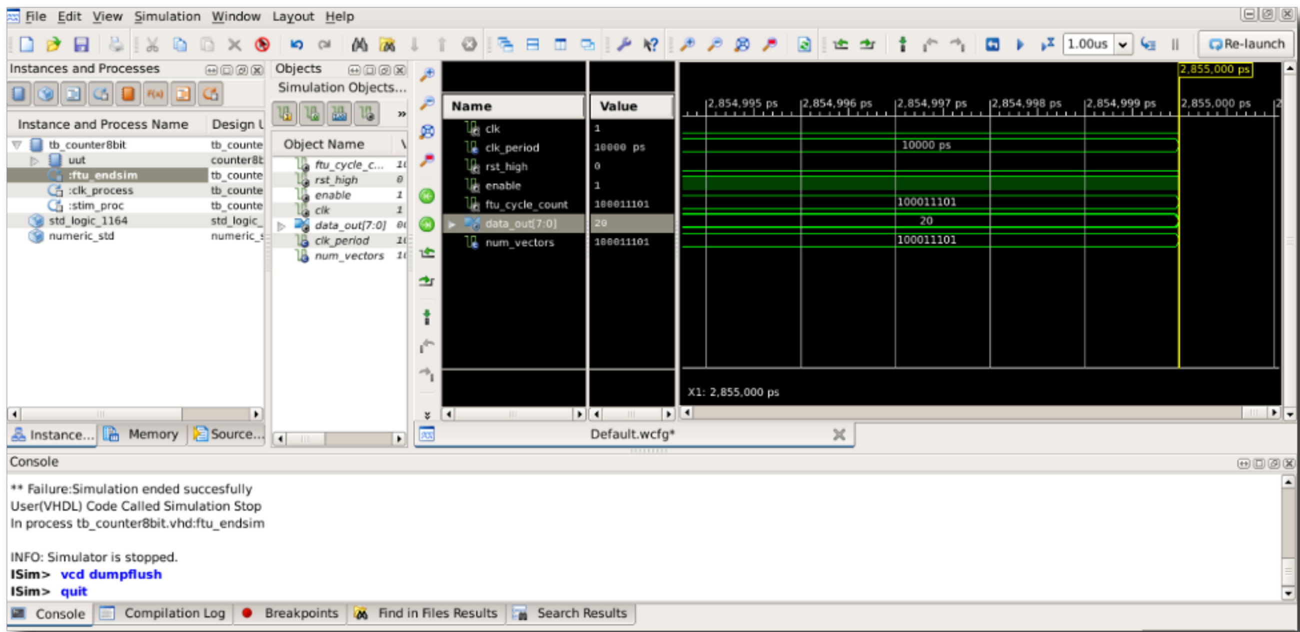




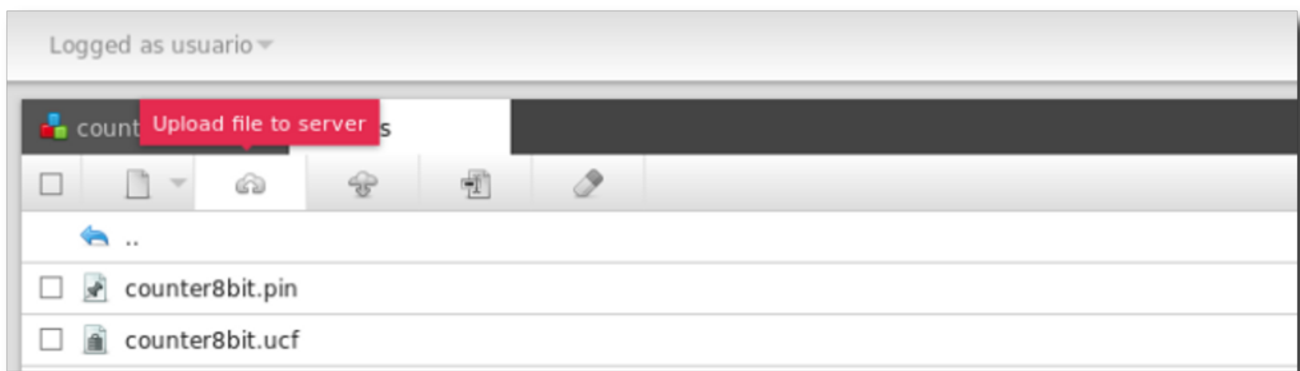


Run the simulation up to the desired time; once it has finished type:

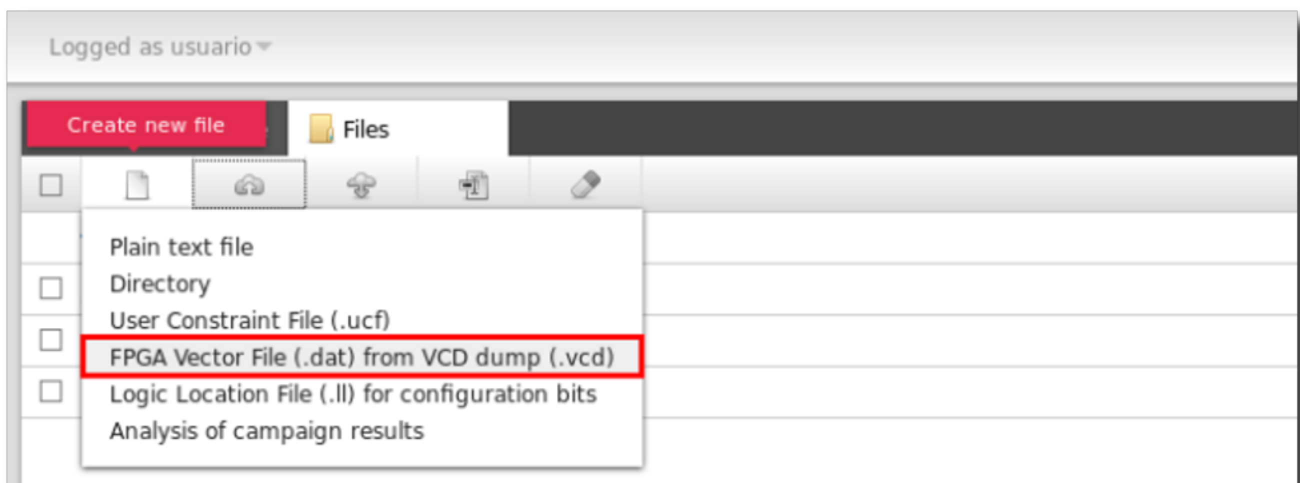
```
vcd dumpflush
quit
```



Now the workload file is in the user's project folder called "counter8bit.vcd", containing the set of stimuli. Using this file in conjunction with the previously created PIN file it is the way to generate the DAT file with a tool from the UFF web server. Firstly, it is necessary to upload the VCD file to UFF as did it previously with the UCF file:



Then click over "Create new file" tab and select the "FPGA Vector File (.dat) from VCD dump (.vcd)" option:



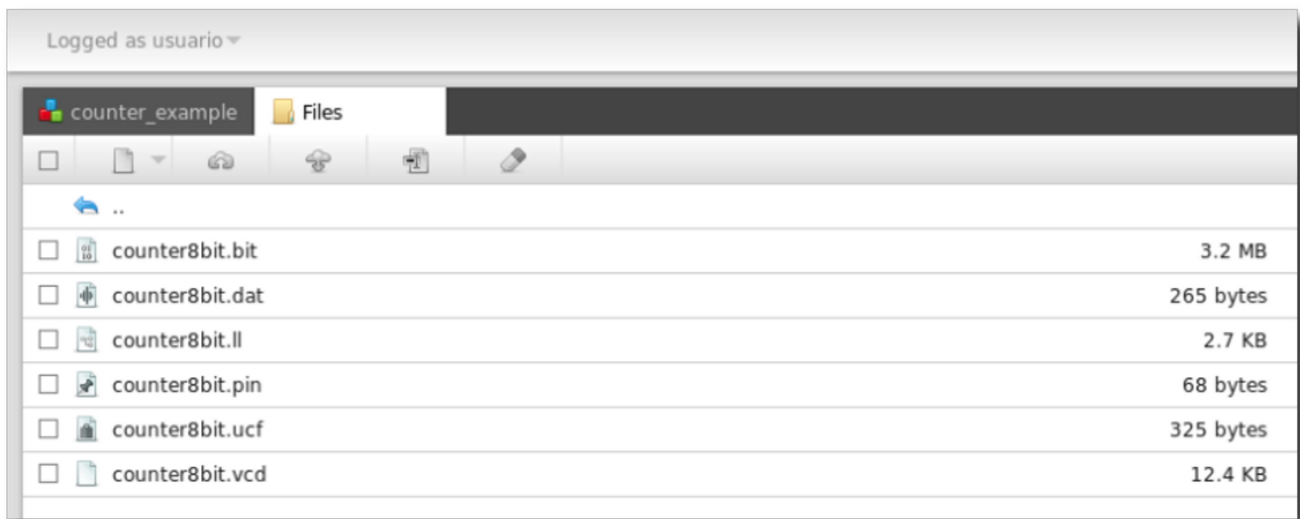
Verify that the options are correct and generate the DAT file:

Generate FPGA vector file

Model of the target FPGA:	xc5vfx70t
Value Change dump (.vcd) file:	counter8bit.vcd
User provided pin (.pin) file:	counter8bit.pin
Name of the Unit Under Test:	uut
Handle "x" values:	Exit with error
Handle "z" values:	Exit with error
Output file name (optional):	

Cancel | Generate .dat

So now the last file called "counter8bit.dat" is in the user's project folder. Finally, add the BIT and LL files to the UFF web server. At this point, the user has a complete set of files to emulate the Design Under Test. It is time to set up an FT-UNSHADES design.



Appendix A

Design Source Code:

counter8bit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity counter8bit is
  port (
    rst_high: in  std_logic;
    clk:      in  std_logic;
    enable:   in  std_logic;
    data_out: out std_logic_vector (7 downto 0)
  );
end counter8bit;

architecture counter of counter8bit is

  signal reg, p_reg: unsigned (7 downto 0);

  begin

  comb: process (reg, rst_high, enable)
  begin
    if (rst_high = '1') then
      p_reg <= (others => '0');
    elsif (enable = '1') then
      p_reg <= reg + 1;
    else
     p_reg <= reg;
    end if;
  end process;

  data_out <= std_logic_vector(reg);

  sinc: process (clk)
  begin
    if (clk = '1' and clk'event) then
      reg <= p_reg;
    end if;
  end process;

end counter;
```

Test Bench:

tb_counter8bit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
```

```
entity tb_counter8bit is
end tb_counter8bit;

architecture tb_counter of tb_counter8bit is

  component counter8bit
    port(
      rst_high: in  std_logic;
      enable:   in  std_logic;
      clk:      in  std_logic;
      data_out: out std_logic_vector (7 downto 0)
    );
  end component;

  -- Simulation will stop at time = clk_period * num_vectors
  constant clk_period:      time    := 10 ns;
  constant num_vectors:     integer := 285;
  signal   ftu_cycle_count: integer := 0;

  -- Inputs
  signal rst_high: std_logic := '1';
  signal enable:   std_logic := '0';
  signal clk:      std_logic := '1';

  -- Outputs
  signal data_out: std_logic_vector (7 downto 0);

  begin

  -- Stops simulation at desired time.
  -- Compatible with FTU2 and FTU1.
  ftu_endsim: process(clk)
  begin
   if (rising_edge(clk))then
     ftu_cycle_count <= ftu_cycle_count + 1;
     if(ftu_cycle_count = num_vectors)then
        report "Simulation ended succesfully"
        severity failure;
     end if;
   end if;
  end process;

  -- Instantiate the Unit Under Test (UUT)
  uut: counter8bit port map (
    rst_high => rst_high,
    enable   => enable,
    clk      => clk,
    data_out => data_out
    );
```

```
  -- Clock process definitions
  clk_process: process
  begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
  end process;

  -- Stimulus process
  stim_proc: process
  begin
    -- 4 cycle rst:
    rst_high <= '1';
    wait for clk_period * 4;
    -- 20 cycles of not reset
    rst_high <= '0';
    enable <= '1';
    wait for clk_period * 20;
    -- 5 cycles of not enable
    enable <= '0';
    wait for clk_period * 5;
    -- 256 cycles of enable
    enable <= '1';
    wait for clk_period * 256;
    wait;
  end process;

  -- Enable count only one each 4 clock cycles
  -- enable <= '1' when ftu_cycle_count mod 4 = 0 else '0';

end;
```

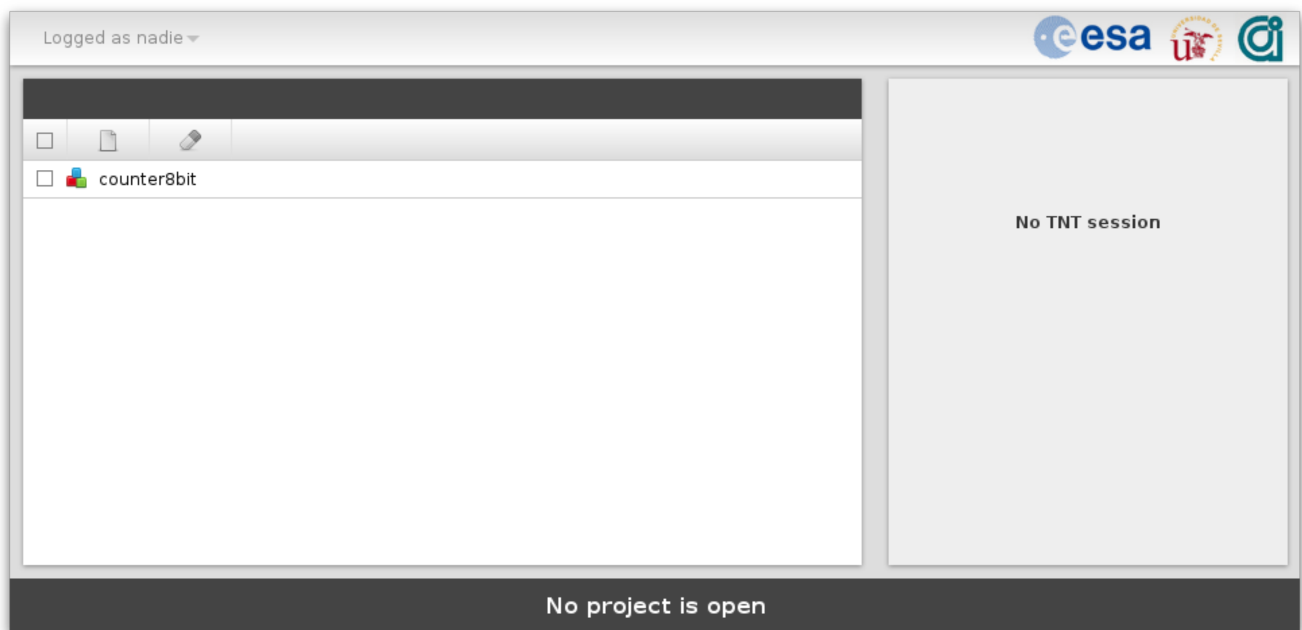
First Steps on UFF

This chapter describes in the simplest way possible the most common tasks you'll perform when using UFF:

- Preparation of the system for emulation of a project,
- Automatic execution of campaigns,
- Basic usage of the debugger.

UFF Workspace

The UFF workspace is divided as follows:



Top Bar (top side)

Contains widgets to manage the session, including the logout button and controls to change the distribution of the rest of the panels.

UFF Panel (left side)

Graphical interface for the current task; during most of this document we'll deal exclusively with its contents, and most screenshots from here onwards will include only it.

TNT Panel (right side)

Text interface for the current task; most actions we perform will automatically execute commands that will be visible here. You will never have to write text here, but it allows a much greater degree of power and flexibility than using the graphical interface.

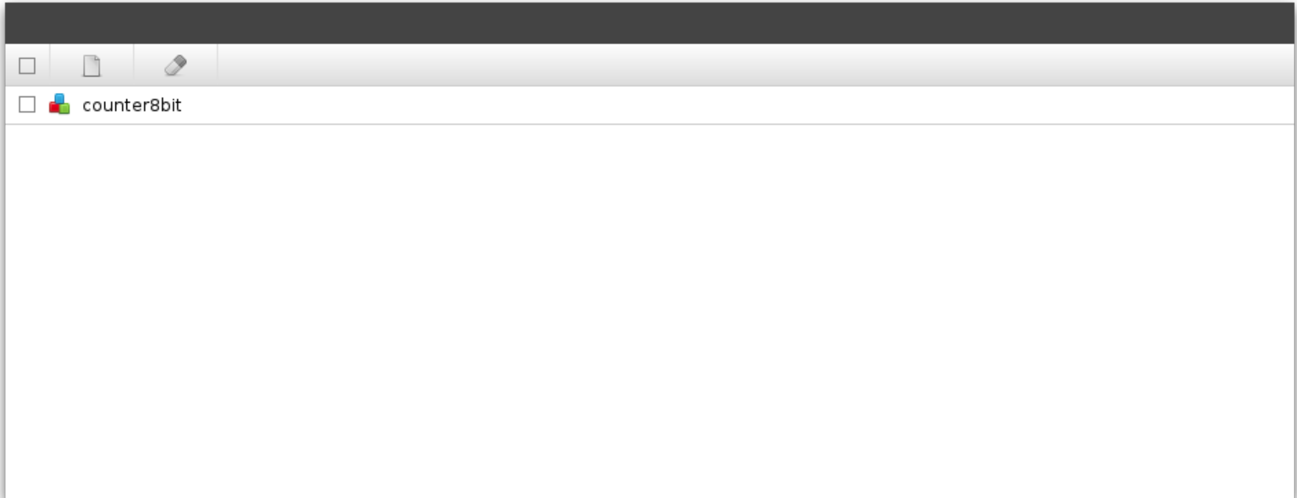
Status Bar (bottom side)

Status of background tasks. Not relevant unless you're running a campaign.

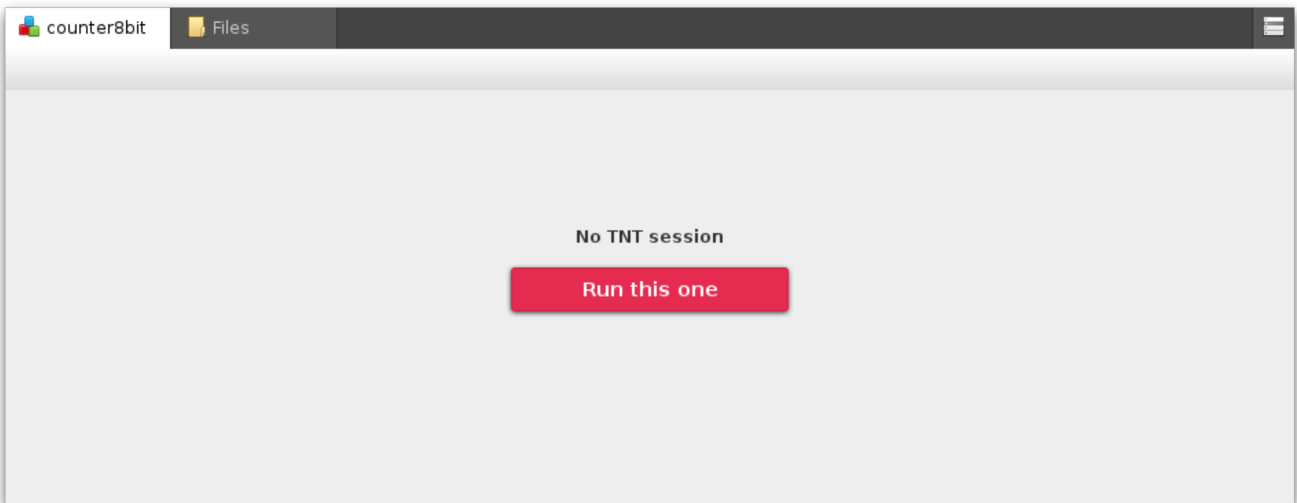
Setting Up The Design

To do so, login into the UFF website at http://ftu.us.es/uff/login/

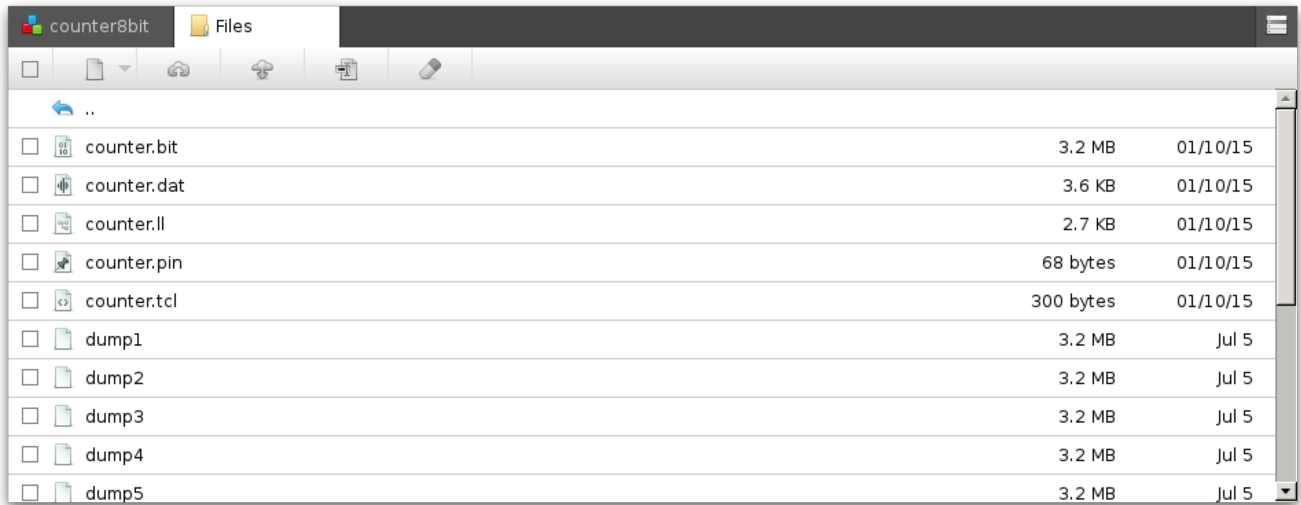
Once you log in, the list of available projects appears in the UFF Panel. You can create, delete, or open existing ones; in this example, we'll open a simple 8 bit counter called "counter_example".



When you open a project, you'll get a message saying "No TNT session"; this means there's no `tntsh` process associated with your user. If you had been working with another project and not closed it properly, you'd get a "You're currently working in project Foo" message instead.



Also, a new tab will be added to the UFF panel: labelled "files", it will contain a list of all files in the current project.

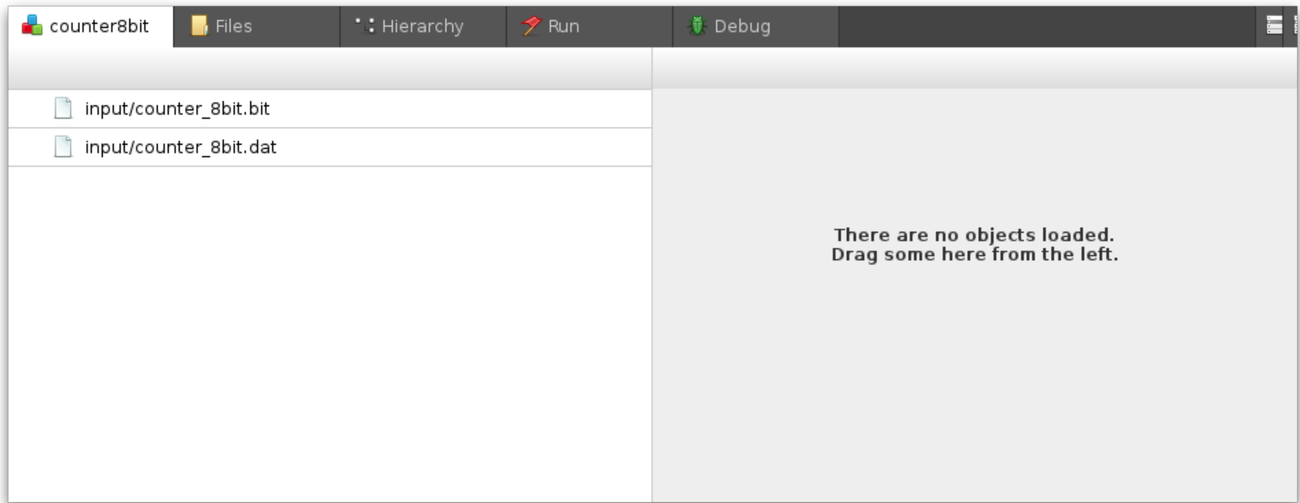


Back to the "counter8bit" tab, press the big, red button that says "Run this one" to proceed; you'll be given an additional tab ("Hierarchy") and prompted to select a device on which to emulate the project.

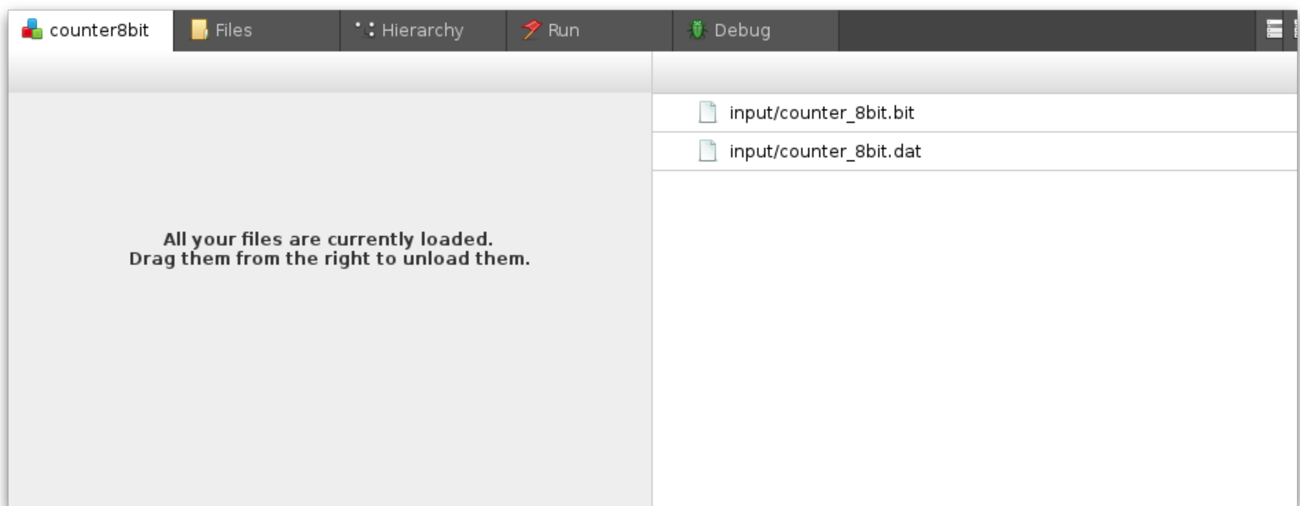
If the device is available, it will be opened by the current tnt session, and you'll be allowed to configure it for emulation.

Configure the Hardware

To prepare the physical device for emulation, we must configure it so the target FPGA is configured to behave as the desire circuit, and there is a set of values that can be used to feed its inputs. This is done by loading objects to the motherboard from the main tab:

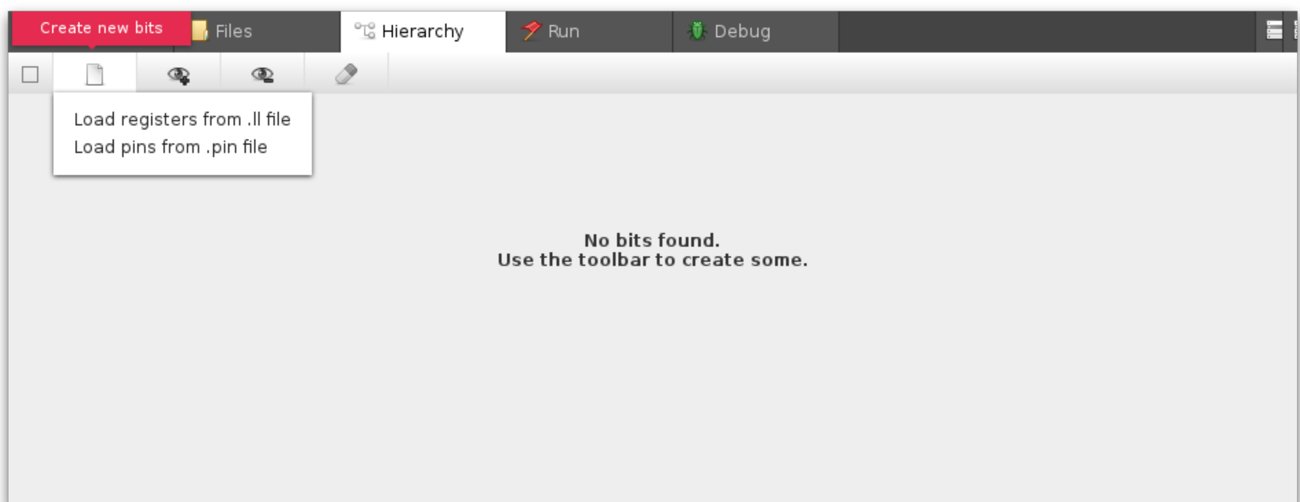


Objects are loaded by dragging them from left (unloaded) to right (loaded) side:

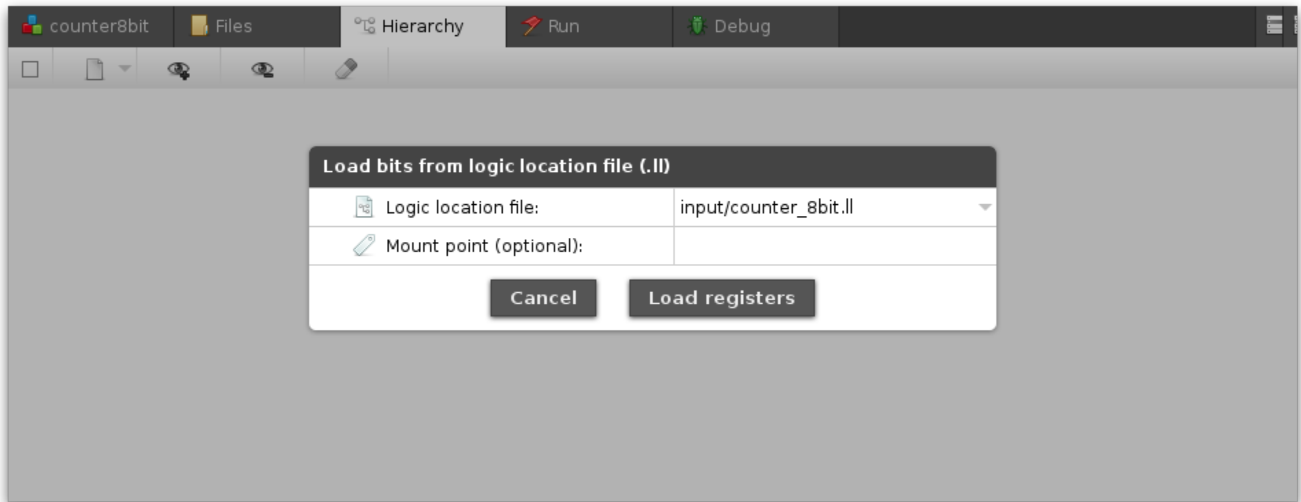


Configure the Software

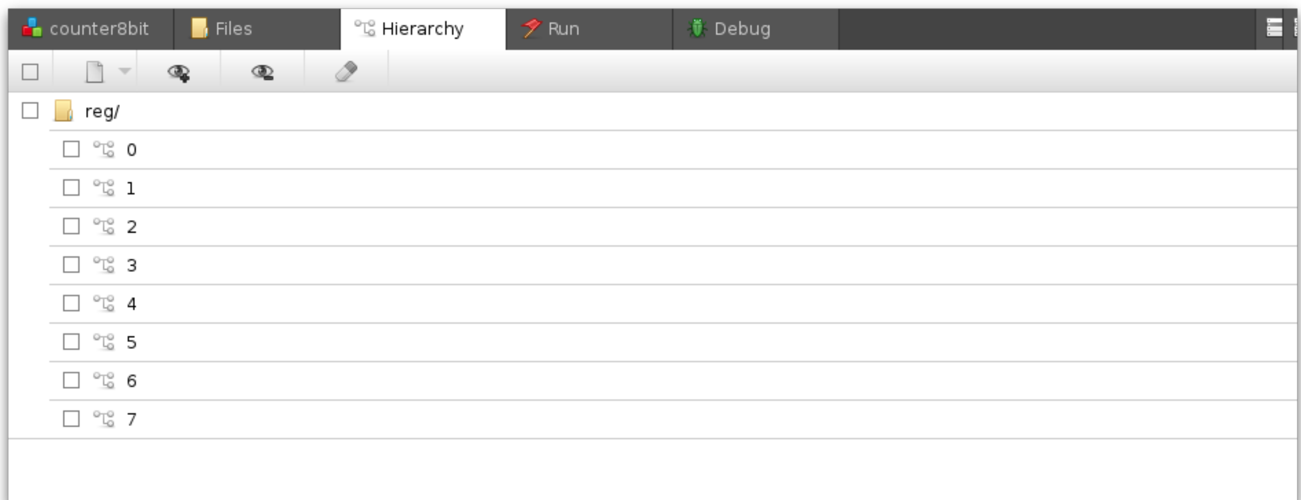
Once the physical device is configured, we prepare the software side to interface with it. The server requires a map of the locations of the target FPGA that we want to analyze, that can be loaded from the Hierarchy tab.



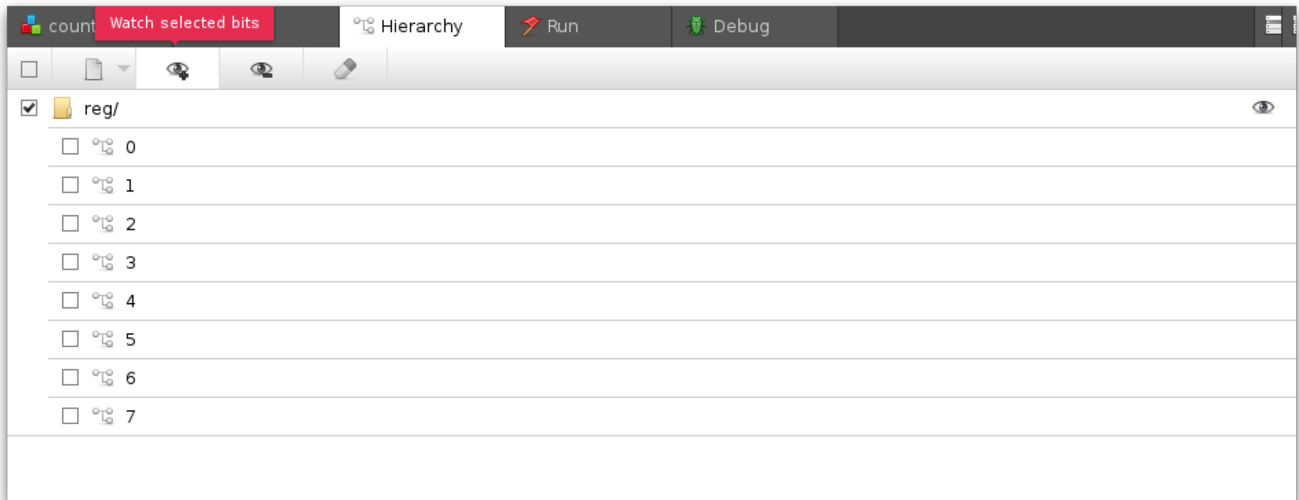
Select "load registers from .ll file" and you'll be provided with a list of options.



Press "Load registers" and the registers will be added to the bit tree; you'll be able to navigate it as if it was a file list.

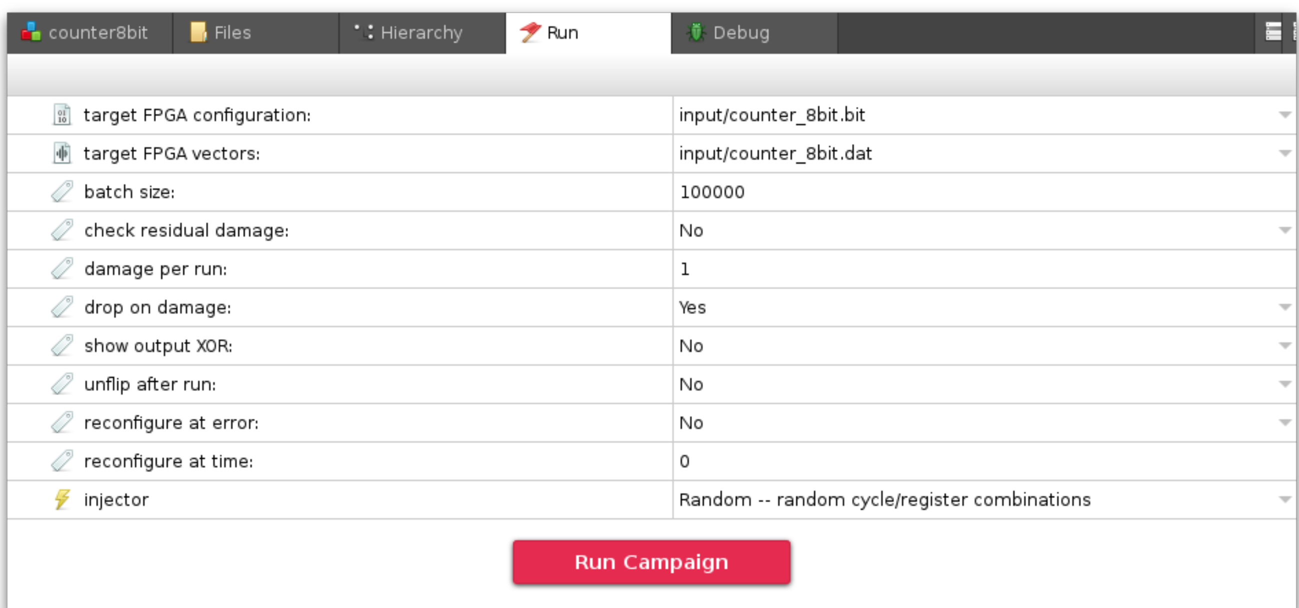


Finally, you must create watches on some registers: having a watch on a register or group of registers means that the system will keep track of their values as emulation advances. You'll be able to request the log of all recorded values at any time as long as the watch exists.

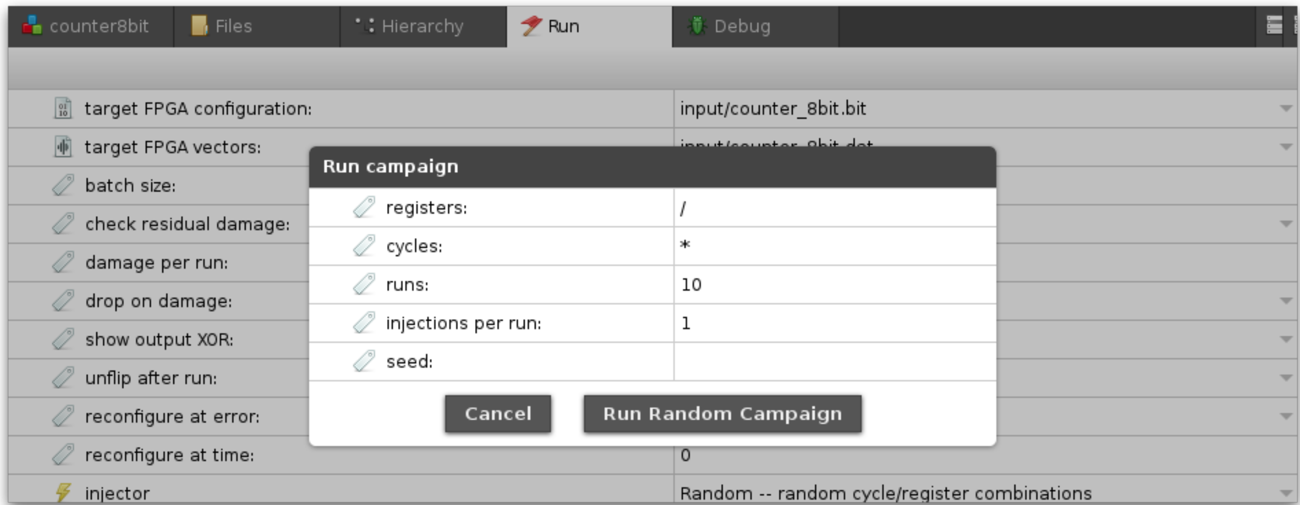


Campaign Runner

Now inside the "Run" tab, the user could change the different parameters [1: For more information refer to the tnt scripting guide.] that are displayed for getting a custom campaign. In this guide the campaign uses the default values.



When all the parameters are ready, click over the "Run Campaign" button and FTU2 will run the process in the background. In this case a random injector is selected so the user could select some options for this kind of injections:



After click over the “Run Random Campaign” button, a notification message is displayed on the bottom. When the “Ready” message is shown, it means that the campaign is completed.

Now it is time to analyze the results. To do that click over the “Files” tab where It appears a new folder called “Results”.

Inside that folder, the results campaigns that the user performs are saved. Each campaign saves the results in a folder that is called with the following format: YY-MM-DD-hh-mm-ss

Inside this folder there are several files, [1: For more information refer to the tnt scripting guide.] as follows:

damages.csv

The results of the full campaign. The contents of this file depend on what information the FTUNSHADES device was configured to collect.

injections.csv

A description of all runs that were executed during the campaign.

reg_names.txt

A list of all registers where faults were injected during the campaign. The purpose of this file is to assign a numeric index to each one of the registers, that is used in the injections.csv file: the first path corresponds to a 0 in, the second to a 1, etc.

run.tcl

A Tcl script that replicates the campaign if executed. It doubles as a log file for what the campaign actually did.

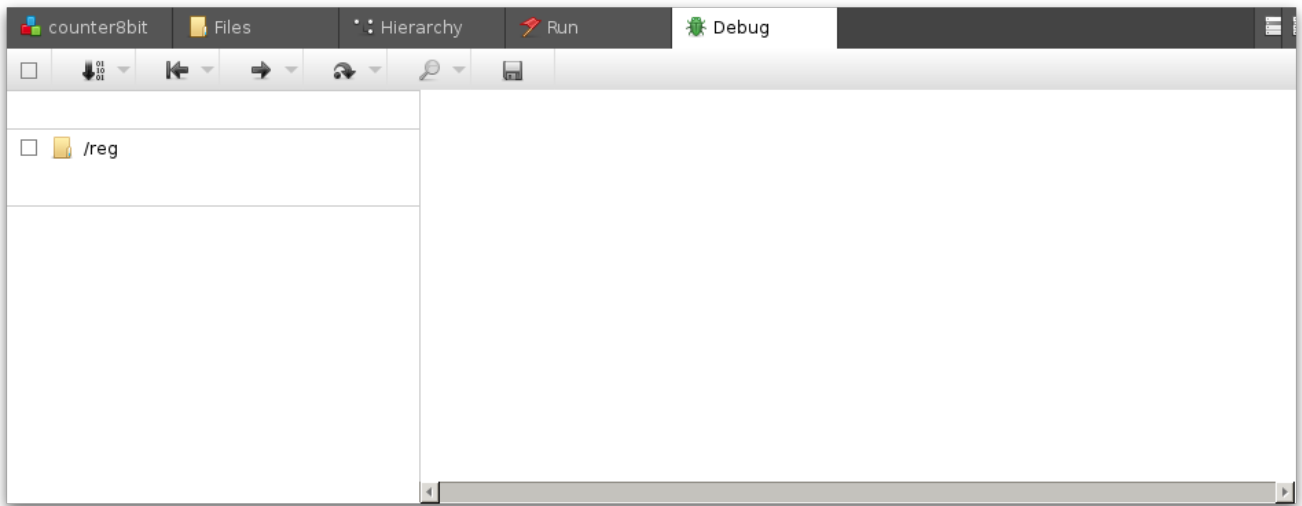
stats.txt

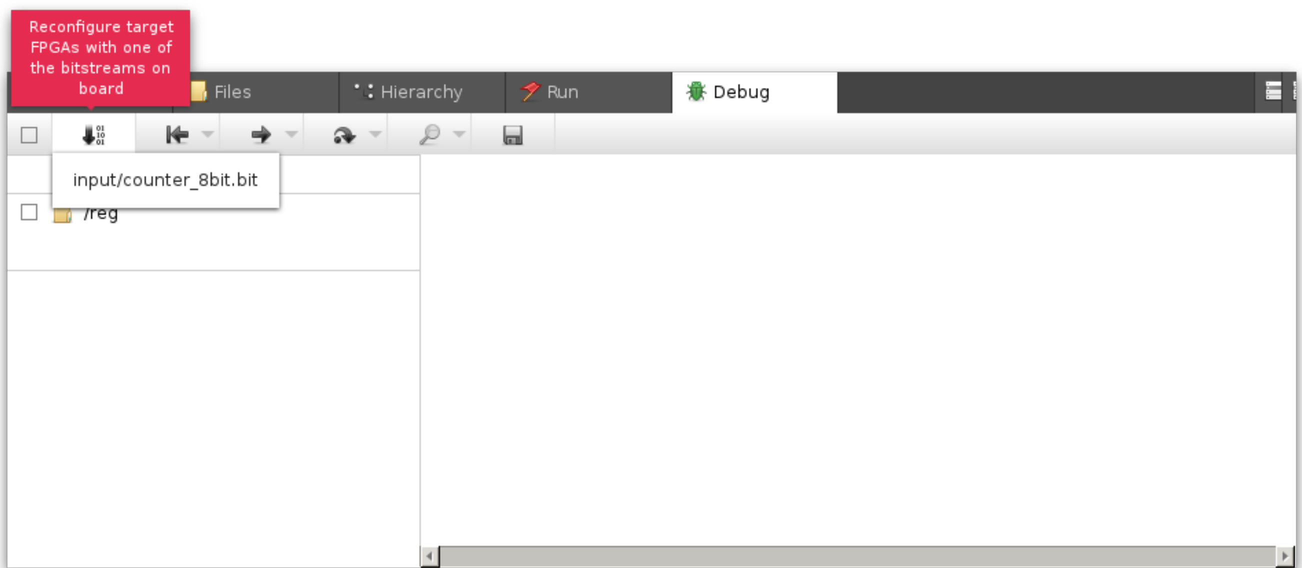
Some statistics taken while running the actual campaign.

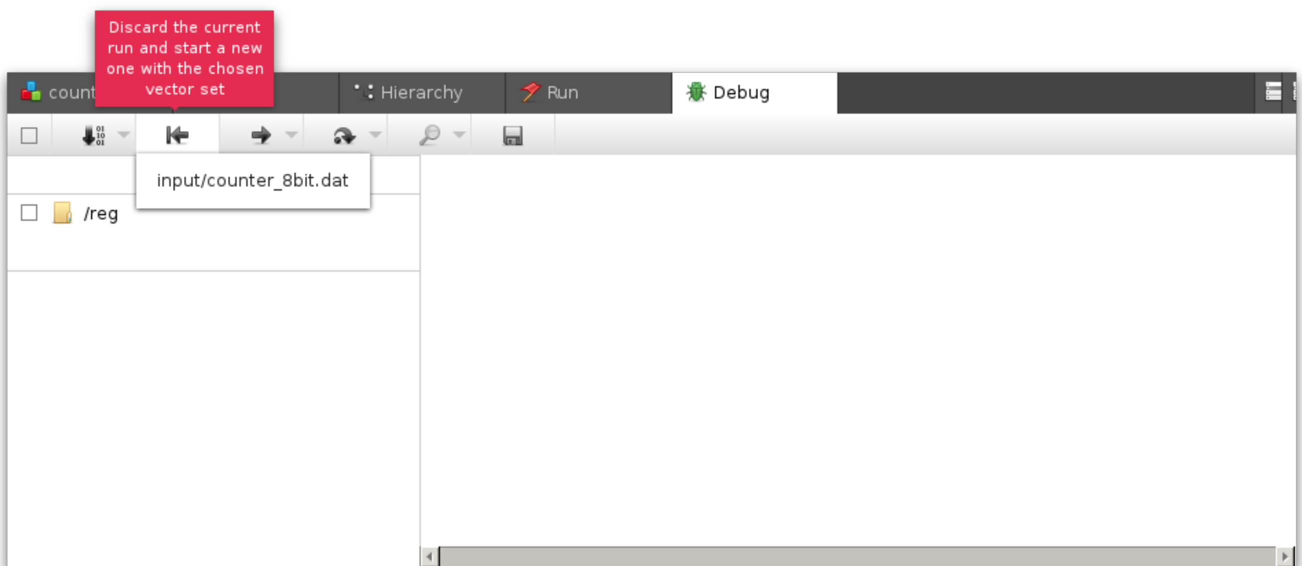
Hardware Debugger

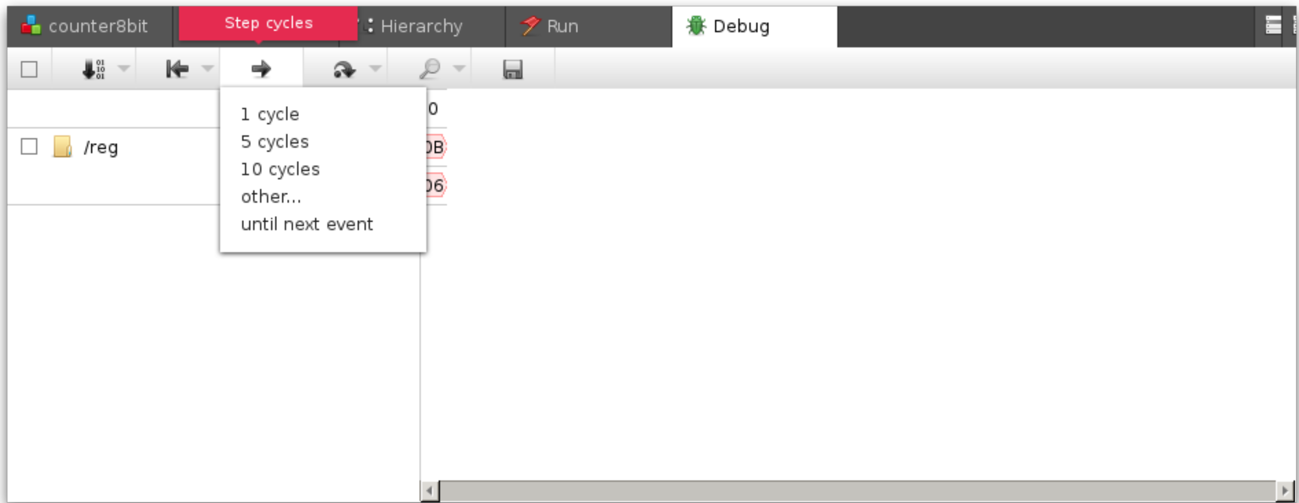
This tool allows to debug the behaviour of the design. So the user could debug the hardware without implementing it first. As in campaign mode, some preliminary steps are required to use

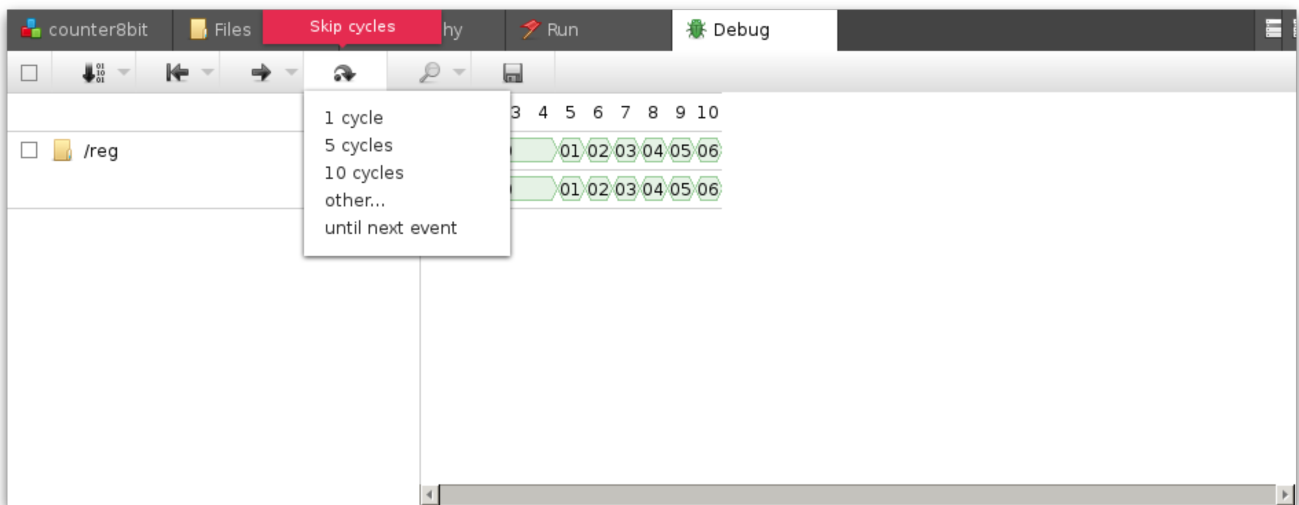
this tool.

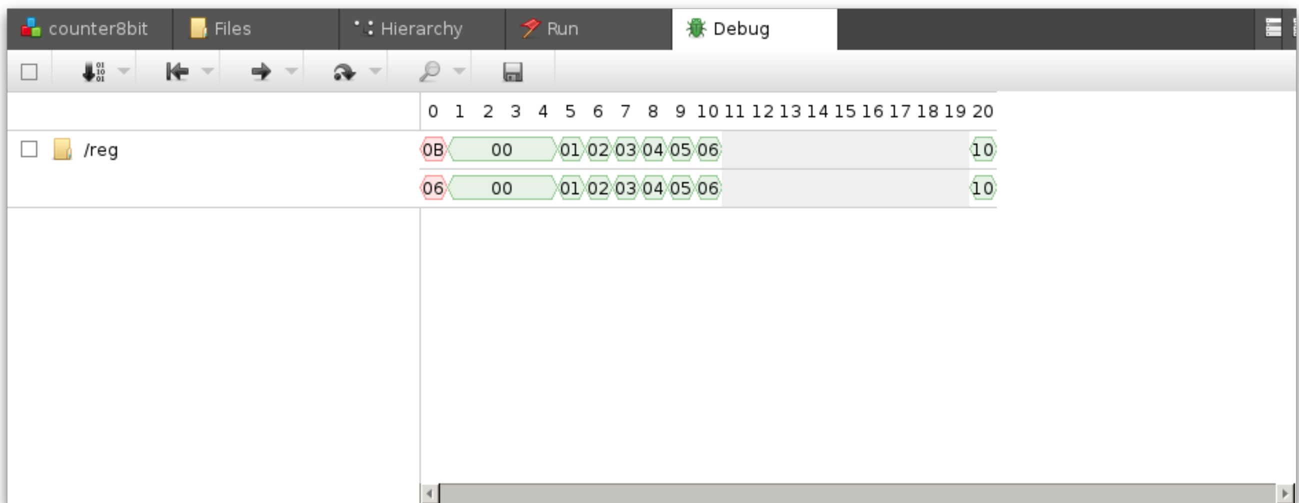












Starting from the beginning, the user has to open the project, select a FPGA and load the bitstream and the vectors files. Also it is necessary to load the registers but in this case the user can create some watches [2: Watches are a way of keeping track of the values of bits as emulation advances.] to see them in the debug process. In this case, select the “reg” bus and click over the “Watch selected bits” option. Then click over the “reg” bus and select the bit “0” or LSB (Less Significant Bit) from the list and click over the “Watch selected bits” option again. The eye symbols mean that watches are created correctly.

Once the watches are established, click over the “Debug” tab. The first step is to reconfigure the bitstream.

The next step is to reload a vector set to feed the target FPGA.

This example needs five clock cycles to start (four cycles for the reset and one more to initialize the count).

Then click over the "Step cycles" option by five cycles again to see the correct behaviour of the counter. For each watch there are two waveforms, the upper waveform is the GOLD reference, this waveform is not affected for the injections. The bottom waveform is the SEU reference, this waveform is affected for the injections. If the waveforms are green it means that both outputs are the same but if the colour is red it means that the outputs have discrepancies.

To generate a manual injection, click over the LSB watch and then change the value to “1”. This action is like to introduce a bitflip into the design.

Now it is possible to see the discrepancies between the GOLD and the SEU outputs.

Closing The Tool

To exit correctly, click over the button displayed in the following picture and select the three options in order. Firstly “Release FTUNSHADES device”, secondly “Close TNT session” and thirdly “Close project”. Finally logout the session.

Working with partial bitstreams

This chapter describes how to work with partial reconfiguration bitstreams, keeping user Flip-flop contents intact. This can be used to work with designs that use partial reconfiguration, either to optimize FPGA resource utilization, to measure effectivenes of adaptative reconfiguration strategies, or to test scrubbing schemas.

This chapter describes the processes needed to work with multiple partial bitstreams in FT-Unshades2:

- Design preparation for partial reconfiguration.
- Adding multiple bitstreams to a user project.
- Working with multiple bitstreams in the debugger.

Design preparation for partial reconfiguration

The user must prepare his/her design for partial reconfiguration. A good document that explains the creation of partial bitstreams is Xilinx's XAPP290

Since the partial reconfiguration is done keeping the internal state of the flip-flops, the user must take care that the flip-flop locations are not changed during the reconfiguration. This can be manually checked by comparing the flip-flop locations in the initial `.ll` with their locations in the `.ll` of the modified design.

Also, constraints may be added to flip-flop and other element locations, see Xilinx Constraints Guide for details.

As a result of the design preparation process, the user should have at least one full bitstream for the initial FPGA configuration and a number of partial bitstreams that can be applied to the full bitstream to change some of its features.

Example design

An example design for the Virtex-5 FX70T FPGA is provided here. This design contains three .bit files:

- `counter8bit.bit`: Full bitstream of an 8-bit counter that counts up
- `up2down.bit`: Partial bitstream to change counter direction from up to down
- `down2up.bit`: Partial bitstream to change counter direction from down to up

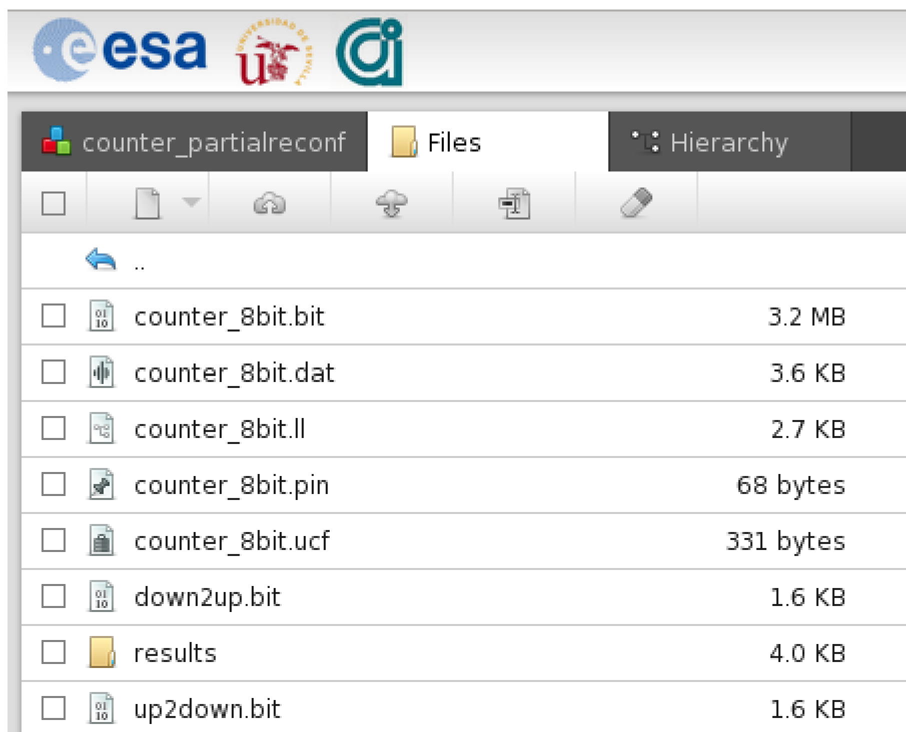
The design has been created following these steps:

1. Create an 8-bit counter design and perform its FPGA implementation for FT-Unshades2.
2. Modify the 8-bit counter sources so the count is decrementing instead of incrementing.
3. Perform an implementation of the modified design, without overwriting the implementation files for the first design.

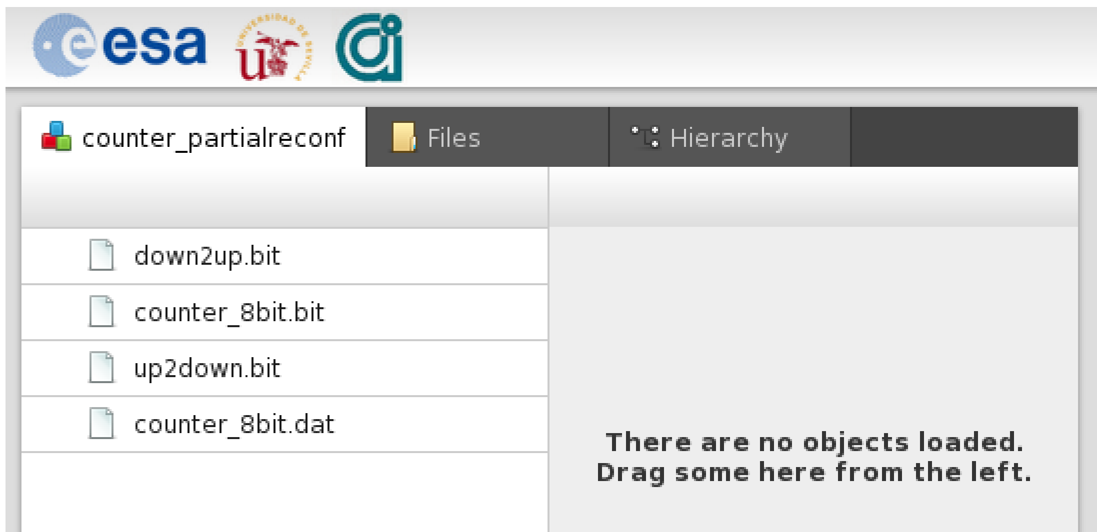
4. Manually check, in the generated `.ll` files for both designs, that the flip-flop locations have not been changed by the implementation processes. In case flip-flop locations have changed, constraints should be added to the design to assure fixed placement of the locations whose value must persist between configurations. See Xilinx Constraints Guide for details.
5. Use bitgen to create a partial bitstream to convert the first design into the second (change counter direction from up to down). See Xilinx's XAPP290 for details on how to perform these operations.
6. Optionally, use bitgen to create a partial bitstream to convert the second design into the first (change counter direction from down to up).

Adding multiple bitstreams to a user project

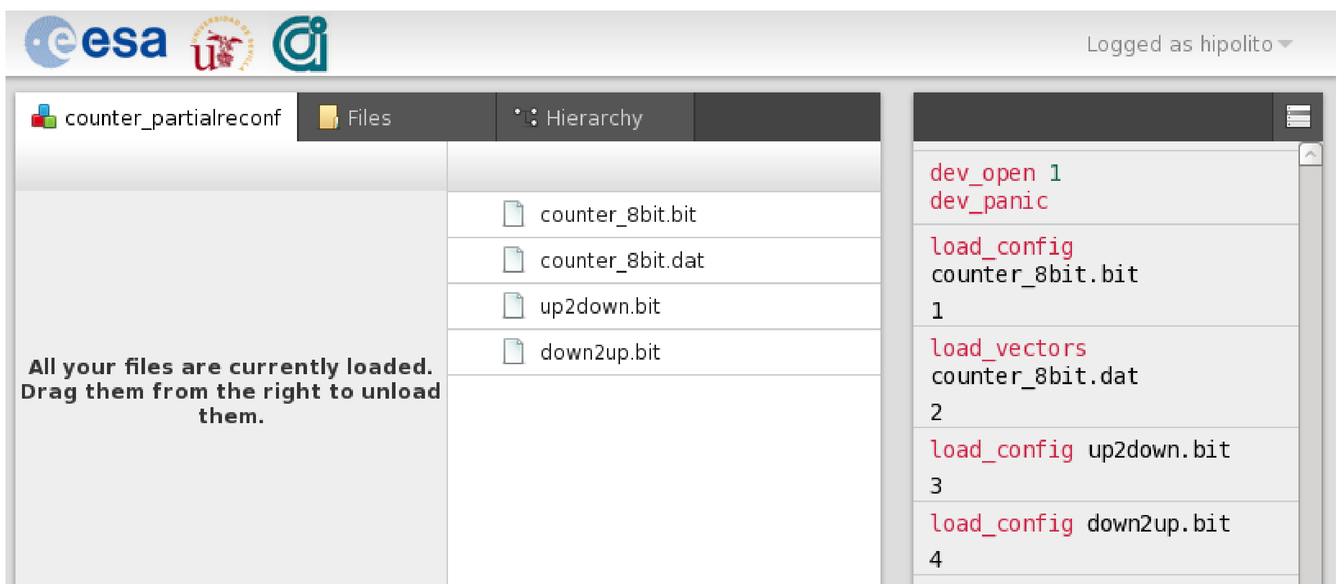
Adding multiple bistreams to a user project is easy using UFF. The user must just upload all the .bit files relevant to his/her design. Please note that the two partial bitstreams are smaller in size compared to the full bitstream:



When opening a device, the user will see all objects that can be loaded into the FT-Unshades2 hardware. In this case, three bitstreams and one vector file:



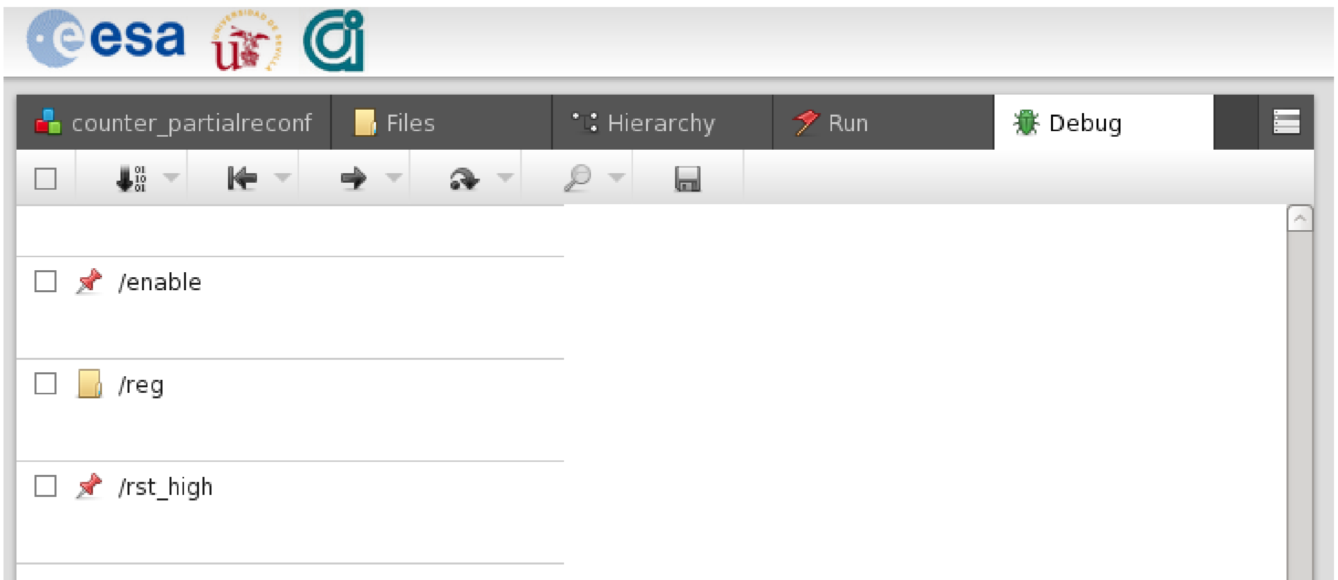
The user may load the objects into the hardware, so they are available to be used later. The shell tab shows the TNT commands generated by the UFF so the user can generate his/her own scripts to automate this process if needed:



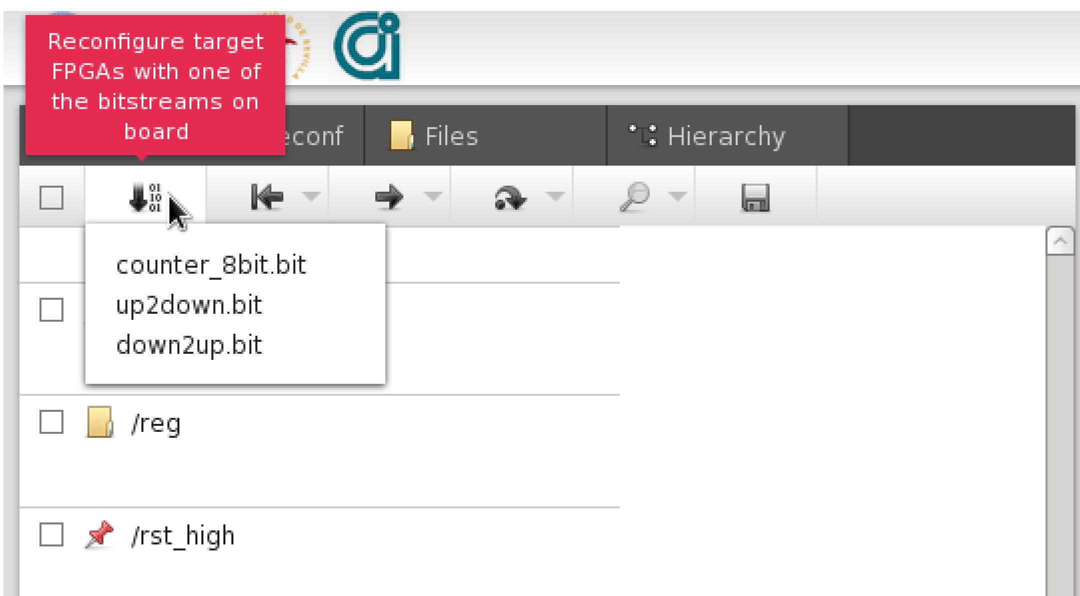
Working with multiple bitstreams in the debugger

The user can load different bitstreams interactively using the hardware debugger, and check the effects of the reconfiguration in the design behavior by observing the waveforms. If batch processing is needed, a script can be made using TCL.

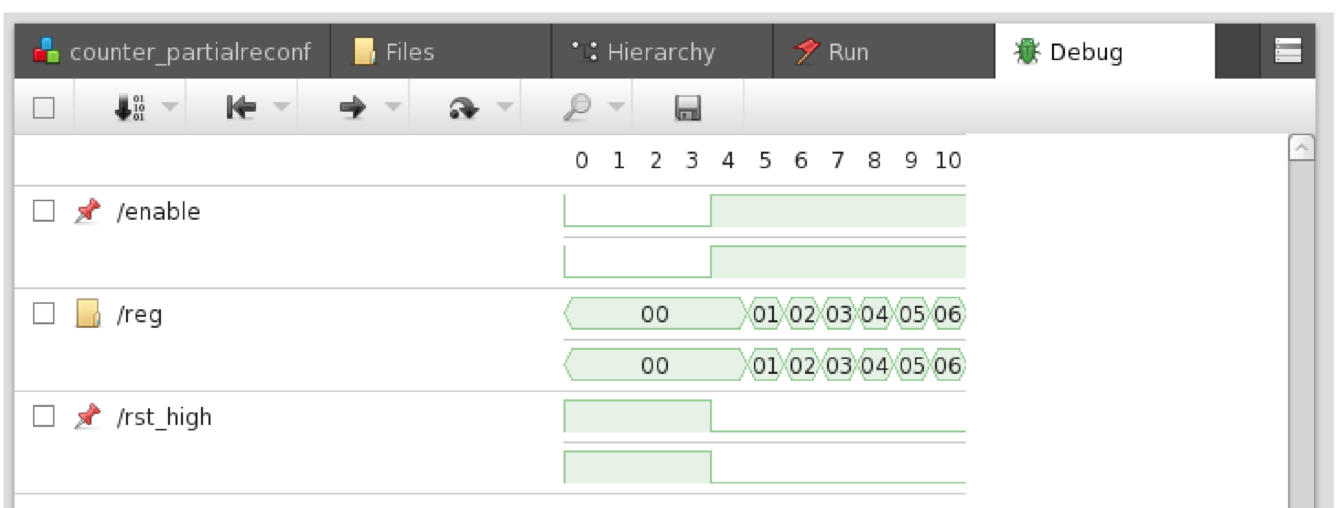
The user can define watches to observe the internal state of the design in the hardware debugger:



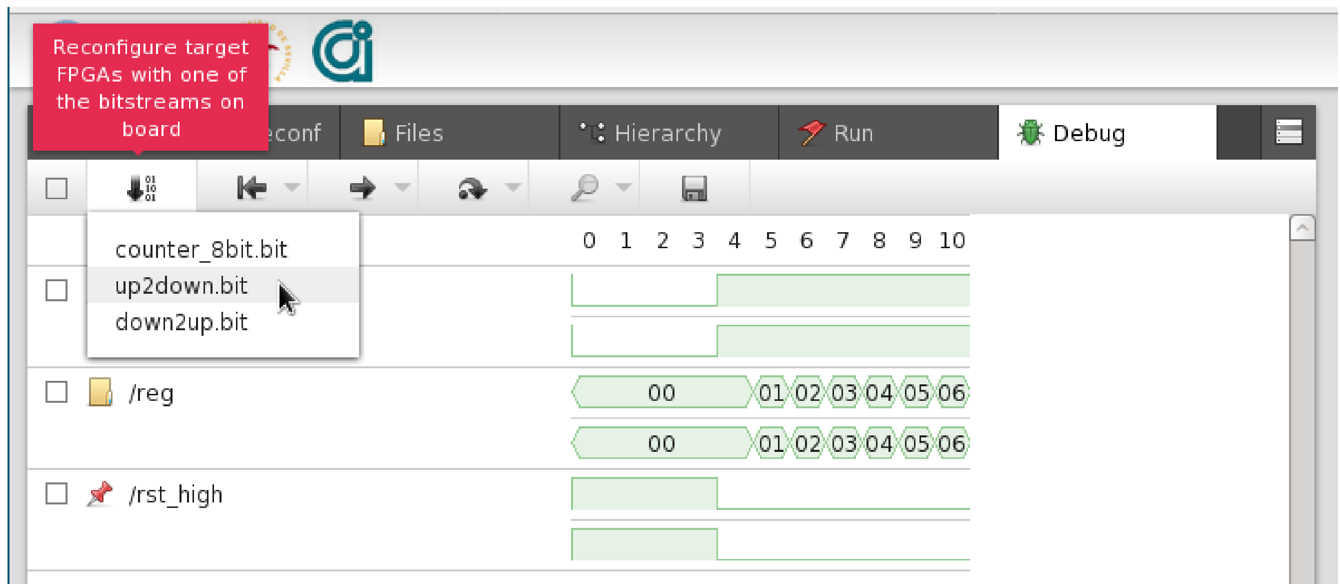
When configuring the FPGA with a bitstream, a drop-down menu lets the user choose which one to use:



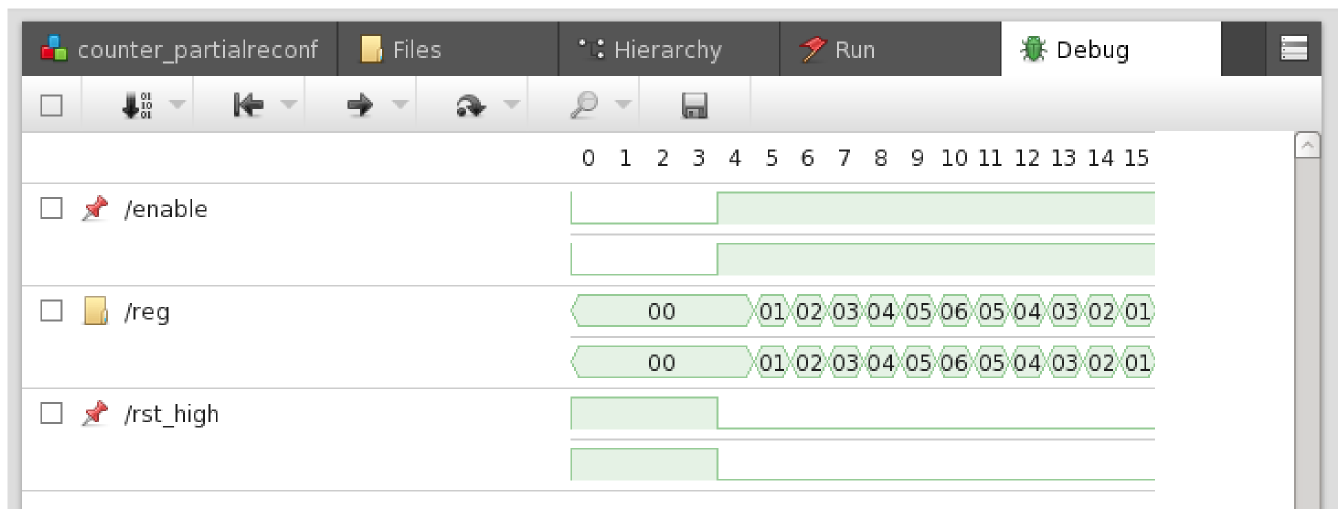
If we configure the FPGA with the full bitstream `counter_8bit.bit`, we can step some clock cycles and watch the counter increase:



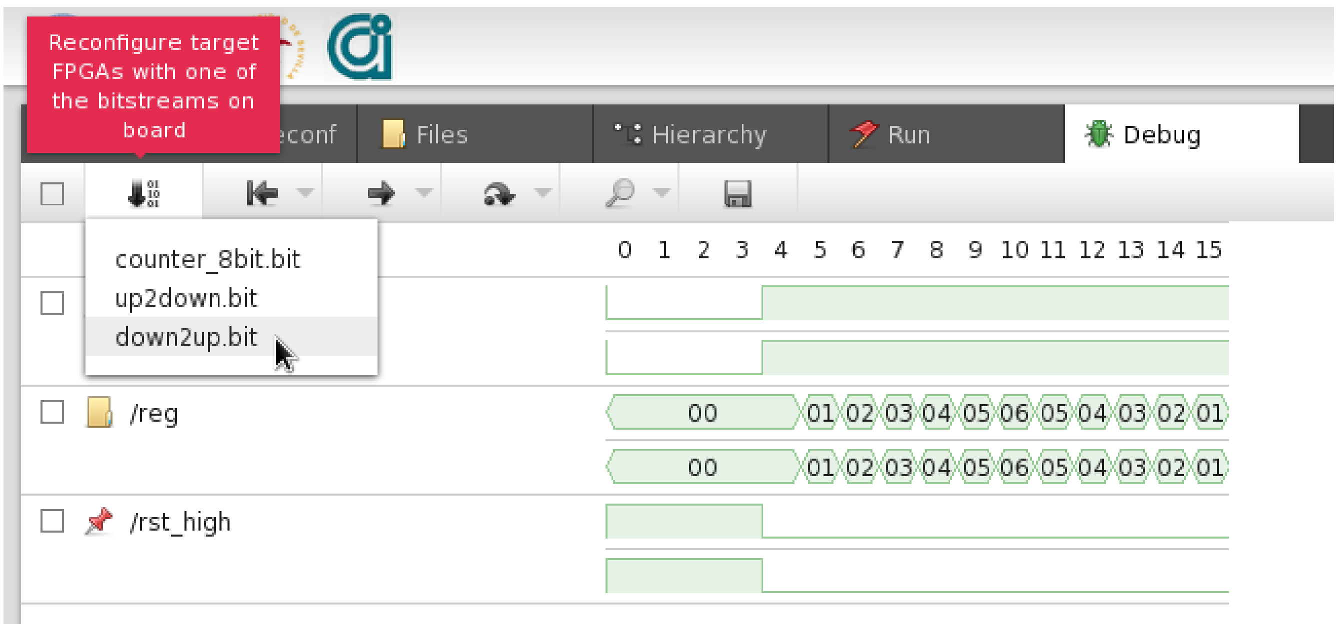
We can then reconfigure the design with the `up2down.bit` bitstream:



If we advance some clock cycles we can see the count going down, from cycle 11 to 15. The state of the user flip-flops is not lost when performing the partial reconfiguration:



Finally, we can reconfigure with the `down2up.bit` bitstream, which will make the counter count upwards again:



If we step the design again, we will see the counter counting up again, from cycle 16 onwards. Note that the flip-flop contents have been maintained between reconfigurations: