FTUNSHADES 2

UFF 3.7 Beginner's Guide



UFF

Table of Contents

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:

	FTUNSHADES Test Analysis Tools
Us Pa U	Isers may only be created through the <u>administration page</u> . You need to enable cookies to log in. Sopyright © 2011-2013 Universidad de Sevilla - AICIA - ESA COC OSA The FTUNSHADES User Friendly Framework v. 1.0 commit: 7aa0dac602019979ca2fbcfed3f51d5374d03c31 Please provide this ID number with all bug reports

And then, the user must create a project:

Logged as usuario 🔻					
Create project					
🗆 ᡖ contador16					
🗆 📥 contador_half7					
🗆 📥 counter8bit					
🗆 📥 counter_8bit_withconfig					
🗌 📥 ejemplo					
🗆 📥 r-vex					
🗌 🛃 zigbee_phy_tx					



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:

() i ftu.us.es/uff/		
Logged as usuario 🔻	Manage files	
ᡖ counter_example	🔓 Files	
		No TNT session
		Run this one

Now, upload the pin file:

Logged as usuario 👻				
hand Upload file to server iles	•: Hierarchy	ፇ Run	🦉 Debug	
★				

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

Log	gged as usuario 👻
C	reate new file
	Plain text file Directory User Constraint File (.ucf) FPGA Vector File (.dat) from VCD dump (.vcd) Logic Location File (.ll) for configuration bits Analysis of campaign results

Select the correct FPGA model and generate the UCF:

Counter_example	_		=	
le				
🗆 📓 counter8bit.pin		68 bytes	13:01	No TNT session
	Generate user constraing file			
	Model of the target FPGA:	xc5vfx70t	7	
	📋 User provided pin (.pin) file:	counter8bit.pin	~	·
	Output file name (optional):			
	Cancel	Generate .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:

L	ogged as usuario 🔻				
	counter_examp fil	ownload selecte es (inside a .zi	ed p)		
	<u> </u>	÷	Ť	2	
	*				
	🖌 counter8bit.pin				
2	a counter8bit.ucf	F			

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:

pecify device and project properties.		
elect the device and design flow for the	project	
Property Name	Value	
Evaluation Development Board	None Specified	•
Product Category	All	\$
Family	Virtex5	•
Device	XC5VFX70T	•
Package	FF1136	\$
Speed	-1	\$
Top-Level Source Type	HDL	\$
Synthesis Tool	XST (VHDL/Verilog)	\$
Simulator	ISim (VHDL/Verilog)	•
Preferred Language	VHDL	\$
Property Specification in Project File	Store all values	\$
Manual Compile Order		
VHDL Source Analysis Standard	VHDL-93	\$
Enable Message Filtering		
	1	

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:

File	<u>Edit View Project Source Process Tools Wi</u>	indow La <u>v</u> out <u>H</u> elp				
) 🖻 🖉 😓 🕹 🖓 🗅 🕞 🗙 🗎	# # Ø Ø # B	🔊 l 🗟 🖻 🖻 🖻	P 12 🕨 🗴	s 🖈 🛛 🖓	
Des	sign 💮 🕀	08	counter8bit.ucf	_ = ×	counter8bit.vhd _	
	View: 🖲 鐷 Implementation 🔿 🛃 Simulation	₹ 1				
89 60	Hierarchy Counter8bit Curver8bit Curver8bit Curver8bit	2 NET 3 NET 4 NET 5 NET	<pre>"clk" LOC = "J20" "rst_high" LOC = " "enable" LOC = " "data_out(7)" L0"</pre>	"K12"; K13"; C = "D25";	<pre>library IEEE; use IEEE.STD_LOGIC_1164.all; use IEEE.NUMERIC_STD.all;</pre>	
00	Counter8bit - counter (counter8bit.vhd Counter8bit.ucf		"data_out(6)" LG "data_out(5)" LG "data_out(4)" LG "data_out(3)" LG	C = "B26"; C = "A25"; C = "B27"; C = "C25";	<pre>encity counterabit is Port (rst_high : in STD_LOGIC;</pre>	
6		<u>¶⊈</u> <u>R</u> un	ta_out (2) " Lo ta_out (1) " Lo	C = "C29";	<pre>data_out : out STD_LOGIC_VECTOR (7 down end counter8bit;</pre>	to
¥		Rerun All Stop	ta_out(0)" L(c - "D26";	architecture counter of counter8bit is signal reg, p_reg: std_logic_vector (7 downto 0);
	No Processes Running	View Text Report			begin	
स्थ इस अप	Processes: counter8bit - counter - 2 Design Summary/Reports - 2 Design Utilities - 2 User Constraints - 2 Synthesize - XST - 3 Synthesize - XST - 4 Implement Design - 5 Generate Regramming File	Force Process Up-to-Da Implement Top Module Design Goals & Strateg Process Properties	ies		<pre>comb: process (reg, rst_high, enable) begin if (rst_high = '1') then p_req <= (others => '0'); elsif (enable = '1') then p_req <= reg + 1; else p_req <= reg; end if;</pre>	×
>	Start Configure larget Device	Design Summa	ry (out of date) 💢 📄	counter8bit.v	/hd 💥 📄 counter8bit.ucf 💥	
-						

B.		Process Properties - Ge	neral Options
<u>C</u> ategory	Switch Name	Property Name	Value
General Options	-d	Run Design Rules Checker (DRC)	V
Configuration Options Startup Options	-j	Create Bit File	V
Readback Options	-g Binary:	Create Binary Configuration File	
Encryption Options	-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 - Configura	tion Options
ategory	Switch Name	Property Name	Value
- General Options	-g ProgPin:	Configuration Pin Program	Pull Up 🔶
- Startup Options	-g DonePin:	Configuration Pin Done	Pull Up
Readback Options	-g InitPin:	Configuration Pin Init	Pull Up 🗘
Encryption Options	-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)	OxFFFFFFF
	-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 Defau
			OK Cancel Apply Help

8.		Process P	roperties - Startup Options x
Category	Switch Name	Property Name	Value
- General Options	-g StartUpClk:	FPGA Start-Up Clock	
- Configuration Options	-g DonePipe:	Enable Internal Done Pipe	
Startup Options Readback Options	-g DONE cycle:	Done (Output Events)	Default (4)
Encryption Options	-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 <u>l</u> evel: <u>Advanced</u> ♀ ♂ Display <u>s</u> witch names <u>D</u> efault
			OK Cancel Apply Help

n •		Proces	ss Properties - Readback Options	
<u>C</u> ategory	Switch Name	Property Name		Value
- General Options	-g Security:	Security	Enable Readback and Reconfiguration	
Configuration Options	-g ReadBack	Create ReadBack Data Files	✓	
Readback Options	-g Persist:	Allow SelectMAP Pins to Persist		
Encryption Options	-1	Create Logic Allocation File		
	-m	Create Mask File		
	-g EssentialBits:	Essential Bits		

∎ .		Process Pr	operties - Encryption Opti	ons
<u>C</u> ategory	Switch Name	Property Name		Value
General Options	-g Encrypt:	Encrypt Bitstream		
Configuration Options	-g Key0:	AES Key (Hex String)		
Readback Options	-g KeyFile:	Input Encryption Key File		
Encryption Options	-g startCBC:	AES Initial Vector		
	-g JTAG_SysMon:	JTAG to System Monitor Connection	Enable	

Finally, run "Generate Programming File" to get the bitstream:

File	<u>Edit View Project Source Process 1</u>	<u>T</u> ools <u>W</u> indow La <u>v</u> out <u>H</u> elp		
) 🌶 🖬 🖗 😓 🛯 🗶 🖻 🖌 🚺	10 » 10 10 10 10 10 10 10 10 10 10 10 10 10	- 🔁 l 🗟 🗄 🖻 🖻 🖉 🖌 😢 🕨	c 🖈 🛛 🖓
De	sign	elox 📄	counter8bit.ucf	counter8bit.vhd _ 🗆 🛪
	View: 🖲 🎆 Implementation 🔿 🧱 Simula	ation 4 1		
a	Hierarchy	2 NE	T "clk" LOC = "J20"; T "rst bigh" LOC = "K12";	library IEEE; use IEEE.STD LOGIC 1164.all:
-	counter8bit	4 NE	T "enable" LOC = "K13";	use IEEE.NUMERIC_STD.all;
80		5 NE	T "data_out(7)" LOC = "D25"; T "data out(6)" LOC = "B26";	entity counter8bit is
=	 Counter8bit - Counter (counter Counter8bit ucf 	rolit.vnd) 7 NE	T "data_out(5)" LOC = "A25";	Port (rst_high : in STD_LOGIC;
	- councerobicider		T "data_out(3)" LOC = "C25";	enable : in STD_LOGIC;
4		" <u>R</u> un	"data_out(2)" LOC = "C29"; "data_out(1)" LOC = "B28";	<pre>data_out : out STD_LOGIC_VECTOR (7 downto - end counter8bit;</pre>
			"data_out(0)" LOC = "D26";	architecture counter of counterPhit is
		Rerun <u>A</u> II		architecture counter of counterobit is
•		👷 Stop		<pre>signal reg, p_reg: std_logic_vector (7 downto 0);</pre>
•	No Processes Running	View Text Report		begin
	Processes: counter8bit - counter	Force Process Up-to-Date		comb: process (reg, rst_high, enable)
-	Design Summary/Reports			begin if (rst_high = '1') then
24	Design Utilities	Implement Top Module		<pre>p_reg <= (others => '0'); cloif (onable = '1') then</pre>
90	Synthesize - XST	Design Goals & Strategies		p_reg <= reg + 1;
-	🗄 🚺 Implement Design	2 Process Properties		preg <= reg:
	Generate Programming File			end if:
_	Configure Target Device	Too Design Summe	anu (aut of data) 😒 📄	
-	Start Cesign Files C Librar	nes 🖉 Design Summa	ary (out of date) 🐹 📋 counterabit.	na 🗶 📄 counterBbit.uct 🗶

Check that the process finish without errors:

Eile Edit View Project Source Process Tools Window	w La <u>v</u> out	<u>H</u> elp							Θ	
🗋 🎓 🗟 🔏 🗋 🖄 🖄 🖄 🖄 🖄 🖉 🖉 🗌	BBB	æ	3 🏹 🕾 🗉 🗖	🖻 🎤 😵 🕨 🗵 📌 📢	7					
Design model S	🔥 🖹 De	sic		counter8bit Projec	t Status (0	8/10/201	6 - 13:20:04)	1		
View: A Implementation A Implementation			Project File:	counter8bit.xise	Parser Er	rors:		No Errors		
Hierarchy	0		Module Name:	counter8bit	Implemen	tation St	ate:	Programming F	ile Generated	
E Cuntersbit	0		Target Device:	xc5vfx70t-1ff1136	• Erro	rs:		No Errors		
counter8bit - counter (counter8bit.vhd)	0		Product Version:	ISE 14.7	• Wan	nings:		No Warnings		
s counterspic.uct	B-En	ror	Design Goal:	Balanced	• Rout	ting Resu	lts:	All Signals Con	pletely Routed	
No Processes Running			Design Strategy:	Xilinx Default (unlocked)	• Timi	ng Const	raints:	All Constraints	Met	
Processes: counter8bit - counter	38		Environment:	System Settings	• Fina	I Timing S	Score:	0 (Timing Repo	ort)	
- x Design Summary/Reports		ē								
🕱 🗉 🎾 Design Utilities	1/1			Deules Utille	ablas Cuma					
User Constraints Synthesize - XST		Ē	Cilian Landa Unillandia	Device Utiliz	ation Sumr	nary	Augustable	Hallingalow	Neteral	
E DA Implement Design	-		Slice Logic Utilizatio	n		Used	Available	Utilization	Note(s)	
COM Translate	≘- De	etai	Number of Slice Regist	ers		8	44,800	1%		_
CAL Map Place & Boute			Number used as Flip	Flops		8				_
Concerte Programming File			Number of Slice LUTs			8	44,800	1%		
E Gonfigure Target Device			Number used as logi	c		7	44,800	1%	•	
Analyze Design Using ChipScope	•		Number using O5	output only		6				-
🍃 Start 🗠 Design 🖺 Files 🖺 Libraries	\Sigma Desig	ın Sum	mary (Programming File	Generated) 💥 📄 counter8	Bbit.vhd 🛛	< 🗋 🕔	counter8bit.uc	:f 🔀		
Console									€ (
Deserve Monorate Deserversion Edited seveniteted										
WARNING:ProjectMomt - File /home/jbr/projects/co	unter8bit	7coun	ter8bit vhdl.pri is n	nissing.						
WARNING:ProjectMgmt - File /home/jbr/projects/co	ounter8bit	coun	ter8bit_vhdl.prj is n	nissing.						=
Concolo C Errors D Warnings R End in Elles R	eculte									
Console Chois a warnings & Find in Files R	esuits									- 1
		_				_				_

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:

File Edit View Project Source Process Tools W	/indow l	.a <u>v</u> out <u>H</u> elp								E	
> →	B B S	8 8 8 8) 🔁 🗟 🖻		🎤 🏘 🕨 🗵 📌 📢	7					
Design	9 X 🔥	E- Desid▲			counter8bit Projec	t Status (08/10/201	l6 - 13:20:04)		-
View: O an Implementation Rehavioral			Project File:	0	counter8bit.xise	Parser E	rrors:		No Errors		
Historia	• 0		Module Name:	: 0	counter8bit	Impleme	ntation S	tate:	Programming F	ile Generated	=
di letarchy	0	- 6	Target Device:		xc5vfx70t-1ff1136	• Err	ors:		No Errors		
B- 🖾 xc5vfx70t-1ff1136	ø	Ę	Product Version	on: I	SE 14.7	• Wa	rnings:		No Warnings		
bb tb_counter8bit - tb_counter(tb_counter8bit - counter8bit - counter	it.vi t.vh	E- Erron	Design Goal:	E	Balanced	• Rot	ting Res	ults:	All Signals Con	pletely Routed	
			Design Strategy	jy: 2	Xilinx Default (unlocked)	• Tim	ing Cons	traints:	All Constraints	Met	
* •	. 30		Environment:	9	System Settings	• Fin	al Timing	Score:	0 (Timing Rep	ort)	
No Processes Running	A										
Processes: tb_counter8bit - tb_counter					Device Utiliz	ation Sum	mary				•
i Sim Simulator		- 🖺	Slice Logic Utili	lization			Used	Available	Utilization	Note(s)	
🕄 🛛 🖓 Behavioral Check Syntax	- 11	⊟ Detai	Number of Slice F	Registers	;			8 44,800) 1%	,	
	" <u>R</u> un		mber used a	as Flip Flo	ops			8			
	Rerun	All	ber of Slice I	LUTs				8 44,800	1%		
	몇 Stop		mber used a	as logic				7 44,800	1%	,	
	Run V	/ith <u>C</u> urrent D	ata Number usir	ina 05 out	tput only			6		1	•
	Proce	ss Properties.	Programmin	ing File Ge	enerated) 🗶 📄 counters	3bit.vhd	×	counter8bit.u	cf 🔀		
Console										++	
▲WARNING:ProjectMgmt - File /home/jbr/project	ts/count	er8bit/count	er8bit_vhdl.prj	j is mis	ssing.						-
Process "Behavioral Check Syntax" completed	success	fully	arobit undi ard	d de mier							
Twwwwitwo.projectegmt - Fite /nome/jbr/project	LS/COUNT	er obit/count	erentc_vnut.pr)	J 15 M18	eeruâ.						
•											
Console Console Errors 🔔 Warnings 🙀 Find in Fil	les Result	's									
Run highlighted process											

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.

File Edit View Simulation Window Layout H	lelp					
ର 🔿 X ଣି ଣ 🐇 🗐 🕤	M 🐹 I 1 😳 📑 🗖	🗆 🖻 🖉 K?	P P B P 🗟 🗠	n i (* *) 🖬	▶ ▶ X 1.00us ✔ 🔙	🔽 Re-launch
Instances and Processes Objects Instance and Process Name Instance and Process Name to countersbit std logic, 1164 std l	n Objects Name Name Correction and a state of the state of th	Value 0 16600 ps 0 1 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10001101	1999,995 ps	10000 ps 1100100 91 100011101	998 ps	1,000.000 pz
A Instance B Memory Source	* • •	Default.wcfg	X1: 1,000,000 ps			
Console						++ [] Ø X
This is a Full version of ISim. Time resolution is 1 ps Simulator is doing circuit initialization process. Finished circuit initialization process. ISim> restart Console Compilation Log Breakpoints	s 🙀 Find in Files Results 🖬	Search Results				•

File Edit View Simulation Window	Layout <u>H</u> elp					-
🗋 🏓 🖬 🗳 X 🗅 🗅 🗙 🔇	i 🖂 🕅 🖂 🖉		5 🔑 K?	p p p ,	2 🖻 🗠 🖻	2 1 10
Instances and Processes 🛛 🕀 🗊 🖉 🗶	Objects ⊕ □ ∅ 🕱	⊕		0 ps		
	Simulation Objects	P Name	Value	0 ps	1 ps	2 ps
Instance and Process Name Design L		🖹 🕌 cik	1			
b tb_counter8bit tb_counter	Object Name \	Clk_period	10000 ps			
numeric std numeric s	l ftu_cycle_c 0	Le rst_high	1			
-	la enable 0	enable	Θ			
	le clk 1	tu_cycle_count	0			
	data_out[7:0] Ul (data_out[7:0]	U			
	la num vectors 16 1	hum_vectors	100011101			
		2r				
		ų.				
		·				
	B ⁴	6				
	-	s				
	-					
	L.	5		X1:0 ps		
		s 4 🗉 🕨		4		
暑 Instance 🛗 Memory [🔁 Source		•	Default.wcfg*		×	
Console						
This is a run version or isim.						
Time resolution is 1 ps						
Simulator is doing circuit initialization pro	cess.					
Finished circuit initialization process.						
ISIM> restart						
	Proskpoints 🙀 End in I	Filos Posults 🔤 Search I	Populto			_
		nies Results En Search F	Results			

Intersection Mathematical Structure Struct	La <u>v</u> out <u>H</u> elp								
🗋 🏓 🖬 🕾 X 🗅 🗅 🗙 🔇	N ⊂ A 🐼 🛛	1015800	≥	1 1 1 1 1	🗟 🗠 🖻	1 1 1 1	■ ► → ^X	1.00us 🖌 🔄	Re-launch
Instances and Processes	Objects Comparison Objects Simulation Objects Object Name V Content N	> Name > Uby clk 12 clk_period Uby rst.high Uby rst.high 13 clk_to_cycle_count 14 tot_cycle_count 15 rst_data_out(7:0) 12 num_vectors 14 num_vectors	Value 1 10000 ps 1 0 0 100011101	0 ps	1 ps	2 ps	13 ps	14.ps, 15.p	<u>, , , , , , , , , , , , , , , , , , , </u>
		Ei		X1:0 ps					
4 III •		8 4 11					11) v
🟯 Instance 💼 Memory [>] Source		901	Default.wcfg	•	×				
Console									+ 0 Ø X
nime resolution is ± ps									
Simulator is doing circuit initialization pro Finished circuit initialization process. ISim> restart ISim> vcd dumpfile counter8bit.vcd	cess.								
Console Compilation Log	Breakpoints 🙀 Find in	Files Results 🗔 Search	Results						
		- Search							

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

vcd dumpflush quit





File Edit View Simulation Window	Layout Help								eøx
🗋 🤌 🖬 😓 🛛 🖧 🗅 🗅 🗙 🔇	🖌 🖂 🗠 🖉	1018800	5 P K?	p p g p	🗟 🗠 🛨	1 10 21	🖬 🕨 🔎 🔳	.00us 🖌 🌜	📮 Re-launch
Instances and Processes (C)	Objects and a second se	 Name Ug clk, period Ug rst.high Ug rst.high<	Value 1 10000 ps 0 1 100011101 20 180011101	2.854.995 ps	2.854,996 ps	2.854,997 ps 10000 ps 100011101 20 100011101	2,854,998 ps	2,854,999 ps	2,835,000 ps 2 2,855,000 ps 2
	1	*I		X1: 2,855,000 ps					·
•		* • •	(4					
🟯 Instance 🖺 Memory 📔 Source		203	Default.wcfg*		×				
Console									
** Failure:Simulation ended succesfully User(VHDL) Code Called Simulation Stop In process tb_counter8bit.vhd:ftu_endsim INFO: Simulator is stopped. ISim> vcd dumpflush ISim> quit © Console © Compilation Log	Breakpoints 🙀 Find in	Files Results 🔽 Search 1	Results						

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:

Logged as usuario 🔻	
Count Upload file to server s	
	1 2
🛳	
🗌 📝 counter8bit.pin	
🗆 📓 counter8bit.ucf	

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

Log	ged as usuario 👻
Cr	eate new file
	i 60 & 1 0
	Plain text file
	Directory
	User Constraint File (.ucf)
	FPGA Vector File (.dat) from VCD dump (.vcd)
	Logic Location File (.II) for configuration bits
	Analysis of campaign results
<u> </u>	

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.

Logged as usuario 🖛	
ᡖ counter_example 🔤 Files	
🛳	
🗆 💼 counter8bit.bit	3.2 MB
🗌 🍈 counter8bit.dat	265 bytes
🗆 🗟 counter8bit.ll	2.7 KB
🗆 📝 counter8bit.pin	68 bytes
🗆 📓 counter8bit.ucf	325 bytes
Counter8bit.vcd	12.4 KB

Appendix A

Design Source Code:

```
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;</pre>
    else
     p_reg <= reg;</pre>
    end if;
  end process;
  data_out <= std_logic_vector(reg);</pre>
  sinc: process (clk)
  begin
    if (clk = '1' and clk'event) then
      reg <= p_reg;</pre>
    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;
         ftu_cycle_count: integer := 0;
 signal
 -- Inputs
 signal rst_high: std_logic := '1';
 signal enable: std_logic := '0';
              std_logic := '1';
 signal clk:
 -- 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;</pre>
    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';</pre>
   wait for clk_period * 4;
    -- 20 cycles of not reset
    rst_high <= '0';</pre>
   enable <= '1';</pre>
   wait for clk_period * 20;
    -- 5 cycles of not enable
   enable <= '0';</pre>
   wait for clk_period * 5;
    -- 256 cycles of enable
   enable <= '1';</pre>
   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:

Logged as nadie -	िesa 😭 🧭
Counter8bit	
	No TNT session
No project is open	

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".

🗌 📥 counter8bit	

When you open a project, you'll get a message saying "No TNT session"; this means there's no thtsh 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.

着 counter8bit 📕 Files
No TNT session
Run this one

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.

💑 counter8bit 🛛 🔜 Files		
🛳		<u>^</u>
🗌 💼 counter.bit	3.2 MB	01/10/15
🗌 🝈 counter.dat	3.6 KB	01/10/15
🗆 🗟 counter.ll	2.7 KB	01/10/15
🗆 📝 counter.pin	68 bytes	01/10/15
🗌 💿 counter.tcl	300 bytes	01/10/15
🗆 📄 dumpl	3.2 MB	Jul 5
□] dump2	3.2 MB	Jul 5
🗆 📄 dump3	3.2 MB	Jul 5
dump4	3.2 MB	Jul 5
🗆 🗎 dump5	3.2 MB	Jul 5 👤

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.

🕹 counter8bit 📑 Files	ී Hierarchy 🦻 Run	الله Debug	
	Load bits from logic location file	(.11)	-
	 Logic location file: Mount point (optional): 	input/counter_8bit.ll	
	Cancel	Load registers	_

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.

-	counter8bit	🔓 Files	ିଅ Hierarchy	ፇ Run	🐧 Debug	
	<u> </u>	ଦ୍ରୁ ଦ୍ର	2			
	📙 reg/					
	□ ° <u>t</u> ° 0					
	0 °T° 1					
	□ °ĩ° 2					
	□ ° <u>⊺</u> ° 3					
	□ ° <u>⊺</u> ° 4					
	□ ° <u>t</u> ° 5					
	□ ° <u>⊺</u> ° 6					
	□ ° <u>⊺</u> ° 7					

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.

-	nt Watch selected bits 😪 🖓 Hierarchy 🛷 Run 🛛 🦥 Debug	
~	reg/	۲
L .	° <u>L</u> 0 0	
	° <u>t</u> ° 1	
	° <u>T</u> ° 2	
	°Ľ ₀ 3	
	ී <u>රි</u> 4	
	°Ľ ₀ 5	
	°Ľ° 6	
	°Ľ ₀ 7	

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.

着 counter8bit 🔤 Files 🔹 Hierarchy	📌 Run	-₩. Debug		
📆 target FPGA configuration:		input/counter_8bit.bit	-	
🐠 target FPGA vectors:		input/counter_8bit.dat	~	
🧷 batch size:		100000		
🧷 check residual damage:		No	~	
🧷 damage per run:		1		
🧷 drop on damage:		Yes	~	
🧷 show output XOR:		No	~	
🧷 unflip after run:		No	~	
🧷 reconfigure at error:		No	~	
🧷 reconfigure at time:		0		
🗲 injector		Random random cycle/register combinations	-	
Run Campaign				

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:

占 counter8bit	📕 Files	•: Hierarchy 🕇	🖻 Run	🦥 Debug	
📓 target FPC	GA configuration:			input/counter_8bit.bit	~
🐠 target FPC	GA vectors:			input/counter Ohit det	
🧷 batch size		Run campaign			
🧷 check res	idual damage:	🧷 registers:		1	~
damage n	er rup	🧷 cycles:		*	
/2 drap ap d	omogo	🧷 runs:		10	
	amage:	🧷 injections per ru	n:	1	
🧷 show outp	out XOR:	seed:			
🧷 unflip afte	er run:	V 0000.	_		
🧷 reconfigur	re at error:	Cancel	Run Ra	andom Campaign	
🧷 reconfigur	re at time:			0	
두 injector				Random random cycle	e/register combinations 🔷 👻

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.

占 counter8bit 🛛 😽 Files	•: Hierarchy	📌 Run	🕷 Debug	=
	a - 2 -			
🗆 📙 /reg				
	4			

Reconfigure target FPGAs with one of the bitstreams on board Files C: Hiera	archy 🦻 Run	🕷 Debug	Ei
	2		
input/counter_8bit.bit			
🗆 🗾 /reg			
	4		×

Discard the current run and start a new one with the chosen vector set	archy 🍠 Run 🕷 Debug	= 1
	2	
input/counter_8bit.dat		
	न	

占 counter8bit	Step cycles	rchy 🤣 Run	🕷 Debug	
	🔿 🔿 - 🖉	0 - .	•	
🗆 🔜 /reg	1 cycle 5 cycles 10 cycles other	о В		
	until next event			
L				<u> </u>



ᡖ counter8bit 🛛 Files 🔹 Hie	rarchy 🦻 Run 🕀 Debug
	₽ - m
	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
🗆 📙 /reg	0B 00 01 02 03 04 05 06 10
	06 00 01 02 03 04 05 06 10
	4

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 .11 with their locations in the .11 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 .11 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:

📀esa 😭 🧭	
뤔 counter_partialreconf 🛛 📙 Files	°C Hierarchy
	2
la	
🗌 🔢 counter_8bit.bit	3.2 MB
🗌 🍈 counter_8bit.dat	3.6 KB
🗆 🗟 counter_8bit.ll	2.7 KB
🗆 📝 counter_8bit.pin	68 bytes
🗌 📓 counter_8bit.ucf	331 bytes
🔲 📓 down2up.bit	1.6 KB
🗆 📙 results	4.0 KB
🔲 🗊 up2down.bit	1.6 KB

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:

💽 esa 👔 🧭				
ᡖ counter_partialreconf 🛛 📑 Files	℃: Hierarchy			
📄 down2up.bit				
📄 counter_8bit.bit				
📄 up2down.bit				
📄 counter_8bit.dat	There are no objects loaded.			
	Drag some here from the left.			

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:

📀 esa 👔 🧭		Logged as hipolito 👻
ᡖ counter_partialreconf 🛛 📙 Files	•: Hierarchy	
	Counter 8bit.bit	dev_open 1 dev_panic
	 counter_8bit.dat	load_config counter 8bit.bit
	📄 up2down.bit	1
All your files are currently loaded.	📄 down2up.bit	load_vectors
Drag them from the right to unload them.		2
		<pre>load_config up2down.bit 3</pre>
		<pre>load_config down2up.bit 4</pre>

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:

📀esa 👔 🧭				
ᡖ counter_partialreconf 🔡 Files	• 🕻 Hierarchy	💅 Run	🕷 Debug	
	2 - a			<u>^</u>
🗆 📌 /enable	-			
🗆 📙 /reg	-			
🗌 📌 /rst_high	-			

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:

ᡖ counter_partialreconf 🛛 🔓 Files	•: Hierarchy	📌 Run	🐺 Debug	
	2 -			
	0 1 2 3 4 5	6 7 8 9 10		
🗌 📌 /enable				
🗆 📙 /reg	00 00:	1 02 03 04 05 06		
	00 00	1/02/03/04/05/06		
🗆 📌 /rst_high				

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

Re FP	configure targe GAs with one o	et 👌 🤇	Ĵ.				
th	e bitstreams o board	n econf	📙 Files	•: Hierarchy	📌 Run	🕷 Debug	
	₽ ⁰¹	K	÷	2			
	counter_8b	oit.bit		01234	5678910		
	up2down.b down2up.b	it 🖒					
	📙 /reg			(00)(02 03 04 05 06		
				00 00	1 02 03 04 05 06		
	📌 /rst_high						

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:

ᡖ counter_partialreconf 🛛 📙 Files	• : Hierarchy	📌 Run	🐮 Debug 📃
	2		
	0 1 2 3 4 5	6 7 8 9 10 11	. 12 13 14 15
🗌 📌 /enable			
🗆 📙 /reg	 00 00 00 00 	1\02\03\04\05\06\05 1\02\03\04\05\06\05	04\03\02\01 04\03\02\01
🗆 🖈 /rst_high			

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

PGAs with one of the bitstreams on			
board conf Files	•: Hierarchy	📌 Run	🕷 Debug
	2		-
counter_8bit.bit	0 1 2 3 4 5	67891011	1 12 13 14 15
] up2down.bit			
down2up.bit			
] 📙 /reg	00 00	1/02/03/04/05/06/05	5 04 03 02 01
	00 00	1\02\03\04\05\06\0	5 04 03 02 01
] 📌 /rst_high			

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:

ᡖ counter_partialreconf	🔓 Files	🗅 Hierarchy				ፇ Run			X	Tebug				
	🔿 👻 – 🚓 👻	2-	-											
		34	5	6	7	8	9	10 11	12 13	14 15	16 17 1	8 19 2	0 21	22
🗌 📌 /enable														
🗆 📙 /reg		00)@)@	1/02 1/02	03 03	04) 04)	05 05	06 05 06 05	04 03 04 03	02/01 (02/01	02\03\0 \02\03\0	4 05 0 4 05 0	6 07 6 07	08 08
🗆 📌 /rst_high														