Reconfigurable Architecture (4)

<u>osana@eee.u-ryukyu.ac.jp</u>

2 weeks ago:

- Continuous assignment (assign) for combinational logic
 - wire type signals
 - Blocking assignment operator (=)
 - Conditional assignment by "?" operator
 - Pay attention for multiplexors

Last week:

- Procedural assignment (always) for sequential logic
 - * **Clock** signal in "always @ (...) " (usually posedge)
 - reg type signal ("output reg" is also OK)
 - Non-blocking assignment operator (<=)
 - Control structures such as "if" and "case" are available



This week:

- Behavior verification with HDL simulator
 - Things under test: last week's contents
 - See how to writing a testbench
 - Simulation flow with Vivado simulator is also shown



Basic design flow







Simulate always

- In all steps of design flow

* RTL simulation, Post-synthesis simulation, Post-place & route…

* RTL simulation is most important, and usually sufficient for FPGAs: because retry is possible if real-chip doesn't work



Testbench is required

- Circuit never work alone itself
 - Some kind of external input is necessary
- Visual confirmation on waveform is not always perfect / possible
 - * printf()-like debug is also powerful in HDL
 - Especially for event detection: then check waveform



Verilog simulators

- * Xilinx Vivado simulator in this class
 - Cadence: NCsim (NC-Verilog)
 - Synopsys: VCS
 - Mentor Graphics: ModelSim
 - Stephen Williams: Icarus Verilog (open source)



Waveform viewers

- Integrated with simulator engine
 - Vivado Simulator, ModelSim
- Or provided separately
 - * VCS (DVE), NCsim (simvision), Icarus Verilog (gtkwave)





Waveform files

- * VCD (Value Change Dump): standard ASCII format
 - * VCD is accessible by all simulators and viewers, but large
- Commercial simulators have their proprietary, compact formats
 - VPD (VCD Plus): Synopsys, SHM (Simulation History Manager): Cadence, WLF (Waveform Log File?): Mentor Graphics

Testbench vs RTL (1)

- * Testbench has "flow of time"
 - initial statement, \$finish, and `timescale'
 - * RTL has only events, but no beginning, history or end

Testbench vs RTL (2)

- Testbench has no port
 - Because testbench has everything outside RTL
- Testbench is not for synthesis: all syntax in Verilog is available
 - System tasks and many other syntax for simulation control

Writing flow of time (1)

- * `timescale : specify unit time and time resolution
 - * "`timescale lns/lps" is commonly used
 - # "#1" for 1ns delay
 - Delays < 1ps are rounded off</p>
- #: delays evaluation or assignment (along `timescale)

Writing flow of time (2)

- initial: procedural assignments evaluated at t=0
 - Time course events with # operator
- * always #: procedural assignments periodically evaluated
 - * always # (10) to be evaluated every 10 unit time
 - Convenient for generating clock signals



Example testbench: Step 1

- Clock period of 10ns:
 100MHz
 - * 11ns to reset
 - * 31ns to release reset
- No "unit under test" yet

`timescale 1ns/1ps

```
module testbench ();
  reg CLK, RST;
```

```
initial CLK <= 1;
always # (5) CLK <= ~CLK;
initial begin
    RST <= 0;
#11
    RST <= 1;</pre>
```

```
#20
    RST <= 0;
    end
endmodule</pre>
```

Example testbench: Step 2

- * With parameter
 - Better abstraction for clock period
 - Real type to prevent loss of digits
 (ex: Step=4 and 1.1*Step)

`timescale 1ns/1ps

```
module testbench ();
  reg CLK, RST;
  parameter real STEP = 10;
```

```
initial CLK <= 1;
always # (STEP/2) CLK <= ~CLK;
initial begin
   RST <= 0;
#(1.1*STEP)
   RST <= 1;
#(2*STEP)
   RST <= 0;
end
endmodule
```

RTL constructs in TB

- * Ex: Clock counter
 - Good with waveform
 viewer
 - Also convenient with
 \$display (shown later) and
 other system tasks

```
initial CLK <= 1;
always # (STEP/2) CLK <= ~CLK;</pre>
```

```
reg [31:0] CLK;
always @ (posedge CLK)
    CNT <= RST ? 0 : CNT+1;</pre>
```

System tasks

- Command-like constructs for simulators
 - Handling waveform files

 - Mathematical functions in real (FP) type: \$sin, \$cos…
 - Mostly ignored by synthesis tools (or causes an error)

* Displaying messages or reading/writing files in simulation



\$display, \$write: stdio (1)

- * \$display ("format", signal1, signal2…);
 - printf()-like function with newline at the end (\$write w/o NL)
 - * %b: binary, %d: decimal, %h: hexadecimal, %f: real
 - * \t and \n for tab and newline
 - Called within initial / always block

Smonitor: stdio (2)

- Similar to \$display, called on its change
 - * \$monitoron / \$monitoroff to suspend and resume



\$f{display, write, monitor}

File access, almost same with standard C library

- * mcd = \$fopen("filename");
- \$fdisplay(mcd, "format", signal, signal...);
- * \$fclose (mcd);

Obtaining simulation time

- * \$realtime
 - Returns "real" time, in second
 - \$display("Time = %f ", \$realtime);
- * \$time
 - * 64bit integer, unit is `timescale

Terminating simulation

- * \$finish: terminates simulation
 - (while others don't support this)
- In Vivado simulator, this is not mandatory because length of simulation can be specified in GUI
 - Important with command-line based simulators *

Some simulators displays CPU time with \$finish(1) or \$finish(2)



Data conversion

- * \$itor, \$rtoi: real <-> integer
- * \$random, \$sin, \$cos,…: many other (mathematical) functions

* \$bitstoreal, \$realtobits: real <-> 64bit signal in IEEE-754 standard

Convenient in debugging scientific computing applications



Saving waveform

- * \$dumpfile("foo.vcd"); save waveform in "foo.vcd"
- \$dumpvars(0); Record all signals in VCD file above
 - Vendor specific system tasks for vendor specific files
 - In Vivado simulator, no \$dumpfile is required to see waveform



System tasks: summary

- * \$display, \$monitor, \$write, \$fdisplay, \$fmonitor, ...
- * \$realtime, \$time
- * \$finish
- \$bitstoreal, \$realtobits, \$itor, \$rtoi, \$sin, \$cos, ...
- \$dumpfile, \$dumpvars *



Module under test (uur)

- * Becomes a submodule of testbench
 - Input signals generated in testbench, usually as reg variable
 - * Other system model (i.e, DRAMs) may also included as submodules, and connected to UUT the by wires
 - Output signal may be connected wires, or left unconnected



Simple example

`timescale 1ns/1ps module sw_led_tb (); reg [3:0] SW; reg PUSH; wire [3:0] LED; sw_led uut (.SW(SW), .LED(LED), .PUSH(PUSH)); initial begin SW <= 4'b0001; PUSH <= 0; $\#(10) SW <= \{ SW[2:0], SW[3] \};$ $\#(10) SW <= \{ SW[2:0], SW[3] \};$ end endmodule

```
module sw led
   input wire [3:0] SW,
   input wire PUSH,
  output wire [3:0] LED
  );
 wire SW1_ = \sim SW[1];
  assign LED[0] = PUSH & SW[0];
  assign LED[1] = SW1_;
  assign LED[2] = SW1_ \& SW[2];
  assign LED[3] = |SW;
```

endmodule



Observing signals

Waveform viewer: walking down module hierarchy

- Can see any signal in the design,
- But usually not easy



Test logic in HDL

- Writing test assistance logic in testbench
 - * Adding signals such as "OK" with simple combinational logic
 - Using \$display in some specific conditions
- Testbench can access signals inside UUT



Accessing signals inside

- * For simulation only:
 - * i.e) uut.SW1_
 - Can go even deeper by: inst1.inst2.inst3.signal
- Not good for synthesis
 - Not possible in VHDL

```
module sw_led
  (
    input [3:0] SW,
    input PUSH,
    output [3:0] LED
  );

wire SW1_ = ~SW[1];
assign LED[0] = PUSH & SW[0];
assign LED[1] = SW1_;
assign LED[2] = SW1_ & SW[2];
assign LED[3] = [SW;
```

endmodule



Try it

- Simulate (and implement next week) in Vivado
 - Write a testbench and RTL
 - * A "project" must be generated first, with target device
- For simulator, the target device is not essential



Where to place source files

Inside project folder, in Vivado's default

- CAD generates a lot of files, separating your own source code is important

Project is sometime broken, or becomes a problem in reuse



Typical directory organization

- Separate source and project
 - Source code is managed manually
 - Project only refers the source files





FPGA ordering # (or model #)

- Device family
 - * Xilinx: Virtex, Kintex, Artix, Zynq, Spartan, …
 - Altera: Stratix, Arria, Cyclone, MAX, …
- Device size and additional features
- * Package and speed grade



Speed Grade: 1 Package: CSG324 Temperature range: Commercial 324 Pins

Hands-on

- Create a Vivado project
 - Write a simple RTL and testbench
 - Then run simulation
- For simpleness, source files in project directory

Launch Vivado

* and "Create Project"

Vivado 2017.3

<u>File Flow Tools Window H</u>elp Q- Quick Access

Quick Start

Create Project > Open Project > Open Example Project >

Tasks

Manage IP > Open Hardware Manager > Xilinx Tcl Store >

Learning Center

Documentation and Tutorials > Quick Take Videos > Release Notes Guide >

Recent Projects

vivado2 /home/osana/work/mblaze/vivado2

vivado-test /home/osana/work/cluster/vivado-test

kc705-riffa /home/osana/work/cluster/kc705-riffa

KC705_Gen1x8lf64 /home/osana/work/riffa/riffa_2.2.2/source/fpga/xilinx/kc705/KC705_Gen1x8lf64/prj

kc705-riffa /home/osana/work/cluster/tmp/kc705-riffa

kc705-riffa /home/osana/work/tmp/cluster/kc705-riffa

vivado /home/osana/work/cluster/netfpga/vivado

vivado /home/osana/work/nexys4/vivado

Recent IP Locations

core /home/osana/work/cluster/src/aurora/netfpga

core /home/osana/work/cluster/src/top/slave/ku040

core /home/osana/work/cluster/src/top/pcie-master/netfpga

core /home/osana/work/cluster/src/top/pcie-master/kc705 core

Tcl Console	
Q, X ♦ II 🗉 III 🖬 III	
start_gui	
Type a Tcl command here	

Create project (1/5)

Just click next

X New Project

Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

To continue, click Next.

< <u>B</u> ack	<u>N</u> ext >	<u>F</u> inish	Cancel

Create project (2/5)

- Name and location
 - * Name "vivado_lab_sim"
 - Folder with the project name is created

• • •	X New Project	t	
Project Name Enter a name	e for your project and specify a directory where the project	ct data files will be stored.	1
Project name: Project location:	vivado_lab_sim /home/osana ct subdirectory eated at: /home/osana/vivado_lab_sim		
		< <u>B</u> ack <u>N</u> ext > <u>F</u> inish	Cancel

Create project (3/5)

* "RTL Project" is the basic

* Still have no source code: "Do not specify sources…"

	New Project
Proj S	iect Type Specify the type of project to create.
۲	<u>R</u> TL Project You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis. <u>In D</u> o not specify sources at this time
0	<u>P</u> ost-synthesis Project You will be able to add sources, view device resources, run design analysis, planning and implementation. Do not specify sources at this time
0]/O Planning Project Do not specify design sources. You will be able to view part/package resources.
0	I <u>m</u> ported Project Create a Vivado project from a Synplify, XST or ISE Project File.
0	Configure an Example Embedded Evaluation Board Design Create a new Vivado project from a predefined IP Integrator template design.
	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish Cancel

Create project (4/5)

* Choose device

- * XC7A100TCSG324-1
- Enter in "Search" field, or search by category / family (Artix-7)

• • •			C	X New Proj	ect				
Default Part									
Choose a default Xi	linx part	or board for	your project.	This can be (changed later				
Select: 🔷 Parts 📓 4 Filter	Boards								
Produ <u>c</u> t category.	All		-		<u>P</u> ackage:	All	-		
<u>F</u> amily:	AI	*			Spee <u>d</u> grade	: Al 🔻			
S <u>u</u> b-Family:	All			*	<u>T</u> emp grade	: All 🔻			
				Reset All Fil	ters				
Search: 🔍 xc7a100t	tcsg		(4	matches)					
Part		I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceivers	GTXE2
🔷 xc7a100tcsg324-3		324	210	63400	126800	135	240	0	0
🔷 xc7a100tcsg324-2		324	210	63400	126800	135	240	0	0
🔷 xc7a100tcsg324-21	L	324	210	63400	126800	135	240	0	0
xc7a100tcsg324-1		324	210	63400	126800	135	240	0	0
•	11111								
					< <u>B</u> a	ack <u>N</u> ex	t > <u>E</u> ir	nish Ca	ancel

Create project (5/5)

Check the settings

Vivado screen

💳 project_1 - [/home/osana	/work/cluster/project_1/project_1.xpr] - Vivado 3	2017.3
<u>F</u> ile <u>E</u> dit F <u>l</u> ow <u>T</u> ools <u>W</u> indow	/ Layout View Help Q+ Quick Access	
	t∺ ✿ Σ ∞ <i>∞</i> ≫	
rl	PROJECT MANAGER - project_1	
✓ PROJECT MANAGER		Project Summary
Settings		Project Summary
Add Sources		Settings Edit
Language Templates	 Design Sources Constraints 	Project name:
👎 IP Catalog	✓ 🖨 Simulation Sources	Project location:
	sim_1	Product family:
✓ IP INTEGRATOR Croate Block Design	Design	Project part:
Open Plack Design	Hiorarchy	Target language:
Generate Plack Design	Пегагспу	Simulator language:
Generate block Design		
✓ SIMULATION		Synthesis
Run Simulation		Status:
	Hierarchy Libraries Compile Order	Messages:
> RTL ANALYSIS		Part:
	Properties ? _ D & X	Strategy:
► Run Synthesis	$\leftarrow \Rightarrow \diamondsuit$	Report Strategy:
> Open Synthesized Design		
		DRC Violations
		Dup Impleme
Run Implementation	Select an object to see properties	Kurimpiene
 Open Implemented Design 	Select an object to see properties	Utilization
✓ PROGRAM AND DEBUG		othization
Senerate Bitstream		Run Synthes
> Open Hardware Manager		< (
	Tcl Console Messages Log Reports Design Run	s ×
	Q ≚ ≑ I ≪ ▶ ≫ + %	
	Name Constraints Status WNS TNS WH	IS THS TPWS Tot
	▶ impl_1 constrs_1 Not started LOgS	and mes

	C	Poody Poody Poody
,		? 🗆 🖒 X
	project 1	
	/home/osana/work/cluster/proje	ect_1
	Artix-7	
	xc7al00tcsg324-1	
e:	Not defined	
0.0	Verilog	
ge:	Mixed	
		Implementation
(
	No errors of warnings OUC,	Message <mark>s:</mark> N
	xc7a100tcsg324-1	Part: x
	Vivado Synthesis Defaults	Strategy: V
	Vivado Synthesis Default Reports	Report Strategy: V
		incremental complie: N
		Timing
eme	ntation to see DRC results	Run Implement
		Power
hes	is to see utilization results	Run Implement
-		
		? _ 🗆 🖸
Tot	al Power Failed Routes LUT FF	BRAMS URAM DSP St
	sages	

"Default Layout" makes reset the screen

Add source code (1/4)

* "Add Sources" to add one

	\chi vivado_lab_sim - [/home/osana	a/vivado_lab_sim/vivado	_lab_sim.xpr] - Vivado 2014.3	
<u>File E</u> dit F <u>l</u> ow <u>T</u> ools <u>W</u> indow La	yout ⊻iew <u>H</u> elp			Q- Search commands
🯄 🖻 🕼 🖉 🐘 🐘 🗙 👂 🕨	🛅 🚳 % ∑ 🤪 😬 Default Layout			Ready
Flow Navigator	Project Manager - vivado_lab_sim			×
🔍 🛣 🖨	Sources	_ 🗆 🖻 ×	∑ Project Summary ×	ロ ピ ×
Project Manager	🔍 🔀 🚔 📷 🔂 📓 🛃		Project Settings	▲
Project Settings	- Constraints		Project name: vivado_lab_:	sim
👌 Add Sources	👳 🗁 Simulation Sources	Properties	Ctrl+E Ctrl+E Ctrl+E	ia/vivado_lab_sim
💡 Language Templates		Hierarchy Undate	duct family. Artix-7	
🖵 IP Catalog		🔿 Refresh Hierarchy	ject part: <u>xc7a100tcs</u>	<u>g324-1</u>
 IP Integrator 		IP Hierarchy	p module name: <u>Not defined</u>	
🍰 Create Block Design	Hierarchy Libraries Compile Order	Edit Constraints Sets.	nthesis	
💕 Open Block Design	🖧 Sources 💡 Templates	Edit Simulation Sets	·atus: 🔶 Not started	
🧠 Generate Block Design	Properties	🚰 Add Sources	Alt+A essages: No errors or warning	gs
Simulation	← → 100 k		Part: xc7a100tcsg324-1	
🚳 Simulation Settings			Strategy: <u>Vivado Synthesis De</u>	faults
🔍 Run Simulation				
RTL Analysis				
Den Elaborated Design				Þ
	Design Runs			_ 🗆 🖻 ×
 Synthesis 	🔍 Name	Constraints WNS	TNS WHS THS TPWS Fa	ailed Routes LUT FF B
🎨 Synthesis Settings		onstrs_1		
Run Synthesis				
Open Synthesized Design	•			
 Implementation 				
🊳 Implementation Settings				
Run Implementation	-			
Open Implemented Desig				
Program and Debug	🔲 Tcl Console 💭 Messages 🔍 Lo	og 🗋 Reports 🗊 Desig	gn Runs	
プロジェクトに追加するソース ファイルを	指定または作成			

Add source code (2/4)

RTL corresponds to "Design Source"

Add Sources Add Sources This guides you through the process of adding and creating sources for your project Add or <u>c</u>reate constraints Add or create design sources Add or create simulation sources 🖸 XILINX ALL PROGRAMMABLE ? < <u>B</u>ack Next > <u>F</u>inish Cancel

Add source code (3/4)

- "Create File" because there's no file yet
 - * Name "sw_led"
- If there's already source file, do "Add Files"

	000	X Create Source File				
	Create a new project.	/ source file and add it to your	4			
	<u>F</u> ile type:	🕡 Verilog	-			
	F <u>i</u> le name:	sw_led				
	Fil <u>e</u> location:		<u> </u>			
		OK Canc	el //			
00		X dd Sources				
Add or Create Design Sources Specify HDL and netlist files, or dire	ctories containi	ng HDL and net st files, to add to	your project. C	ireate a new so	ource file on	
disk and add it to your project.						
Index Name Library	Location	rt>				-
Switca.t xitacidakiis						1
						*
	<u>A</u> dd Files	Add Directories	<u>C</u> reate File			
Scan and add RTL <u>i</u> nclude files into	project					
Copy <u>s</u> ources into project						
Add so <u>u</u> rces from subdirectories						
			< <u>B</u> ack	<u>N</u> ext >	Einish	Cancel

Add source code (4/4)

 Vivado will ask the module's port organization

Just ignore it

• • •		X Define Module		
Define a module and s For each port specifier MSB and LSB values Ports with blank na	specify I/O Ports to ad d: will be ignored unless mes will not be written	d to your source file. s its Bus column is checked.).	4	
Module Definition				
<u>M</u> odule name: sv	/_led		\otimes	
I/O Port Definitions	5			
Port Name	Direction Bus input ■	MSB LSB 0 0 1 0 <td>OK Cancel</td> <td></td>	OK Cancel	
			The module definition Are you sure you war	e Module n has not been changed. nt to use these values? <u>No</u>

Add testbench

* "Simulation Source" is that

* Name "sw_led_test"

Rest is same to RTL

Design hierarchy

- * Design Sources = RTL
- Simulation Sources =
 Testbench + RTL
 - Multiple testbenches are
 possible (create other simulation set than sim_1)

Example

Source code in the last slide today

Design hierarchy review

* RTL module under testbench

 "Instance name - module name" is shown

Simulation settings

- "Settings" in Flow navigator
 - * Simulation settings \rightarrow Simulation \rightarrow runtime to 0
 - Default is 1000ns

00	X	Project Setting	gs				
	Simulation						
General	Target simulator:	Vivado Simulator 🔹					•
	Si <u>m</u> ulator language:	Mixed					-
Simulation	Simulation set:	<u>a</u> sim_1					•
>	Simulation top module name:	sw_led_test					8 -
Synthesis	✓ Clean up simulation files						
Implementation	Gene <u>r</u> ate scripts only						
1010				· · · · · · · ·			
Bitstream	Compilation Elaboratio	✓ Simulation	<u>N</u> etlist	Advanced			
	xsim.simulate.runtime*		0				
<u> </u>	xsim.simulate.uut						
<u>I</u> P	xsim.simulate.wup						
	xsim simulate xsim more o	ntions					
		priority					
	vcim cimulata suntimat						
	Specify simulation run time						
	Speeny Sinaletten Fan tine						
				ОК		Cancel	Apply

Run compilation

- ∗ Run Simulation →
 Run Behavioral Simulation
 - * Compiler will be launched
 - Post-synthesis and other
 simulation is also possible

Simulation

To run again…

To add signal(s) to waveform, reset and run again

Recompile when source code is modified

Today's source code

```
`timescale 1ns/1ps
module sw_led_tb ();
  reg [3:0] SW;
  reg PUSH;
 wire [3:0] LED;
  sw_led uut (.SW(SW), .LED(LED),
              .PUSH(PUSH));
  initial begin
    $monitor("%t SW: %b, PUSH: %b", $time, SW, PUSH);
       SW <= 4'b0000; PUSH <= 0;
   #10 SW <= 4'b0001;
   #10 PUSH <= 1;
   #10 SW <= { SW[2:0], SW[3] }; PUSH <= 0;
   #10 SW <= { SW[2:0], SW[3] };
   #10 SW <= { SW[2:0], SW[3] };
  end
endmodule
```

Just copy and paste to TB + RTL file, then it'll work

```
module sw_led
  (
    input wire [3:0] SW,
    input wire PUSH,
    output wire [3:0] LED
  );

wire SW1_ = ~SW[1];
assign LED[0] = PUSH & SW[0];
assign LED[1] = SW1_;
assign LED[1] = SW1_;
assign LED[2] = SW1_ & SW[2];
assign LED[3] = [SW;
```

endmodule

