再構成型アーキテクチャ特論(4)

osana@eee.u-ryukyu.ac.jp

前々回のまとめ

- * 継続代入文 (assign)
 - - ∗ wire 型の信号 (ポート宣言は暗黙の wire 宣言)
 - * ブロッキング代入演算子 (=)

* 組み合わせ回路ができる。必要な演算器などは合成ツールが判断

*?を使った条件付き代入も可能だが、マルチプレクサに注意

前回のまとめ

- * 手続き代入文 (always)
 - * always @ (…) の () の中にはクロック信号「だけ」を書く
 - * reg 型の信号 (output regも): 必要に応じてFFが合成される
 - * ノンブロッキング代入 (<=)
 - * 条件付き代入に加えて if や case といった制御構造が使える



今回の予定

* HDL シミュレータによる動作検証

* 検証の対象は先週までの内容を利用

テストベンチの書き方

* Vivado simulator によるシミュレーションフロー



設計フロー





シミュレーションは重要

* 設計のあらゆる局面で

* RTLを書いて検証、論理合成して検証…

* とはいえ、FPGA では気を付けていれば RTL だけでもなんとか

* 実チップが動かなくてもやり直せる、というのは大きいメリット





テストベンチが必須

- * 回路は勝手には動かない
 - * 外からの入力が必要
- * 波形を目で見て全部チェック、というのは実際ムリ
 - * printf() 的なデバッグは結局のところ、偉大
 - * 特定の状況が発生した時間などがわかれば検証の助けになる





シミュレーションしましょう

- - Cadence: NCsim (NC-Verilog)
 - Synopsys: VCS
 - Mentor Graphics: ModelSim
 - Stephen Williams: Icarus Verilog (フリーソフトウェア)

* Verilog シミュレータが必要 (講義ではXilinx Vivado simulatorを使用)



波形ビューワ

* シミュレータと統合されているもの

- Vivado Simulator, ModelSim
- * 分離されているもの
 - * VCS, NCsim (simvision), Icarus Verilog (gtkwave)





波形 (waveform) ファイル

- * VCD (Value Change Dump): 標準フォーマット。ASCII形式
 - * VCD はどのシミュレータでも生成できる
- * 各社独自フォーマットのほうがファイルサイズが(ずっと)小さい
- WLF (Waveform Log File?): Mentor Graphics

* VPD (VCD Plus): Synopsys, SHM (Simulation History Manager): Cadence,

波形ファイルを何に使うか

* 波形ビューワで見る: 当然です

* 消費電力推定ツールへ入力する:より正確な推定のために必要

* On-chip debug のツールから出力する: 実機の波形みながら問題解決

テストベンチとRTLの違い(1)

* テストベンチには時間の概念がある

initial 文と \$finish、それから timescale

* RTLではイベントはあっても、基本的に時間の経過はない



テストベンチとRTLの違い (2)

テストベンチには入出力ポートがない

* RTLの世界の外側全部を書きます

* テストベンチは論理合成しない: 論理合成を前提としない記述が自由

* システムタスクなど、シミュレーションの制御に関するいろいろ

時間の経過に関連する記述(1)

* `timescale:単位時間と時間解像度を指定 * "`timescale 1ns/1ps" とかが普通 * #1 で 1ns の遅延が生じる * 1ps 未満の遅延は四捨五入する #:文の実行や代入を遅延させる (timescale に依存)





時間の経過に関連する記述(2)

- * initial:時刻ゼロで実行される手続き代入文 * # を使って時系列でイベントを書いていくことができる
- * always #: 一定の時間おきに実行される手続き代入文
 - * always # (10) で 10ns おきに動く、とか
 - クロック作ったりするのに便利

テストベンチの例: Step 1

* クロックは10ns周期: 100MHz

* 11ns でリセット

* 31ns でリセット解除

テスト対象はまだない

`timescale 1ns/1ps
module testbench ();

```
reg CLK, RST;
```

```
initial CLK <= 1;
always # (5) CLK <= ~CLK;
initial begin
    RST <= 0;
#11
    RST <= 1;
#20
    RST <= 0;
end
```

endmodule

テストベンチの例: Step 2

* parameter を使用

- * 周期をわかりやすくする
- * 整数型だと *1.1 などで 桁落ちの可能性がある

`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 で書くものはなんでも書ける

* たとえばクロックのカウンタ

* 波形ツールみながら、
 何クロック目かわかると便利

* \$display(後述)とかでも便利

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

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



システムタスク

* 主にシミュレータが解釈していろいろするためのコマンド的なもの

* シミュレーション中のメッセージ表示や波形ファイルの保存

* 実数型の取り扱いをするための関数 (sin とか cos も) など

* デバッグ用のメッセージを作るのに使う

* 基本的に論理合成ツールは無視する (エラーになるものもある)

\$display, \$write: 標準出力(1)

- * \$display ("フォーマット", 信号, 信号…);
 - * printf() 的なもの。ただし最後は改行される (\$write は改行しない)
 - * %b: 2進、%d: 10進、%h: 16進、%f: 実数 など
 - * \t と \n でタブや改行も OK
 - 表示したいタイミングは initial / always の中で自分で決める



\$monitor: 標準出力 (2)

* \$display と違って、値が変化したら表示される

* \$monitoron / \$monitoroff で停止 / 再開ができる

\$f{display, write, monitor}

- * ファイルに出力する版で、だいたい C と同じ
 - * mcd = \$fopen("filename");
 - \$fdisplay(mcd, "format", signal, signal...);
 - * \$fclose (mcd);

シミュレーション時間の関数

- * \$realtime
 - * real 型で実時間
 - * \$display("Time = %f ", \$realtime); とか
- * \$time
 - ★ 64bit 整数で、timescale をかければ実時間



シミュレーション制御

- * \$finish: シミュレーションを終了
 - * \$finish(1) とか \$finish(2) とするとCPU時間とかが出る (かも)
- シミュレーションを走らせるので必要ない
 - * コマンドラインで使うシミュレータでは重要

* Vivado simulator などではシミュレーションの終了時刻を指定してから



テータ型変換など

* 実数計算をする回路のデバッグで重宝します

* \$itor, \$rtoi: real型とinteger型の変換

* \$sin, \$cos,…: いろいろな関数もあります。 乱数とかも

* \$bitstoreal, \$realtobits: real型と64bitの信号(IEEE-754倍精度)の変換



波形ファイル保存

* ベンダ固有のものもある

■ Vivado simulator などGUI統合型では指定しなくても波形が見られる

* \$dumpfile("hogehoge.vcd"); hogehoge.vcdを波形保存先として開く * \$dumpvars(0); すべての信号を \$dumpfile で開いたファイルに記録



システムタスクまとめ

- * \$realtime, \$time
- * \$finish
- * \$bitstoreal, \$realtobits, \$itor, \$rtoi, \$sin, \$cos, …
- \$dumpfile, \$dumpvars *



テスト対象モジュールの接続

* 通常のサブモジュールと同じ

* inputは基本的にテストベンチの持つ reg で生成

それとの間を wire で接続することも

* output は wire に接続

* テストベンチに他のシステムモデル(メモリなど)を含むなら、



たとえば・・・

`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_ = ~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



信号を観測する

* 波形をビューワで見る: モジュール階層を下へたどっていける

* つまりデザインに含まれるどの信号でも見ることができる

* 波形見てやるだけ、というのは意外としんどい



テスト用のロジックを HDL で書く

テスト用のロジックをテストベンチに作る

* テストベンチで "OK" みたいな信号を作るとか

* 特定の条件で \$display を使うとか

* テスト対象モジュールのポートに出ていない信号は見えないの?



見えます!

* そう、Verilog ならね…

* uut.SW1_とか

* インスタンス名をつないでいけ ばもっと深い階層でもOK

* 合成するコードでは使えない

* VHDL ではムリだそうです

```
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[1] = SW1_;
assign LED[2] = SW1_ & SW[2];
assign LED[3] = |SW;
```

endmodule



やってみる

* Vivadoでシミュレーションする (合成までそのままいけます)

* テストベンチとRTL記述も書きましょう

* ターゲットデバイスを決めてプロジェクトを作る必要あり

* 普通のシミュレータを使う場合

* 純粋なRTLのシミュレーションにはデバイスの指定は不要



ソースファイルをどこに置くか

* Vivado のデフォルトはプロジェクトフォルダの下

* CAD はたいていたくさんのファイルを作ります

* 自分のソースファイルと混じるとややこしい

* プロジェクトを作り直したり、再利用するときに困る



おすすめのディレクトリ構成

* ソースコードとプロジェクト

- * ソースコードのフォルダは 自分で中身を管理する
- * プロジェクトからはソースを 参照するだけにする





FPGAの型番って何…?

- * ファミリ
 - * Xilinx: Virtex, Kintex, Artix, Zynq, Spartan, …
 - Altera: Stratix, Arria, Cyclone, MAX, …
- * デバイスサイズ・追加機能
- * パッケージ・スピードグレード









* Vivado でプロジェクトを作成 * 簡単なRTLとテストベンチを作成 * シミュレーションしてみる * 簡単にするためにプロジェクトフォルダの中にソースを作成



Vivadoを起動

Create Project する

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





プロジェクトを作る (1/5)

* とりあえず次へ

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



プロジェクトを作る (2/5)

* プロジェクトの名前と場所

* "vivado_lab_sim" にしました

* プロジェクト名でフォルダが
 作成されます

	Now Project	
Project Name	X New Project	
Enter a name	e for your project and specify a directory where the project d	ata files will be stored.
Project name:	vivado_lab_sim	
Project <u>location</u> :	/home/osana	
🗹 Create proje	ct subdirectory	
Project will be ci	reated at: /home/osana/vivado_lab_sim	
		Canad



プロジェクトを作る (3/5)

* 基本は"RTL Project"

まだソースがないので、
 "Do not specify sources…"

	New Project	
Pro	r oject Type Specify the type of project to create.	
	 <u>RTL Project</u> 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>Do not specify sources at this time</u> <u>Post-synthesis Project</u> You will be able to add sources, view device resources, run design analysis, planning and implementation. <u>Do not specify sources at this time</u> <u>J/O Planning Project</u> Do not specify design sources. You will be able to view part/package resources. <u>Imported Project</u> Create a Vivado project from a Synplify, XST or ISE Project File. 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





プロジェクトを作る (4/5)

* デバイスを選ぶ

* XC7A100TCSG324-1

* "Search" に入力しても、 フィルタで絞り込んでもOK

New Project									
Default Part									
Choose a default Xi	Choose a default Xilinx part or board for your project. This can be changed later.								
Select: 🛛 🔷 Parts 📓	Boards								
Produ <u>c</u> t category:	All		-		<u>P</u> ackage:	All	-		
<u>F</u> amily:	All	Ŧ			Spee <u>d</u> grade	: Al 👻			
S <u>u</u> b-Family.	All			-	<u>T</u> emp grade	: All 🔻			
				Reset All Fil	ters				
<u>Search:</u> <u>Q-xc7a100</u>	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
🔷 🔷 xc7a100tcsg324-2	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> in	iish Ci	ancel





プロジェクトを作る (5/5)

* 最後確認しましょう

• • •	X New Project
	New Project Summary
	 A new RTL project named 'vivado_lab_sim' will be created.
	 The default part and product family for the new project: Default Part: xc7a100tcsg324-1 Product: Artix-7 Family: Artix-7 Package: csg324
	Speed Grade1
VIVADO.	To support the supplicate slight Finish
	To create the project, click Finish
	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish Cancel





Vivadoの画面

			III Defeult Levent
			😬 Default Layout 🗸 🗸
PROJECT MANAGER	PROJECT MANAGER - project_1		
Settings	Sources ? _ O C	Project Summary	? 🗆 🖒 X
Add Sources		C Sottings Edit	
Language Templates	🗅 Design Sources	Settings Eur	
	> 🚍 Constraints	Project name: project_1	
r IP Catalog	V D Simulation Sources	Project location: /home/osana/work/cluster/proje	ct_1
	l⊒ sim_1	Product family: Artix-/	
Consta Black Design		Project part: xc/a100tcsg324-1	
Create Block Design	レック アリイノ 陌眉	Target language: Verilag	
Open Block Design		Simulator language: Mixed	
Generate Block Design		Sindlator language. Plined	
IMULATION		Synthesis	Implementation
Run Simulation		Status: Not started	Stat
	Hierarchy Libraries Compile Order	Messages: No errors or warnings	Messaues: No
TL ANALYSIS		Part: xc7a100tcsg324-1	Part: xc
	Properties ? _ 🗆 🖸	× Strategy: Vivado Synthesis Defaults	Strategy: Vi
YNTHESIS		Report Strategy: Vivado Synthesis Default Reports	Report Strategy: Vi
			Incremental compile: No
Open Synthesized Design			
		DBC Violations	Timina
PLEMENTATION			
 Run Implementation 		Run Implementation to see DRC results	Run Implementa
Open Implemented Design	Select an object to see properties		
		Utilization	Power
ROGRAM AND DEBUG		Due Crethonic to one utilization require	Due leelee et
🖁 Generate Bitstream		Run synthesis to see utilization results	Run implementa
Open Hardware Manager			
,			
	Tcl Console Messages Log Reports Desig	n Runs ×	2 _ 0 6
	Q ≍ ≑ I ≪ ▶ ≫ + %		
	Name Constraints Status WMS TH	S WHS THS TRWS Total Power Failed Pourtee LLIT FE	
			STURNS OTHER DSF SU

画面のレイアウトが わからなくなったら ここを "Default Layout"



ソースコードを追加(1/4)

* デザイン階層に ソースを追加

COO	X vivado_lab_sim - [/home/osana	a/vivado_lab_sim/vivado	_lab_si	m.xpr] - Viv	ado 2014	.3	O comete com			
File Edit Flow Tools window D	wout view Help						Q+ search con	nmanus		
🖉 📴 🖾 🖉 🖷 🆷 🗙 💊 🕨	🐔 1 🍪 🎉 🔎 🤮 💾 Default Layout	¥ 🕸 🕅 🛃)						Re	ady
Flow Navigator «	Project Manager - vivado_lab_sim									×
< \	Sources	_ 🗆 🖻 ×	Σ	Project Sum	imary ×					×
Project Manager	🔍 🛣 😂 📾 🔂 📓 🛃			Project Set	tings					
	Design Sources		⊜	Project nan	ne:	vivado_lab,	_sim			
😚 Add Sources	Onstraints Onstraints Onstraints	_		- P-pject loca	ation:	/home/osa	ana/vivado_lab_s	sim		
💡 Language Templates	↓ sim_1	Properties	Ctrl+	e duct fan	nily:	Artix-7				
手 IP Catalog		Hierarchy Update)ject par	t:	<u>xc7a100tc</u>	:sg324-1			
4 ID Integrator		P. Lienensky		p module	e name:	Not define	<u>d</u>			
Create Block Design	Hierarchy Libraries Compile Order	Frit Constraints Sate		hthesis						
Open Block Design	& Sources 7 Templates	Edit Constraints Sets		icitesis						-1
Generate Block Design	Properties	Add Sources	Alt+/	atus:	🤿 Not	started				
				essages:	No erro	rs or warnir	ngs			
 Simulation 				Part:	xc/alo	Otcsg324-:	1			
Simulation Settings				Strategy.	<u>vivado :</u>	synthesis Di	erauits			
W Kun Simulation										
 RTL Analysis 			l r	ppc ve-t-e	·					
👂 🔂 Open Elaborated Design				· ·						
▲ Synthesis	Design Runs	Constanting White	TNC	whe	TUC 1					×
Synthesis Settings		Constraints WINS Dinstrs_1	TINS	WHS	THS		Falled Koutes	LUI	FF	BI
Run Synthesis	impl_1 co	onstrs_1								
Open Synthesized Design										
 Implementation Implementation Cottings 										
Run Implementation	4									
Onen Implemented Design										
v open implemented besit										►
Program and Debug	📃 🔲 Tcl Console 💭 Messages 🔤 Lo	og 📄 Reports 🜗 Desi	gn Run	s						
プロジェクトに追加するソース ファイル	き指定または作成									



ソースコードを追加 (2/4)

RTLは "Design Source"





ソースコードを追加 (3/4)

* これから書くので"Create File"

- * 今回はsw_led という名前で
- * 他のフォルダにあるのを 使う場合は "Add Files"

	00	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	8			
	Fil <u>e</u> location:	🔂 <local project="" to=""></local>	-			
		OK Can	el 📈			
• • •		X dd Sources				
Add or Create Design Sources					611	
disk and add it to your project.	ctories containii	ng HDL and net st files, to add to	your project. Crea	ate a new :	source file on	
Index Name Library	Location					
1 sw_led.v xil_defaultlib	<local proje<="" th="" to=""><th>ct></th><th></th><th></th><th></th><th></th></local>	ct>				
						1
	Add Files	Add Directories	Create File	٦		
Scan and add PTL include files into a	project					
\square Convision and add RTE include messines	project					
\square Add sources from subdirectories						
			< Back	Vext >	Finish	Cancel



ソースコードを追加 (4/4)

* モジュールのポートどうする?

* 聞かれるけど気にしない

00		X Define Mod	le		
Define a module For each port spe MSB and LSB va Ports with blan	and specify I/O Ports cified: alues will be ignored ik names will not be v	to add to your source unless its Bus column i vritten.	file. s checked.	4	
Module Definition					
<u>M</u> odule name	: sw_led			8	
I/O Port Defin	itions				
Port Nam	ne Direction	Bus MSB LSB		+	
	input 🕑				
				Ť	
_					
				OK Cancel	
				X	
				🗋 🔿 🛛 🕅 🕅 🕅	Module
			2	The module definition	n has not been change
			9	Are you sure you war	nt to use these values?
				Yes	No
				<u> </u>	



テストベンチも追加

* "Simulation Source" がそれ

* sw_led_test という名前で

* 手順は RTL と同じ





デザイン階層

- Design Sources
 - * RTL
- Simulation Sources
 - * テストベンチ+RTL
 - * 複数のセットを作れる(sim_1)





これ書きましょう

* さっきのスライドのと同じ





デザイン階層ふたたび

* テストベンチの下にRTLが入る

* インスタンス名-モジュール名





シミュレーションの設定

- * Flow navigator から
 - * Simulation settings \rightarrow Simulation で runtime を 0
 - * デフォルトは 1000ns

00	X	Project Setting	js				
	Simulation						
General	<u>T</u> arget simulator:	Vivado Simulator 🔹					
	Si <u>m</u> ulator language:	Mixed					-
Simulation	Simulation set:	👼 sim_1					•
Synthesis	Simulation top module name:	sw_led_test					
	☑ Clean up simulation files						
Implementation	Gene <u>r</u> ate scripts only						
1010	Compilation Elaboration	Simulation	Netlist	Advanced			
Bitstream	xsim.simulate.runtime*	_	0	_			\mathbf{x}
=	xsim.simulate.uut						
	xsim.simulate.wdb						
<u>IP</u>	xsim.simulate.saif						
	xsim.simulate.xsim.more_o	ptions					
	vsim simulata runtima*						
	Specify simulation run time						
	Speeny Smalaton run tine						
				ОК		Cancel	



コンパイルを実行

- * Run Simulation →
 Run Behavioral Simulation
 - 論理合成後のモデルなどでの
 シミュレーションも可能
 - シミュレーションするための
 コンパイルが実行される





シミュレーション





やり直しが必要な場合

波形に信号を追加したら一度リセットが必要 📢 📐 🕅 70 ns 🕒 🧏 📗 🗔





おまけ:今日のソース

```
`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
```

```
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

