## チュートリアル6:WaveFormerを使ったシステム・モデリングとVerilogシミュレーション

本チュートリアルは, WaveFormer Proを使って素早くシステムをモデリングしシミュレーションできることを,ステート・マシンを含む中程度の規模のディジタル・システムを例に示します.

## 1) 例題

12 ビット A/D コンバータ(ADC)によってサンプルされた 64K(64×1024)サンプルのデータに対して , ヒストグラ ムを計算する回路をモデリングする .

この回路は以前に伝統的な EDA ツールを使ってモデリングし,シミュレーションを行うのに,数ヶ月を要した, 実際のシンプルな VME ボードの例です.しかし,WaveFormer Pro を使うことで,きわめて短時間のうちにモデ リングとシミュレーションを完了できます.また,VME バス・インターフェース・プロトコルを扱うフル機能の回路モ デリングおよびシミュレーションもすばやく行えます.

## 2) 本チュートリアルで解説する内容

- 論理式インターフェースを使ったステート・マシンのモデリング
- テンポラル / ラベル式を使った入力信号の生成
- シミュレーション・ログを使ってデザインの入力ミスを発見
- インクリメンタル・シミュレーション
- Verilog-HDL コードの直接入力によるモデリング
- 外部 Verilog-HDL ソース・コード・モデルの利用
- 条件命令を使った 3 ステート・ゲートのモデリング
- リダクション・オペレータを使った n ビット・ゲートのモデリング
- トランスペアレント・ラッチのモデリング方法
- \$display を使った Verilog-HDL ソース・コードのデバッグ
- Time Maker を使ったシミュレーション時間のコントロール
- Wave Former の Report ウィンドウを使った外部 Verilog-HDL ファイルの編集

チュートリアルを始める前に,これから作成する WaveFormer Pro のダイヤグラムの完成版(tutsim.tim)を先 に見ておくとよいでしょう.しかし,このファイルを本チュートリアルで直接に使うのではなく,このチュートリアル 実行の途上で必要に応じて参照してください.

# 3) 回路仕様

#### 3.1) メモリ容量

このヒストグラムは, ADC から受け取った 12 ビット長の異なる各値が何回出現したかをグラフで表現するものです.サンプルした値をヒストグラムとして出力するにあたり, 12 ビットの各値(0~2<sup>12</sup>-1)をもれなくカ バーするのに 4K の SRAM(2<sup>12</sup>=4096)が必要です.また, SRAMのワード幅は対象とするデータの出現頻度に依存しますが,最悪の場合,一つの値に 64K 個のデータが集中することを想定し,64K=2<sup>16</sup>により,8 ビット幅の 4K SRAMを2個使うことで(8 ビット×2=16 ビット),どのような場合でもヒストグラムはメモリ上に実現されます.

### 3.2) 回路の動作

回路が起動したとき, SRAM は初期値として各アドレスにおいてゼロ・データを保持するものとします. ADC からのサンプル・データは SRAM のアドレスとして使われ, そのアドレスに対するメモリ・セルの保持する カウント値をインクリメントすることによりヒストグラムを形成します.この動作を 64K 個のデータを受け取るまで 繰り返します.

### 3.3) 試験システムの構成

システムを、

## ディジタル・システム=データ・パス+コントロール

の考えに基づいて分析し,次の仕様(図)を得ます.

\* 図の右半分は「データ・パス部」で機能ブロック間で処理されるデータの流れを示しています.

\* 左半分は「コントロール部」を表し,ステート・マシンとなっています.状態信号を「データ・パス部」の一つであるカウンタより得て,制御信号を「データ・パス部」に与えています.

#### 3.4) コントロール部の構成

回路を制御するために,ワンホット・ステート・マシンを使います.ワンホット・ステート・マシンは,各状態に一つのフリップフロップを割り付けます. したがって現在の状態を表すフリップフロップだけが1を保持し,他のステート・フリップフロップは0になります(名前 one-hot はここに由来する). ワンホット・ステート・マシンは FPGA のアーキテクチャによくマッチするので,FPGA ベースの設計ではとてもポピュラーなもので,伝統的なバイナリ・エンコード・ステート・マシンにくらべて比較的高速に動作し,少ないリソースで実現できます.



### 3.5) システムの動作

ステート・マシン(SM)は最初 IDLE ステートに初期化されます.START が High になった後,クロックの立ち下 がりエッジで SM は READ ステートに遷移し, ADC から出力された SRAM アドレス値に対するカウントをインク リメントしたのち,トランスペアレント・ラッチ DBUS\_INC に保持します.次のクロックで,SM は WRITE ステートに 遷移し,トランスペアレント・ラッチ DBUS\_INC に保持された更新値を SRAM に書き戻します.

このステート・マシンはREAD とWRITE ステート間を,決められた回数(バイナリ・カウンタCOUNT のサイズに よって決められる)だけ,交互に繰り返します.そして既定値に到達すると,SM はIDLE ステートに戻ります.

## 4) コンポーネントのモデリングとシミュレーション

4.1) 論理式インターフェースを使ったステート・マシンのモデリング

## ヒストグラムの回路をモデリングするための新しいダイヤグラムの作成

-[File]-[New]メニュを選択し,新しいダイヤグラムを作ります.

-このチュートリアルでは使わない Parameter ウィンドウを最小化しておきます.

- [Window] - [Tile Horizontally]を選択し,ダイヤグラムと Report ウィンドウを見やすくします.

## システム・クロックのモデリング

-[Add Clock]ボタンをクリックして, CLK0 という名前で信号を追加します.

-[OK]をクリックして,デフォルトの設定でCLK0を登録します.

WaveFormer Pro - untitled.tim	×
<u>F</u> ile Export Edit Draw Bus Libraries <u>V</u> iew Options <u>R</u> eport <u>W</u> indow <u>H</u> elp In	
Add Grant Add Due Dates Cates Sente	
Add Signal Add Sus Feedy Setup Sample HIGH LOW TRI VAL INVal WHI WLO HEX Add Clock Add Spacer Hold Text Marker	Inactive Zoom Out
71.68ns 71.68ns Ons 50ns 100ns 150ns	200ns   250ns   300ns
	Signal Properties 🔀
	Name: CLK0 Properties
	active low name (adds bar on top, \$BAR suffix)
	Boolean Equation: ex. (SIG1 and SIG2) delay 5
	Clock: Unclocked Edge/Level: Neg
Started Perl script caching	Clock Properties
	Startup State: Unknown Hold: 0
	🔶 Boolean Equation <> HDL Code
	Simulate Once 📃 🗔 Continuousty Simulate
	W/m Eqn 8ns=Z (5=1 5=0)*5 9=H 9=L 5=V 5=X 💌
	Export Signal Direction: shared output
	VHDL Type: std_logic
INS Row: 1 Line: 1 Col: 1	Verilog Type: wire
	Badix: bin Bus MSB: 0 LSB: 0
waveperlerr /	
Parameter 🖪 🗆 🔀	UK Cancel Apply Prev Sig Next Sig

図2 システム・クロック信号の作成

START とADDR 信号のモデリング

本システムは,以下の2つの入力信号を持っています.

(1)処理を始める START

(2)ADC の出力データで, SRAM のアドレス・ラインに供給される ADDR

次の 2 つのセクションでは,この入力信号の波形データの作成方法について説明します.START 信号の波 形は比較的シンプルなのでマウス操作だけで作成できます.ADDR 信号の波形はより複雑で,人手で入力す るのはやや大変です.そこで,テンポラル式やステート・ラベル式を使ってこの信号を作成します. START 信号のモデル化 - グラフィカルに入力信号を定義する

-[Add Signal]ボタンをクリックします.

-作成された SIGO をダブル・クリックして, Signal Properties Dialog を表示させます.このダイアログを使って, デフォルトの設定を変更します. Name の項目に START と入力します.

-[OK] ボタンをクリックして,現在の値を設定します.

## Tips:

START 信号の設定を変更したときに現れた Signal Properties Dialog ウィンドウは, このチュートリアルを行っている最中,開いたままにしておくことができます([Apply]ボタンをクリックして結果を更新できる). 他の信号の設定を変更したいときには,ダイヤグラム・ウィンドウの信号名をダブルクリックします.

-START 信号として 60[ns]程度の Low 信号をはじめに描きます . Low 信号を描くには , ダイヤグラム・ウィンド ウ内で[Low]ボタンをクリックして , START の名前のある項目の右 60[ns]のあたりを左クリックします .

-100[ns]程度の High 信号を START に継ぎ足します. [HIGH]が押されていることを確認して, ダイヤグラム・ ウィンドウ内で, START の名前のある項目の右 160[ns]のあたり(はじめの Low 信号の終わりから 100[ns])を 左クリックします.

-CLK0 の最初のネガティブ・エッジのときに START 信号が Low, 次のネガティブ・エッジのときに High である かを確認してください. そのようになっていれば, ステート・マシンは正常に動作します. もしそうでない場合には, 入力波形をチェックし, 波形を描き直します.

- 最後に 800[ns]程度の Low 信号を継ぎ足します.

🙅 TestBencher Pro - tutor	6.tim					
<u>F</u> ile E <u>x</u> port <u>E</u> dit <u>D</u> rav	v <u>B</u> us <u>L</u> ibraries <u>V</u> iew	Options <u>R</u> epo	rt <u>W</u> indow <u>H</u>	<u>t</u> elp		
🙋 Diagram	C					_ <b>_</b> ×
Add Signal Add Bus Add Clock Add Spacer	Delay Setup Sample Hold Text Marker –	GH LOW TRI Y	AL INVal WH	WLO HEX	Simulation Zoom In Inactive Zoom Out	
19.46ns 19.46ns C	)ns [50]ns	100ns		Ins	Signal Properties	
ССКО Г	N		<u>\</u>		<u> </u>	
START			Ŋ		Name: START	Properties
	0		U		active low name (adds bar	on top, \$BAR suffix)
					Boolean Equation: ex. (SIG1 and	ISIG2) delay 5
					J	
	-1-1				Clock: Unclocked 🔽 Edg	e/Level: neg 🔹
					Clock to Out: 0	Setup: 0
Started Perl script cach	ing				Startup State: unknown	Hold: 0
					🔶 Boolean Equation	HDL Code
Simulate Once				Simulate Once	Continuously Simulate	
					Wfm Eqn 8ns=Z (5=1 5=0)*5	9=H 9=L 5=V 5=X 🗸
					Export Signal Direction:	shared output 💌
					VHDL Type: std_logic	
INS		Row: 1	Line: 1 C	ol: 1	Verilog Type: wire	
tuto	ró.v		veriwell.log		Radix: 🗾 🗾 Bus MS	B: 0 LSB: 0
Parameter 🗗 🗖	×				OK Cancel Apply	Prev Sig Next Sig

図3 START 信号の作成

波形とラベル式を使った入力信号の生成 - A/D コンパータ・データのモデリング

SRAM のアドレス・ラインを駆動する信号 ADDR (ADC からの出力)を仮想バス信号として扱えば十分なので, A/D コンバータを単なるデータ・ソースとみなしてモデリングします.

ADDR 信号の生成

-[Add Signal]ボタンをクリックし, ADDR 信号を作ります.ここで, [Add Bus]ボタンをクリックしないでください. 仮想バス信号で十分です, バスの要素信号は必要ありません.

- 信号名をダブル・クリックして, Signal Properties Dialog ウィンドウを開きます.

-信号の MSB を 11 にセットして, Radix を hex に設定します.

ADC はステート・マシンのクロック CLK0 の半分の周波数で動作します. それにより, ADDR の値は, 一つおきのクロック・サイクルで変化します. 以下の波形方程式を ADDR 信号の内容として使います.

170=X(200=V)\*20

上の式は 170[ns]間不定, つぎに 200[ns]間有効値を 20 回繰り返すことを意味します.

-ADDR の Signal Properties Dialog ウィンドウを開きます.

- [Wfm Eqn]ボタンの右隣の項目に上記の波形方程式を入力します.

- [Wfm Eqn]をクリックすると,波形方程式に対応する波形が自動生成されます.

-[OK]をクリックして,現在の値を保存します.

次に , ラベル式を使って , ADDR バスの各有効値に値(ラベル)を割り当てます . 次の式が , ADDR 信号に適用 されるステート・ラベル方程式です .

ADDR:1 Rep((0,1,2,3,4,),4)

上の式は 0 から 4 までの 16 進数をカウントし, それを 4 回繰り返すことを意味します.":1"は, はじめの不定ステートの後, ラベル付けを開始することを意味します.

-[Export]-[Label Waveform Equation]を選択します.

- ステート・ラベル方程式を入力します.

-[OK]をクリックして,現在の値を保存します.

🙅 TestBencher Pro - tutorô.tim	
<u>File Export</u> Edit <u>Draw</u> Bus Libraries <u>View</u> <u>Options</u> <u>Report</u> <u>Window</u> <u>Help</u>	
Export Signals As Add Generate Test Bench Edit Object Properties Big High Low TRI VAL INVa WHI WLD HEX	Signal Properties
Add         Exclude Selected From Export         500ns         1.0us         1           Show Non-Exported Signals         500ns         1.0us         1	Name: ADDR Properties active low name (adds bar on top, \$BAR suffix)
CLk Add/Execute Script Draw Waveform Equation ST/ Label Waveform Equation	Boolean Equation: ex. (SIG1 and SIG2) delay 5
ADDR[11:0] (0 (1) (2) (3) (4) (0) (1	Clock: Unclocked  Edge/Level: neg
	Clock to Out: 0 Setup: 0
Penart - C#TBenakitutar6u	Startup State: unknown Hold: 0
Timescale 1ns / 1ns	📀 Boolean Equation 🔿 HDL Code
	Simulate Once 📃 Continuously Simulate
module top;	
wire CLKD:	Wfm Eqn 170=X(200=V)*20
wire START:	Export Signal Direction: shared output
Enter State Equation	
Enter state labeling equation, Example: SIG0 Hex(Inc(0,2,5))	
ADDR:1 Rep((0,1,2,3,4),4)	Verilog Type: wire
wavepe	Radix: hex 💽 Bus MSB: 11 LSB: 0
	OK Cancel Apply Prev Sig Next Sig

図4 ADDR 信号の作成

## フリップフロップの式を使ったステート・マシンのモデリングとシミュレーション

ステート・マシンは WaveFormer Pro 上では, Signal Properties Dialog の Boolean Equation 項目使ってモデリングできます. IDLE, READ, WRITE の3つの信号を作成します.

作成した3つの信号各々について Signal Properties Dialog で以下のステップを実行します.

- ステート・マシンの式を信号に対して Boolean Equation の項目に入力します.

IDLE	(WRITE & DONE) ¦ (	START	& IDLE)
READ	(IDLE & START) ¦ (\	NRITE &	DONE)
WRITE	READ		

-3 つの信号それぞれについて, CLK0 をダイアログ内の Clock のドロップダウン・リスト・ボックスから選択します. これで, フリップフロップがCLK0 の立ち下がりエッジに同期して動作するようになります(Edge/Level のデフォルト値は neg).

- 各フリップフロップのパワー・アップ時の値(デフォルト"Unknown")を編集します.まず, POWER 信号を追加し(RESET 信号に相当), 最初の 80[ns]間を Low とし, その後 2[us]までを High とします. つぎに, 信号 IDLE の Signal Properties Dialog を開き,以下のように設定します.

Signal Properties				
Name: IDLE	<u>Analog</u> Props			
🗖 Active Low Name (\$B	BAR suffix) <u>G</u> rid Lines		チェックをつける	
Boolean Equation: ex. (SIG	G1 and SIG2) delay 5	/		
(WRITE & DONE)   (~STA	RT & IDLE) 📃			
Clock: CLK0	Edge/Level: neg 💌		Set <b>制御信号として</b>	
Set POWER -	Clear: Not Used 💽	(	POWER を選択する	
Clock EI CLK0 START	Advanced Register			
ADDR IDLE Sid BEAD	n C HDL Code			
Wfm gn Bns=7 (b)				
図5 IDLE 信号のパワ	ー・アップ時の値の設定	例		

これで, POWER 信号がLow の間, IDLE 信号はHigh(1)にセットされます(IDLE 1,b1). なお, WaveFormer Pro では,指定された信号の値を元に,ロー・アクティブ/非同期でレジスタの値を High にセット,またはLow にリセットするよう初期設定されています.また,この初期設定は変更が可能です.



図6 レジスタ/ラッチのセット/リセット動作属性の設定

同様に, READ 1'b0, WRITE 1'b0 を設定します.これは以下に示す図にしたがって操作すれば設定できます.なお,どちらの信号についても Simulated Signal の項目にチェックをつけます.

Signal Properties	X Signal Properties	×
Name: WRITE Analog Props	s Name: READ <u>Analog Prop</u>	\$
Active Low Name (\$BAR suffix)	Active Low Name (\$BAR suffix)	
Boolean Equation: ex. (SIG1 and SIG2) delay 5	Boolean Equation: ex. (SIG1 and SIG2) delay 5	_
READ	(IDLE & START) I (WRITE & ~DONE)	-
Clock: CLK0  Clock: CLK0 Clcck: CLK0 Clcck	Clock: CLK0     CLK0     Edge/Level: neg	-
Set: Not Used 🔽 Clear: Not Used	Set: Not Used Clear: POWER	-
Clock Enable: Not Used  A CLK0 START	Clock Enable: Not Used  A CLK0 START	
Boolean Equation C ADDR	Boolean Equation C ADDR	
Simulate Once	Simulate Once	
₩fm Eqn 8ns=Z (5=1 5=0)*5 9	Wfm Eqn 8ns=Z (5=1 5=0)*5 9.4 9=L 5=V 5=X	

図7 WRITE 信号とREAD 信号のパワー・アップ時の値の設定例(0クリア動作の設定)

-以上の設定を行っても , シミュレーション結果が波形として現れません . これは , まだ定義していないDONE 信号を IDLE とREAD の式が参照しているためです .

💁 VeriLogger Pro – [Diagra	m - Tutsim.tim]	
🕂 <u>F</u> ile Export <u>E</u> dit <u>B</u> us	Libraries Project Editor Simulate Report View Option	ns <u>W</u> indow <u>H</u> elp <u>_ 문 ×</u>
📗 Auto Run 🔃 🕨 🕪	🚥 🕅 🔍 🛛 S 💈 🕸	<b>~</b>
Add Signal Add Bus Delay Add Clock Add Spacer Hold	y Setup Sample HIGH LOW TRI VAL INVal WHI WLO	HEX Zoom In Zoom Full Simulation
759.7ns = -2.188us	Ons  500ns  1.0us  1.5u	Signal Properties
CLKO		Name: READ Analog Props
START		☐ Acti <u>v</u> e Low Name (\$BAR suffix) <u>G</u> rid Lines
ADDR[11:0]		Boolean Equation: ex. (SIG1 and SIG2) delay 5
IDLE		(IDLE & START) I (WRITE & ~DONE)
READ		Clock: CLK0   Edge/Level: neg
WRITE		Set Not Used  Clear POWER
POWER		
		Clock Enable: Not Used  Advanced Register
		Boolean Equation     O HDL Code
		Simulate Once Simulated Signal
		<u>₩</u> fm Eqn 8ns=Z (5=1 5=0)*5 9=H 9=L 5=V 5=X 💌
		Label Eqn Hex(Inc(0,2,5))
		Export Signal Direction: shared output
		VHDL: std_logic  Verilog: wire
		Radix: bin 💌 Bus MSB: 0 LSB: 0
• •	<b>1</b> -1	OK Cancel Apply Prev Next

図8 IDLE, READ, WRITE の作成とシミュレーション結果

## シミュレーション結果の検証- WaveperLlog のレポートを見る

Waveperl.log ファイルには,作成されたフリップフロップを Verilog-HDL コードに変換する Perl スクリプトにより 生成されたステータス・レポートが記述されています.このファイルには,シンタックス・エラーや,設計式中の不 明信号の情報について記述されています.

-Report ウィンドウの下にある[Waveperl.log]ボタンをクリックすると,このファイルを見ることができます.IDLE または READ 信号について, Signal Properties Dialog 内の Simulate Once ボタンをクリックしてシミュレーション を行うと,このファイル中に"Unknown signal name DONE"という記述が現れます.

-IDLE, READ, WRITE の各信号については, Simulated Signal の項目がチェックされているので, ダイヤグラム 中の任意の信号を変更するたびにシミュレーションが行われます.これで DONE 信号を定義すると, その結果 をすぐにダイヤグラム・ウィンドウで見ることができます.



図9 Waveperl.log ファイルの内容

### シミュレーション結果の検証- シミュレーション・ログ・ファイルを見る

Report ウィンドウ内のシミュレーション・ログ・ファイル(verilog.log) 中にも, DONE 信号が定義されていないと いうレポート・メッセージを見ることができます.また, このログ・ファイルは WaveFormer が自動生成した Verilog-HDL ソースの対応するエラー行もレポートします. Verilog-HDL のソース・コード・ファイル名はダイヤグ ラムと似ており, 拡張子が.tim から.v に変わります(ダイヤグラム名が untitled.tim のとき, Verilog-HDL のソー ス・コード・ファイル名は untitled tim.v となる).

これらのファイルは, WaveFormer がソース・ファイルを生成するたびに, 自動的に Report ウィンドウに表示されます(デフォルトでは, 設計を変更してシミュレーションを行うたびに自動生成される).

以下の方法でエラーの起きた Verilog-HDL ソースの行を参照できます.

-verilog.log ファイルでエラーが起きた箇所を確認します.以下はその例です.

🚍 Report – verilo	e.log	<u> </u>
WaveFormer Pro si	imulation log created at Fri Mar 05 14:32:47 19	399 🔟
Beginning Compile	-	_
Beginning Phase I		
Compiling source fi	le as library: C:\WAVEFORM\wavelib.v	
Finished Phase I		
Entering Phase II		
Compiling auto-gen	ierated top level module file: C:\WaveForm\T61	fim.v
C:\WaveForm\UNTI	ITLEDTim.v: L91: error: 'DONE' not declared	
C:\WaveForm\UNTI	ITLEDTim.v: L94: error: 'DONE' not declared	
INS	Row: 7 Line: 7 Col: 68	-
		•
(verilog.log) wavep	erllog / untitledTim.v /	

図10 verilog.log ファイルの内容

図10に示すように, C¥untitledtim.v: L91: error: 'DONE' not declared, と表示されています.これからエラーは, 91行目で起きていることがわかります(実際のエラー行番号は本チュートリアルと異なる場合がある).

-Report ウィンドウの下にある, [\* tim.v]ボタン(\*は現在のタイミング・ダイヤグラム名に対応)をクリックし, Verilog-HDL のソース・ファイルを見ます.

-<ContIol>-<Shift>-L を押して, Jump to Line Number ウィンドウを表示させます.そして, 91 を入力します. - すると, IDLE と READ をシミュレーションしている Verilog-HDL コードの行が見えます.

Eile       Export       Edit       Bus       Libraries       Report       View       Options       Window       Help         #200.000000       // Abs       Time:       3770.000000       Jump To         ADDR = 12'h3;       Jump To	<u>_ 8 _</u> _
#200.000000 // Abs Time: 3770.000000 ADDR = 12h3; Jump To	<b>^</b>
#200.000000         // Abs Time: 3970.000000           ADDR = 12'h4;         Jump to Line Number: 91           #200.000000         // Abs Time: 4170.000000            OK	
end wire IDLE_wf0 = (WRITE & DONE )  (~START & IDLE ); registerN_Asyn #(1,0,0,1,1) registerN_Asyn_IDLE(IDLE,CLK0,IDLE_wf0,1'b0,POWER,1'b1); wire READ_wf0 = (IDLE & START )  (WRITE & ~DONE ); registerN_Asyn #(1,0,0,1,1) registerN_Asyn_READ(READ,CLK0,READ_wf0,1'b0,1'b1,POWER);	_
wire WRITE_wfD = READ;	
INS  Row: 9 Line: 91 Col: 38	ك 

NOTE:

ソース・コードを編集しないでください.ソース・コードを編集しても,次回のシミュレーション時に,自動的にソ ースが上書きされてしまい,編集結果は反映されません.設計を変更する場合は,直接ソース・コードを変更す るのではなく, Signal Properties Dialog ウィンドウを使ってください. WaveFormer は,正しい Verilog-HDL ファイ ルを生成し,シミュレーションを行います.

## 出力信号をモデリングしたシミュレーション(インクリメンタルな設計)

ディジタル・システムのシミュレーションやデバッグを行うにあたり,一方のコンポーネントの出力信号が他方のコンポーネントの入力信号となっているので,システムの大部分を設計し終わらないとテストが始められない, という問題が起こります.

しかし,システムを細分化して,その各々の部分に対して,他の部分の出力をテスト・ベクタとして与えれば, 小規模化した分だけ容易にテストを開始できます.しかし,いずれにしろテスト・ベクタの生成に時間を要します が,WaveFormerは設計の小さい部分のテストに対して,とてもシンプルで素早い方法を提供します.

まだ設計情報のない部分に対しては、代りにグラフィカルに信号を描いて、これを他の部分に対する入力信号としてテストを行い、後でこの信号を生成するような設計情報を付加すればよいのです.言い換えると、一時的にある部分の期待される出力を、WaveFormer上で波形を描いてエミュレーションすることになるのです.

以下では,この方法を使って,いままでに作成したステート・マシンの動作を,DONE 信号を生成する論理式 を入力する前に確認します.

-DONE 信号を追加します.

-[Low]ボタンをクリックし, 1.6[us]長の Low 波形を描きます. この後, 1クロック・サイクル以上続く High 信号を描きます(DONE 信号のエミュレーション).

-生成されたダイヤグラムと図1 と比較して,作成したステート・マシンが正しいことを確認してください.

-もし,相違点があれば,まず verilog.log ファイルを調べ,作成したダイヤグラムのコンパイルが成功している かどうかを確認します.エラーがあれば,ダイヤグラムのエラー箇所がエラー・メッセージと一緒に表示されます. また,IDLE,READ,WRITE の各信号の Signal Properties Dialog ウィンドウで, Simulated Signal ラジオ・ボタンに チェックがついているかを確認してください。

-シミュレーションはできるが出力が誤っている場合は,設計式,START,DONE およびPOWER 信号を確認してください.



図12 DONE 信号のモデリング(エミュレーション)とステート・マシンのシミュレーション結果

なお,たとえばSTART 信号の High の期間が非常に短い場合,ステート・マシンがどのような動作をするかを 確認する場合,単にSTART 信号のエッジ位置を変更するだけで即 WaveFormer が変化した START 信号を基 に再シミュレーションを行い,結果をダイヤグラム・ウィンドウに表示してくれます.

126.4ns <b>-2.510us</b>	Ons  200ns  400ns  600ns  800ns  1.0us  1.2us  1.4us  1.6us
CLKO	
START	
ADDR[11:0]	
IDLE	
READ	
WRITE	
POWER	
DONE	

図13 START 信号の変化によるシミュレーション結果

図13 に示すように START 信号の High の期間が非常に短い場合(CLK0 の立ち下りエッジで Low)の場合, 図1 に示したステート・マシンの定義どおり IDLE 状態にとどまることがすぐに確認できます.

確認が終わったら START 信号を最初の状態に戻しておいてください.

### 組み合わせ論理のモデリング

IDLE, READ, WRITEの状態信号に加え, ステート・マシンはENABLE信号を持ち, SRAM, カウンタ, A/D コンバータをイネーブルにします.

以下ではこの ENABLE 信号をモデリングしてみます.

-新たに ENABLE 信号を加えます.

-ENABLE 信号に対して,以下の式を Boolean Equation の項目に入力します.また, Simulated Signal の項目 を忘れずにチェックしてください.

READ | WRITE

上の式は, READ とWRITE の論理和演算を行うことをあらわします.

なお , 入力を完了すると同時に , ENABLE 信号の値がシミュレーションされてダイヤグラム・ウィンドウに表示 されます . 4.2)Verilog-HDL インターフェースを使ったモデリング

カウンタ回路のモデリング

カウンタの出力 COUNT は Verilog-HDL コードを使ってモデリングします.はじめに,4 ビット・カウンタを動作 確認のために作成し,その後にこれを17 ビットに拡張します.

-COUNT 信号を入力します.

- Signal Properties Dialog で,信号の MSB を3に, Radix を hex に設定し, Simulated Sigal の項目をチェックします.

-HDL code の項目をチェックし, Equation ビューから, HDL Code ビュー/エディタに切り替え, 以下に示すコードを入力します(コメント文は//ではじめ, その行の//以降は実行時にスキップされる).

reg [3:0] COUNTER;	
always @(negedge CLK0)	//on each falling edge of CLK0
begin	
if (ENABLE) COUNTER = COUNTER + 1	1; // count while ENABLE is high
else COUNTER = 0;	<pre>// synchronous reset if ENABLE is low</pre>
end	
assign COUNT = COUNTER;	//drive wire COUNT with reg COUNTER value

Verilog-HDL Note:

WaveFormer 内のすべての信号は wire としてコーディングされます. COUNTER の値で COUNT (wire)を駆動 させるのに, Verilog-HDL のコーディング・ブロックの最後で assign 文が必要です.

64K のデータを扱えるように, COUNT の MSB を 16 に変更して, HDL コード上のカウンタの宣言を reg [16:0] COUNTER

とすることもできますが,本チュートリアルでは行いません.

💁 WaveFormer Pro							- O ×
<u>File Export Edit Bus</u>	<u>L</u> ibraries <u>R</u> eport	<u>V</u> iew <u>O</u> ptions <u>W</u> ine	dow <u>H</u> elp				
🗗 Diagram - T6.tim							<u> </u>
Add Signal Add Bus Add Clock Add Spacer	Delay Setup Sample Hold Text Marker	HIĞH LOW TRI V		WLO HEX	Zoom In Zoom Full Zoom Out Zoom Range	Simulation Good	, 
1.954us 1.804us	Ons  200ns	400ns  600ns	800ns	1.0us  1.2	2us  1.4us  1.6us	s  1.8us	2.0us
CLKD	$\sim \sim $				$\mathcal{M}$	$\sim$	
START							
ADDR[11:0]		<u>( 1 ) 2</u>	) 3 )	<u>4 )(</u>	0 ( 1 )( 2	)(3	<u> </u>
IDLE							
READ							
WRITE					orginal Properties		
POWER					Name: COUNT		Analog Props
DONE					Active Low Name (	\$BAR suffix)	<u>G</u> rid Lines
ENABLE					Boolean Equation: ex. (S	IG1 and SIG2]	) delay 5
COUNT[3:0]	bx 0 1	2 3 4 5	<u> </u>	<u>(9 ( A )</u>	reg[3:0] COUNTER		-
<u> </u>	×				always @(negedge	CLKO)	
Finding handle to synca Finding handle to synca	d_top.IDLE d_top.READ				if(ENABLE) COU	NTER = CO	UNTER +
Finding handle to synca	d_top.WRITE				else COUNTER =	= U;	ت_
Finding handle to synca Finding handle to synca	d_top.COUNT				C Boolean Equa	tion 🕢 HD	⊥ LCode
Compile Complete					Circulate Orace	l ⊑ ciuda	te d Ciencel
Running					Simulate Unce		ited Signal
0 Errors, 0 Warnings	Lood time $= 0.110$	00 Execution time :	- 0 0000		<u>₩</u> fm Eqn 8ns=Z (5=1	5=0)*5 9=H 9	=L 5=V 5=X 💌
Complie time - 0.10000	, Load time – 0. m	ioo, Execution time -	- 0.22000		Label Egn Hex(In	c(0,2,5))	*
Normal exit							
INS		Row 14	Line: 24 Col:	1	✓ Export Signal D	irection: share	ed output
1		, itom in	2.110.21 001.		VHDL: std_logic 💽	Verilog: wi	re 🔹
verilog.log / waveperl.log	s / T6Tim.v /				Radix: hex 💌 B	us MSB: 3	LSB: 0
Parameter 🗗 🗖	×				OK Cancel <u>4</u>	Apply En	ev <u>N</u> ext
Left click & drag to zoom in: Left click &	anag with (CTFL) key to block o	apy. Flight click to zoom out.					

図14 4 ビット・カウンタの実装とシミュレーション結果

既存のVerilog-HDL モデルをWaveFormer に加える(SRAM のモデリング)

ここでは, すでに Verilog-HDL で記述された SRAM モジュール(sram.v)を Wave Former に取り込んでみます. このモデルは非同期のインターフェースを正確にモデリングしているほか, SRAM 起動時のメモリ・セルのゼロ・クリアを行う記述も含んでいます(実際の回路では,アドレス・バスを通して繰り返しゼロ・データを書き込む回路が別に必要になる).

Verilog-HDL を使った完全な SRAM のモデリングはこのチュートリアルの範囲外ですが, [Report]-[Open Report Tab] メニュを使って, おおまかに sram.v を見てください.

🚍 Report – Sram. <del>v</del>	_ 🗆 🗵
module sram(CSB,WRB,ABUS,DATABUS	S);
input CSB; input WRB; input [11:0] ABUS; inout [7:0] DATABUS;	// active low chip select // active low write control // 12-bit address bus // 8-bit data bus
reg [7:0] DATABUS_driver; wire [7:0] DATABUS = DATABUS_driver; reg [7:0] ram[0:4095];	// memory cells
integer i; initial //initialize all RAM cells to begin	o O at startup
INS Row: 10 Line: 10	Col: 69
verilog.log / waveperl.log / T6Tim.v ; Sram.v	<u>ے</u>

図 15 SRAM モジュールの Verilog-HDL 記述(sram.v)

次にこの Verilog-HDL ソースを WaveFormer に取り込みます.これは, WaveFormer が使う wavelib.v に include 文を使って sram.v ファイルを取り込みます.

- [Report] - [Open Report Tab...]メニューを使って wavelib.v ファイルを開きます.

-以下の一行を wavelib.v ファイルの先頭に追加します(すでに追加されている場合がある).

`include "sram.v";(include の前にダッシュをつける)

Report - wavelib.v*		
`include "sram.v"		
//*************************************		
// Register, Negative Edge Triggered, Asynchronous Set and Clear // (first use v5.0)		
module registerN_Asyn(out,clock,in,clockEnable,set,clear);		
parameter n = 1; parameter clock2outMin = 0; parameter clock2outMax = 0; parameter ce_activeLow = 1; //active low by default parameter sc_activeLow = 1; //active low by default		
i output [n-1:0] out; input clock; input [n-1:0] in; input clockEnable; //active low by default		
INS Row: 14 Line: 14 Col: 3		
verilog.log / waveperl.log / T6Tim.v / wavelib.v /		

図16 SRAM モジュール(sram.v)の追加

## インクリメンタ , ラッチのモデリング

以前にネガティブ・エッジ・トリガ型のレジスタ(フリップフロップ)を生成する論理式を使ってステート・マシンをモ デリングしましたが,ここではレベル・トリガ型のレジスタ(ラッチ)を同様の方法でモデリングしてみます. SRAMの値が読み出されるとこれを DBUS に送り,インクリメンタで1を加算され,ラッチで保持されます.

-DBUS\_INC という名前で新しい信号を作ります. -以下の論理式を Boolean Equation の項目に入力します.

DBUS+1

-Clock の項目から READ を選び, Edge/Level の項目から High を選びます(High レベル・ラッチの実現). -MSB を 15 に, Radix を hex に, そして Simulated Signal をチェックします.

なお,この時点ではDBUSが定義されていないのでシミュレーション結果はあらわれません.

Signal Properties		
Name: DBUS_INC Analog Props		
Active Low Name (\$BAR suffix)		
Boolean Equation: ex. (SIG1 and SIG2) delay 5 DBUS+1		
Clock: READ		
Set: Not Used 🔹 Clear: Not Used 💌		
Clock Enable: Not Used 💌 Advanced Register		
Boolean Equation C HDL Code		
Simulate Once		
<u>₩</u> fm Eqn 8ns=Z (5=1 5=0)*5 9=H 9=L 5=V 5=X ▼		
Label Eqn Hex(Inc(0,2,5))		
Export Signal Direction: shared output		
VHDL: std_logic  Verilog: wire		
Radix: hex 💌 Bus MSB: 15 LSB: 0		
OK Cancel <u>Apply</u> <u>Prev</u> <u>N</u> ext		

図17 インクリメンタ, ラッチのモデリング

条件演算子を使った3ステート・ゲートのモデリングと,SRAM モデルの実装

-DBUS という名前で新しい信号を作ります.

-MSB を 15, Radix を hex に, そして Simulated Signal 項目をチェックします.

-次の Verilog-HDL コードを DBUS の HDL コード・ウィンドウに入力します.

wire CSB = !ENABLE; sram BinMem1(CSB,READ,ADDR,DBUS[7:0]); sram BinMem2(CSB,READ,ADDR,DBUS[15:8]); assign DBUS = WRITE ? DBUS\_INC : ' bz;

最初の行では ENAB LE 信号を反転した信号を生成します(SRAM はアクティブ・ロー). 次の 2 つの行は, 8 ビット×4K の SRAM を実装したもので,入出力信号のマッピングを行っています. (最初の SRAM は DBUS の下位バイト,次の SRAM は上位バイトに接続されている)

最後の行は, DBUS を3ステート・ゲートとしてモデリングしています.ここで If-then-else(If 条件 then a else b)の働きをする条件演算子(条件? a: b)を使っています. WRITE が High のときは DBUS は DBUS\_INC によって駆動され, Low のときは DBUS は他と切り離されます(bz はすべてのビットがハイ・インピーダンスになるこ

とをあらわす).

以上の作業を行ったらシミュレーションを行ってみます.以下に示すようなダイヤグラムが得られます.



図18 DBUS およびDBUS\_INC をモデリングした後のシミュレーション結果

**リダクション・オペレータを使った**n ビット・ゲートのモデリングによるDONE 信号のモデリング 最後にステート・マシンに対する入力信号としてエミュレーションした DONE 信号をモデリングします. DONE 信号は COUNT 信号の各ビットの論理積を実行させることで生成されます.すでに作成ずみの DONE 信号をダブル・クリックして, Simulated Signal にチェックをつけた後, 次の論理式を入力します.

&COUNT

&オペレータはリダクション・オペレータと呼ばれ,入力信号の各ビット同士の論理積(AND)を演算することを 意味します.これは次の式と同等です.

COUNT[0] & COUNT[1] & COUNT[2] & ...

リダクション・オペレータの利点は,ビット長を自動的に計算して上記のような論理式を自動生成するので,入 力の手間が格段に省けることにあります.

453.9ns 303.6ns	0ns   200ns   400ns   600ns   800ns   1.0us   1.2us   1.4us   1.6us   1.8us		
CLKO			
START			
ADDR[11:0]			
IDLE			
READ			
WRITE			
POWER			
DONE			
ENABLE			
COUNT[3:0]	bx(0)(1)(2)(3)(4)(5)(6)(7)(8)(9)(A)(B)(C)(D)(E)(F)(		
DBUS_INC[15:0]	("bx)(1))((1))((1))((1))(2))((2))		
DBUS[15:0]	("bz )),0 ),1 )),0 ),1 )),0 ),1 )),0 ),1 )),0 ),1 )),0 ),1 ),1 ),2 )),1 ),2 )),1 ),2 )),1 ),2 ))		

図19 DONE 信号をモデリングした後のシミュレーション結果

\$display,\$monitor コマンドを使った外部Verilog-HDL モデルのデバッグ

Verillog-HDL はシステム・タスク, \$display,\$monitor を装備しています. これらは Verilog-HDL ソースのデバッ グを目的に用意されています.

\$display は C 言語の printf 文に相当し, \$display が実行されると, シミュレーション・ログ・ファイルの verilog.log に処理結果が出力されます.

\$monitor は,パラメータに指定された信号が変化したときに,ログ・ファイルにレポートを出力します.

SRAM モデル・ファイル sram.v は2つの\$display 文を含み, SRAM に対してデータの読み/書きが行われるときにレポートを出力します.



図20 sram.v に記述された\$display システム・タスクのレポート例

以上でヒストグラム回路のモデリングが完了しました. 図 19 に示すような波形が描けているでしょうか. 自分の作成したダイヤグラムと違う部分を見つけたら, verilog.log を使いながら間違いを訂正していきます.

SRAM から不確定なデータを読み込んだことを示す x が DBUS 上に現れているときは , \$display コマンドの出 力を使ってチェックを行うのが有効です . この場合 , 不確定なアドレスに対して SRAM に書き込みを行っていな いかを確認してください .

End Diagram Marker を使ったシミュレーション時間の制御

デフォルトでは, WaveFormer は信号があるところまでシミュレーションを行いますが, タイム・マーカによって シミュレーション時間を制御することもできます.

-[Marker]ボタンをクリックし,ダイヤグラム・ウィンドウ内の任意の位置でシミュレーションを終えたい時刻付近で右クリックしてマーカをつけます.

- マーカを左クリックで選択すると緑色の表示となるので,これをダブル・クリックして Edit Time Marker ダイア ログ・ウィンドウを開きます.

-marker type を"End Diagram"に設定します.

-ダイアログを閉じマーカを動かすと,自動的にマーカーの置かれた時刻まで再シミュレーションを行います.

💁 WaveFormer Pro - [	Diagram - T6.tim]	
🕂 <u>F</u> ile Export <u>E</u> dit	<u>B</u> us <u>L</u> ibraries <u>R</u> eport <u>V</u> iew <u>O</u> ptions <u>W</u> indow <u>H</u> elp	<u>_8×</u>
Add Signal Add Bus Add Clock Add Spacer	Delay Setup Sample High LOW TRI VAL INVal WH WLD HEX Zoom In Zoom Full Simulation Good	
1.954us 183.5ns	Ons  200ns  400ns  600ns  800ns  1.0us  1.2us  1.4us  1.6us  1.8us  2.1	Dus 2.2us 2.4us 2.6us
CLKD		
START		
ADDR[11:0]		Edit Time Marker 🔀
IDLE	/ 1770.468	Name: MARKO
READ		Marker Type: End Diagram 🔹
WRITE		Return to Marker
POWER		ns
DONE		
ENABLE		Attached to Signal: IDLE 1770.468
COUNT[3:0]	bx(()()(	Keep current attachment
DBUS_INC[15:0]	bx (1) (1) (1) (1) (1) (2) (2) (2	C Attach to time  1770.468
DBUS[15:0]	bz <u>)</u> ;0 <u>,</u> 1 <u>,</u> 1 <u>,</u> 2 <u></u> ;21 <u>,</u> 2 <u>,</u> 21 <u>,</u> 2 <u></u> ;	C Attach to edge 0
		Signal ends snap to marker
		OK Cancel

図21 タイム・マーカーの設定例

WaveFormer Pro を使ったVerilog-HDL ソースの編集

WaveFormer Pro に加えられた Verilog-HDL ソース・ファイルの編集は次のように行います.なお,ここでは SRAM モデル・ファイル sram.v を編集してみます.

- Report ウィンドウ内で sram.v ファイルを開きます. - 17行目を, ram[i]=0; から ram[i]=8へ変更します.

以上の作業で, SRAMセルは, 0の代わりに8で初期化されるようになります.

-Signal Properties Dialog の Simulate Once ボタンをクリックするか,入力信号のエッジを動かして再シミュレーションを行います.

NOTE:

sram.v ファイルをシミュレーションする前にセーブしましょう.sram.vを表示させた状態で, [Report]-[Save Report Tab]メニュを起動すると, Report ウィンドウ内の sram.v ファイルはセーブされます

結果は DBUS に 8 が出力されると予想されますが, 808 が表示されます.これは DBUS が 2 つの SRAM の初期化値 08 (hex 0808= hex 808) から 16 ビットのデータを構成しているためです.

905.1ns =536.6ns	Ons  200ns  400ns  600ns  800ns  1.0us  1.2us  1.4us  1.6us  1.8us  2.0us
CLKO	
START	
ADDR[11:0]	
IDLE	
READ	
WRITE	
POWER	
DONE	
ENABLE	
COUNT[3:0]	bx(0)(1)(2)(3)(4)(5)(6)(7)(8)(9)(A)(B)(C)(D)(E)(F)(
DBUS_INC[15:0]	("bx)(809)((80A)(80B)((80C)(80D))(80E)(80F)()(
DBUS[15:0]	( "bz ),\$08%809%809%80A%80A%80B%80B%80C%80C%80D%80D%80E%80F%%80F%%808%809%

図22 SRAMの初期化値を8に変更したシミュレーション結果(ADDR 信号をすべて0とした)

# 5) **まとめ**

WaveFormer Pro は,

・単なる波形エディタではなく, Verilog-HDL シミュレータを内蔵したモデリング・ツールとしても使える 細かい単位で設計 検証 設計...が容易に行えるツールである(インクリメンタルな設計作業ができる) WaveFormer が自動生成した Verilog-HDL ソースは他の Verilog-HDL シミュレータでも使える

という特徴を備えた強力なツールです.

以上でチュートリアルは終了です.本チュートリアルについてのご質問などは,小社デザインウェーブ企画室 までお願いたします.

WaveFormer Pro, TestBencher Pro そして SynaptiCAD は SynaptiCAD Inc.のトレード・マークです.

CQ 出版株式会社 デザインウェーブ企画室 〒170-8461 東京都豊島区巣鴨1-14-2 CQ ビル4F TEL :03-5395-2126 FAX :03-5395-2127 E-mail :<u>edasupport@cqpub.co.jp</u> web :<u>http://www.cqpub.co.jp/</u>