

<HDL 入門>

基本論理ゲートを組み合わせた簡単な論理回路は、回路図を描いて論理回路を表現した。簡単なカウンタの設計でも同様であった。適切に描かれた論理回路図は見やすく、論理回路の動作を検討する際にとても有効である。しかし、CPU のような大規模の論理回路になると、論理回路の設計はとても大変な作業となり、回路図を描画することも困難になる。

現代における論理回路の設計は、ハードウェア記述言語 HDL を使用する。論理回路やシミュレーションの条件・手順を HDL でソースコードとして記述する。ソースコードをコンパイラでコンパイルし、シミュレータで論理回路の動作を確認する。ハードウェア設計であるが、その設計手段はアプリケーションプログラムの開発手段とほぼ同じものとなっている。

(1)HDL による論理回路

・従来の方法

真理値表や特性表をもとに論理回路を設計し、論理回路図を描き、論理 IC を使い、論理回路を実現する

・現代の方法

真理値表や FF の知識を前提に、論理回路の動作を HDL で記述し、シミュレータで動作を確認し、CPLD や FPGA で論理回路を実現する

(2)使用する素子

論理回路は複数のデバイスで置換可能

	組み込み用 コンピュータ	マイクロ コントローラ	論理 IC	プログラマブル ロジックデバイス	ダイオード・FET・ トランジスタ等
利点	プログラム次第で いろいろできる	小さくて安価 プログラム次第で いろいろできる	複雑な動作をする論 理回路は部品数が 多くなる 簡単なものには最適	小さい プログラム次第で いろいろできる	安い 工夫次第でアナロ グ回路・デジタル 回路が混在できる
応答 速度	とても遅い 1mS	遅い 1mS～1μS	速い 100nS	速い 10nS	とても速い 1nS 未満
欠点	コスト 高価 サイズ 大きい 消費電力大	消費電力が小さい 開発には PC が必 要	部品数が多くなると 大きくなり消費電力 も増える	価格はいろいろ 開発設備が必要	超高速ロジックを つくるにはかなりの 技術が必要

(3)単語

PLD Programable Logic Device

HDL Hardware Description Language

RTL Register Transfer Level

(4)HDL ハードウェア記述言語

- ・論理回路を記述する
- ・論理回路をシミュレートする条件を記述する

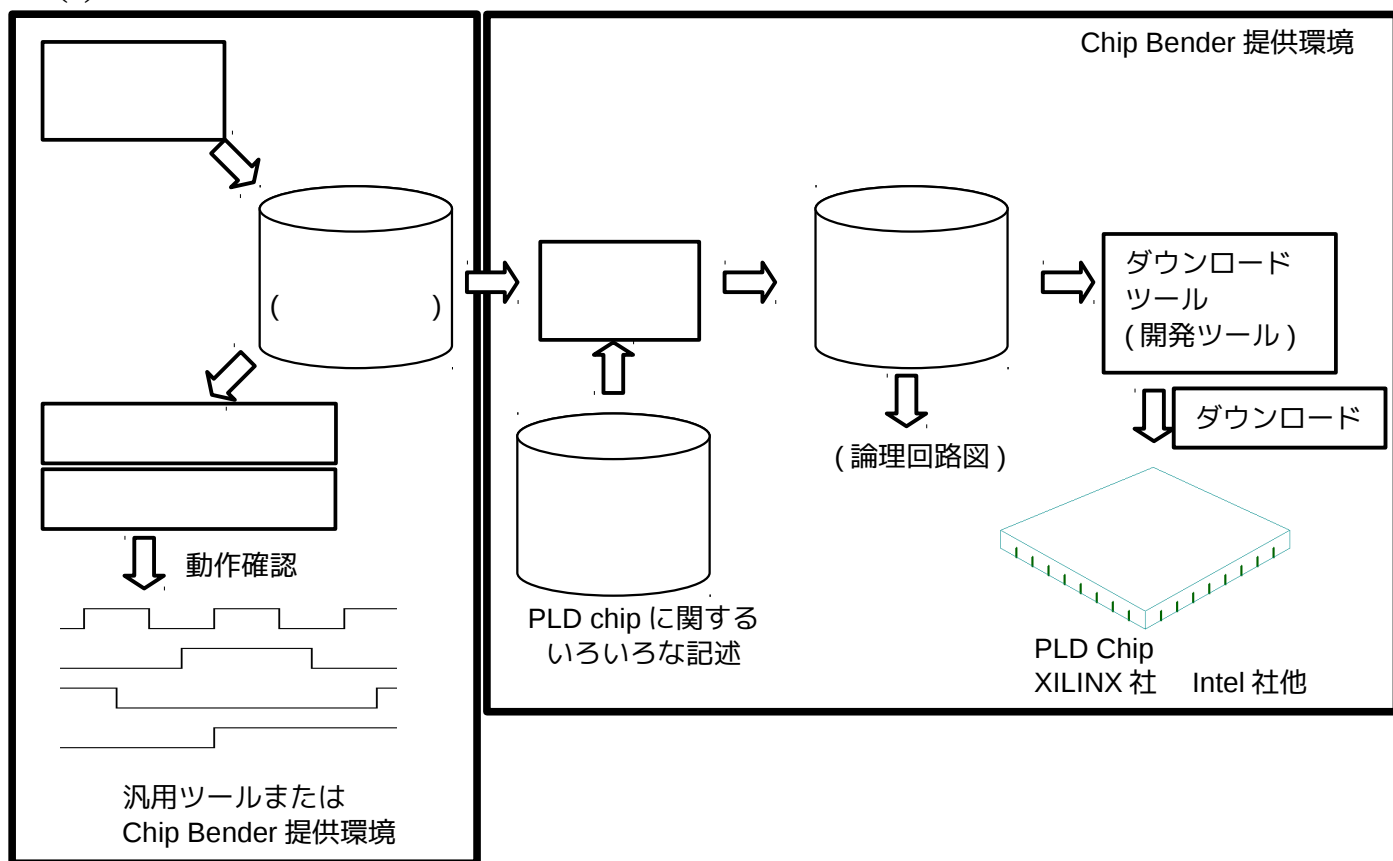
(5)HDL ハードウェア記述言語にはいろいろなものがある

- ・ VHDL
- ・ Verilog HDL
- ・ その他

(6)HDL の利点・欠点

利点	欠点

(7)開発手順



<Verilog HDL 基本>

(1)命名規則

- ・識別子 信号名、モジュール名などにつける名前
- ・使用できる文字は半角文字 アルファベット、数字、\$、_
- ・識別子は予約語を除くアルファベットまたは_ではじまる任意の文字列 大文字小文字は区別する

(2)論理値と数値

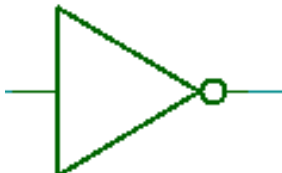
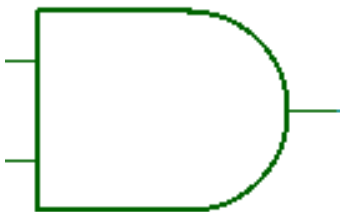

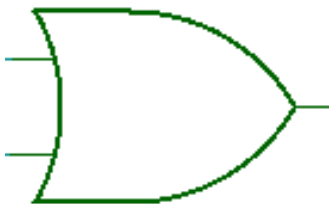
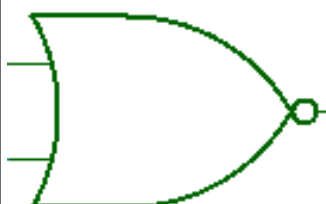

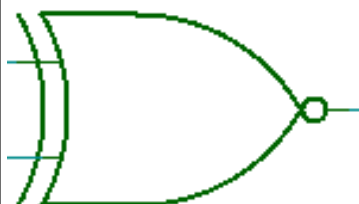
- ・論理値 0,1,Z,z,X,x
- ・数値 数値はビット幅・基数・値の順に表示

2ビット 2進数不定値	4ビット 2進数 値 1101	8ビット 10進数 値 32	8ビット 16進数 値 7f
2'bxx	4'b1101	8'd32	8'h7f

(3)簡単な回路記述

<pre>module gate01(a,b,c); input a,b; output c; assign c=a&b; endmodule</pre>	module で始まり endmodule で終わる 入力と出力を定義 入力と出力の関係
---	--

(4)基本論理ゲート

	<pre>assign c=~a; not(c,a);</pre>		
	<pre>assign c=a&b; and(c,b,a);</pre>		<pre>assign c=a~&b; nand(c,b,a);</pre>
	<pre>assign c=a b; or(c,b,a);</pre>		<pre>assign c=a~ b; nor(c,b,a);</pre>
	<pre>assign c=a^b; xor(c,b,a);</pre>		<pre>assign c=a~^b; xnor(c,b,a);</pre>

(5)テストベンチ

論理回路記述をシミュレートする条件・手順を記述する。回路記述で作成したモジュール gate01 をインスタンス化して g01 とする。入力として与える信号の値(0,1)やタイミング(#1)を記述する。

<pre> `timescale 100ns/100ns module testbench01; reg x,y; wire z; gate01 g01(x,y,z); initial begin \$dumpfile("tmp.vcd"); \$dumpvars(); \$monitor("%t %d %d %d", \$time x,y,z); end initial begin x=1'b0; y=1'b0; #1 y=1'b1; #1 x=1'b1; y=1'b0; #1 y=1'b1; #1 x=1'b0; y=1'b0; #1 \$finish; end endmodule </pre>	<p>シミュレータに時刻単位を 100nS にする。(省略時は 1S)</p> <p>テストベンチはモジュールとして記述する。 テストベンチ名は testbench01 とした。</p> <p>変数宣言</p> <p>モジュール gate01 をインスタンス化して g01 とする。</p> <p>シミュレーション結果の保存先は tmp.vcd</p> <p>シミュレーション結果として表示するものを指定</p> <p>initial シミュレーション開始時、初期条件として動作 t=0 で x=0, y=0 #1 100nS 後 y=1 (途中省略)</p> <p>100nS 後 シミュレーション終了(\$finish)</p>
--	--

<FA を verilogHDL で記述してみよう>

回路記述	テストベンチ
<pre> module fulladder(u,v,w, x,y); input _____ ; output _____ ; assign _____ ; assign _____ ; endmodule </pre>	<pre> `timescale 1ms/1ms module testbench; reg a,b,cin; wire cout,s; fulladder fa(a,b,cin, cout,s); initial begin \$dumpfile("tmp.vcd"); \$dumpvars(); \$monitor("%t %d %d %d %d %d", \$time,cin,a,b, cout,s); end initial begin #0 cin=1'b0; a=1'b0; b=1'b0; #1 cin=1'b0; a=1'b0; b=1'b1; #1 cin=1'b0; a=1'b1; b=1'b0; #1 cin=1'b0; a=1'b1; b=1'b1; #1 cin=1'b1; a=1'b0; b=1'b0; #1 cin=1'b1; a=1'b0; b=1'b1; #1 cin=1'b1; a=1'b1; b=1'b0; #1 cin=1'b1; a=1'b1; b=1'b1; #1 cin=1'b0; a=1'b0; b=1'b0; #1 \$finish; end endmodule </pre>