2017/11/21

電気電子情報工学 実践演習B

FPGAを用いた組込みシステムの協調設計

峯脇 さやか, 吉田 太一

情報コース　テーマ3

概要

- Verilog-HDLの基礎文法

- 課題（Verilog-HDLソース記述）

Verilog-HDLの基礎文法

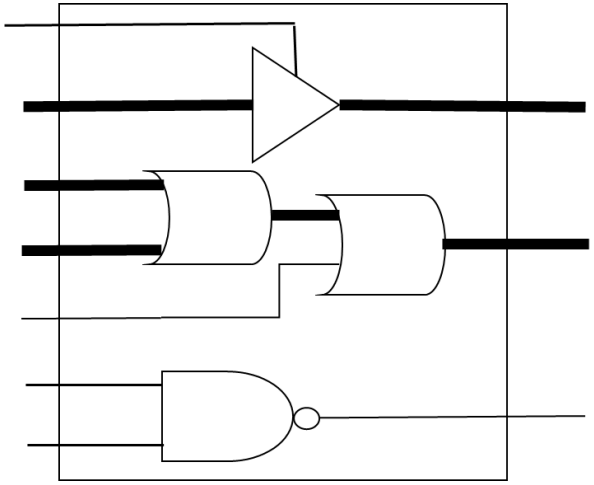
Verilog-HDL

ソフトウェア的にディジタル回路の設計を行なうプログラム言語

```
//PwmCtrl.v
//
module PwmCtrl (  RST_N,   CLK,  LED0 );

input CLK, RST_N;
output LED0;
reg [27:0] counter0;

always@( negedge RST_N or posedge CLK )
begin
   if (RST_N == 1'b0 ) begin
      counter0 <= 0;
   end else begin
      counter0 <= counter + 1;
   end
end

assign LED0 = counter0[27];
endmodule
```

C言語っぽい
回路構成を記述した
プログラム

Verilog-HDL
コンパイラ



回路が出力

Verilog-HDL：数値表現

ソフトウェア的にディジタル回路の設計を行なうプログラム言語

Verilog	ビット幅	基数	2進表現
1'b0	1	2進	0
4'b0100	4	2進	0100
4'd4	4	10進	0100
4'd7	4	10進	0111
8'hf0	8	16進	11110000
8‘b1000_1111	8	2進	10001111
1	32	10進	000...0001

Verilog-HDL：数値表現

定数の記述

書式：<ビット幅>'<基数><数値>

ビット幅　　：定数のビット幅を10進数で記述
基数　　：基数(b 2進数、o 8進数、d 10進数、h 16進数)
数値　　：定数値（区切り文字にアンダースコア'_'を使用できる）
基数に指定した文字と、論理値のx,zを記述できる
※10進記述の場合は、x,zを使用できない

＜数値表現例＞

Verilog	ビット幅	基数	2進表現
1'b0	1	2進	0
4'b0100	4	2進	0100
4'd4	4	10進	0100
4'd7	4	10進	0111
8'hf0	8	16進	11110000
8‘b1000_1111	8	2進	10001111
1	32	10進	000...0001

Verilog-HDL：論理値

Verilog HDLで扱う論理値は以下の4種類

論理値	意味
0	Low
1	High
x	不定値
z	High Impedance

x：0か1かは不定だが、どちらかの値をとる（valid）
z：0でも1でもない（invalid、回路状態としてHigh Impedance）

Verilog-HDL：演算子

名称	記号	定義
算術演算	+	加算（プラス符号）
	-	減算（マイナス符号）
	*	乗算
	/	除算
	%	剰余
ビット演算	~	ビットごとの論理反転
	&	ビットごとのAND
	|	ビットごとのOR
	^	ビットごとのExclusive OR
	~^	ビットごとのExclusive NOR

Verilog-HDL：演算子

名称	記号	定義
リダクション演算	&	各桁ビットのAND（単項演算）
	~&	各桁ビットのNAND（単項演算）
	|	各桁ビットのOR（単項演算）
	~|	各桁ビットのNOR（単項演算）
	^	各桁ビットのExclusive OR（単項演算）
	~^	各桁ビットのExclusive NOR（単項演算）
論理演算	!	論理値のNOT
	&&	論理値のAND
	||	論理値のOR
等号演算	==	論理等号
	!=	論理不等号
	===	ケース等号
	!==	ケース不等号

Verilog-HDL：演算子

名称	記号	定義
関係演算	<	左辺が小さい
	<=	左辺が小さいか等しい
	>	左辺が大きい
	>=	左辺が大きいか等しい
シフト演算	<<	右オペランド分左シフト（空いたビットは0）
	>>	右オペランド分右シフト（空いたビットは0）
条件演算	?:	条件？真の場合の結果：偽の場合の結果
連接演算	{ }	2つ以上のオペランドをひとつのビット表現にまとめる

Verilog-HDL：モジュール

回路を記述する基本単位

module　モジュール名（入出力ポートリスト）；

< 宣言部 >

下記の素子を宣言

入出力端子、内部の信号線、レジスタの変数名

< 回路記述部 >

宣言部で宣言したものの相互接続関係を記述

endmodule

Verilog-HDL：サンプル1

回路を記述する基本単位

モジュール名と入出力

宣言部

回路記述部

```
//PwmCtrl.v
//
module PwmCtrl (  RST_N,   CLK,  LED0 );

input CLK, RST_N;
output LED0;
reg [27:0] counter0;

always@( negedge RST_N or posedge CLK )
begin
   if (RST_N == 1'b0 ) begin
      counter0 <= 0;
   end else begin
      counter0 <= counter + 1;
   end
end

assign LED0 = counter0[27];
endmodule
```

Verilog-HDL：宣言部

モジュール内の各素子を宣言

- ポート宣言（入出力端子）

```
input       入力信号名;
output      出力信号名;
inout       双方向信号名;

input  [4:1]  入力バス名;
              (4bitバス信号)
```

Verilog-HDL：宣言部

モジュール内の各素子を宣言

- ネット宣言（線で繋ぐ）

  wire　　信号線名;

- レジスタ宣言（記憶素子）

  reg　　レジスタ名;

- パラメタ宣言（ただの変数）

  parameter　変数名;

Verilog-HDL：回路記述部

- assign文

  組合せ回路の記述に使用

  継続的な代入

- always文

  順序回路の記述に使用

  クロック等による同期処理を実現

ネット　宣言された変数への代入は、assign文のみ可能
レジスタ宣言された変数への代入は、always文のみ可能

Verilog-HDL：回路記述部

- assign文

Verilog-HDLは回路を設計しますので、

C++などのソフトウェアと異なり

１命令ずつ処理を行なわず、

assign文は**同時に動作**します。

設計しているのは「**回路**」です。

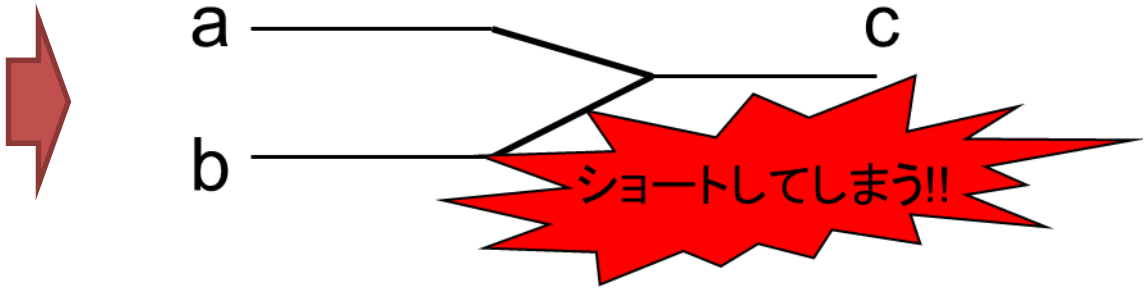
注意してプログラムしてください。

- レジスタ宣言された変数への代入は、always文のみ可能

Verilog-HDL：注意点

回路を設計しているという意識をもつ事

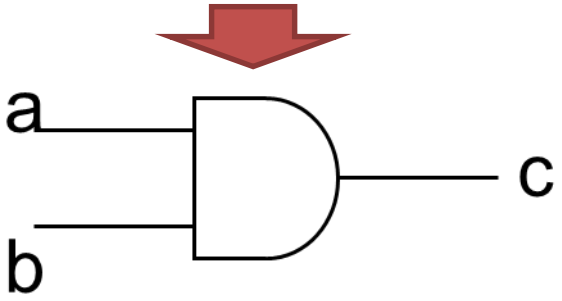
```
wire    a, b, c;
assign  c = a;
assign  c = b;
```



Verilog-HDL：assign文

継続的代入文、「線でつながっている」と考える

```
module   my_and(a, b, c)
    input     a, b;
    output    c;
    wire      a, b, c;
    assign    c = a & b;     // 2入力AND
endmodule
```



Verilog-HDL：always文

「イベント式」に書かれた信号が変化したときに、
beginからend内の動作を行なう

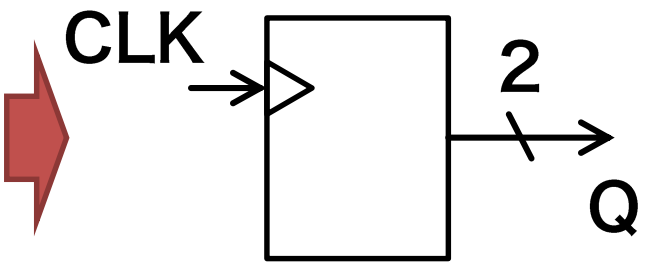
```
always  @(イベント式)
begin
        所望動作
end
```

イベント式

- 信号名　　：信号の立ち上がりと立ち下がり
- posedge 信号名　：信号の立ち上がりのみ
- negedge 信号名　：信号の立ち下がりのみ

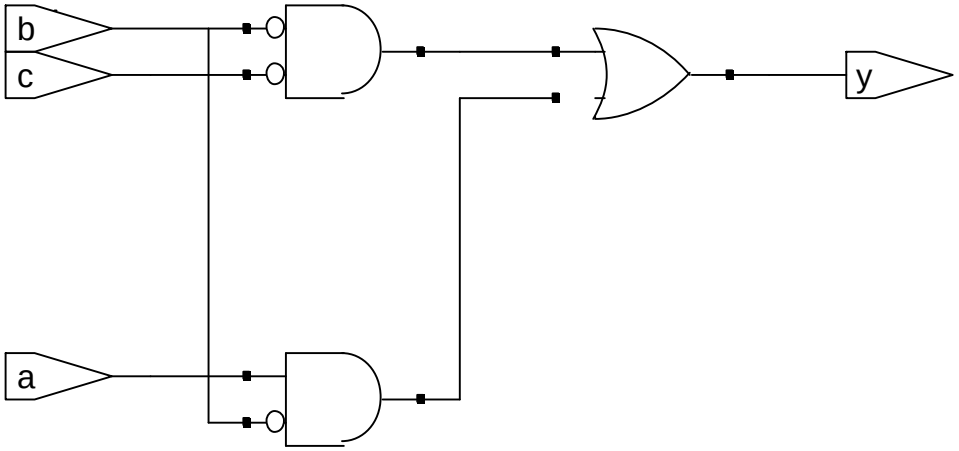
Verilog-HDL：always文

```
module   my_2counter(CLK, Q)
    input     CLK;
    output    [1:0] Q;
    wire      CLK
    reg       [1:0] Q;

    always   @(posedge CLK);
    begin
        Q <= Q + 1'h1;
    end
endmodule
```



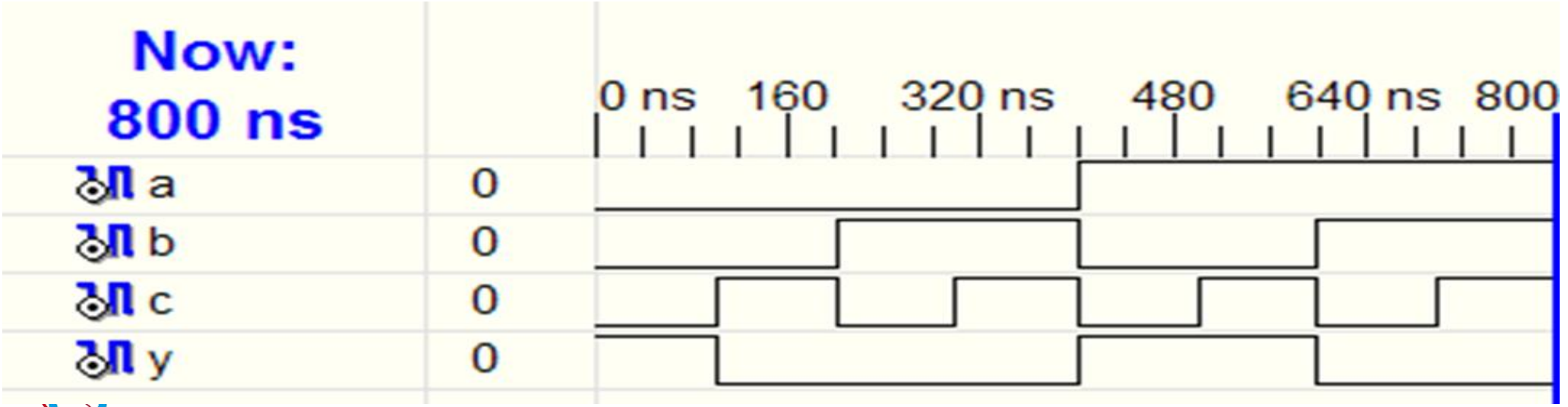
Verilog-HDL：サンプル2

```
module   sample(a, b, c, y)
    input     a, b, c;
    output    y;
    assign    y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
endmodule
```



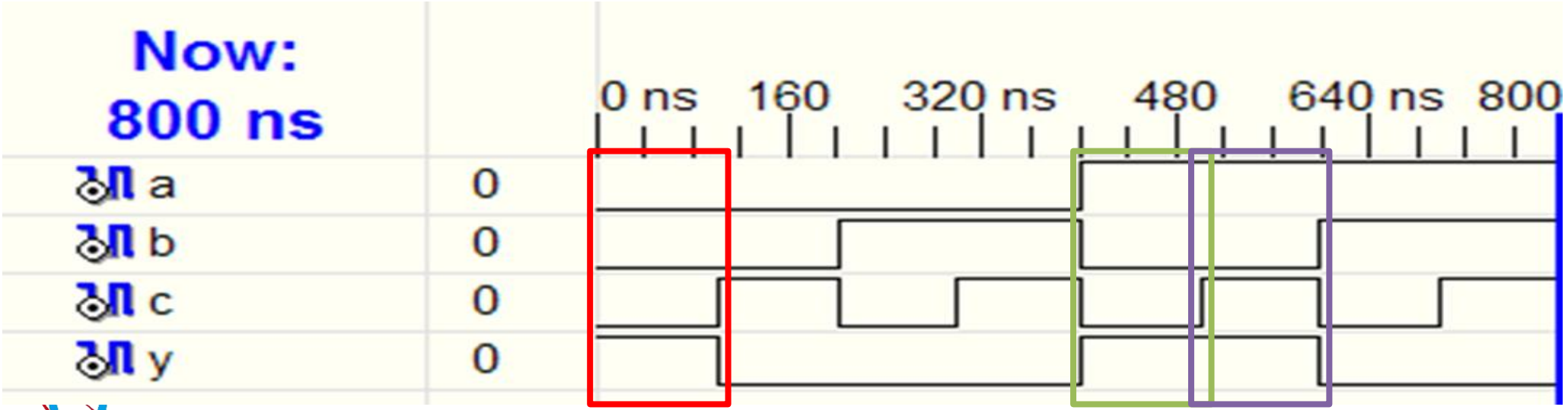
Verilog-HDL：サンプル2

```
assign   y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
```



Verilog-HDL：サンプル2

```
assign  y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
```



Verilog-HDL：条件分岐

- if文

```
if (条件式)
    ～～～～～;
else
    ～～～～～;
```

- case文

```
case (対象信号)
    状態A : ～～～;
    状態B : ～～～;
        ⋮
    default : ～～～;
endcase
```

Verilog-HDL：遅延制御

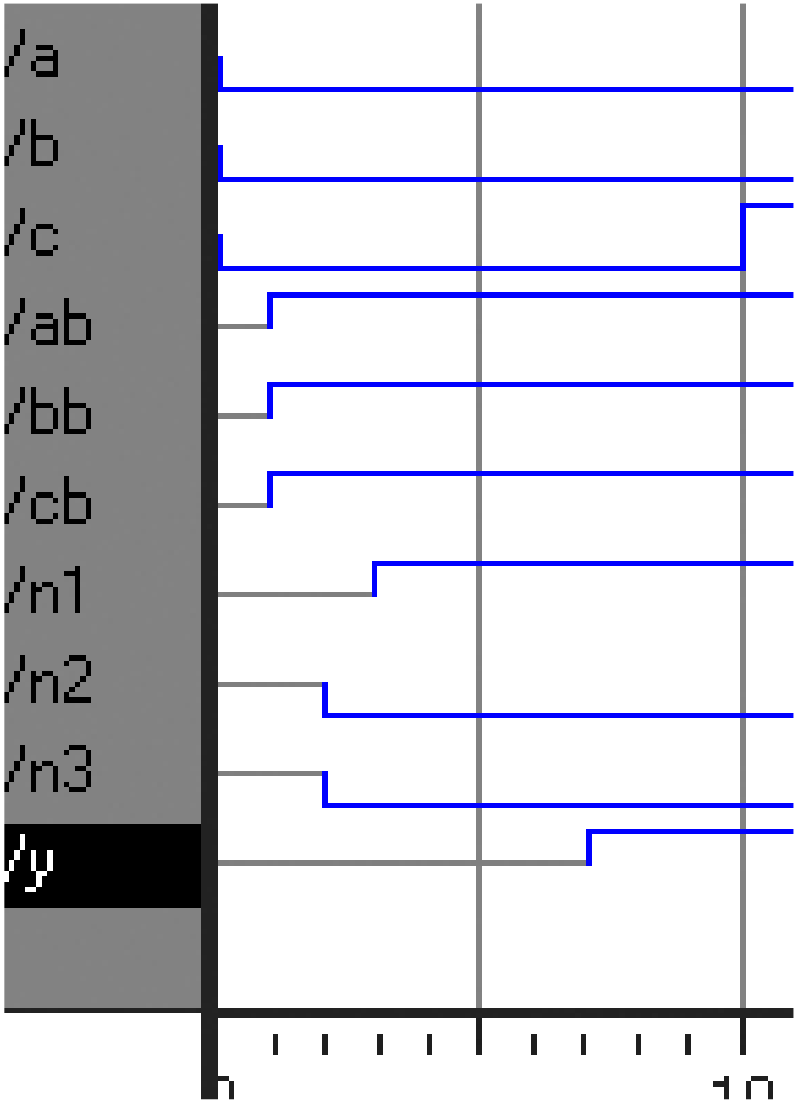
「#[遅延値]」で遅延を設定できる

遅延値は整数、単位はない

単位を指定する場合は`timescale

`timescale 「1」の単位/丸め値

```
module example(input  a, b, c, output y);
  wire ab, bb, cb, n1, n2, n3;
  assign #1 {ab, bb, cb} = ~{a, b, c};
  assign #2 n1 = ab & bb & cb;
  assign #2 n2 = a & bb & cb;
  assign #2 n3 = a & bb & c;
  assign #4 y = n1 | n2 | n3;
endmodule
```



Verilog-HDL：ブロッキング文

initial内の非代入「=」で書かれた構文は逐次処理される

逐次処理

ブロッキング代入文	ノンブロッキング代入文

initial
begin
a = 3;
b = 5;
 	
#10
a = b + 1;
b = a + 1;
end
	initial
	begin
	a <= 3;
	b <= 5;
	
	#10
	a <= b + 1;
	b <= a + 1;
	end

右辺が計算され同時に代入される

右辺が計算され同時に代入される

a=6, b=7

a=6, b=4