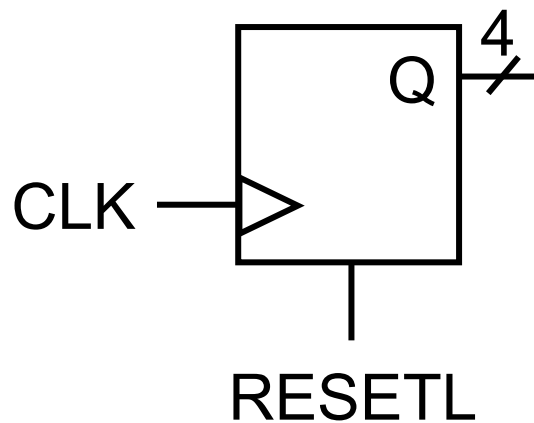


Verilog-HDL ゼミ 第 3 週・練習問題「順序回路とテストベンチ」

問 1.

プログラムリスト 1 に示す 4 ビットバイナリカウンタを完成させ、シミュレーションによりカウンタが正しく動作することを確認してください。シミュレーションに使用するテストベンチについては、プログラムリスト 2 を使用してください。なお、4 ビットバイナリカウンタの仕様は図 1 の通りとします。



入力：クロック CLK, リセット RESETL

出力：Q (4 ビット)

動作：CLK の立ち上がりでカウントアップ,
RESETL が 0 のときは, CLK に関わらず
非同期的に Q を 0 にする

図 1. 4 ビットバイナリカウンタの仕様

```

module counter ( CLK, RESETL, Q ); // 4bit counter
input  CLK, RESETL;
outbut ...;
reg    ...;

// 以下より always 文によるリセット及びカウントアップ記述
always @( posedge CLK or negedge RESETL ) begin
    if ( RESETL == 1'b0 )
        ...;
    else
        ...;
end

endmodule

```

プログラムリスト 1. counter.v

```

`timescale 1ps / 1ps

module counter_tb;
reg    CLK, RESETL;
wire   [3:0] Q;

parameter STEP = 1000;

// 4ビットカウンタを呼び出す
counter counter( CLK, RESETL, Q );

// クロック生成
initial begin
    CLK = ~CLK; #(STEP/2); // 半ステップ毎に CLK を反転してクロック生成
end

// テスト入力
initial begin
    $dumpfile("counter_tb.vcd");
    $dumpvars(0, counter_tb);
    $monitor( $stime, " CLK=%b RESETL=%b Q=%b", CLK, RESETL, Q);

    CLK = 0; RESETL = 1;
    #STEP    RESETL = 0;
    #STEP    RESETL = 1;
    // 20 回カウントアップして終了(20 クロック分進める)
    ...      $finish;
end

endmodule

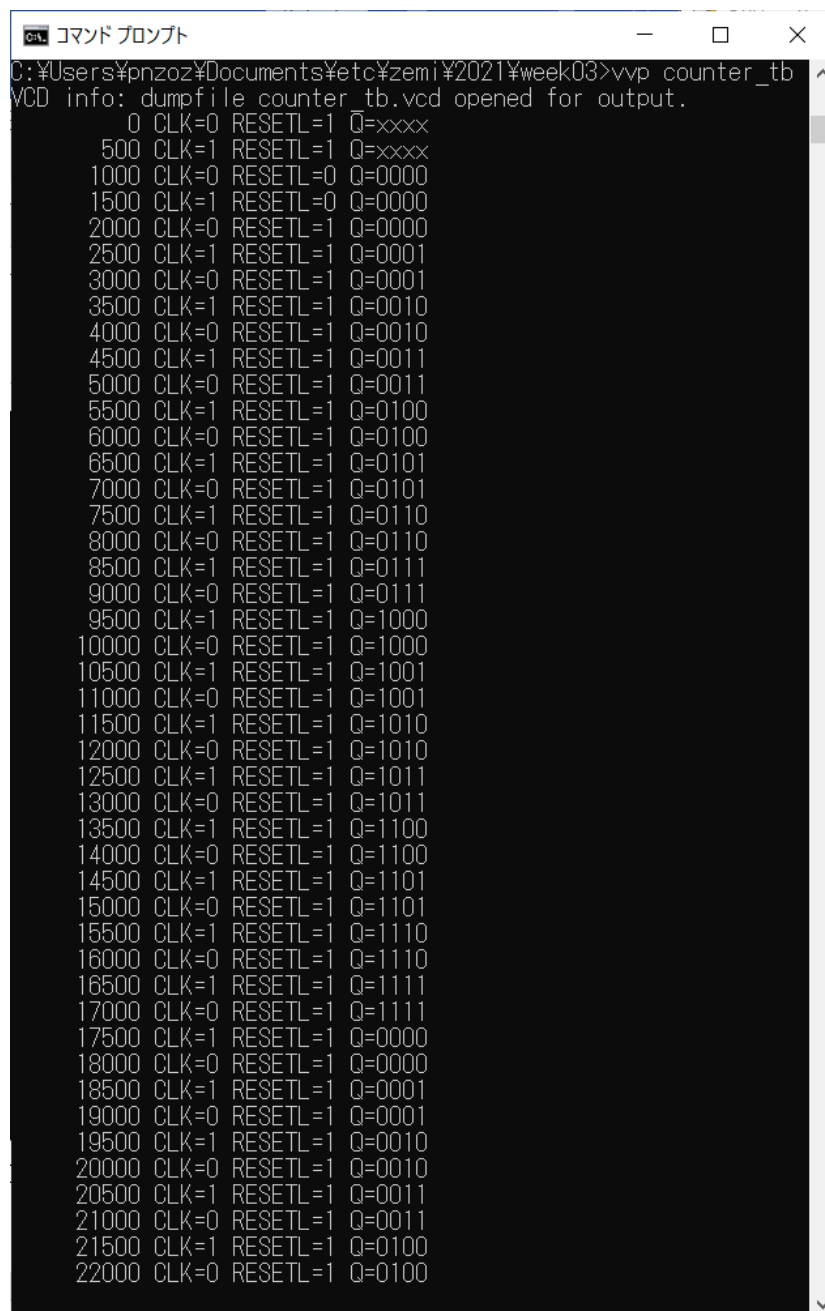
```

プログラムリスト 2. counter_tb.v

コマンド例) > iverilog -o counter_tb -s counter_tb counter_tb.v counter.v
vvp counter_tb
gtkwave counter_tb.vcd

or

cver counter_tb.v counter.v
gtkwave counter_tb.vcd



```
C:\Users\pnzoz\Documents\etc\zemi\2021\week03>vvp counter_tb
VCD info: dumpfile counter_tb.vcd opened for output.
  0 CLK=0 RESETL=1 Q=xxxx
  500 CLK=1 RESETL=1 Q=xxxx
 1000 CLK=0 RESETL=0 Q=0000
 1500 CLK=1 RESETL=0 Q=0000
 2000 CLK=0 RESETL=1 Q=0000
 2500 CLK=1 RESETL=1 Q=0001
 3000 CLK=0 RESETL=1 Q=0001
 3500 CLK=1 RESETL=1 Q=0010
 4000 CLK=0 RESETL=1 Q=0010
 4500 CLK=1 RESETL=1 Q=0011
 5000 CLK=0 RESETL=1 Q=0011
 5500 CLK=1 RESETL=1 Q=0100
 6000 CLK=0 RESETL=1 Q=0100
 6500 CLK=1 RESETL=1 Q=0101
 7000 CLK=0 RESETL=1 Q=0101
 7500 CLK=1 RESETL=1 Q=0110
 8000 CLK=0 RESETL=1 Q=0110
 8500 CLK=1 RESETL=1 Q=0111
 9000 CLK=0 RESETL=1 Q=0111
 9500 CLK=1 RESETL=1 Q=1000
10000 CLK=0 RESETL=1 Q=1000
10500 CLK=1 RESETL=1 Q=1001
11000 CLK=0 RESETL=1 Q=1001
11500 CLK=1 RESETL=1 Q=1010
12000 CLK=0 RESETL=1 Q=1010
12500 CLK=1 RESETL=1 Q=1011
13000 CLK=0 RESETL=1 Q=1011
13500 CLK=1 RESETL=1 Q=1100
14000 CLK=0 RESETL=1 Q=1100
14500 CLK=1 RESETL=1 Q=1101
15000 CLK=0 RESETL=1 Q=1101
15500 CLK=1 RESETL=1 Q=1110
16000 CLK=0 RESETL=1 Q=1110
16500 CLK=1 RESETL=1 Q=1111
17000 CLK=0 RESETL=1 Q=1111
17500 CLK=1 RESETL=1 Q=0000
18000 CLK=0 RESETL=1 Q=0000
18500 CLK=1 RESETL=1 Q=0001
19000 CLK=0 RESETL=1 Q=0001
19500 CLK=1 RESETL=1 Q=0010
20000 CLK=0 RESETL=1 Q=0010
20500 CLK=1 RESETL=1 Q=0011
21000 CLK=0 RESETL=1 Q=0011
21500 CLK=1 RESETL=1 Q=0100
22000 CLK=0 RESETL=1 Q=0100
```

図 1. 出力結果の例(vvp)

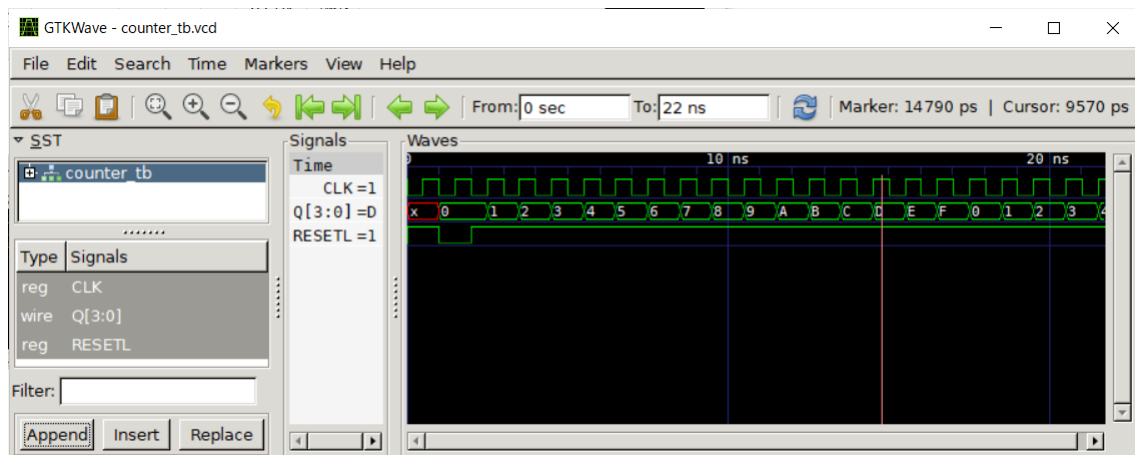


図 2. 波形表示例(gtkwave)

一般的に回路は、演算などのデータ処理を行うデータパスと、それを制御する制御部に分けられます。制御部においては、データパスに対する制御信号を生成します。この制御信号を作り出すための順序回路をステートマシンと呼びます。

ここでは、デジタルウォッチの動作を制御するための制御部として働くステートマシンを作成します。図 3 にデジタルウォッチの表示とスイッチの仕様を示します。表示は時・分・秒それぞれ 2 桁の全 6 桁です。スイッチは、SW1, SW2, SW3 の 3 種類あります。モードは通常表示のノーマルモードと、時刻修正モードの 2 種類があります。また、図 4 にデジタルウォッチのブロック図を示します。入力、システムリセット信号 RESETL、クロック信号 CLK に加え、スイッチ SW1, SW2, SW3 となります。内部は、表示ブロックとカウントブロック、制御部から構成されます。今回は、これら 3 つの要素のうち制御部の作成のみを行います。

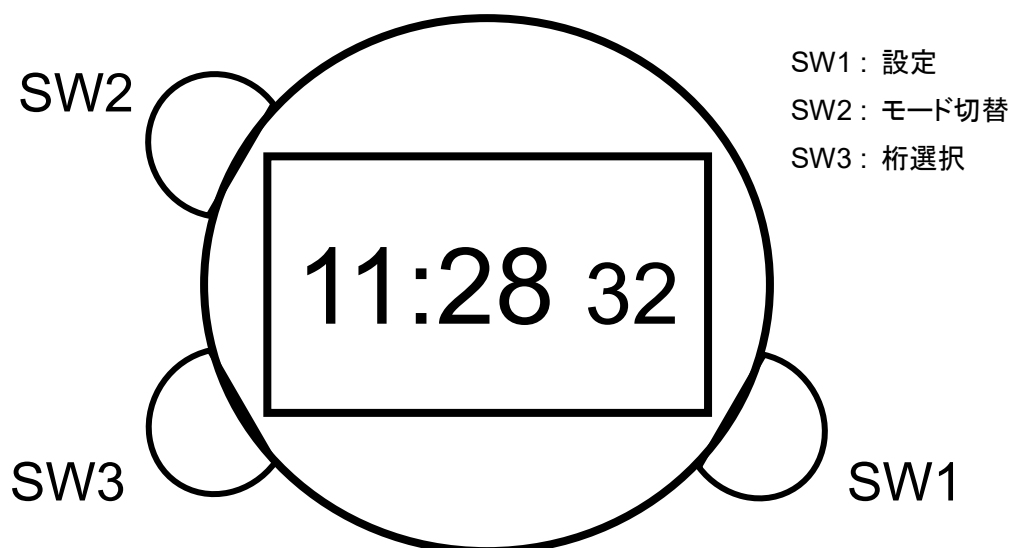


図 3. デジタルウォッチの仕様

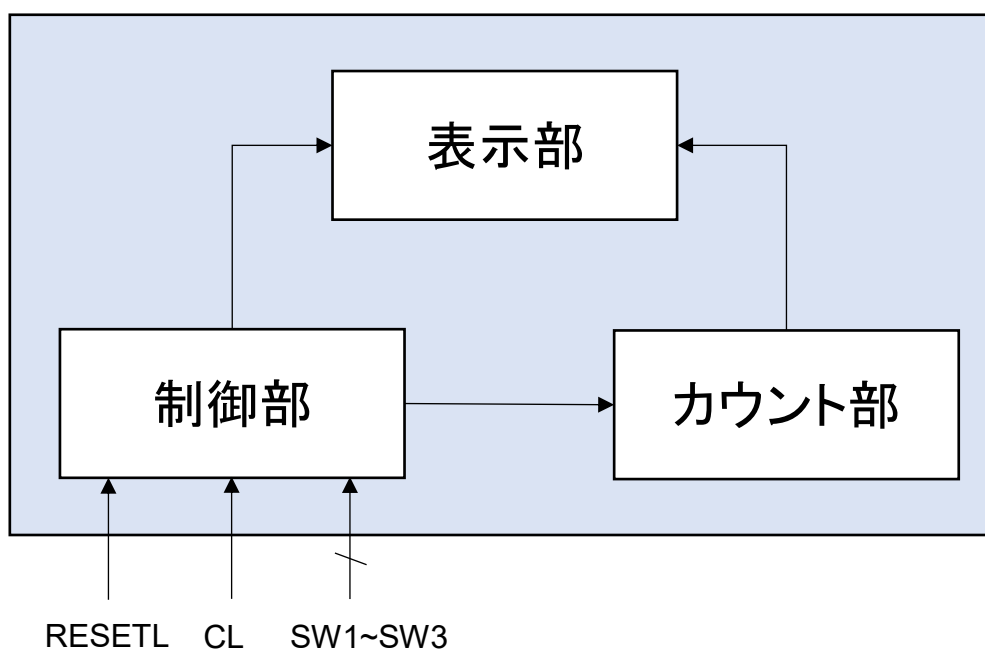


図 4. デジタルウォッチのブロック図

前述したように、デジタルウォッチはノーマルモードと時刻修正モードの 2 つのモードを持ちます。さらに時刻修正モードについて細かく分けると、デジタルウォッチは、以下に示す 4 つの状態を持つことになります。

- 通常状態
- 時修正状態
- 分修正状態
- 秒修正状態

これらの状態についての状態遷移図を図 5 に示します。状態は **RESETL** によって通常状態に遷移します。**SW2** によって秒修正状態へ遷移し、その後 **SW3** によって秒修正、時修正、分修正と修正状態を繰り返します。どの状態でも、**SW2** によって通常状態に戻ります。また、表 1 に各状態における **SW1** の動作を示します。

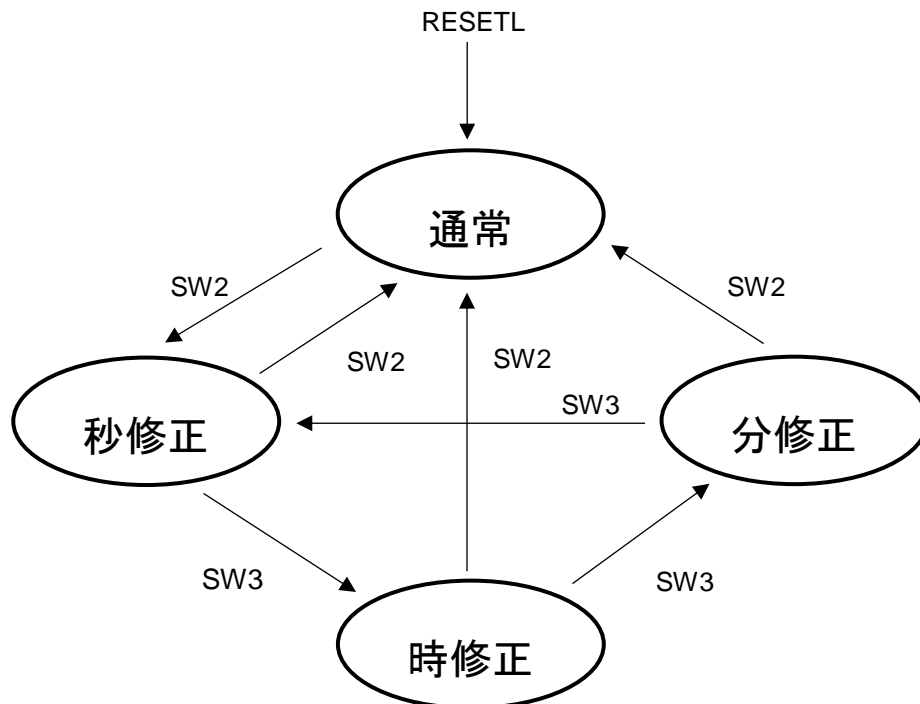


図 5. デジタルウォッチの状態遷移図

表 1. 各状態の SW1 動作と表示

状態	ステート名	SW1 動作	表示
通常	NORMAL	なし	通常
秒修正	SEC	秒リセット	秒 2 桁点減
分修正	MIN	+1 分	分 2 桁点減
時修正	HOUR	+1 時	時 2 桁点減

問 2.

プログラムリスト 3, 4 に示すデジタルウォッチのステートマシンを完成させ、シミュレーションによりステートマシンが正しく動作することを確認してください。ステートマシンを作成する際には、ステート生成回路のレジスタ部分は図 5 を参考に、制御信号の生成部分は表 1 を参考に、各々記述してみてください。また、シミュレーションに使用するテストベンチについては、プログラムリスト 5, 6 を使用してください。ステートマシンの正しい動作を確認する際は、最後のコマンド例に示したように、出力ファイルと期待値ファイルの内容を `fc` または `diff` コマンドによりファイルの内容に相違がないことを確認してみてください。

```

module state ( CLK, RESETL, SW1, SW2, SW3,
               sec_resetl, min_inc, hour_inc, sec_onoff, min_onoff,
hour_onoff );
input CLK, RESETL, SW1, SW2, SW3;
output sec_resetl, min_inc, hour_inc; // 秒リセット(負論理), +1 分, +1 時
のカウンタ部への制御信号
output sec_onoff, min_onoff, hour_onoff; // 秒・分・時点減の表示部への制
御信号

reg [1:0] cur; // ステータレジスタ
reg [1:0] nxt; // ステータ生成回路(組み合わせ回路)

// ステータ名の定義
parameter NORMAL = 2'b00, SEC = 2'b01, MIN = 2'b10, HOUR = 2'b11;

// ステータレジスタ
always @( posedge CLK or negedge RESETL ) begin
    if ( !RESETL )
        cur <= NORMAL;
    else
        cur <= nxt;
end

// ステータ生成回路 (組み合わせ回路)
always @( cur or SW1 or SW2 or SW3 ) begin
    case ( cur )
        NORMAL: if ( SW2 )
                    nxt <= SEC; // 通常時に SW2 が押されたとき
                else
                    nxt <= NORMAL; // それ以外の場合
        SEC:    if ( SW2 )
                    nxt <= NORMAL; // 秒修正時に SW2 が押された
とき
                else if ( SW3 )
                    nxt <= HOUR; // 秒修正時に SW3 が押されたとき
    endcase
end

```

プログラムリスト 3. state.v (前半部)


```

        else
            nxt <= SEC; // それ以外の場合
// 以下 NORMAL, SEC 時と同様に状態遷移図を参照して記述
    HOUR: if ( SW2 )
        nxt <= ...;
    else if ( SW3 )
        nxt <= ...;
    else
        nxt <= ...;
    MIN:  if ( SW2 )
        nxt <= ...;
    else if ( SW3 )
        nxt <= ...;
    else
        nxt <= ...;
    default:nxt <= 2'bxx; // cur が不定値になってしまったとき
    endcase
end

// 制御信号の生成
// 秒修正時に SW1 が押されたとき
assign sec_reset1    = ~(cur==SEC) & SW1);
// 分修正時に SW1 が押されたとき
assign min_inc       = ...;
// 時修正時に SW1 が押されたとき
assign hour_inc      = ...;
// 秒修正時は秒点滅状態
assign sec_onoff = (cur==SEC);
// 分修正時は分点滅状態
assign min_onoff = ...;
// 時修正時は時点滅状態
assign hour_onoff = ...;

endmodule

```

```

`timescale 1ps / 1ps

module state_tb;
reg    CLK, RESETL, SW1, SW2, SW3;
wire sec_resetl, min_inc, hour_inc;
wire sec_onoff, min_onoff, hour_onoff;
integer mcd;

parameter STEP = 1000;

// ステートマシンを呼び出す
state state( CLK, RESETL, SW1, SW2, SW3, sec_resetl, min_inc,
hour_inc, sec_onoff, min_onoff, hour_onoff );

// クロックは遷移とそれに伴う制御信号の変化に必要な
always begin
    CLK = ~CLK; #(STEP/2);
end

// テスト入力
initial begin
    $dumpfile("state_tb.vcd");
    $dumpvars(0, state_tb);
    $monitor( $stime, " CLK=%b : cur=%b RESETL=%b SW1=%b SW2=%b
SW3=%b sec_resetl=%b, min_inc=%b, hour_inc=%b, sec_onoff=%b,
min_onoff=%b, hour_onoff=%b",
            CLK, state.cur, RESETL, SW1, SW2, SW3,
sec_resetl, min_inc, hour_inc, sec_onoff, min_onoff, hour_onoff);

    CLK = 0; RESETL = 1; SW1 = 0; SW2 = 0; SW3 = 0;
    #STEP      RESETL = 0;
    #STEP      RESETL = 1;
    // 通常時の遷移と制御信号のテスト
    // 2 つ以上の SW が同時に 1 になることを想定しない
    #STEP      SW1 = 1;

```

プログラムリスト 5. state_tb.v (前半部)

```

#STEP          SW1 = 0; SW2 = 1;
#(STEP*2)      SW2 = 0; SW3 = 1;
#STEP          SW3 = 0; SW2 = 1;
// 秒修正時の遷移と制御信号のテスト
#STEP          SW2 = 0; SW1 = 1;
#STEP          SW1 = 0; SW2 = 1;
#(STEP*2)      SW2 = 0; SW3 = 1;
// 時修正時の遷移と制御信号のテスト
#STEP          SW3 = 0; SW1 = 1;
#STEP          SW1 = 0; SW2 = 1;
#(STEP*2)      SW2 = 0; SW3 = 1;
// 分修正時の遷移と制御信号のテスト
#(STEP*2)      SW3 = 0; SW1 = 1;
#STEP          SW1 = 0; SW2 = 1;
#(STEP*2)      SW2 = 0; SW3 = 1;
#(STEP*3)      SW3 = 0; SW2 = 1;
#STEP $finish;

end

endmodule

```

プログラムリスト 6. state_tb.v (後半部)

コマンド例) > iverilog -o state_tb -s state_tb state_tb.v state.v
 vvp state_tb > state_tb_out.txt # テストベンチの出力 txt に保存
 fc state_tb_out.txt state_tb_true.txt # 出力と期待値を比較

or

```

cver state_tb.v state.v > state_tb_out.txt
fc state_tb_out.txt state_tb_true.txt

```

※ ファイル比較コマンドの fc は、Linux 系の OS では diff となります。

参考文献

小林 優, 「入門 Verilog HDL 記述」, CQ 出版社, 1996 年(初版)