ANALOG DEVICES

TNJ-017
アナログ電子回路技術ノート

スイッチ読み出しでのチャタリング防止の 3 種類のアプローチ

著者: 石井 聡

はじめに

「スイッチのチャタリングはアナログ的振る舞いか？デジタル的振る舞いか？」ということで、アナログ・チックだろうという考えのもと技術ノートの話題としてみます（「メカ的だろう！」と言われると進めなくなりますので…ご容赦を…）。

さてこの技術ノートでは、スイッチのチャタリング対策（「チャタ取り」とも呼ばれる）について、電子回路の超初級ネタではありますが、デジタル回路、マイコンによるソフトウェア、そして CR 回路によるものと、3 種類を綴ってみたいと思います。

チャタリングのようすとは？

まずは最初に、チャタリングの発生しているようすをオシロスコープで観測してみましたので、これを図 1 にご紹介します。こんなふうにバタバタと変化します。チャタリングは英語で「Chattering」と書きますが、この動詞である「Chatter」は「ぺちゃくちゃしゃべる。〈鳥が〉けたたましく鳴く。〈サルが〉キャッキャッと鳴く。〈歯・機械などが〉ガチガチ［ガタガタ］音を立てる」という意味です（weblio 辞書より）。そういえばいろんなところで Chatter を聞くなあ…（笑）。

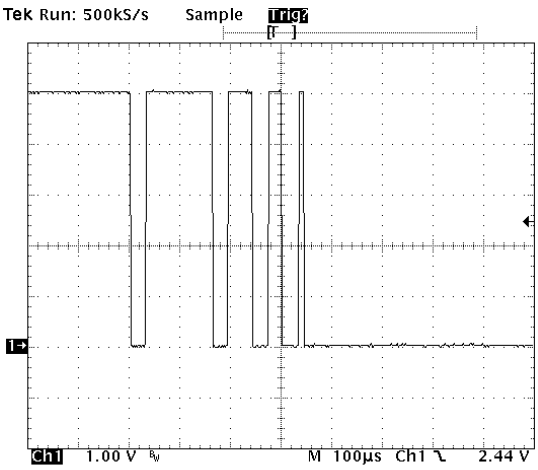

図 1. スイッチのチャタリングが発生しているようす
（横軸は 100us/DIV）

先鋒は RTL（デジタル回路）

余談ですが、エンジニア駆け出し 4 年目位のときに 7k ゲートのゲートアレーを設計しました。ここで外部からの入力信号のストローブ設計を間違えて、バグを出してしまいました…（汗）。外部からの入力信号が非同期で、それの処理を忘れたというところです。チャタリングと似たような原因でありました。ES チェックで分かったのでよかったのですが、ゲートアレー自体は作り直しでした。中はほぼ完ぺきでしたが、がっくりでした。外部との I/F は（非同期ゆえ）難しいです（汗）…。

当時は FPGA でプロトタイプを設計し（IC は XC2000！）、回路図（紙）渡しで作りました。テスト・ベクタは業者さんに 1 か月入り込んで、そこのエンジニアの方と一緒にワーク・ステーションの前で作り込みました。その会社の偉い方がやってきて、私を社外の人と思わず、私の肩に手をやり「あれ？誰だれ君はどした？」と聞いてきたりした楽しい思い出です（笑）。

図 2. スイッチのチャタリング防止を
FPGA 上で RTL でやってみた

それでは本題に…

閑話休題ということで…。さて、図 2 の上側の 7 セグメント LED の表示を変えるべく（というより内部動作パラメータをスイッチで操作すべく）、下側のスイッチで設定するための RTL（VHDL）を書きました。

図 3 にこの RTL を示します。チャタリング対策のアルゴリズム（アプローチ）も多岐の方法論があろうかと思いますが、一例としてご覧ください。TNJ-016 でも書きましたが、Verilog が書けないので、VHDL です。一応これで動いています。

応用される場合は、必要に応じてカウンタに toggle enable を追加して strobe を入れるなど、カウンタのビット数が無用に伸びないようにしてください（この回路のクロックは低速になっているので、カウンタは 5 ビットで済んでいます）。マイコンのソフトより論理回路のほうがタイミング（時間）予測が簡単だから楽ですね（マイコンで割り込み処理すれば別ですが）。

Rev. 0

アナログ・デバイセズ株式会社

本　　社／〒105-6891 東京都港区海岸 1-16-1 ニューピア竹芝サウスタワービル
電話 03（5402）8200
大阪営業所／〒532-0003 大阪府大阪市淀川区宮原 3-5-36 新大阪トラストタワー
電話 06（6350）6868

```
architecture Behavioral of SwChat is
signal LastSwState : std_logic;
signal CurrentSwState : std_logic;
signal DurationCounter : std_logic_vector(4
downto 0);

begin
  process(CLK, RESET) begin
    if (RESET = '1') then
      LastSwState <= '0';
      CurrentSwState <= '0';
    elsif (CLK'event and CLK = '1') then
      LastSwState <= CurrentSwState;
      CurrentSwState <= SwitchIn;
    end if;
  end process;

  process(CLK, RESET) begin
    if (RESET = '1') then
      DurationCounter <= "00000";
    elsif (CLK'event and CLK = '1') then
      if ((LastSwState xor CurrentSwState) = '1')
      then
        DurationCounter <= "00000";
      elsif (DurationCounter /= "11111") then
        DurationCounter <= DurationCounter + 1;
      end if;
    end if;
  end process;

  process(CLK, RESET) begin
    if (RESET = '1') then
      RiseEdge <= '0';
      FallEdge <= '0';
    elsif (CLK'event and CLK = '1') then
      if (DurationCounter = "11110") then
        if (LastSwState = '1') then
          RiseEdge <= '1';
          FallEdge <= '0';
        else
          RiseEdge <= '0';
          FallEdge <= '1';
        end if;
      else
        RiseEdge <= '0';
        FallEdge <= '0';
      end if;
    end if;
  end process;
end Behavioral;
```

図 3. VHDL で書いたチャタリング対策回路の RTL

簡単に動作説明

LastSwState と CurrentSwState は 1 クロックごとに読んだ、入力ポートの状態履歴です。これを赤字で示した部分のように xor すると、同じ状態（チャタっていない）であれば結果は false (0) になり、異なっている状態（チャタっている）であれば結果は true (1)になります。

チャタっている状態を検出したらカウンタ（DurationCounter）をクリアし、継続しているのであればカウントを継続します。このカウンタは最大値で停止します。

その最大値ひとつ前のカウント値になるときに LastSwState が 0 であるか 1 であるかにより、スイッチが押された状態が検出されたか、スイッチから手を離した状態が検出されたかを判断し、それにより RiseEdge, FallEdge をアサートします。なお本質論とすれば、スイッチの状態と RiseEdge, FallEdge のどちらがアサートされるかについては、スイッチ回路の設計に依存しますが…。

メタステーブル（準安定）はデジタル回路でのアナログ的ふるまいだ！

やれやれ出来たぞ、と思っていたところ、「この前段に気分的にメタ・ステーブル殺しを入れたくなりますね」というコメントをいただきました。

もともと FPGA/RTL からのスタートでしたので「技術ノートにするにも、デジタル・ネタだなあ」と後悔ぎみでありましたが、「メタ殺し」という、デジタル回路に生じるアナログ的現象である「メタ・ステーブル」についてのコメントでありました。こういう進展するとは思いもせず、ただただ嬉しく思いました。

メタステーブル（準安定）はセットアップ・ホールドが満足できないために生じる、中間レベルが「ある時間」維持される現象です。調べてみると wikipedia には記述が無いようです。

自分も図 3 のソースでは（これをミーリ・マシン／ムーア・マシンと言っていいのか判りませんが）入力をワン・ライン（SwitchIn）のみとしており、複数の F/F で準安定による誤動作が生じないように

```
CurrentSwState <= SwitchIn;
```

として F/F でクロッキングしてあります。この SwitchIn が外部からのポートです。CurrentSwState でメタ・ステーブル（準安定）が発生することがありますが、システム・クロックも低速ゆえ、時間が経ってメタ・ステーブルが消滅する可能性が高いことから、1 段いれておけば大丈夫だろう、という考え方で「メタ殺し」設計してあります。

中堅は C 言語（マイコンでのソフトウェア）

余談ですが先日、立川市のお客様のところにお邪魔しました。車両での移動でしたが、立川飛行場跡地前を通りすぎました。

私は大学生だったころ、それは昭和 60 年の夏、この「タチヒ（立飛）地区」にあった会社にソフトウェア・エンジニア（Z80 のアセンブラ）としてバイトに来ていたのでした。たしか AD コンバータの値を Z80 マイコンで取り込み、それを PC9801 と I/F し（RS232C だったか？今は記憶が無いのですが 8251 などの SCI は使わなかったはず…。PC98 BUS だったかも？）、さらにそれを N88 BASIC で画面表示させ、HP-GL でプロッタにプロットするというものでした。当然デバッガなども無く、いきなりオブジェクトを EPROM に焼いて確認という開発スタイルでした。

それは大学 4 年生として最後の夏休みの 1.5 か月程度のバイトでした。昼休み時間には青い空の下で、若手社員さんから仕事の大変さについて教わっていたものでした…。

今回そのお客様訪問後に、このことを思い出し、ネットでサーチしてみると（会社名さえ忘れかけていました）、今は違うところで会社を営業されていることを見つけ、私の設計したソフトが応用されている装置も「Web 歴史展示館」上に展示されているものを見つけることができました（感動の涙）。

それではここでも本題に…

またまた閑話休題ということで…。図 4 はマイコンを利用した回路基板です。これらの設定スイッチが正しく動くように C 言語でチャタリング防止機能を書きました。これも一応これで問題なく動いています。

ソースコードを図 5 に示します。こちらもチャタリング対策のアプローチとしても、多岐の方法論があろうかと思いますが、一例としてご覧ください（汗）。

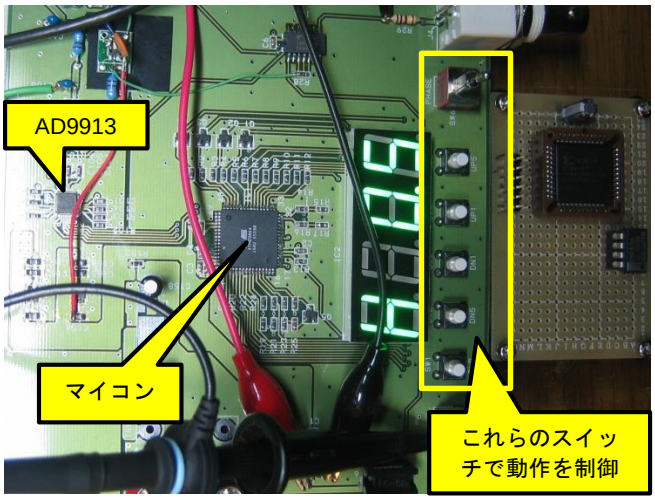

図 4. こんなマイコン回路基板のスイッチのチャタリング防止を C 言語でやってみた

```
// 5 switches from PE2 to PE6
swithchstate = (PINE & 0x7c);

// wait for starting switch
if (switchcount < 1000) {
  if (swithchstate == 0x7c) {
  // switch not pressed
    switchcount = 0;
    lastswithchstate = swithchstate;
  }
  else if (swithchstate != lastswithchstate) {
    switchcount = 0;
    lastswithchstate = swithchstate;
  }
  else {
    // same key is being pressed
    switchcount++;
  }
}
// Perform requested operation
if (switchcount == 1000) {
  ※ ここで「スイッチが規定状態に達した」として、目的の
  動作をさせる処理を追加 ※
  // wait for ending of switch press
  switchcount = 0;
  while (switchcount < 1000) {
    if ((PINE & 0x7c) != 0x7c) {
      // some of switches are still pressed
      switchcount = 0;
    }
    else {
      switchcount++;
    }
}
// return
switchcount = 0;
```

図 5. C 言語で書いたチャタリング対策ソース

簡単に動作説明

良く見てみると、この図 5 の C コードも図 3 の RTL コードと考え方はほとんど同じです。この場合は 5 つのスイッチが Port E の B2～B6 に接続されており、全てのスイッチはプルアップされています。スイッチが押されると各ビットが 0 になります。スイッチが押されていない状態であれば、

```
swithchstate == 0x7c
```

が true となり、図 3 の RTL では DurationCounter となっている switchcount（int です）がゼロになります。チャタリングが起きればこれがゼロに戻りますので、チャタリングによる誤動作を防止できます。

どこかのスイッチを押していると switchcount がメイン・ループ 1 周ごとにカウントアップしていきます。switchcount は（10 進で）1000 になるまでカウントアップし、1000 になったときに目的のスイッチ動作を行います。

これは簡易な動作用のソースですが、「複数のスイッチが同時に押されたときにはどう動くのだ？」というソフトウェア設計レビュー・ミーティング的な突っ込みにも else if のあとで対応しています、いや、いるつもりです（笑）。

大将はやっぱり CR 回路（アナログ回路）

図 6 も複数の設定スイッチがあるデジタル回路基板です。このデジタル回路基板は FPGA もマイコンもない基板で、HCMOS でアクロバティックなシーケンサ回路を実現している（という自己満足？的な）基板です。HCMOS が多数見えるところかと思います…。

ということでデジタル信号処理やソフトウェア的なチャタリング対策が施せませんので、ここでは図 7 のような回路にして、CR 回路とシュミット・トリガ・インバータ 74HC14 を用いたかたちで実現してみました。

チャタリング取りは CR 回路により実現します。時定数は後でも詳しく説明しますが、約 0.1sec です。この時定数で波形が大きく鈍りますので、それを安定に検出するためにシュミット・トリガ・インバータ 74HC14 を用いています。

74HC16x のカウンタは同期回路の神髄が詰まったもの

この回路でスイッチを押すと、74HC16x のカウンタを使った自己満足的なシーケンサ回路が動作し、デジタル信号波形のタイミングが変化していきます。波形をオシロで観測しながらスイッチを押していくと、波形のタイミングがきちんとずれていくようすを確認することができました。

74HC16x とシーケンサと聞いてピーンと来たという方は、「いぶし銀のデジタル回路設計者」の方と拝察いたします。74HC16x は、同期シーケンサの基礎技術がスマートに、煮詰まったかたちで詰め込まれ、応用されている HCMOS IC なのであります。動作を解説するだけでも同期回路の神髄に触れることもできると思いますし（半日説明できるかも）、いろいろなシーケンス回路も実現できます。

不適切だったことは後から気が付く！

「やれやれ出来たぞ」というところでしたが、基板が完成して数か月してから気が付きました。使用したチャタリング防止用コンデンサは 1uF ということで容量が大きめでありますが、電源が入ってスイッチがオフである「チャージ状態」では、コンデンサ（図 7 では C15/C16）は 5V になっています。これで電源スイッチを切ると 74HC14 の電源電圧が低下し、IC の入力端子より「チャージ状態」の C15/C16 の電圧が高くなってしまいます。ここから IC 内部のダイオードを通して入力端子に電流が流れてしまい、IC が劣化するとか、最悪ラッチアップが生じてしまう危険性があります。

ということで、本来であればこの C15/C16 と 74HC14 の入力端子間には 1kΩ程度で電流制限抵抗をつけておくべきでありました…（汗）。この基板は枚数も大量に作るものではなかったので、このままにしておきましたが…。

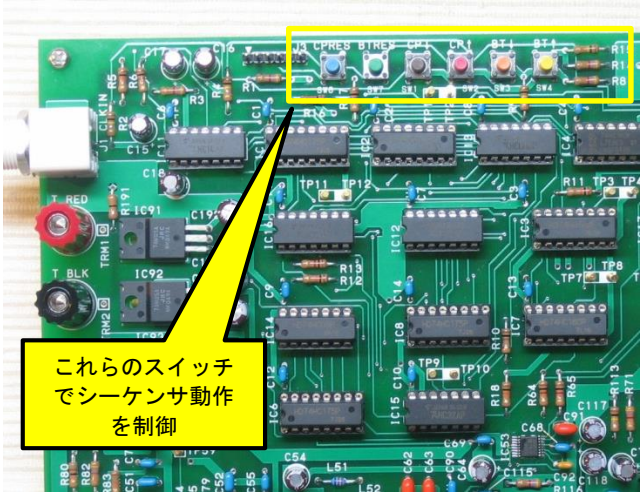

図 6. 複数の設定スイッチのある回路基板のチャタリング防止を CR 回路でやってみた

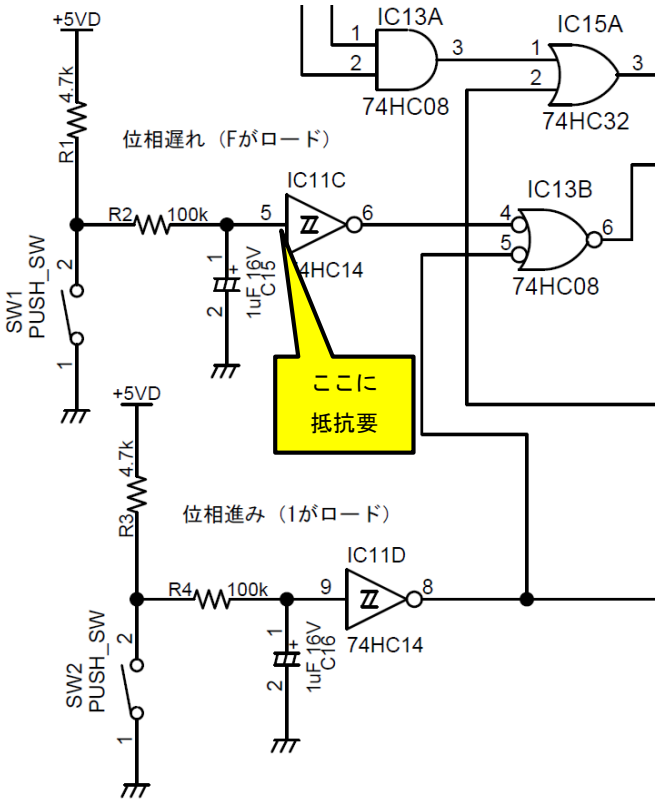

図 7. 図 6 の基板の CR 回路によるチャタリング防止
（気づくのが遅かったが C15/C16 と 74HC14 の間には
ラッチアップ防止の抵抗を直列に入れるべきであった！）

回路の動作をオシロスコープで一応確認してみる

図 7 の回路では 100kΩ（R2/R4）と 1uF（C15/C16）が支配的な時定数要因になっています。スイッチがオンしてコンデンサから電流が流れ出る（放電）ときは、時定数は 100kΩ×1uF になります。スイッチが開放されてコンデンサに電流が充電するときは、時定数は（100kΩ＋4.7kΩ）×1uF になりますが、ほぼ放電時の時定数と同じと考えることができます。

図 8 にスイッチが押されたときの 74HC14 の入力端子（コンデンサの放電波形）と同出力端子（シュミット・トリガでヒステリシスを持ったかたちで L から H になる）の波形のようすを示します。

また図 9 にスイッチが開放されたときの 74HC14 の入力端子（コンデンサの再充電波形）と同出力端子（シュミット・トリガでヒステリシスを持ったかたちで H から L になる）の波形のようすを示します。このときは時定数としては（100kΩ ＋ 4.7kΩ）×1uf ということで、先に示したとおりですが、4.7%の違いなのでほぼ判別することはできません。

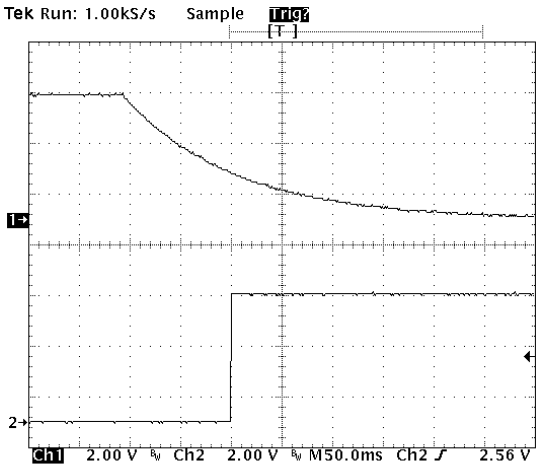

図 8. 図 6 の基板でスイッチを押したときの CR 回路の
放電のようすと 74HC14 出力（時定数は 100kΩ×1uF になる。
横軸は 50ms/DIV）

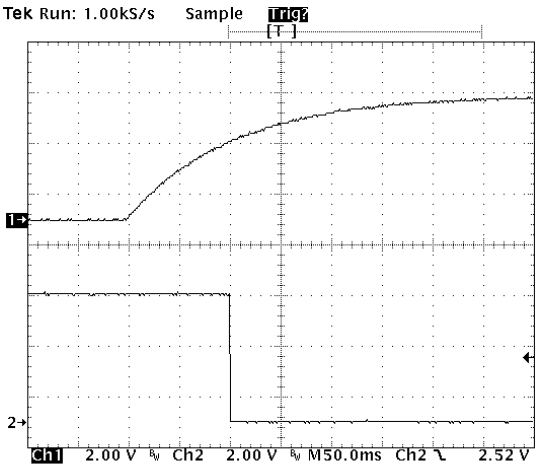

図 9. 図 6 の基板でスイッチを開放したときの CR 回路の
充電のようすと 74HC14 出力（時定数は 104.7kΩ×1uF に
なるが 4.7%の違いなのでほぼ判別できない。横軸は 50ms/DIV）