ANALOG DEVICES

TNJ-016
アナログ電子回路技術ノート

10 ビットのプチ DAC を RTL で動かしてみる（おまけソースつき）

著者: 石井 聡

はじめに

このところ「デジタル・コンサート・ホール」というもので楽しみ始めました（音だけで楽しんでいます）。チケット購入は週間視聴コースからで、PayPal でも決済できます。

http://www.digitalconcerthall.com/

さて、AD5611 という 10bit DAC があります。小型の 6 ピンのパッケージで「nanoDAC」と呼ばれていますが、「プチ DAC」という感じです。

AD5611: 10 ビット D/A コンバータ、nanoDAC(TM)、2.7～5.5V で 100μA 未満

http://www.analog.com/jp/ad5611

謎の一石二鳥をめざし（笑）、これを FPGA から駆動させてみます。この技術ノートの後半にも、その RTL も公開します。図 1, 2 は SC70 パッケージの AD5611AKSZ です。パッケージが SC70 というのも「プチ DAC」のとおり、小さくて素敵ですね。

図 1. 10 ビット nanoDAC AD5611AKSZ
（入手した状態。静電バッグに入っている）

図 2. 10 ビット nanoDAC AD5611AKSZ
（静電バッグから取り出したリール）

やはり遊ぶには（ホントは）オーディオ信号か？

知り合いの方からもこんなコメントもいただきました。「44.1kHz/16bit のオーディオ信号を 25 倍くらいにオーバ・サンプリングして、簡単に 2 次くらいでΣΔ変調して、この DAC に突っ込んであげるのも楽しそうですね」。

私も「デジタル・コンサート・ホール」を聞いたりする都合（？）上、HD (High Definition)なディジタル・オーディオの信号再生にもこだわりたいところです。

その点からすれば、この方のアイディアはなかなか面白いです！私は絶対に気がつかないところでした。たしかに AD5611AKSZ でもレートも十分間に合います。AD5611AKSZ では SCLK = 30MHz まで行けますので、たとえば 18 SCLK で 1 SEQ（1DAC コード）を組むとすると、1.666Msps ですから、サンプリング 44kHz とすれば 37 倍程度までのオーバ・サンプリングまでが実現できますね。

とか何とか言っても、現実には時間的余裕もないことから、この AD5611AKSZ を用いて HD な再生システムなど作り込むなんて夢の夢ではありますが…。

なお、この DAC ファミリは分解能が複数あって、

AD5601	8 ビット
AD5611	10 ビット（今回使うもの）
AD5621	12 ビット
AD5641	14 ビット

となっています。AD5641 以外はデータシートが共用です。

ΣΔ変調をかけると SN 比が向上する

ここでこの方から頂いた「簡単に 2 次くらいでΣΔ変調して」という話題をアナログ・デバイセズの技術資料をネタに少しご紹介してみます。図 3、図 4 は参考文献[1]「The Data Conversion Handbook」の Chapter 3「Data Converter Architectures」から抜粋

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

したものです（CQ 出版社からも「A-D/D-A 変換 IC の実用技術」として翻訳本が発売されています）。図 3 はナイキスト・サンプリングと、オーバ・サンプリングと、ΣΔ変調を用いた場合のそれぞれの AD 変換（これは AD 変換です）結果として得られるノイズのようすを示しています。ΣΔ変調により、ノイズ成分が高周波帯にシェーピングされる（押し出される）ことにより、不要な部分「REMOVED NOISE」とあるところをフィルタリングすれば高い SN 比が実現できるというものです。

また図 4 はΣΔ変調の次数と、それぞれのオーバ・サンプリング比により実現できる SN 比の改善率（dB）を表しています。この方のおっしゃる「2 次で」そして「AD5611AKSZ で SCLK = 30MHz、44kHz レートなら 37 倍程度までのオーバ・サンプリング」というところを見てみると、60dB の SN 比の改善ができそうです。1 ビットは約 6dB なので 10 ビット相当になります。AD5611AKSZ は 10 ビットの DAC なので、20 ビット相当の DAC の性能が実現できることになりそうです。

14 ビット DAC の AD5641 を使えば、24 ビット相当が実現できそうなわけですね！

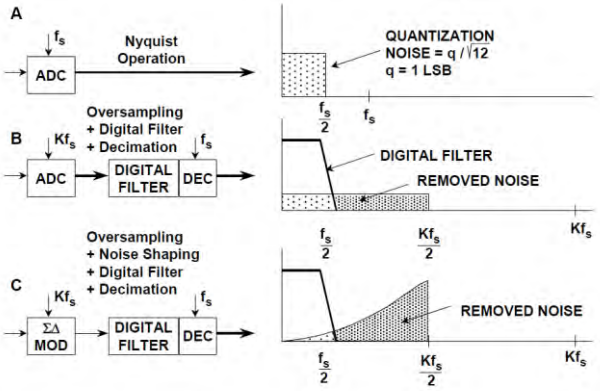

Figure 3.125: *Oversampling, Digital Filtering, Noise Shaping, and Decimation*

図 3. ADC アーキテクチャごとによる SN 比改善のようす。一番下がΣΔ変調を用いたもの。ADI 技術資料「The Data Conversion Handbook」Fig. 3.125 から抜粋

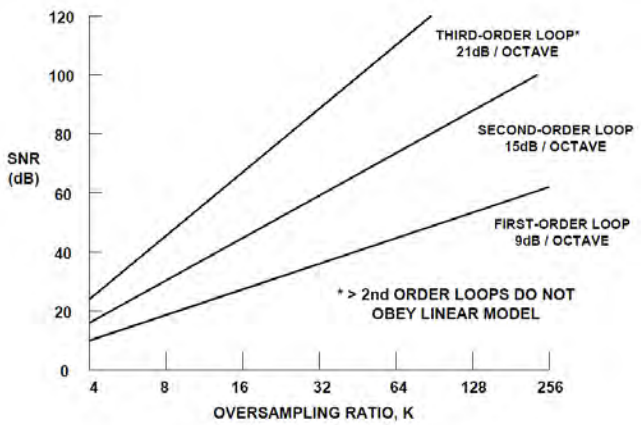

Figure 3.131: *SNR Versus Oversampling Ratio for First, Second, and Third-Order Loops*

図 4. ΣΔ変調の次数と、それぞれのオーバ・サンプリング比により実現できる SN 比の向上率。ADI 技術資料 ADI 技術資料「The Data Conversion Handbook」Fig. 3.131 から抜粋

はんだ付けしてみたようす

ということで実装のようすをお見せします。SC70 ということで、ピンピッチが 0.65mm なのですが、図 5 に示す秋月電子の SOT23 の変換基板に図 6 のように実装してみました。見ていただいて分かるようにピンピッチ（SC70 = 0.65mm）と基板のパッド（SOT23 = 0.95mm）のサイズが合わないのではんだ付けは慎重に行いました。

ピッチが合わないとしても、片側 3 ピンずつだから出来る荒業かもしれません（笑）。

はんだ付けといえば、0.5mm の 400 ピンクラスの FPGA を手はんだではんだ付けしたことがあります。最初は失敗しましたが、コツ（いろいろあるのですが）をつかめばキレイに手はんだできるようになりました。最初は高価な IC を複数個「お釈迦」にしてしまった覚えがあります。0.5mm ピッチの長いリードが、複数、曲がって、グチャグチャに、ショートしていく、そのようすは、涙なしには語れません…。

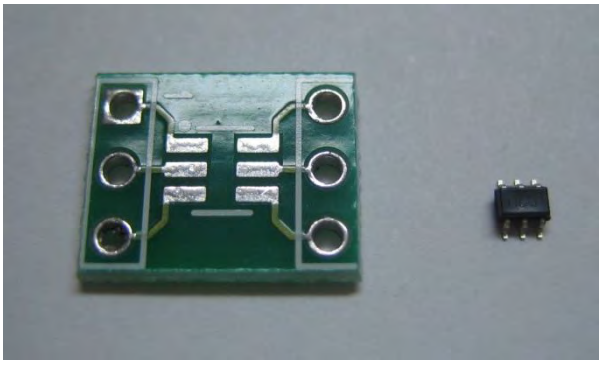

図 5. nanoDAC AD5611AKSZ（SC70 パッケージ 0.65mm ピッチ）と秋月電子の SOT23 変換基板（0.95mm ピッチ）

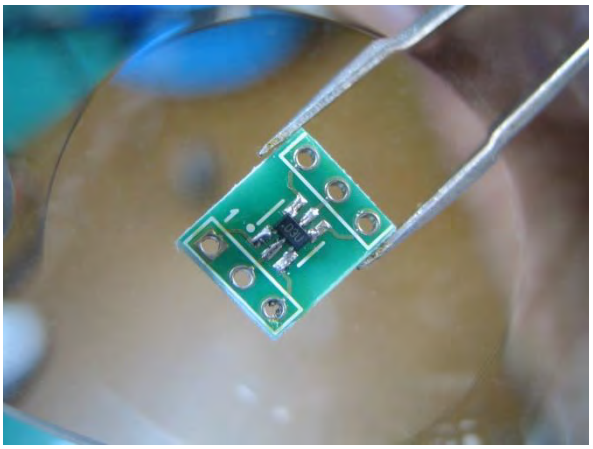

図 6. nanoDAC AD5611AKSZ を SOT23 変換基板に実装したようす

FPGA は書籍の付録で

AD5611AKSZ にデータを送り込む FPGA についてご紹介します。図 7 は CQ 出版の書籍の付録としてついていた、XC3S100E を使ったボードです。これは図 8 に示す本で、Verilog の本なのですが、私は Verilog が書けないので VHDL で使っています（汗）。

動作させたい GCLK ピンに CLK を接続するために、33MHz の XTAL からワイヤリングしてあります。

SPI の SCLK（16.5MHz）の信号は、ピンヘッダ端子から 1kΩを経由させて AD5611AKSZ にデータを与えてみました。大丈夫かと思ったところ、波形が「なまりすぎ」でしたので、330Ωに変更して、図 9 のオシロの波形となりました。

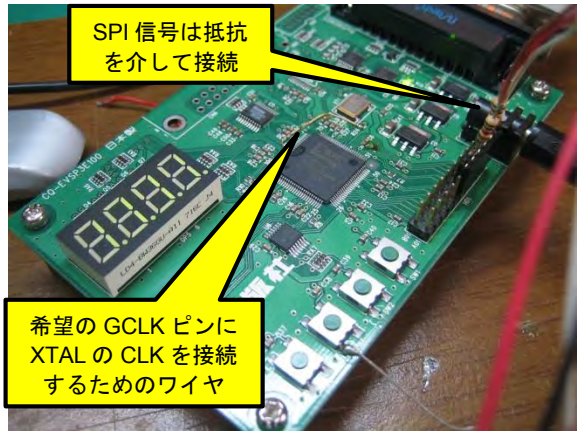

図 7. AD5611AKSZ にデータを送り込む FPGA には書籍の付録を利用

図 8. FPGA ボードがついていた書籍。この書籍自体は Verilog での記述だが Verilog が書けないので VHDL でコーディング

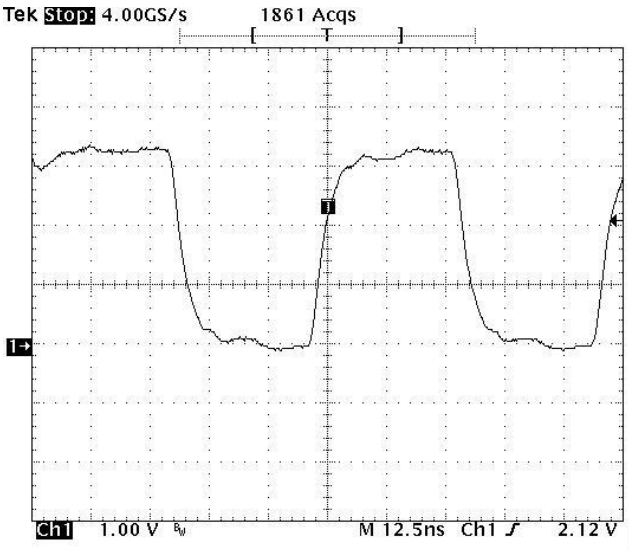

図 9. FPGA ボードからの 16.5MHz の SCLK 信号
330Ω で信号なまりを取りつつ波形を安定化

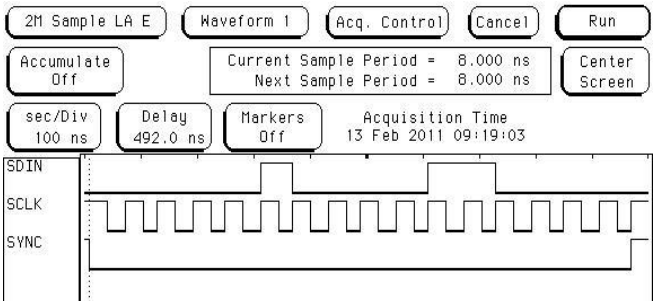

図 10. SPI の 1 シーケンス（SYNC が L になっている）

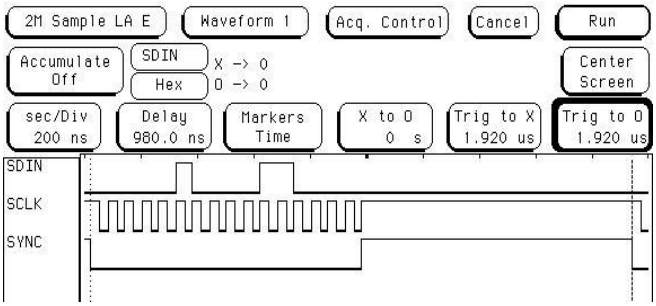

図 11. SPI のシーケンス間（SYNC の立下りが 2 箇所みえる）

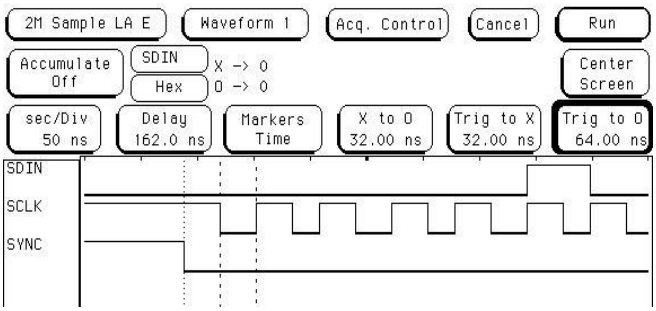

図 12. 1 シーケンスのスタート（SYNC の立下りがみえる）

シリアル波形がでた！でた！

実際に FPGA にプログラミングしてみて、SPI のシリアル波形をロジアナで観測してみました。図 10 は SPI の 1 シーケンスのようすです。1 フレームを表す SYNC 信号が L になっているのが見えます。図 11 はシーケンス間のようすを観測してみたものです。SYNC の立下りが 2 箇所みえます。1 フレーム全長と次のフレームのスタートというところです。図 12 は 1 シーケンスのスタートのようすです。SYNC の立下りがみえています。

動きそうな動作を一応しています。

プチ DAC を実装したようす

AD5611AKSZ を実装したようすをお見せします。図 13 に示すような、手もちの DIP 用実験プラットフォーム（？）のアキのところにグラウンドがきちんと取れるようにはんだ付けしてあります。手前は SCLK などのデジタル入力の接続です。さきの説明のように FPGA とは 330Ω で分離しています。

AD5611AKSZ は電源電圧を 1/1024 に分割する DAC で（つまりリファレンス電圧が無い）、精度を出そうとすれば「電源電圧の精度」を高めることが基本になります。ここでは高精度 LDO である ADP3301 を使ってみました（図 14）。この LDO は電圧精度が室温で±0.8%というもので、結構な特性が得られる優れモノです。

ADP3301: リニア・レギュレータ、100mA の低ドロップアウト、高精度、anyCAP(R)

http://www.analog.com/jp/ADP3301

三角波信号波形がでた！でた！

最終的にできた三角波信号の波形を図 15 に示します。1 変換サイクルで 1LSB インクリメント（デクリメント）しているので、周期は 2048 / *fs*（*fs* = 515.6kHz）となり結構低めです。

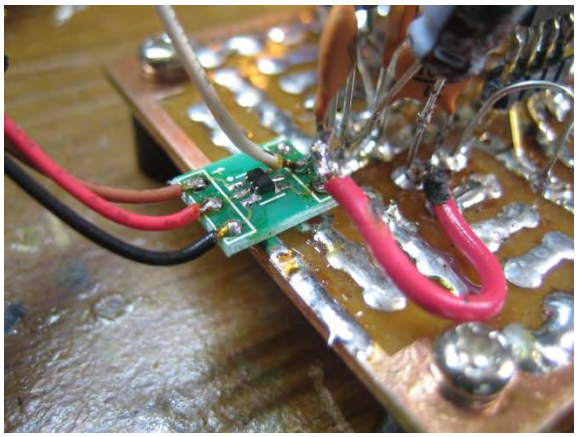

図 13. DIP 用実験プラットフォームと言えば聞こえがいいが、実験基板のアキのところ実装したようす

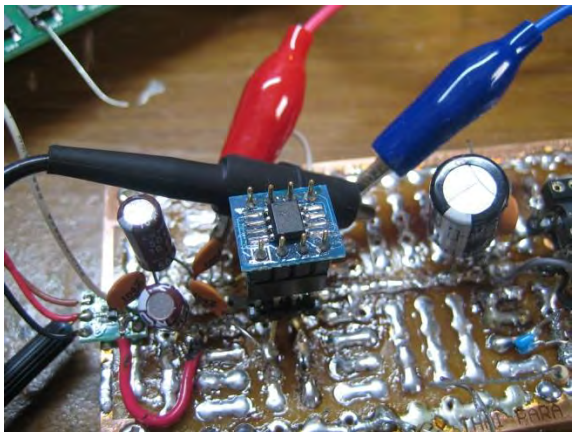

図 14. 高精度 LDO ADP3301 を DAC 電源（リファレンス電圧）として利用し精度を高める

RTL ソースのご紹介（三角波を発生）

以下は「おまけソースつき」のソースです（駄菓子のおまけみたいですが）。VHDL で書いてあります。図 16 のソースコードは図 10～図 12 で示した AD5611AKSZ への SPI 通信を制御するモジュールのソースコードです。

図 17 はその上位モジュールとして、三角波波形生成と CONVSTART 信号を生成するモジュールのソースコードです。ここのコード部分は上位との接続用としての entity 記述や archtecture 記述は省略してあります。謎の「一石二鳥」を目指しているために、2 つの AD5611 を設定できるようになっています。それから制御ビット 2 ビットは"00"固定にしてあります。

ステート・マシン設計が大好きなので、どうでもいいようなところをステート・マシンで作っています（汗）。

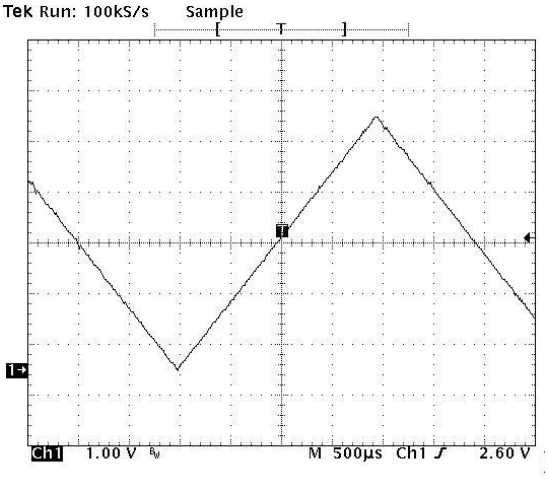

図 15. AD5611AKSZ 出力で得られた三角波信号

おまけのおまけ

これまでのところで、この技術ノートも終わりにしようと思っていましたが、このシリーズの特設ページがありましたのでご紹介しておきます。

nanoDAC ファミリの拡張

http://www.analog.com/jp/products/landing-pages/001/cu_over_analog_devices_introduces_nanodac_family.html

参考文献

[1] アナログ・デバイセズ技術資料; The Data Conversion Handbook, http://www.analog.com/library/analogDialogue/archives/39-06/data_conversion_handbook.html

```
-------------------------
-- AD5611 serial control
-------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- This module dedicates for two DACs
entity CtrlAD5611 is
  port(RESET : in std_logic;
    CLK : in std_logic;
    CONVST : in std_logic;
    DAData1 : in std_logic_vector (9 downto 0);
    DAData2 : in std_logic_vector (9 downto 0);
    SYNC_low : out std_logic;
    SCLK : out std_logic;
    SDIN1 : out std_logic;
    SDIN2 : out std_logic
  );
end CtrlAD5611;
architecture Behavioral of CtrlAD5611 is
type LOAD_STATE is (Idle, Loading);
signal ShiftReg1, ShiftReg2 : std_logic_vector (15 downto 0);
signal SclkTimer : std_logic_vector (3 downto 0);
signal SclkToggle : std_logic;
signal STATEREG, NEXTSTATE : LOAD_STATE;

begin

SCLK <= SclkToggle;
SDIN1 <= ShiftReg1(15);
SDIN2 <= ShiftReg2(15);

  process(CLK, RESET) begin
    if (RESET = '1') then
       STATEREG <= Idle;
    elsif (CLK'event and CLK='1') then
       STATEREG <= NEXTSTATE;
    end if;
  end process;

  process(STATEREG, CONVST, SclkTimer, SclkToggle)
begin
    case STATEREG is
      when Idle =>
        if (CONVST = '1') then
          NEXTSTATE <= Loading;
        else
          NEXTSTATE <= Idle;
        end if;
      when Loading =>
        if (SclkTimer = "1111" and SclkToggle = '0')
then
          NEXTSTATE <= Idle;
        else
          NEXTSTATE <= Loading;
        end if;
      when others =>
        NEXTSTATE <= Idle;
    end case;
  end process;

    -- Set chip select pin
  process(CLK, RESET) begin
    if (RESET = '1') then
      SYNC_low <= '1';
    elsif (CLK'event and CLK='1') then
      if (NEXTSTATE = Loading) then
        SYNC_low <= '0';
      else
        SYNC_low <= '1';
      end if;
    end if;
  end process;

    -- Set SCLK internal signal w/16 times toggle
  process(CLK, RESET) begin
    if (RESET = '1') then
      SclkToggle <= '1';
    elsif (CLK'event and CLK='1') then
      if (STATEREG = Loading) then
        SclkToggle <= not SclkToggle;
      else
        SclkToggle <= '1';
      end if;
    end if;
  end process;

    -- SCLK toggle timer
  process(CLK, RESET) begin
    if (RESET = '1') then
      SclkTimer <= "0000";
    elsif (CLK'event and CLK='1') then
      if (STATEREG = Loading and SclkToggle = '0')
then
        SclkTimer <= SclkTimer + 1;
      end if;
    end if;
  end process;
```

```
    -- Data load registers
  process(CLK, RESET) begin
    if (RESET = '1') then
      ShiftReg1 <= x"0000";
      ShiftReg2 <= x"0000";
    elsif (CLK'event and CLK='1') then
      if (CONVST = '1') then
        ShiftReg1 <= "00" & DAData1 & "0000";
        ShiftReg2 <= "00" & DAData2 & "0000";
      elsif (STATEREG = Loading and SclkToggle = '0')
then
        ShiftReg1 <= ShiftReg1(14 downto 0) & "0";
        ShiftReg2 <= ShiftReg2(14 downto 0) & "0";
      end if;
    end if;
  end process;

end Behavioral;
```

図 16. AD5611AKSZ への SPI を制御するモジュールの RTL

```
-------------------------
-- triangle waveform generation
-------------------------
process(CLK, RESET) begin
  if (RESET = '1') then
    DACtimer <= "000000";
    CONVST <= '0';
  elsif (CLK'event and CLK='1') then
    DACtimer <= DACtimer + 1;
    if (DACtimer = "111111") then
      CONVST <= '1';
    else
      CONVST <= '0';
    end if;
  end if;
end process;

-- For triangle waveform
process(CLK, RESET) begin
  if (RESET = '1') then
    Result1 <= "0000000000";
  elsif (CLK'event and CLK='1') then
    if (CONVST = '1') then
      if (UpDirection = '1') then
        Result1 <= Result1 + 1;
      else
        Result1 <= Result1 - 1;
      end if;
    end if;
  end if;
end process;

-- For triangle waveform
process(CLK, RESET) begin
  if (RESET = '1') then
    UpDirection <= '1';
  elsif (CLK'event and CLK='1') then
    if (UpDirection = '1' and Result1 = "1111111110")
then
      UpDirection <= '0';
    elsif (UpDirection = '0' and Result1 =
"0000000001") then
      UpDirection <= '1';
    end if;
  end if;
end process;
```

図 17. 三角波を発生させる RTL（一部）