USB データ転送レートの向上対策

～ Smart-USB Plus 製品ファミリへの適用 ～

【概要】

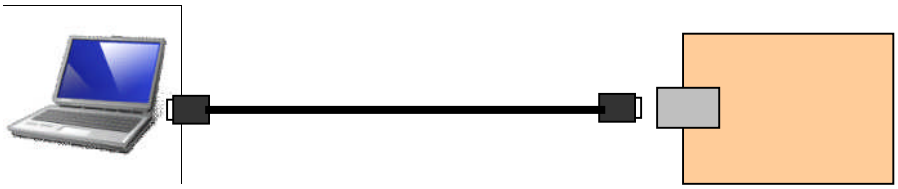
USB2.0 バス仕様「Ch. 7.1.9 Bit Stuffing」によれば、USB バス上のシリアルデータ列に“1”が6個連続した場合、データの受信側では、“0”を 1 ビット追加する仕様になっています。このビットスタッフ機能により、“1”が連続するようなシリアルデータでも、正しくデータ通信が行われます。 通信を正しく行うための機能ですが、不要なビットが自動的に追加されるので、大容量データで“1”が連続するような場合、ビットスタッフ機能によりデータ容量が増え、データの実効転送レートが低下します。 転送したいデータに“1”が連続しないようにすれば、自動的に処理されるビットスタッフを防ぎ、転送レートを向上させることができます。

ここでは、USB バス上のシリアルデータで “1” が連続しないように FPGA 内の制御回路でデータをスクランブル化させ、実効データ転送レートを少しでも向上させるための方法を紹介します。

【スクランブル化の概要】

Smart-USB Plus 製品の各ボード側で、FPGA によりデータのスクランブル／デスクランブル化を行い、ボードを制御するソフトウエアでデスクランブル／スクランブル化を行います。逆の場合も同様です。

ボードの FPGA 側だけスクランブル化しても、対応する制御ソフトウエアがデスクランブル機能を有効にしていないと、誤ったデータを受信することになります。同様に、制御ソフトウエアがスクランブル機能を有効にしても、ボード側でデスクランブル化を有効にしなければ、正しいデータを受信できません。



ボードにデータを送信時、
制御ソフトウエアで**スクランブル化** ⟶ ボード側の FPGA により**デスクランブル化**

ボードからデータを受信時、
制御ソフトウエアにより**デスクランブル化** ⟵ ボード側の FPGA により**スクランブル化**

＜図 1. スクランブル／デスクランブル化の概要＞

【FPGA 回路によるスクランブル化】

<外部メモリ(SSRAM)を搭載していないボードの場合>

表 1 に示す HDL コードは、製品に添付するサンプル回路にスクランブル／デスクランブル化の機能を追加する内容です。FPGA の内部メモリは容量が小さいので、スクランブル化による恩恵は受けませんが、外部システムと接続したときに、内部メモリを FIFO 化して大容量データを取り扱うときに参考にしてください。

GPIF_Verilog/GPIF_VHDL サンプル回路での変更点:

[変更するファイル]—— GPIF_Mem.v または GPIF_Mem.vhd

```
//      Verilog-HDL@CX-Card2（アルテラ FPGA）の変更例
//      scramble ノードはスクランブルを行う／行わないの切り替え用です。

        M4K64  m4k_ram      (.clock( clk ),
                            .data( ram_d ),
                            .wren( write_enable ),
                            .address( mem_adrs ),
                            .q( ram_q )
                            );

        assign  up_data     =       ( scramble ) ? ram_q ^ 16'hAAAA : ram_q;
        assign  ram_d       =       ( scramble ) ? data ^ 16'hAAAA : data;
```

<表1. GPIF_Verilog サンプル回路の修正内容>

<外部メモリ（SSRAM）を搭載しているボードの場合>

表2に示す HDL コードは、製品に添付するサンプル回路にスクランブル／デスクランブル化の機能を追加する内容です。SSRAM を搭載していないボードでも、外部システムと接続するときの回路例として参考にしてください。

GPIF_Verilog/GPIF_VHDL サンプル回路での変更点：
［変更するファイル］―― GPIF_Mem.v または GPIF_Mem.vhd

＊メモリリード部分の変更箇所

外部メモリから読み出したデータを 2 段バッファする「read_dt_buf0－read_dt_buf1」間で処理します。

read_dt_buf0 は、I/O レジスタ・セルを使用しています。read_dt_buf1 では、セル内の LUT をバイパスして同じセル内のレジスタを使用しているので、FPGA リソース消費量にあまり影響を与えません。

```
//      Verilog-HDL の変更例
//      scramble ノードはスクランブルを行う／行わないの切り替え用です。
        if( scramble )      begin
        read_dt_buf1        <=      read_dt_buf0 ^ 32'hAAAAAAAA;
        end
        else  begin
        read_dt_buf1        <=      read_dt_buf0;
        end
```

＜表2. 1 GPIF_Verilog サンプル回路（メモリ・リード部）の修正内容＞

＊メモリライト部分の変更箇所

USB 制御 IC（FX2）が出力する 16 ビットのデータ（FD）を FPGA 内部の 32b ノード “pbus”に組み上げます。
pbus から外部メモリに出力する為の出力バッファ“write_dt_buf”に遷移する箇所で行うと FPGA リソース消費が少なく、有効です。

```
//      Verilog-HDL の変更例
//      scramble ノードはスクランブルを行う／行わないの切り替え用です。
        if( scramble )      begin
                            write_dt_buf      <=      pbus ^ 32'hAAAAAAAA;
                end
                else begin
                            write_dt_buf      <=      pbus;
                end
```

<表２．１　GPIF_Verilog サンプル回路（メモリ・ライト部）の修正内容>

【制御ソフトによるスクランブル化】

スクランブル化に対応する制御アプリケーション・ソフトウエアは RefApp2.exe と RefApp3.exe(注)です。
以下、RefApp2.exe を例として開設します。 RefApp2 制御アプリの「メモリ操作」タブで、ツールバーの「メニュー」をプルダウンして、メモリリードの場合、メモリライトの場合、両方の場合を選択することができます。
「オプション」で設定した後、この他の操作は従来のメモリ転送操作と全く同じです。

（注）RefApp3.exe は、USB Plus 開発キットを適用した場合の制御ソフトウエアです。USB Plus 開発キットは、製品購入後にユーザ登録をすると、ダウンロード可能になります。