AC97 サウンド入出力およびグラフィックス表示システムで必須の FIFO モジュール

Appendix 3

同期 FIFO モジュールと簡易非同期 FIFO モジュールの設計事例

山武 一朗

1 FIFO とは何か

● FIFO が必要なところ

FIFO（First-In First-Out）とは，先に書き込んだものを先に取り出すバッファ動作のことを言います．**図 1** に示すように，回路 1 で出力したデータを回路 2 に入力する場合を考えましょう．この二つの回路のタイミングが合えばデータの受け渡しに問題はありませんが，内部の動作の都合から，回路 1 がデータを出力するタイミングと，回路 2 がデータを入力するタイミングが合わない場合があります．この場合，回路 1 と回路 2 の間に FIFO を実装します．回路 1 が都合の良いタイミングで出力したデータを FIFO でいったん保持し，回路 2 が都合の良いタイミングでデータを入力できるようにします．回路 1 の出力を FIFO のライト・ポートに接続し，回路 2 の入力を FIFO のリード・ポートに接続します．

● 同期 FIFO と非同期 FIFO

図 2 に FIFO の基本構造を示します．FIFO の中のメモリも一般的な 1 次元配列メモリになっていて，ライト・ポートから書き込んだデータをどのアドレスに書き込むかを管理するライト・アドレス・カウンタと，読み出しデータのアドレスを管理するリード・アドレス・カウンタがあります．つまり FIFO の基本構造は，一般的なメモリと二つのアドレス・カウンタで成り立っています．

大きく分けると FIFO には，**図 3** のように回路 1 と回路 2 がともに同一クロックで動作する場合に使用する同期 FIFO と，**図 4** のように回路 1 と回路 2 が別クロックで動作する場合に使用する非同期 FIFO があります．つまり，ライト・アドレス・カウンタもリード・アドレス・カウンタも同一のクロックでカウントするものが同期 FIFO，ライト・アドレス・カウンタはライト・ポート側のクロックで，リード・アドレス・カウンタはリード・ポート側のクロックでカウントするものが非同期 FIFO というわけです．

● FIFO フロー制御信号

FIFO を使えば，回路 1 が自分に都合のいいタイミングでデータを出力できるといっても，回路 2 が FIFO からデータを取り出さないと，いずれ FIFO がいっぱいになってしまいます．これを検出するため，一般にライト・ポート側には FIFO フル信号が用意されています．回路 1 はこの信号によって FIFO に空きが

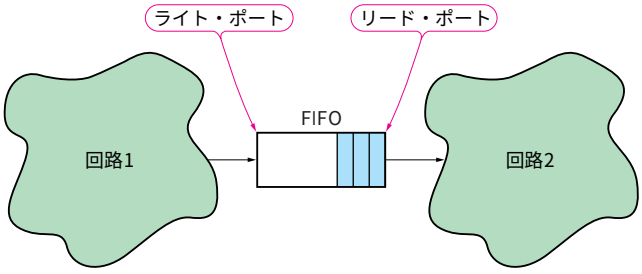

図 1　FIFO を使う場面

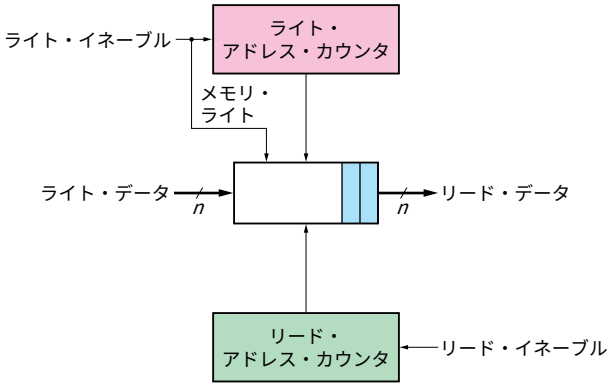

図 2　FIFO の基本構造

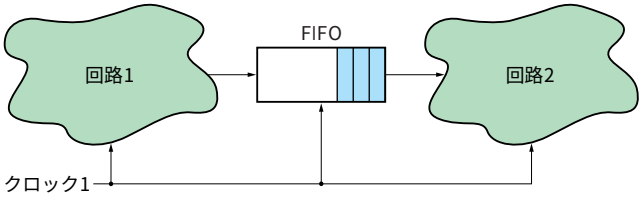

図 3　同期 FIFO

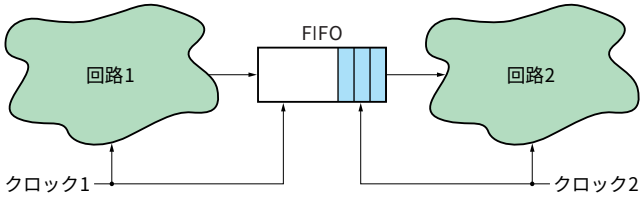

図 4　非同期 FIFO

KEYWORD——同期 FIFO，非同期 FIFO，フロー制御，FIFO フル，FIFO エンプティ，FIFO ハーフ・フル，FIFO ハーフ・エンプティ

あるかどうかを確認し，空きがあればデータを出力します．

逆に回路2では，FIFOにデータが用意されていないときに読み出し動作を行っても無意味です．そこで，FIFOにデータが書き込まれているかどうかを判定する信号が用意されています．1ワードでもデータが書き込まれればFIFOは空ではないことになるので，一般的なリード・ポート側にはFIFOが空きそうかどうかを示すFIFOエンプティ信号が用意されています．

● FPGAにおけるFIFO

ROMやSDRAMと同じように，FIFOも汎用メモリの一種として容量やアクセス速度，データ・バス幅などについていろいろな仕様のものが市販されています．

FPGAを使う場合，容量が数ワード程度であればメモリにはフリップフロップで構成した配列を使い，カウンタ回路の組み合わせでFIFOを構成します．容量が数百ワード以上であれば，メモリにはFPGA内蔵ブロックRAMを使うのが一般的です．さらにFIFOは汎用的な部品でもあるので，FPGA開発ツールの機能として，容量やデータ・バス幅，必要なフロー制御信号などの仕様を選択するだけで，FIFOモジュールを生成してくれる機能が用意されています(**図5**)．

しかし，「組み込みシステム開発評価キット」は，ハードウェアの基礎を理解するための教材でもあるので，FIFOモジュールを専用ユーティリティで自動生成してしまっては意味がありません．ここではFIFOを自前で設計してみることにしましょう．

2 同期FIFOの設計事例

● FIFOメモリにはFPGA内蔵ブロックRAMを使う

ここでは数Kバイトの容量をもつFIFOを実現してみます．このぐらいの容量になると，FIFOのメモリにはFPGA内蔵ブロックRAMを使う必要が出てきます．ここではFPGAとしてSpartan-3内蔵のブロックRAM機能(RAMB16_S36_S36)を使います．Spartan-3に内蔵されたブロックRAMは，完全なデュアルポート構造になっています．これによってライト・ポートとリード・ポートを完全に独立して実装できるため，FIFOを実現するには最適です．

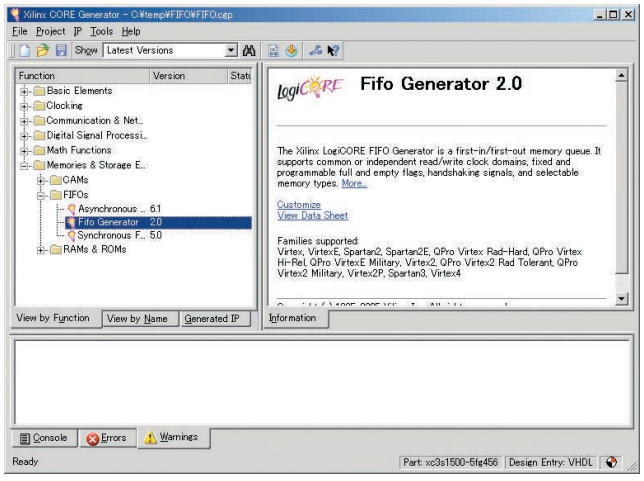

図5　モジュール自動生成ツールの例(Xilinx CORE Generator)

とはいえ，このブロックRAM機能をそのまま使ってしまうと，ほかのデバイスにFIFOモジュールを移植しにくくなります．そこで**リスト1**のように，一般的なデュアルポートRAMモジュールとしてモジュール名や入出力信号名を定義し，このモジュールの中でSpartan-3内蔵のブロックRAM機能を組み込んでいます．

なお，Spartan-3内蔵のブロックRAMにはパリティ用のビットも用意されていますが，ここではパリティを使用していません．

● デュアルポートRAMをFIFOメモリ化

リスト2に，設計した同期FIFOモジュールを示します．FIFOメモリとしては，**リスト1**のデュアルポートRAMモジュールを組み込んでいます．デュアルポートRAMモジュールはポートA/Bの両方に，リード/ライト・データ・バスやライト・イネーブル信号が用意されています．しかし，FIFOの場合，片方のポートは書き込み専用，もう片方のポートは読み出し専用です．従って，デュアルポートRAMのポートAは書き込み専用にしてリード・データ・バスはオープン状態にします．また，ポートBは読み出し専用にして，ライト・データ・バスはとりあえずオール0を出力し，ライト・イネーブルは常時0にします．

● 同期FIFOのアドレス・カウンタ

ライト・ポート側のアドレス・カウンタは，FIFOフル状態ではないときにライト・イネーブル信号が入力されたら，カウント・アップさせます．アドレス・カウンタは`std_logic_vector`型なので，カウントの上限に達すると次はゼロになります．そのため，カウンタの上限処理を記述していません．

同じようにリード・ポート側のアドレス・カウンタは，FIFOが空ではないときにリード・イネーブル信号が入力されたらカウント・アップさせます．こちらのアドレス上限の処理もライト・ポート側と同じです．

● FIFOフル/エンプティの生成

FIFOがいっぱいになったかどうかは，ライト・アドレス・カウンタがリード・アドレス・カウンタに追い付いたかどうかによって判定できます．また，FIFOが空かどうかも，リード・アドレス・カウンタがライト・アドレス・カウンタに追い付いたかどうかによって判定できます．しかしカウンタが追い付いたからといって，単純にカウンタを比較するだけで判定すると，前者なのか後者なのかを判定できません．

そこで，次のような判定ロジックを採用します．FIFOフルでない状態でライト・イネーブル信号が入力されたとき，現状のライト・アドレス・カウンタの値を+1した値がリード・アドレス・カウンタと同じ値なら，この書き込み完了後にFIFOフルに