第7章

野球ゲームの製作

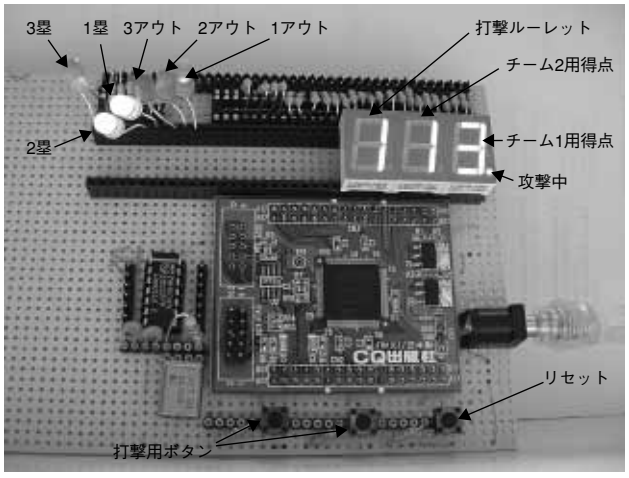
―― オリジナルCPUコアを実装して ソフトウェアで実現

山際伸一

ここでは，独自に設計したCPUコアとソフトウェアで実現したゲームの製作例を紹介する．シンプルなアーキテクチャにより最低限の機能だけを持たせることで，回路規模を抑えた．EP1C3の1/4以下の630ロジック・エレメントで実現できた．ソフトウェア開発ツールとしてアセンブラも用意した．

（編集部）

以前，CPLD（EPM7256A）で実現した「野球ゲーム」[1]と同じものを，Cycloneを使って設計しました（**写真1**）．しかし，256マクロ・セルで実装可能なアプリケーションを，2,910ロジック・エレメントのFPGAにリターゲットするだけではおもしろくありません．そこで，独自のCPUコアを設計し，アプリケーションはソフトウェアだけで記述しました．同じ機能のハードウェアによる実現事例とソフトウェアによる実現事例として，参考にしてみてください．



〔写真1〕野球ゲームの外観

ソフト・マクロのCPUを作る

アプリケーションをソフトウェアで開発するためにはどうしたらよいでしょうか？

- FPGAの外部にCPUを接続する
- ソフト・マクロのCPUをFPGAに実装する

という二つの方法が考えられます．

今回は，ソフト・マクロのCPUをFPGAに実装する方法で実現することにしました．また，ソフト・マクロのCPUについても独自に設計しました．

●設計コンセプト

CPUの設計には，二つの方向性が考えられます．

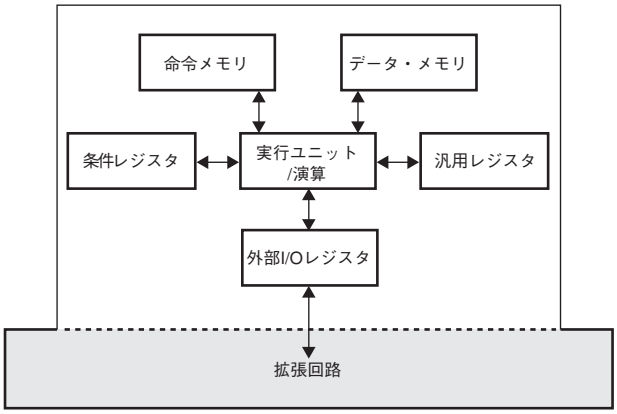
- 複雑なアーキテクチャをチューニングして処理性能を引き上げる
- 極限まで簡素化したアーキテクチャを採用して回路の規模を抑える

また，FPGAを用いた設計では，

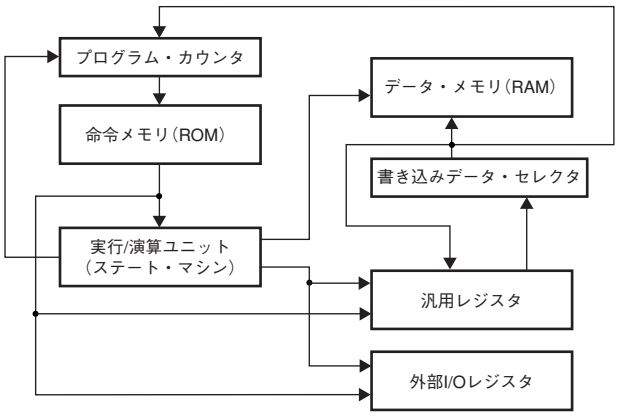
- 回路構成をシンプルにすれば動作周波数が上がる
- 回路規模を小さくして多くの回路を入れたい

といった点をよく考えます．

FPGAにソフト・マクロのCPUを実装することを考えた場合，CPUの機能をソフトウェアが100%活用するのであれば，ハードウェア・リソースをむだにしていないと言えます．しかし，多くのアプリケーションでは，CPUの持つ命令セットのすべてを利用することはまずありません．CPUの内部レジスタをすべて使いきらない場合すらあります．その反面，アプリケーションに最適な命令があれば，ソフトウェアはとてもシンプルになります．すなわち，カ



〔図1〕設計したCPUコアのブロック図
実行/演算ユニット，メモリ，レジスタのみのシンプルな構成である．外部回路とのインターフェースは，すべてレジスタを介して行う．



〔図2〕設計したCPUコアのデータフロー
命令メモリとデータ・メモリを分離するハーバード・アーキテクチャを採る．汎用レジスタの数は，ソフトウェアで設定可能．命令/実行ユニットは，ステート・マシンで制御する．

〔表1〕設計したCPUコアの仕様

レジスタ長/データ・バス幅	16ビット
命令数	10
メモリ・ワード数	FPGAの内蔵メモリ容量に応じて可変（今回は1,024ワード）
条件レジスタ数	1

スタマイズ可能なソフト・マクロのCPUは，FPGAにとって理想的なものになる可能性があります．

そこで，今回は「できるだけ簡素に，必要な機能だけをできるだけ小規模に」実現することにしました．

●アーキテクチャ設計

今回設計したCPUのブロック図を**図1**に示します．仕様を**表1**に示します．実行/演算ユニット，メモリ，レジスタのみのシンプルな構成です．外部回路とのインターフェースは，すべてレジスタを介します．

CPUを作り上げていくうえでは，メモリ，汎用レジスタ，外部I/Oレジスタ，分岐用条件レジスタ，制御論理，データフローなどを考える必要があります．今回はFPGAに実装することから，小規模化するためのくふうも必要です．

データフローを**図2**に示します．

1）メモリ

命令メモリとデータ・メモリを分離するハーバード・アーキテクチャを採用します．データ・メモリに対しては読み出しと書き込みを行いますが，命令用メモリは読み出ししか行いません．FPGAでは読み出しを行うだけ，つまりROMで実現したほうが配線リソースを節約できます．

メモリへのデータの書き込みと読み出しは，ラッチを介して行います．必要なハードウェア・リソースは少し増えますが，これによって動作周波数を上げることができます．

2）汎用レジスタ

汎用レジスタの数をどうするかは重要な問題です．CPUコアを論理合成するときに，ソフトウェアがいくつ必要としているかを判断できません．すなわち，使用していない汎用レジスタがあっても，自動的に削減することはできないのです．

今回はソフトウェアが用いるレジスタ数に応じて，ハードウェアで実装するレジスタの個数を定義できるようにします．この指定については手動で行うわけですが，ソフトウェアが使用するレジスタ数はプログラムを書く際に把握できているはずです．その数に応じてレジスタ回路を増減できるようにします．これにより，論理合成では不可能なレジスタ数の制御が可能となり，必要最小限の回路に抑えることができます．

3）外部I/Oレジスタ

外部I/Oレジスタは，拡張回路とのインターフェースで使用します．拡張回路には，レジスタを割り当てます．

しかし，回路によっては，割り当てたレジスタのすべてのビットを使用するとは限りません．例えば，1個のLEDをON/OFFする回路であれば，1ビットあれば十分です．16ビット幅のレジスタを割り当ててしまうと，15ビット分がむだになります．

そこで今回は，使用しないビットにラッチを生成しない