DALLAS SEMICONDUCTOR MAXIM

Microcontroller ENGINEERING REVIEW

Volume 3

MAXQ™アーキテクチャの紹介

MAXQ RISC アーキテクチャは、高性能・低消費電力で数多くの複雑なアナログ機能を実現します。

マイクロコントローラは8ビットに16ビット、RISC、CISC、その中間とさまざまな種類があり、プロジェクトに適したマイクロコントローラを選ぶのは大変です。選ぶときには、通常、数多くの条件を考慮します。たとえば、価格、性能、消費電力、コード密度、開発期間、そして場合によっては将来的な移行の可能性なども考慮します。選ぶのを難しくしている原因の1つに、1つの条件を追求すると、他の条件に影響が出てしまうという点があります。あるアプリケーションで重要なファクタが、他のアプリケーションではまったく問題にならないこともあります。結局のところ、すべてのプロジェクトに最適なマイクロコントローラなど存在しないのです。しかし、成功するためには、検討項目の多くにすぐれたマイクロコントローラを選ぶ必要があります。

世界的に評価の高いアナログチップメーカー、マキシムが、業界をリードする高性能マイクロコントローラサプライヤ、ダラスセミコンダクタと一緒になったとき、最先端のマイクロコントローラと高度なアナログ機能を組みあわせることが可能になりました。その結果生まれたのが、MAXQ RISCアーキテクチャです。これは新開発のマイクロコントローラコアで、高性能・低消費電力で数多くの複雑なアナログ機能を実現します。

複雑なアナログ回路と高性能ディジタルブロックを一体化するとき重要な点は、動作環境のノイズをいかに抑えるかです。クロックノイズとスイッチングノイズがマイクロコントローラコアのディジタル回路で発生し、影響を受けやすいアナログ部に混入してしまいます。このように、高いマイクロコントローラ性能を実現すると同時に、敏感なアナログ回路に影響するクロックノイズを最小限に抑えなければならないため、アナログ/ディジタルの混在回路は設計が難しいのです。

MAXQアーキテクチャでは、インテリジェントなクロックの管理・使用によってノイズを抑えています。つまり、MAXQのコアは、クロックが必要な回路にのみクロックを供給し、消費電力を抑えるとともにアナログ回路との混在に必要な低ノイズの環境を実現しているのです。また、1クロックで複数の機能を実行することによって、性能の向上も図っています。本稿では、MAXQアーキテクチャとその特長を紹介します。

無駄のないクロックサイクル利用

MAXQアーキテクチャは、消費電力に対して高い性能が得られるように設計されています。高効率マシンとするためにまず必要となる条件は、クロックサイクルを最大限に活用してユーザコードを実行することです。

MAXQでは、シングルサイクル命令実行によってクロックサイクルの有効活用を実現しています。この結果、命令帯域が拡大し、性能が向上します。または、クロック周波数を落として消費電力を下げることが可能になります。MAXQの命令は、ロングジャンプ/ロングコールと拡張レジスタアクセスの一部をのぞき、すべて、シングルクロックサイクルで実行されます。RISCマイクロコントローラにもシングルサイクル実行をサポートしていると言われるものが数多くありますが、ほとんどは命令やアドレッシングモードの一部をサポートしているにすぎません。これに対しMAXQでは、シングルサイクル実行が基本になっています。

もう1つ、MAXQアーキテクチャでクロックサイクルが有効活用できる理由として、命令パイプライン(RISCマイクロコントローラによく採用されている)を使わずにシングルサイクル動作を

目次

...MAXQのコアは、クロックが必要な回路にのみクロックを供給し、消費電力を抑えるとともにアナログ回路との混在に必要な低ノイズの環境を実現しています。

実現していることがあげられます。MAXQの命令デコード/実行ハードウェアが非常にシンプルであるため(かつ、タイミングが非常に速いため)、プログラムが自身をフェッチする形でこれらの動作を同一クロックサイクル上で実行し、最大動作周波数への影響を最小限に抑えているのです。命令パイプラインを使わないことの利点は、パイプライン実行による一般的なRISC CPUについて検討するとよくわかります。プログラムが分岐する際、CPUは、1つ、あるいは複数のクロックサイクル(パイプラインの深さによる)を使って分岐先アドレスをフェッチし、すでにフェッチしていた命令(複数の場合もある)を廃棄します。命令を実行するのではなく廃棄するためにクロックサイクルを使うことは明らかに無駄であり、性能を低下させ消費電力を増やす望ましくない行為です。このようにユーザにとって望ましくない動作ですが、パイプラインのリロードにクロックを消費してしまうことは、このアーキテクチャでは避けることができません。MAXQアーキテクチャでは、命令パイプラインなしで(つまりパイプラインで起きてしまうクロックサイクルの無駄なしで)シングルサイクル実行を実現している点が、8ビット・16ビットのRISCマイクロコントローラとは大きく異なる点です。

MAXQ命令ワード

MAXQ命令ワードはユニークで、古典的な意味合いでは、「MOVE」命令1つしかありません。「MOVE」命令に付随するソースオペランドとデスティネーションオペランドによって、命令やメモリアクセス、ハードウェア動作のトリガなどを行います。MAXQ命令ワードは、7ビットのデスティネーションフィールドと8ビットのソースフィールド、それに1ビットのソースフォーマットビットという2つのコンポーネントからなる16ビットです。ソースフォーマットビットが0のとき、イミディエートまたはリテラルバイト(#OOh－#FFh)はソースオペランドとして与えることができます。1つの命令ワード内でイミディエートバイトソースが無制限にサポートされているのは、レジスタ初期化ルーチンやALUオペレーションにおいて非常に便利なことです。非リテラルのソースとデスティネーションは、より小さなモジュールと呼ばれるグループに分割されています。図1に、16ビットのMAXQ命令ワードの構成を示します。

フォーマット	デスティネーション	ソース
f	ddd dddd	ssss ssss
1	INDEX MODULE	INDEX MODULE
0	INDEX MODULE	IMMEDIATE BYTE DATA (i.e., 00h–FFh)

*図1. **MAXQ命令ワードはシンプルでパワフルです。***

マシン命令は、すべて、ソースオペランドとデスティネーションオペランドによる転送動作になります。オペランドによって、MAXQデバイスの物理的なレジスタを指定することもできます。このような転送はもっとも基本的で、動作を思い描くことも簡単です。ただし、MAXQマシンでは、物理的レジスタとソース/デスティネーションオペランドが厳密な関係を持っているわけではありません。

MAXQアーキテクチャでは、間接メモリアクセスにも、このソース/デスティネーションによる転送を使用します。デスティネーションやソースのエンコーディングの一部が、スタックやアキュムレータ行列、データメモリなどに対する間接アクセスポータルとなっているのです。間接メモリアクセスポータルでは、物理的なポインタレジスタを使い、アクセスするメモリアドレス位置を定義します。データメモリに対する間接アクセスをする方法の一例をあげると、"@DP[0]"オペランドを使う方法があります。このオペランドをソースやデスティネーションとして使うと、データポインタ0(DP[0])レジスタでアドレッシングされたデータメモリに対し、間接的に読込み、あるいは書込み処理を行います。

また、MAXQアーキテクチャでは特殊なデスティネーションエンコーディングやソースエンコーディングによっても、ハードウェア演算をトリガします。このようなトリガメカニズムをベースとして、各リソースに対し暗黙的にリンクされたMAXQ命令が作られます。たとえば、算術演算(ADD、SUB、ADDC、SUBB)は、ワーキングアキュムレータのいずれかを暗黙的にターゲットとしたデスティネーションエンコーディングとして実行されます。ユーザソースオペランドは、ユーザが提供します。条件ジャンプは命令ポインタ(IP)を修正の暗黙ターゲットとしており、評価するステータスごとに独立したデスティネーションエンコーディングとして実行されます。

間接メモリアクセスとハードウェア演算のトリガは、可能な限り組みあわせてソース/デスティネーションオペランドとしていますが、これは別の意味でも便利です。自動増分/減分間接アクセスニーモニックを使ったデータポインタがよい例です。DP[0]を使ってデータメモリを読み出すとき、ソースオペランドを"@DP[0]++"や"@DP[0]--"とするだけで、読み出しの後にポインタを増分/減分することができます。

このような形式になっていることによって、さまざまな利点があります。モジュールとしてグループ化されたソースオペランドとデスティネーションオペランドが命令ワードに含まれているため、ハードウェアによる命令デコーディングがシンプルで高速になるとともに、そのようなモジュールでは転送時に信号スイッチングを制限するため消費電力とノイズが小さくなります。また、命令ワードの16ビットすべてがソースオペランドとデスティネーションオペランドに使うことができるため、物理レジスタ、間接メモリアクセス、ハードウェアトリガ動作に十分なアドレス空間が得られます。最終的には、ソース/デスティネーションの組み合わせに関する制限が少ない大きなソース/デスティネーションアドレス空間を使うことができることから、直交性の高いマシンを構築することができます。

…ソース/デスティネーションの組み合わせに関する制限が少ない大きなソース/デスティネーションアドレス空間が使えることから、直交性の高いマシンを構築することができます。

MAXQシステムの特長

MAXQシステムはマイクロコントローラユーザが要求する基本的なハードウェアリソースと機能を提供するだけでなく、そのようなリソースを強化し、デバイスのさまざまな機能や有用性を拡張することができます。ここでMAXQシステムのリソースすべてを紹介するわけにはいきませんが、例をいくつかご紹介しましょう。

ワーキングアキュムレータ

MAXQアーキテクチャはここまで、ひとつのエンティティとして扱ってきましたが、実際は、2004年のはじめに発売されるMAXQファミリには、MAXQ10とMAXQ20という細かな点で異なる2つのバージョンがあります。MAXQ10とMAXQ20の一番大きな違いは、ワーキングアキュムレータの幅とサポートする論理演算装置(ALU)にあります。MAXQ10は、8ビット(バイト幅)のアキュムレータとALU演算をサポートしており、MAXQ20は16ビット(ワード幅)のアキュムレータとALU演算をサポートしています。MAXQデバイスのアキュムレータ数は最小で8本であり、アプリケーションに応じて16本まで増やすことができます。ソース/デスティネーション転送マップにおいて、これらのアキュムレータはシステムレジスタモジュールであり、A[n](*n*は各アキュムレータのインデックス)でアクセスできるようになっています。つまり、16本のアキュムレータを持つMAXQデバイスは、アキュムレータA[0]、A[1]...A[14]、A[15]を持ちます。アキュムレータはいずれもアクティブアキュムレータであり、*Acc*ニーモニックによってアクセスすることができます。アキュムレータポインタ(AP)レジスタを、アクセスしたいアキュムレータのインデックスにセットします(Acc = A[AP])。APレジスタはアキュムレータ行列にバイナリデコードを提供するために必要な数のビットのみを実行するため、16本のアキュムレータを持つためには4ビットが必要になります。ALU演算はすべて、演算デスティネーションとしてアクティブアキュムレータが暗黙に指定されています。例として、「ADD C src」命令を見てみましょう。この命令は、アクティブアキュムレータとキャリーフラグ、指定されたソース(src)オペランドの加算演算を行う命令です。アクティブアキュムレータ周辺でさまざまなビット操作とシフト/ローテート命令が実行されます。

アキュムレータポインタには特殊なハードウェアが付属しており、アキュムレータファイルに順序よくアクセスすることができるようになっています。アキュムレータポインタコントロール(APC)レジスタから、APをリセットするビットと、インクリメント、ディクリメント、モジュロの各演算をストリームラインするビットがアキュムレータポインタレジスタに提供されます。

プロセッサステータスフラグ(PSF)という、5種類のステータスフラグを持つレジスタもあります。フラグは(C)arryと(Z)ero、(S)ign、(E)qual、(OV)erflowで、それぞれ、アクティブアキュムレータの状態やALU演算に対して特別な意味を持ちます。これらのフラグを評価して、条件ジャンプやリターンを行うことも可能です。PSFレジスタはまた、ユーザがソフトウェアから利用できる2つの追加汎用フラグ(GF1とGFO)を持っています。

MAXQは転送トリガアーキテクチャにより、広帯域、高効率、高直交性を実現しています。

専用ハードウェアスタック

MAXQアーキテクチャは、専用ハードウェアスタックを備えています。スタックの深さは製品によって異なります。専用ハードウェアスタックには2つのはっきりとした利点があります。まず、データメモリをスタックで消費することなく、アプリケーションで利用することができること。もう1つは、専用ハードウェアスタックが専用のリード/ライトポートを持ち、ポートをデータメモリと共有しなくてよいため、PUSH/POP動作が高速になることです。ハードウェアスタックの深さが不足した場合には、データポインタをスタック的に操作し(事前にプリインクリメント/プリディクリメントしてライトし、リード後にポストインクリメント/ポストディクリメントする)、データメモリにソフトウェアスタックを作ることが可能です。

MAXQアーキテクチャでは、命令パイプラインなしで(つまりパイプラインで起きてしまうクロックサイクルの無駄なしで)シングルサイクル実行を実現している点が、8ビット・16ビットのRISCマイクロコントローラとは大きく異なります。

柔軟な割込アーキテクチャ

MAXQ10もMAXQ20も、1本のユーザ構成可能な割込ベクトルアドレスレジスタを持っています。この方式によって、割込識別・処理ルーチンをユーザが好みの位置に置くことが可能です。割込ソースに優先順位が強制されることはありません。通常の個別及びグローバルな割込イネーブルとフラグに加え、マスキングフラグと識別フラグがモジュールレベルで用意されています。個別ソースのイネーブルとモジュールからグローバルレベルのマスキング、割込ソースの優先順位は、すべてユーザコードで自由に設定することができます。このような形で割込がサポートされていることには、大きな利点があります。まず、使用されないコード空間が発生しません。これに対して、ソースごとに専用の割込ベクトルアドレスを持つマイクロコントローラでは、使用しない割込ベクトルに割り当てられたコード空間は使えないのがふつうです。次に、どの割込をイネーブルにするのか、また、割込優先順位をどうするのかを、ユーザが自由に決めることが可能になります。

ハードウェアループカウンタによるオーバヘッドの削減

MAXQアーキテクチャでは、2本の16ビットループカウンタのいずれか(LC[0]かLC[1])に対して動作するDJNZ命令を持ちます。「DJNZ LC[n], src」命令によって、シングルクロックサイクルで、ループカウンタレジスタをディクリメントし、カウンタが0に達しなければ、プログラムの実行を指定されたアドレスに条件分岐します。これに対してRISCマイクロコントローラでは、カウンタレジスタの更新とループ終了条件のチェックは、2つの動作になります。この2つのアクションがMAXQでは1つにまとめられているということは、マイクロコントローラのアプリケーションによくあるソフトウェアループでループカウンタを管理する際、コード量もサイクルオーバヘッドも少なくてすむことを意味します。シングルサイクルのループカウンタディクリメントと条件分岐がDJNZ命令でトリガできるというのは、クロックサイクルを最大限に活用するという当社の基本方針に合致します。

拡張データポインタ

MAXQには16ビットのデータポインタが3つ(DP[0]、DP[1]、BP[Offs])あります。各データポインタは、個別にワードアクセスモードかバイトアクセスモードに構成することができます。設定はデータポインタコントロール(DPC)レジスタのワード/バイト選択(WBSn)レジスタビットによって行います。データポインタは3本とも、シングルサイクル間接メモリアクセスが可能で、書込み時にはプリインクリメント/プリディクリメントが、読出し時にはポストインクリメント/ポストディクリメントがサポートされます。このデータポインタの1つ、フレームポインタ(FP = BP[Offs])は、16ビットベースポインタ(BP)レジスタと8ビットオフセット(Offs)レジスタの符号なし加算によって生成されます。このようなポインタはCコンパイラ開発ツールにとって重要であり、その中でも特に、スタックフレームの処理で重要な役割を果たします。

フォン・ノイマンの利点を併せ持つハーバードメモリアーキテクチャ

MAXQアーキテクチャはハーバードメモリアーキテクチャを採用しているため、プログラムバスとデータメモリバスが分離されており、命令ワードとデータワードに同一クロックサイクルで同時にアクセスすることが可能です。データメモリにアクセスする命令をシングルサイクル実行し、最高の性能を得るためには、このようなメモリ構成が必要なのです。フォン・ノイマンメモリインタフェースを使ったマイクロコントローラでは、プログラムメモリとデータメモリ、I/O、周辺機器へのアクセスでバスを共有するため、バス帯域によって性能にボトルネックが発生します。

フォン・ノイマンメモリアーキテクチャを推す人々は、プログラム空間をデータメモリとしてアクセスしたり、その逆をすることができないのは困るといいます。アクセス可能性があれば、定数の保存やルックアップテーブル、インシステムまたはインアプリケーションのプログラミング柔軟性などの面で利点があるからです。MAXQでは、そのような弱点に対し、メモリマネージメントユニット(MMU)と固定ユーティリティROMを追加するという方法で対処しました。なお、固定ユーティリティROMには、論理メモリマッピングと固定ユーティリティコードルーチンがあり、これによって、インシステムプログラミングと便利なアクセスモードをサポートします。

リソースアクセスの1点集中

MAXQアーキテクチャが持つもう1つの特長は、1つの転送マップに、すべてのリソースに対するアクセスポイントを持たせている点です。これをレジスタマップではなく転送マップと呼んでいるのは、MAXQアーキテクチャが転送トリガコンセプトをベースにしているからです。

転送マップは16のモジュールに仕切られています。各モジュールには、インデックスと呼ばれる32個の個別アクセスポイントがあります。これらのアクセスポイントで重要な点は、アクセスポイントでレジスタを直接読み書きできるだけでなく、間接メモリアクセスやハードウェア演算のトリガもできるという点です。16のモジュールのうち、最初の6モジュール(M0～M5)は、デバイス固有の周辺機能用に割り当てられています。つまり、周辺機器のレジスタ用とアクセス用として、豊富な転送マップ(6 x 32 = 192箇所分)が用意されているということです。これらのモジュールには、指定されたMAXQデバイスオプションに応じて、ディジタルI/Oやタイマ、シリアルポート、ハードウェア乗算器、LCDドライバ、ADC、インサーキットデバッガなどの機能を実行するレジスタが用意されます。後半の10モジュール(M6～M15)は、MAXQシステム機能用です。このシステムモジュールには、MAXQシステムの動作に必要なレジスタがあります。ウォッチドッグやシステムクロック、割込制御などに使うレジスタです。また、システムモジュールには間接メモリアクセスや特殊なマシン操作をトリガするワーキングアキュムレータファイルとデータポインタ、ソース/デスティネーションエンコーディングもあります。このような基本システムレジスタ空間は、MAXQのデバイスオプションが異なってもできる限り同一になるようにしてあります。図2に、MAXQのソース/デスティネーション転送マップの例を示します。

MAXQには、すべてのリソースに対するアクセスポイントを持つ16分割の転送マップが1つだけあります。

MAXQアーキテクチャの特長としてもう一点、挙げるべきものにプリフィックスレジスタモジュールがあります。あるデータ(デフォルト = 00h)を特殊な転送動作で使うための1つのプリフィックスレジスタがあります。このプリフィックスレジスタは、ロードされると、1クロックサイクルだけデータをホールドし、00hに戻ります。プリフィックスレジスタ(PFX[n])を選択するときには、インデックスとして(n)をつける必要があります。MAXQの転送マップは16のモジュールで構成されており、モジュールごとに32のインデックスがあるということは、1つの命令ワードで使えるソース/デスティネーションエンコーディングビットで直接アクセスできない部分があることを意味します。各モジュールのソースインデックスの後半16個とデスティネーションインデックスの後半24個がそうです。このようなインデックスに対するアクセスウィンドウを、1サイクルの間だけ開け、この問題を解決するのがプリフィックスレジスタなのです。PFX[n]レジスタがロードされると、インデックスの"n"によって高次ソースとデスティネーションビットがすぐ後の命令(*n = dds*)に供給されます。このように、プリフィックスレジスタモジュールという手段を通じて、不足のデコードビットがアクセス拡張(または保護)レジスタに提供されます。プリフィックスレジスタのロードが必要な動作やアクセスはアセンブラが自動生成するため、ユーザがプログラミングする必要はありません。プリフィックスレジスタモジュールはまた、16ビットデスティネーションに書込む際にソースバイトを連結するためにも使うことができます。ユーザから見えない部分での処理になりますが、16ビットの絶対アドレスへのジャンプやコールでも、このような形でプリフィックスレジスタが使われています。将来的なMAXQアーキテクチャの拡張という意味では、プリフィックスレジスタモジュールによって、未使用のシステムモジュール空間にMAXQ命令セットをシームレスに拡大・拡張することが可能です。

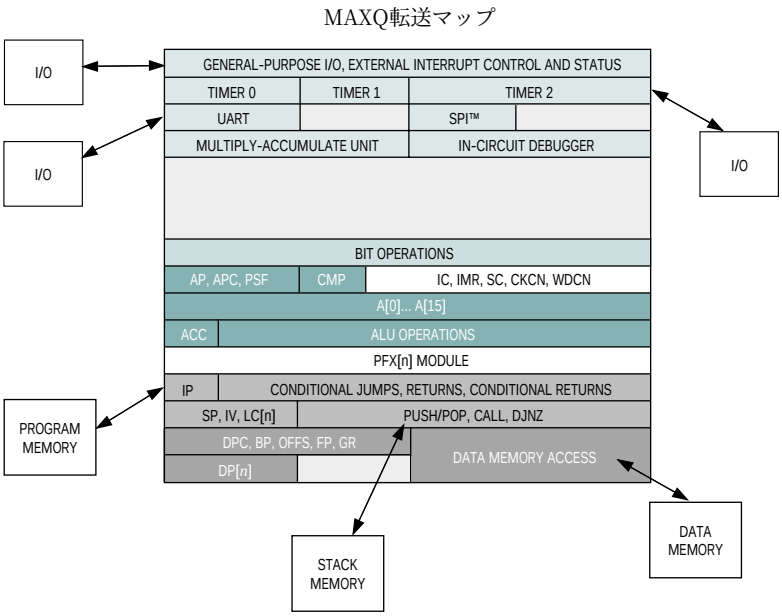

*図2. **MAXQでは、1つの転送マップを経由してあらゆるリソースにアクセスします。***

まとめると、MAXQデバイスの転送マップには、デバイスが持つすべてのシステムレジスタと周辺機器レジスタがマッピングされています。この同じマップに、データメモリやスタックメモリ、アキュムレータ行列への間接アクセスポイントも用意されています。また、MAXQのマシン命令やその下層で行われる動作をトリガするアクセスポイントや、将来的な命令セットの拡張メカニズムも、同じマップに用意されています。すべてのリソースに対するアクセスポイントが1つの転送マップにまとめられているということは、さまざまなソース/デスティネーション転送方法が可能だということを意味します。このようにアクセスがまとめられているため、クロックを必要とするリソースに対するクロック分配もシンプルに行うことができます。この結果、とても「静かな」環境が得られ(MAXQの"Q"はQuietを意味します)、アナログとの混在をする上でメリットになります。MAXQアーキテクチャでは、非常に高い周辺機能のモジュール性と可搬性が実現できます。このような形式としたのは、製品開発サイクルが短く、エンド

...MAXQシステムと周辺リソースをモジュールとした結果、コンパイラの最適化が可能となっただけでなく、モジュールの可搬性が高くなり、MAXQの拡張などが短期間で行えるようになりました。

ユーザの周辺に対する要求がどんどん変化しているため、高い柔軟性と再利用性が必要だと考えたからです。周辺機能が高いモジュール性を持つことから、ある市場やアプリケーション向けにMAXQデバイスを作ろうとしたとき、標準的なMAXQ周辺モジュールを複製したり追加したり、取り除いたりするために必要な設計期間を最小限に抑えることができます。

結論

MAXQアーキテクチャは、マイクロコントローラ業界における真の革新だといえます。MAXQでは、転送トリガアーキテクチャにより、広帯域、高効率、高直交性を実現しています。また、MAXQシステムと周辺機器リソースをモジュールとした結果、コンパイラの最適化ができただけでなく、モジュールの可搬性が高くなり、MAXQの拡張などが短期間で行うことができるようになりました。また、次世代製品に適した命令セットの拡張が行うことができるメカニズムも組みこまれています。このようにMAXQアーキテクチャは、どのような基準で選んでも高い評価を受けることが確実で、既存プロジェクトに対しても今後のプロジェクトに対しても理想的なソリューションとなります。