

MAXQの秘密の解明

MAXQ®プロセッサファミリは、1クロックサイクルで複数の処理を実行する高性能の8ビット、16ビット、および32ビットのシングルサイクルマイクロコントローラです。この記事ではMAXQ20コアの内部機能を考察し、その優れた能力を紹介します。

プログラムのモデル

MAXQ20コアは16ビットCPUであり、これは全アキュムレータと大部分のワーキングレジスタ(スタック、データポインタ、カウンタ)が16ビット長であることを意味します。MAXQ20は、64kワードのコード空間(すなわち64kBの命令)と64kワード(128kB)のデータ空間をアドレス指定することができます(図1)。

なお、MAXQ20コアに基づくプロセッサの場合は、このメモリ空間の大部分は空き空間です。また、ユーティリティROMとデータRAMは上位32kBのコード空間に存在するため、この領域のユーザコードにアクセスするには、この記事の対象外であるコアの特別機能が必要です。

アキュムレータ

「アキュムレータ」と呼ばれる16個のレジスタが、汎用レジスタアレイを構成しています。アキュムレータポインタ(Accumulator Pointer)レジスタ(AP)が示すレジスタは、算術演算および論理演算の対象である「アクティブアキュムレータ」として指定されます。このため、APレジスタの値を変更すると、16個のアキュムレータの任意のアキュムレータを算術論理ユニット(ALU)演算の対象として指定することができます。アキュムレータポインタ制御(Accumulator Pointer Control)レジスタ(APC)を通じて、アクティブアキュムレータにアクセスするごとに、APを自動的にインクリメントまたはデクリメントするため、高精度の演算が容易になります。図2ではA[0]がアクティブアキュムレータですが、アキュムレータへのアクセスによって、APCレジスタの値に応じてA[1]またはA[15]がアクティブアキュムレータになることができます。

GRレジスタ

汎用レジスタ(General Register) (GR)は、16ビットワードからの各バイトの抽出に便利です。プログラマはこのGRを使って、バイトをワードに組み立てることができます。すなわち、下位バイトをGRL (汎用レジスタの下位バイト(General Register-Low byte))に、上位バイトをGRH (汎用レジスタの上位バイト(General Register-High byte))にロードし、組み立てられたワードをGRで読み取ります。また、プログラマはGRレジスタを使って、ワードをその各構成バイトに分解することができます。GRにロードされたワードをGRS (汎用レジスタのスワップ(General Register-Swapped))を用いてバイトスワップ形式で読み取ることができます。最後に、GRXL (汎用レジスタの拡張下位バイト(General Register eXtend Low byte))を読み取って、GRLレジスタにロードされたバイトを符号拡張してワードにすることができます。図3を参照してください。

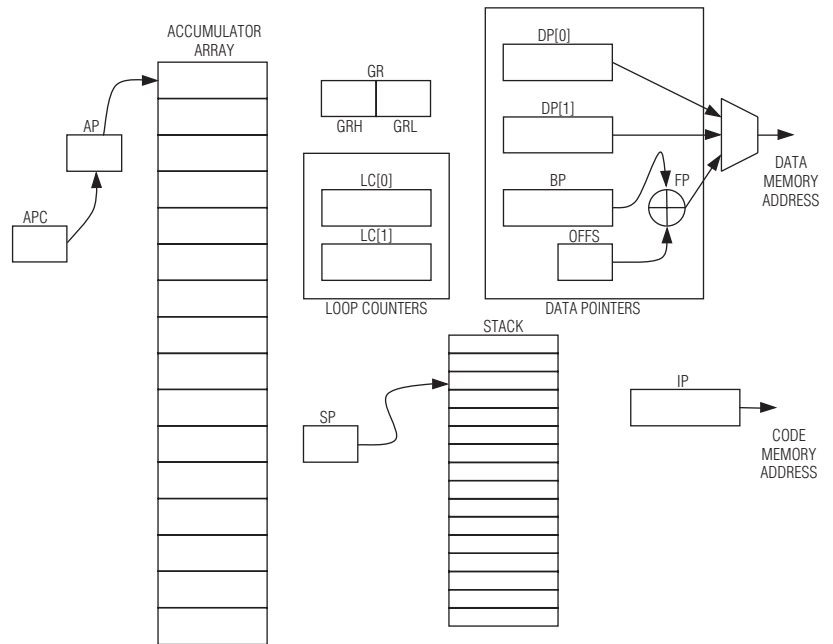


図1. MAXQ20コア用のプログラムモデルは、16個の汎用アキュムレータ、2個のループカウンタ、および1組のデータポインタから構成されています。

目次

MAXQの秘密の解明 ..1

SDメディア
フォーマットによる
MAXQ2000の
不揮発性データ
ストレージ空間の
拡張8

Rowley CrossWorks
とMAXQ3100*の
評価キットによる
温度ロギング
アプリケーションの
作成.....13

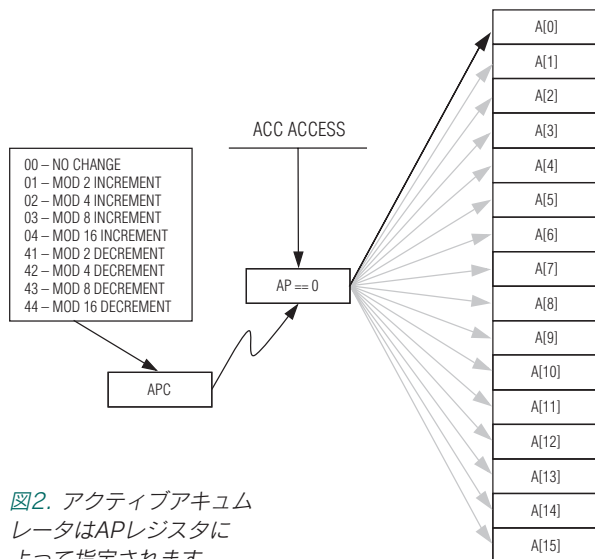


図2. アクティブアキュムレータはAPレジスタによって指定されます。このAPレジスタ自体をアキュムレータアクセス命令によって変更することができます。

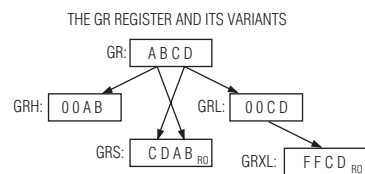


図3. GRレジスタは、バイト抽出、バイトスワップ、および16ビット符号拡張をサポートしています。

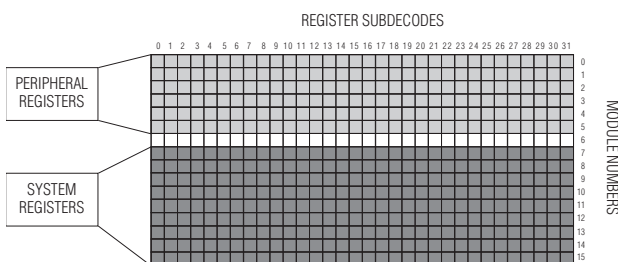


図4. MAXQ20コアのレジスタ割当は、以下のように2つの領域に分割されます。すなわち、レジスタバンク0~5はペリフェラルレジスタで、あるMAXQデバイスと別のMAXQデバイスでは変更可能です。バンク7~15はシステムレジスタで、いずれのMAXQデバイスにおいても比較的固定されています。

MAXQの命令が実行されると、デスティネーションレジスタにソースレジスタの内容または即値がロードされます。また、このデータ転送によって、ポイントのインクリメントやデクリメント、あるステータスビットの設定、またはその他のある関数などの他のイベントをトリガすることができます。したがって、このアーキテクチャは転送トリガアーキテクチャです。このアーキテクチャをサポートするには、大きなレジスタの補足が必要です。MAXQ20コアには、ペリフェラルレジスタ空間とシステムレジスタ空間の2つに大別される合計512のレジスタアドレスがあります(図4)。

最初の6つのレジスタモジュール(モジュール0~5)はペリフェラルレジスタ専用で、最後の9つのモジュール(モジュール7~F)はシステムレジスタとして割り当てられています(モジュール6は予備)。ペリフェラルレジスタモジュールがMAXQプロセッサのあるタイプから別のタイプに変更されても、システムレジスタはいずれのMAXQプロセッサでも同じ状態を維持します(図5)。

ループカウンタ

ループカウンタ0 (LC[0])とループカウンタ1 (LC[1])の2個のループカウンタがあります。これらのレジスタを汎用レジスタとして使用可能ですが、カウンタが非ゼロ(DJNZ)命令である場合は、デクリメントおよびジャンプ用のループカウンタになります。

スタック

MAXQ20コアは、専用の16レベルスタックを内蔵しています。スタックポインタは使用される次のスタック位置か、またはPUSH命令またはCALL命令を示します。

データメモリポインタ

MAXQマイクロコントローラは、データメモリにアクセスするための3個のポインタを備えています。DP[0]およびDP[1]の2個のポインタは、単純な16ビットポインタです。第3のポインタは、ベースアドレスポインタ(BP)を8ビット無符号オフセット(OFFS)に追加すると生成されます。

なお、3個のデータポインタのいずれか1つでアドレス指定されるデータメモリは、命令ポインタでアドレス指定されるコードメモリとは異なります。MAXQのプロセッサはすべてメモリ管理ユニット(MMU)を内蔵しているため、いずれのメモリセグメントもコードやデータとして扱うことが可能であり、またコードバスとデータバスは分離独立しています。コードおよびデータのフェッチ命令用のバスがこのように分離しているのはMAXQ20技術の基盤となる要素であり、シングルクロックサイクルでコードとデータの同時アクセスが可能になります。

転送トリガアーキテクチャ

プログラマモデルを調べると、命令をロードし、命令をデコードして、CPUの特定要素を作動させる従来型の命令フェッチ-デコードユニットが存在すると結論付ける人がいるかもしれません。しかし、その結論は誤っています。MAXQアーキテクチャがその他の従来型CPUと異なっているのは、MAXQコアの転送トリガ特性です。

転送トリガは、単純なMOVE (転送)命令によってCPUで利用可能なあらゆる関数を実行させる手法です。MAXQのアセンブラは30を超える命令コードをサポートし、以下のようなMAXQ命令セットでいずれの命令でもエンコードすることができます。

```
move Ma[b], Mc[d]
または
move Ma[b], #immediate_value
```

ここで、記号表示Ma[b]はレジスタモジュールaとレジスタサブコードbを示します。簡単に言うと、ADD、ビット操作、外部メモリの参照などのいずれの命令も、2個のレジスタ間の転送として、またはレジスタへの即値の転送としてコーディングされます。

MAXQの命令のデコード

いずれのMAXQ命令も実際にはMOVE（転送）であるため、いずれの命令も以下の3つの領域に分類することができます。すなわち、データの転送元を指定するSOURCE（ソース）領域、データの転送先を指定するDESTINATION（デスティネーション）領域、およびソースが即値(FORMAT == 0)か、またはレジスタ指定子(FORMAT == 1)であるかを指定するフォーマットビットです(図6)。

命令コード0x0923を例にしましょう。この命令では、FORMATビットはクリア状態で、ソース指定子(23)を8ビットの即値として扱う必要があることを示しています。デスティネーションモジュールはモジュール9であるアキュムレータアレイです。このアレイ内のレジスタ0はアキュムレータA[0]です。このため、この命令によって、値0x0023がレジスタA[0]にロードされます。この場合、ソースまたはデスティネーション指定子に関連する副作用はありません。

もう1つの例として、0xBF09を検討しましょう。この命令では、FORMATビットが設定され、ソース指定子をレジスタと見なす必要があることを意味します。モジュール9のレジスタ0は、上述したようにアキュムレータA[0]です。デスティネーション側では、モジュールFはデータポインタモジュールであり、レジスタ3（命令ではビット14:12）はデータポインタDP[0]を表します。このため、この命令は、A[0]の内容をDP[0]に転送します。

なお、場合によって、レジスタモジュール内の各場所は実際のレジスタを示す場合と、示さない場合があります。また、各場所が実際のレジスタを示すことができても、そのレジスタサブデコードにアクセスされるときに、副作用が発生することもあります。たとえば、前の例を0xAF09で少し変更してみましょう。デスティネーションサブデコードのみが変更されました。すると今度はレジスタDP[0]にロードする代わりに、命令はDP[0]をデクリメントしてから、DP[0]が指定する新しいメモリ位置への格納操作を開始します。すなわち、この命令はプリデクリメントされたポインタで間接格納を実行します。MAXQのアセンブラでは、これは「move @--DP[0], A[0]」としてコーディングされますが、「move M15[2], M9[0]」と同様に容易にコーディングすることができます。

プレフィックスレジスタ

モジュール当り32個のレジスタがありますが、ソースレジスタの選択用は4ビットのみで、デスティネーションレジスタの指定用は3ビットのみです。一見すると、これは、レジスタサブデコードの半分を読み取りできず、レジスタサブデコードのまるまる4分の3も書き込むことができないことを意味します。幸いにも、MAXQアーキテクチャ設計はこれに対処しています。MAXQプロセッサはすべて、これらの追加レジスタアドレスビットを提供し、ワード幅のmove（転送）の上位バイトを提供するプレフィックスレジスタを備えています。詳細については、「モジュール11—プレフィックス」の項を参照してください。

一度に1モジュールのMAXQ命令セットの作成

以下の項では、システムレジスタモジュールと、これらのモジュールが連携して定義/未定義命令を作成する方法について詳述します。まず、MAXQ20コアの中核であるアキュムレータアレイを調べてみましょう。

		REGISTER SUBDECODE NUMBER															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MODULE NUMBER	7	BOOLEAN VARIABLE MANIPULATION															
	8	AP	APC			PSF	IC	IMR	CMP	SC			HIR			OKCN	WDCN
	9	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]	A[11]	A[12]	A[13]	A[14]	A[15]
	10	ACCUMULATOR OPERATIONS															
	11	PREFIX															
	12	IP UNC	IP Z	IP C	IP E	IP S	IP NZ	IP NC	IP NE								
	13	POP PUSH	SP	IV	CALL	DUNZ LCO	DUNZ LCI	LCO	LCI	POPI							
	14	@FP	@FP ++	@FP --	OFFS	DPC	GR	GRL	BP	GRS	GRH	GRXL	FP				
	15	@DP0	@DP0 ++	@DP0 --	DP0	@DP1	@DP1 ++	@DP1 --	DP1								

図5. MAXQシステムレジスタマップは、MAXQ20ベースのすべてのプロセッサ内にあるレジスタと、命令セットを実装する追加デコードから構成されます。

DECODING A MAXQ INSTRUCTION

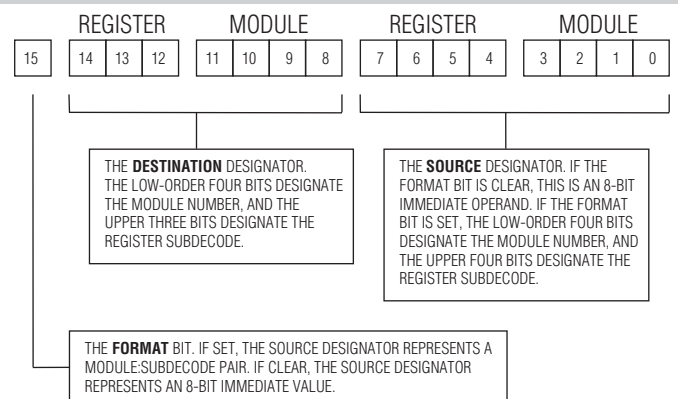


図6. MAXQ命令は、ソース指定子、デスティネーション指定子、およびソースが即値オペランドかまたはレジスタオペランドかを決定するソースフォーマットビットの3つの領域から構成されています。

MAXQプロセッサファミリは、シングルクロックサイクルで複数の処理を実行する高性能の8ビット、16ビット、および32ビットの各シングルサイクルマイクロコントローラから成ります。

MAXQ20コアは16ビットCPUであり、これは全アキュムレータと大部分のワーキングレジスタ(スタック、データポインタ、カウンタ)が16ビット長であることを意味します。

モジュール9—アキュムレータ

MAXQアーキテクチャは、大部分のバリエーションでは16個しか実装されていませんが、最大32個のアキュムレータをサポートします。これらのアキュムレータには、モジュール9を通じて直接アクセスされます。このモジュール内の各サブデコードは、1個のアキュムレータを表します。モジュール9は概念的には最も簡易なモジュールですが、アキュムレータアレイに影響を及ぼすモジュールがさらに2つあります。

モジュール8—システム制御

このモジュールは、割込み制御やプログラムステータスフラグなどのシステム処理の各状況を管理する多くのレジスタを備えています。これらのレジスタの多くはこの記事の範囲外であるため、詳細についてはデバイス仕様を参照してください。

APおよびAPCレジスタは、特に注目する価値があります。APレジスタはいずれのアキュムレータレジスタがアクティブアキュムレータであるかを決定します。すなわち、算術演算、論理演算、およびビット単位の演算の対象を指定します。APレジスタは、アレイ内のいずれのアキュムレータも指定することができます。

APCレジスタは、アキュムレータ処理後にAPレジスタの変更方法を設定するビットセットを備えています。このため、選択可能な2のべき乗の係数でカウントをロールオーバーして、APレジスタをインクリメントまたはデクリメント可能であるため、高精度の演算が容易になります。

モジュール10—アキュムレータ機能

モジュール10において、アキュムレータの実際の作業の大部分が実行されます。モジュール10によって、従来型のALU機能にアクセスし、アクティブアキュムレータにビットレベルでアクセスすることができます。モジュール10はユニークであり、ソース、デスティネーション、またはソースおよびデスティネーションとして機能しているかに応じて、さまざまに動作します。

MAXQ20は、64kワードのコード空間(すなわち64kBの命令)と64kワード(128kB)のデータ空間をアドレス指定することができます。

ソースがモジュール10で、デスティネーションがモジュール10以外のモジュールの場合は、アキュムレータの内容はデスティネーションに転送されます。サブデコードがゼロの場合は、APレジスタはAPCレジスタのビットに応じて変更されます。サブデコードが1の場合は、APレジスタは変更されません。

なお、マクロアセンブラの現行バージョンはサブデコード1をサポートしていません。これは、サブデコード1を指定するニーモニックまたは指定子がないためです。このため、命令「`move A[1], ACC`」によって、命令コード0x990Aが必ず生成され、0x991Aは生成されません。

モジュール10がデスティネーションとして指定され、ソースが即値またはモジュール10以外のモジュールの場合は、ソースはALUを通じて送出されます。デスティネーションはソースから直接供給されず、ALUの出力から供給されます。このようにして、算術命令と論理命令が実装されます。

なお、ソースレジスタとして機能することに対しては制限がありません。即値、間接メモリ位置、またはスタックの値やペリフェラルレジスタにもなることができます。

ソースおよびデスティネーション指定子がモジュール10の場合は、アキュムレータ専用命令か、またはキャリービットを含むビット操作です。いずれの場合でも、ソースとデスティネーションサブデコードを使って、命令を指定することができます。

デスティネーションサブデコード0は、補数、否定、全シフト、ローテート、および交換命令などのアキュムレータ専用命令のホームです。デスティネーションサブデコード1、2、3、6、および7は、キャリービットを使用する命令とビット単位のロードで構成されます。最後に、デスティネーションサブデコード5は、ロード0および1と補数というキャリー専用命令を備えています。

「アキュムレータ」と呼ばれる16個のレジスタが、汎用レジスタアレイを構成しています。

なお、デスティネーションサブデコード5のうちの1つのソースサブデコードは、専用のNOP(無演算)命令です。副作用がなく、かつ空のレジスタ位置をアドレス指定する命令がNOPとして機能し、また「`MOVE M10[5], M10[3]`」は現行または今後のMAXQデバイスにおいてノーオペレーション(無演算)であることが特に保証されています。これは、NOPニーモニック用の現行の全アセンブラで生成される命令コードです(0xDA3A)。

モジュール12—命令ポインタ

モジュール12は、多数の条件付きロード命令を備えているためユニークです。モジュール12がソースモジュールとして使用される場合は、IPがデスティネーション指定子にコピーされるだけです。ただし、モジュール12がデスティネーションである場合は、指定条件を満たさなければ、命令は実行されません。

また、モジュール12は8ビット即値ソースからロードされるためユニークであり、ソース値は符号付き整数と見なされ、命令ポインタの直前のプリインクリメントされた内容に追加されます。この追加によって相対短分岐が容易になるため、コードサイズが大幅に節減されます。また、いずれの短分岐命令または長分岐命令も条件付きにすることができることを意味します。

このモジュールは、命令ポインタ(Instruction Pointer)レジスタ(IP)の簡易なロードと格納のみをサポートしています。CALL命令は、スタックにプッシュするIP命令ではなく、IPのロードも行うスタック命令と見なされます。このため、CALL命令の転送はスタックポインタモジュール(モジュール13)における転送です。また、明示的なRET命令がなく、これはPOP IPとしての働きをします。

モジュール13—スタックポインタ

モジュール13はスタックポインタ関連のレジスタだけでなく、ループカウンタおよび割込みベクトルも備えています。なお、複数のサブデコードはデスティネーションとしてのみ有効です。サブデコード8はソースとしてのみ有効で、値をスタックからポップし、サービスビットの割込みをクリアすると、RETI命令が容易になります。

サブデコード3、4、および5は、IPレジスタのプロキシとして機能します。サブデコード3は、インクリメントされた命令ポインタがスタックにプッシュされると、命令ポインタをロードするため、従来型のCALL命令が実装されます。サブデコード4および5は指定したループカウンタのプリデクリメントされたバージョンをループカウンタにロードし、プリデクリメントされたループカウンタが非ゼロの場合は命令ポインタにソースオペランドをロードします。このデスティネーションサブコードにロードするソースはいつでも可能です。命令[DJNZ LC[0], A[1]]は完全に有効です。この場合は、この命令はLC[0]をデクリメントし、デクリメント演算の結果が非ゼロの場合はA[1]のアドレスに分岐します。

モジュール14—GR、BP、およびDPC

モジュール14はDPCレジスタ、GRレジスタ、およびベースポインタやオフセットレジスタに関連する全レジスタを備えています。

データポインタ制御(Data Pointer Control)レジスタ(DPC)は、データポインタの動作方法を示します。特に、データポインタがワードモードまたはバイトモードで動作しているかどうかを定義する1ビットがデータポインタごとに含まれます。また、このレジスタは、いずれのポインタが現在のソースポインタであるかを定義するフィールドも備えています。ソースポインタのロード時にソースがアクセスされ、オペランドデータ用のバスは1つしかないため、このフィールドが必要です。

16ビットデータ用にバイトアクセスが必要な場合に、GRレジスタは便利です。GRに16ビットデータがロードされると、低位および高位バイトをそれぞれGRLおよびGRHレジスタを通じて検索することができます。GRSレジスタはGRのバイトスワップバージョンを備え、また上位バイトは低位バイトの符号拡張である点を除いてGRXLレジスタはGRLレジスタと同じです。

ベースポインタ(Base Pointer)レジスタ(BP)は、MAXQアーキテクチャ内の3個のデータメモリポインタレジスタの1つであり、オフセットレジスタをサポートする唯一のレジスタです。BPは通常、データ構造の基本を指定し、8ビット無符号オフセットレジスタは構造内のデータエレメントを指定します。なお、このレジスタのインクリメント/デクリメントバージョンはオフセットレジスタのみを変更し、ベースレジスタを変更しません。

モジュール15—データポインタ

モジュール15は、MAXQアーキテクチャの3個のデータポインタのうちの2個を備えています。サブデコードに応じて、このモジュールへのアクセスによって、直接または間接ロードや格納が実行され、場合によっては間接アクセスの後にデータポインタがインクリメントまたはデクリメントされます。これらのレジスタサブデコードをソースまたはデスティネーションレジスタとして使用することができます。

MAXQアーキテクチャがその他の従来型CPUと異なっているのは、MAXQコアの転送トリガ特性です。

MAXQのプロセッサはすべてメモリ管理ユニット(MMU)を内蔵しているため、いずれのメモリセグメントもコードやデータとして扱うことが可能であり、またコードバスとデータバスは分離独立しています。

MAXQプロセッサはすべて、これらの追加レジスタアドレスビットを提供し、ワード幅のmove(転送)の上位バイトを提供するプレフィックスレジスタを備えています。

モジュール7—ブール変数操作

ブール変数操作(BVM)モジュール(モジュール7)によって、標準的なMAXQプロセッサにおいて多数のレジスタのビット抽出とビット設定/クリアが可能です(図7)。なお、すべてのモジュールがBVMマシンと接続されているわけではありません。通常は、ペリフェラルモジュールのみがBVMと接続され、システムレジスタは未接続です。このため、BVMとシステムレジスタ間のデータの転送は、おそらく予測不可能な結果をもたらします。

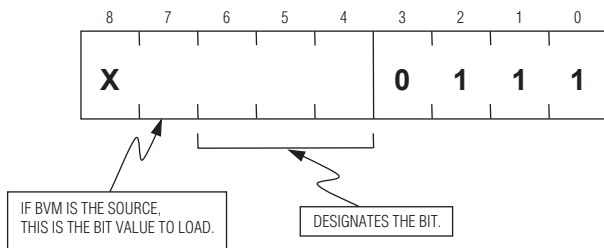


図7. モジュール7のサブデコードは、ソース指定子、即ビット値の場合、抽出または置換するビットを指定します。

デスティネーション指定子としてBVMはキャリービットのプロキシとして機能します。ソースの1ビットが抽出され、キャリービットにコピーされます。BVMがソース指定子の場合は、サブデコードのビット3(完全ソース指定子のビット7)に与えられた値がデスティネーションの指定ビットにコピーされます。

なお、BVMは、ペリフェラルレジスタのビット0~7でのみ動作します。これには大部分のペリフェラルレジスタが対応可能です。多くのレジスタ(特にI/Oポート)はわずか8ビット長であるためです。ただし、16ビットペリフェラルレジスタにアクセスする場合は、下位8ビットのみを利用可能です。

モジュール11—プレフィックス

プレフィックスモジュールは、あらゆる16ビットマイクロコントローラの制限に対処するMAXQアーキテクチャの独自機能です。16ビットレジスタの場合、即値ロード命令には16ビットオペランドが必要です。これは、有効な即値ロード命令には16ビット以上が必要であることを意味します。

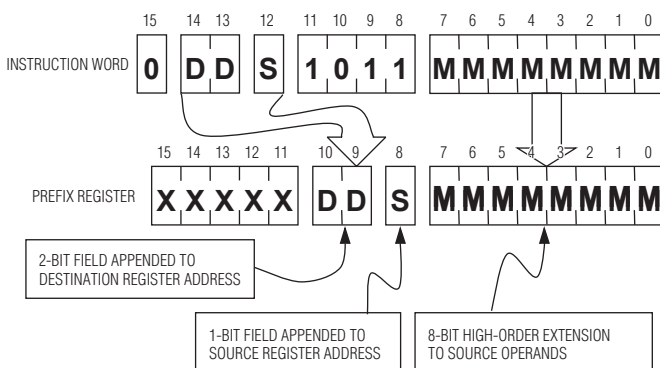


図8. プレフィックスレジスタがデスティネーションの場合、8ビット即値ソースは16ビット即値オペランド用に高位バイトを提供し、デスティネーションサブデコードはソースおよびデスティネーションオペランド用にモジュールごとに32個の全レジスタのアドレス指定を可能にする追加ビットを提供します。

下位および上位バイトへの個別のアクセスを可能にするレジスタや可変長命令など、この制限のソリューションが複数あります(MAXQ GRレジスタはこの一例です)。各ソリューションはデコードロジックを複雑化したり、または新しいレジスタを必要とするため、いずれのソリューションも理想的ではありません(図8)。

プレフィックスメカニズムを通じて、2通りの方式でこのプロセスが向上します。第1に、追加ビットが特に必要な命令のみをプレフィックスすると、このメカニズムはコード空間と実行時間を節減します。第2に、即値オペランド用のほかに、レジスタ指定子用の追加ビットを備えると、このメカニズムはレジスタ空間のサイズを拡張しながら、アーキテクチャ全体を保持します。

レジスタモジュール当り32個のレジスタがありますが、4ビットのみでソースレジスタを指定し、3ビットのみでデスティネーションレジスタを指定していることに留意してください。プレフィックスメカニズムを通じて、これらの追加ビットが提供されます。

プレフィックスメカニズムは、いくつかの点でユニークです。第1に、命令のデスティネーション部の特定ビットを即値ソースビットとして使用して、ソースアドレス用に15を上回るレジスタサブデコードと、デスティネーションアドレス用に7を上回るレジスタサブデコードにアクセスすることができます。このように、単一のプレフィックス命令によって、任意のレジスタまたは即値から任意のレジスタサブデコードにアクセスすることができます。

第2に、プレフィックスレジスタにロードされる値は1クロックサイクルの間しか存続しないため、プレフィックスレジスタはユニークです。その後、レジスタはゼロに自動的にクリアされます。すなわち、プレフィックスレジスタへの転送は、プレフィックスレジスタで変更される命令の直前の命令である必要があります。また、プレフィックス命令は割込み不可能であることも意味します。割込みがプレフィックス処理に続いて発生した場合は、割込みがメイン関数に戻るとプレフィックス情報は消失します。

図9に示すように、プレフィックスレジスタのビットはソース指定子、デスティネーション指定子、および即値に移動します。このため、ほとんどの命令はシングルサイクルで実行されますが、以下の命令には2サイクルが必要です。すなわち、7を上回るデスティネーション

レジスタサブデコードに対処する命令、15を上回るソースレジスタサブデコードに対処する命令、または255を上回る即値をロードする命令です。

このプロセスを示すには、命令「`move A[0], #010h`」を考察してください。これは即値をモジュール9のレジスタ0に転送し、アセンブラは0910という命令コードを生成します。ただし、命令が「`move A[10], #0320h`」である場合は、アセンブラはプレフィックス命令2B03 2920を自動的に挿入する必要があります。

プレフィックス命令がない場合は、命令コード2920は、「`move A[2], #020h`」に変換されます。ただし、プレフィックスによって、1ビットがデスティネーション指定子に、追加ビットが即値に追加され、プロセッサが2サイクルを上回るサイクルを必要とせずに、任意の値を任意のレジスタサブデコードにロードすることができます。

まとめ

たとえMAXQコアが小型で一見単純でも、転送トリガアーキテクチャによって速度と柔軟性は抜群です。ペリフェラルはレジスタインタフェースを通じて直接アドレス指定されるため、組込みペリフェラルを通じたデータ転送の速度は驚異的になります。結局、いずれのMAXQコアも、広範囲のマイクロコントローラアプリケーションに最適です。

この記事の詳述版については、以下を参照してください japan.maxim-ic.com/AN3960 (英文)。

MAXQはMaxim Integrated Products, Inc.の登録商標です。

たとえMAXQコアが小型で一見単純でも、転送トリガアーキテクチャによって速度と柔軟性は抜群です。

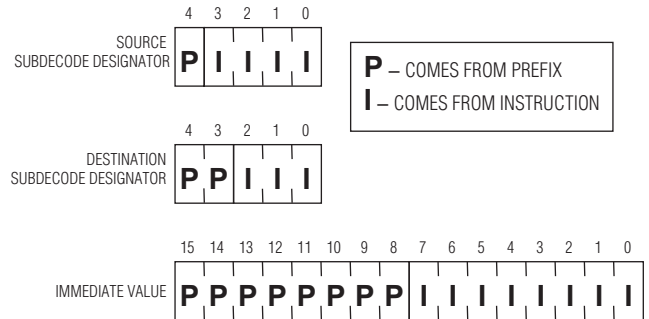


図9. プレフィックスレジスタは、16ビット即値オペランドに必要な、各モジュールの32個の全レジスタをソースおよびデスティネーションとしてアドレス指定する追加ビットを備えています。

SDメディアフォーマットによる MAXQ2000の不揮発性データ ストレージ空間の拡張

MAXQ2000はフラッシュメモリ内に不揮発性データを格納し、ユーザコード空間と共用する32kワード(64kB)のフラッシュ容量を備えています。

低電力で低ノイズのMAXQ2000マイクロコントローラは、幅広いアプリケーションに適しています。MAXQ2000はフラッシュメモリ内に不揮発性データを格納し、ユーザコード空間と共用する32kワード(64kB)のフラッシュ容量を備えています。しかし、読者のアプリケーションがもっと多くの不揮発性ストレージを必要とする場合はどうしますか?この記事では、Secure Digital (SD)メディアフォーマットによってMAXQ2000の不揮発性データストレージを拡張する方法について検証します。

外部ストレージ設計の注意点

アプリケーションに関する第1の設計注意事項は、電源電圧および電流要件です。標準的なMAXQ2000のアプリケーションでは、デュアルリニアレギュレータを使って、選択した設計クロック速度に必要な最低電圧でプロセッサコアの電圧(V_{DD})を動作させます。MAXQ2000の V_{DD} 電源電圧は、1.8Vまで低くすることができます。MAXQ2000のI/O端子は V_{DDIO} から供給され、 V_{DD} の許容下限範囲と3.6Vの上限を備えています。外部ストレージの許容電流消費量は電源の電流定格によって規定され、バッテリー駆動デバイスの場合は、バッテリーシステムの容量によって規定されます。

第2に、外部ストレージ接続用のMAXQ2000のI/Oラインの数は、対象アプリケーションに十分な帯域幅を提供する一方で、最低本数にしておく必要があります。たとえば、Atmel社のフラッシュチップAT29LV512は、ホストマイクロコントローラとインタフェースする場合には、15本のアドレスライン、8本のデータライン、および3本の制御ラインを必要とします。MAXQ2000は外部アドレス/データバスを備えていないため、この場合、ソフトウェアがバス伝送を制御する必要があります。一部のアプリケーションでは、この方式はMAXQ2000のI/O端子の効率的な使い方ではありません。

ただし、SPI™およびI²Cベースの外部フラッシュデバイスには、3または4つのインタフェース端子しか必要ありません。MAXQ2000はハードウェアSPIモジュールを備えていますが、ユーザがソフトウェアでMAXQ2000にI²Cを実装する必要があります(すなわち、「ビットバンキング」)。この統合能力は、SPIインタフェースは外部の不揮発性ストレージにアクセスするのに不可欠な手段であることを意味します。

SDメモリーカードフォーマット

SDメディアフォーマットは、多くのアプリケーションに上記の注意事項を満たす不揮発性外部メモリです。SDフォーマットは、「MultiMedia Card」フォーマット、すなわちMMCの後継フォーマットです。SDカードのメモリは、緩やかな電流要件で標準3.3Vの電源電圧で動作します。SDカードの容量の範囲は、数メガバイトから最大4GBまでです。この広範囲の利用可能な容量によって、多くのアプリケーションに十分な外部ストレージがもたらされます。

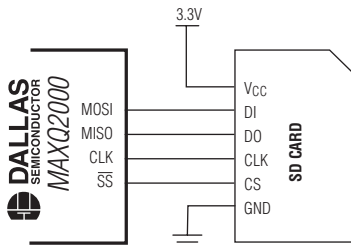


図1. MAXQ2000はSDメモリーカードと容易にインタフェースします。

一見したところ、SDはSD独自の共用バスのためにMAXQ2000とインタフェースするのは容易ではないように見えるかもしれません。しかしながら、SDはMMCの第2バス形式SPIを継承しています。したがって、MAXQ2000はSPI用のハードウェアサポートを備えているため、インタフェースするのは容易です。

図1の回路図は、標準動作回路を示しています。SDカードは、フルデュープレックス、8ビットSPI動作を必要とします。データはMAXQ2000のMOSI端子からカードのDI端子にクロック入力され、カードのDOラインからMAXQ2000のMISO端子にクロック出力されます。データは、CLKラインの立上りエッジで、カードとの間で同時にクロック入力および出力されます。8個の追加クロックを各トランザクションの最後に供給し、SDカードが未処理の動作を完了にする必要があります。これらの追加クロックの間の入力データは、すべて1である必要があります。クロック速度は、識別フェーズ時には最高400kHzに制限する必要がありますが、SDカードが識別されると最高25MHzまで上げることができます。

MAXQ2000 SPIモジュール

MAXQ2000は、SDカードインタフェース用に容易に設定されるハードウェアSPIモジュールを備えています。クロック極性とデータ長を設定するには、SPICFレジスタをすべてゼロに設定します。これによって、SPIモジュールがクロックの立上りエッジでデータをラッチするように設定され、データ長が8ビットに設定されます。このアプリケーションの場合は、MAXQ2000のシステムクロック周波数は16MHzです。この場合、SPICKレジスタは0x28に設定され、SPIクロックは約380kHzになります。SPICNレジスタの下位2ビットを設定して、SPIマスターモードをイネーブルする必要があります。

MAXQ2000は、SDカードインタフェース用に容易に設定されるハードウェアSPIモジュールを備えています。

SDのSPIデータフォーマット

SDカードのSPIプロトコルは、SDバスプロトコルに類似しています。全クロックエッジでSDカードのDO端子から有効データを受信せずに、送信するデータがないカードは、DO端子をすべて1のアイドル状態に保持します。カードがホストに返送するデータを持っている場合は、スタートビットがゼロの専用トークンがそのデータの前に送信されます。SDカードから送信されるすべてのデータは、これらのトークンの直後に送信され、固定長です。受信側が预期するバイトの数を事前に認識しているため、バイト長は応答には含まれません。さらに、開始トークンおよびデータの送信が終了するまでアイドル状態は発生しないため、すべてのデータバイトは変更なしで、プレフィックスを伴わずに送信されます。バス上のその他のトラフィックと共に、トークンはSPIトランザクションの8ビット境界で整合されます。ホストからカードへのコマンドおよびデータは同様の形式に従い、すべて1はアイドルバスを示します。ステータストークンを除く全トランザクションは、データの最後に付加される巡回冗長検査(CRC)コードによって保護されます。データの短ブロック用のCRC-7と、データの長ブロック用のCRC-16という2つのCRCアルゴリズムが提供されます。CRCはSD SPIインタフェースのオプション部分ですが、アプリケーションの制約がその利用を妨げない限り、このCRCによってデータ完全性を保証する必要があります。

SDのコマンドフォーマット

コマンドは、6バイトフォーマットでカードに送出されます(図2)。16進数0x40付きの6ビットコマンドコードをOR-ingして、コマンドの先頭バイトを構築することができます。コマンドが要求する場合は、その次の4バイトは単一の32ビット引数を備え、最終バイトは、バイト1~5に対するCRC-7チェックサムを備えています。表1は、重要なSDコマンドを一覧表示しています。

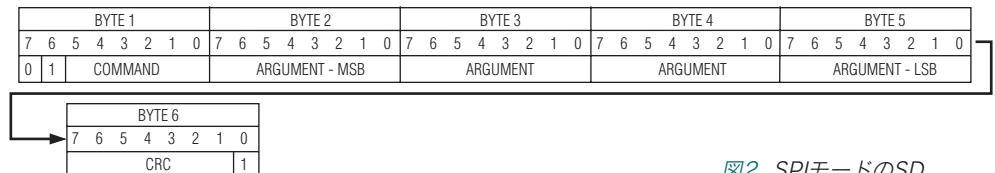


図2. SPIモードのSDコマンドは、6バイト形式でカードに送出されます。

SPIモードでのSDカードの初期化

電源投入時に、SDカードは独自のSDバスプロトコルにデフォルト設定されます。カードをSPIモードに切り替えるには、ホストがコマンド0 (GO_IDLE_STATE)を送出します。このコマンドやその他の全SPIコマンドに対してカードの選択(CS)端子がローに保持されるため、SDカードはSPIモードの選択を検出します。カードは、応答形式R1で応答します(図3)。アイドル状態ビットはハイに設定され、カードがアイドル状態に入ったことを示します。MMCカードとの互換性を維持するために、SPIクロック速度はこの段階で400kHzを超えてはいけません。

表1. 主なSDメモ리카ードのコマンド

コマンド	ニーモニック	引数	応答	説明
0 (0x00)	GO_IDLE_STATE	<なし>	R1	SDカードをリセット
9 (0x09)	SEND_CSD	<なし>	R1	カード固有のデータを送信
10 (0x0a)	SEND_CID	<なし>	R1	カード識別を送信
17 (0x11)	READ_SINGLE_BLOCK	アドレス	R1	バイトアドレスのブロックを読み取り
24 (0x18)	WRITE_BLOCK	アドレス	R1	バイトアドレスのブロックに書き込み
55 (0x37)	APP_CMD	<なし>	R1	アプリケーションコマンドのプレフィックス
59 (0x3b)	CRC_ON_OFF	ビット0のみ	R1	引数がCRCオン(1)またはCRCオフ(0)を設定
41 (0x29)	SEND_OP_COND	<なし>	R1	カードの初期化を開始

一見したところ、SDは独自の共用バスのためにMAXQ2000とインタフェースするのは容易ではないように見えるかもしれませんが、しかしながら、SDはMMCの第2バス形式のSPIを継承しています。

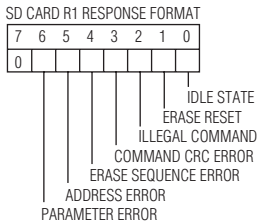
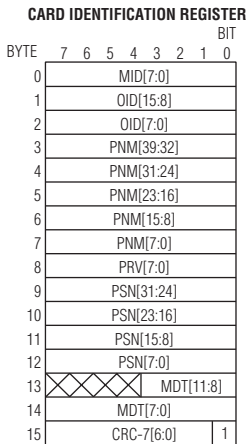


図3. 応答形式R1は、送出されたコマンドの成功または失敗を通知します。

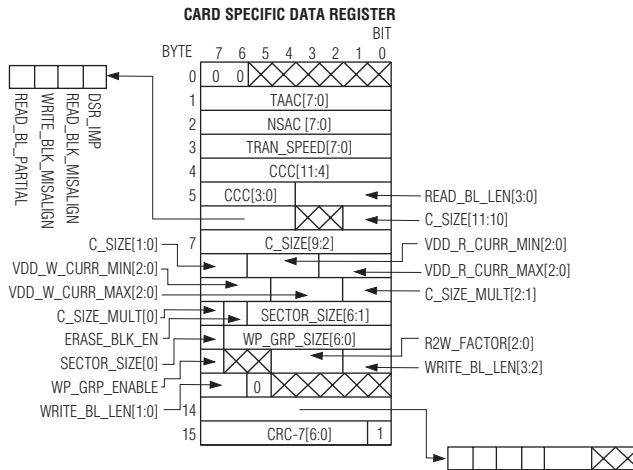
SDカードがSPIモードになると、SDの仕様は、その他の要求が処理可能となる前に、ホストが初期化コマンドを送出することを必要とします。MMCカードとSDカードを識別するために、SDカードは、MMCカードが応答しない別の初期化コマンドを実装します。コマンド55 (APP_CMD)、続いてアプリケーションコマンド41 (SEND_OP_COND)をカードに送出すると、この重要なステップが終了します。MMCカードはコマンド55に回答せず、このコマンドを使ってMMCカードを無効メディアとして拒否することができます。カードからのR1の応答における全ビットがゼロになる(すなわち、IDLEビットがローになる)まで、このコマンドシーケンスが繰り返されます。

```
while(status && (errors < retries)) {
    printf("-> Send CMD55_APP_CMD\r\n");
    xmitcmd(CMD55_APP_CMD, arg);
    if (waitForR1(&rxdata, 0) < 0) {
        /* If this is a MultiMediaCard (not SD), it will not respond here */
        printf("ERROR: Timeout! Perhaps this is a MMC card?\r\n");
        return TR_TIMEOUT;
    }
    check_r1(rxdata, R1_IDLE);

    printf("-> Send ACMD41_SEND_OP_COND\r\n");
    xmitcmd(ACMD41_SEND_OP_COND, arg);
    if (waitForR1(&rxdata, 0) < 0) {
        printf("ERROR: Timeout on ACMD41_SEND_OP_COND\r\n");
        return TR_TIMEOUT;
    }
    status = rxdata & R1_IDLE;
    if (status) {
        /* Pause here for a bit to let the card start up */
        for (i = 0; i < 10000; i++); /* busy loop */
    }
}
}
```



NAME	TYPE	DESCRIPTION
MID	BINARY	MANUFACTURER ID
OID	ASCII	OEM/APPLICATION ID
PNM	ASCII	PRODUCT NAME
PRV	BCD	PRODUCT REVISION
PSN	BINARY	SERIAL NUMBER
MDT	BCD	MANUFACTURER DATE CODE
CRC-7	BINARY	CRC-7 CHECKSUM



LEGEND

☒ DENOTES RESERVED BIT LOCATION. SHOULD BE ZERO.

SDカードは、SDカードに関する情報を提供する重要なレジスタを複数備えています。最も重要なレジスタは、カード固有データ (Card Specific Data) レジスタ (CSD) です。サンプルアプリケーションの場合は、メモリのブロックサイズと合計サイズに留意しています。カード識別データ (Card Identification Data) レジスタ (CID) はカードのメーカーおよびシリアル番号に関する詳細を含んでいるため、このレジスタにも留意する必要があります。図4は、CSD およびCIDレジスタのレイアウトを示しています。

SDカードの応答の検証

カードのレジスタまたはブロックをカードから読み取るには、まずカードがどのように照会に回答するかを理解する必要があります。SPIモードでは、SDカードは、コマンド SEND_CSD (9)、SEND_CID (10)、およびREAD_SINGLE_BLOCK

図4. CSDおよびCID レジスタは、SDカードに関する情報を備えています。

(17)にR1形式の応答で応答します。開始トークン、要求されたデータ、そして最後にデータに対するCRC-16チェックサムが続きます。R1応答とデータ開始トークンが次々とすぐに実施されると想定してはいけません。というのは、バスはこれらの2つのイベントの間にしばらくの間、アイドル状態に移行する可能性があるためです。図5は、データ応答を詳述しています。

CSDおよびCIDレジスタの メタデータ読み取り

SEND_CSDおよびSEND_CIDコマンドは、SDカードパラメータの設定に使用されるレジスタ内容を返送します。これらのコマンドは固定数のバイトを返し、これはそれぞれCSDまたはCIDレジスタのサイズに対応します。コマンドバイト内に含まれる引数は、これらのSENDコマンドのためにSDカードによって無視されます。

SDカードからのデータブロックの 読み取り

SDカードからのデータブロックの読み取りはごく簡単です。ホストは、引数として開始バイトアドレスとともにREAD_SINGLE_BLOCKコマンドを送出します。このアドレスは、メディアのブロックの先頭と整合される必要があります。次に、SDカードはこのバイトアドレスを評価し、R1コマンド応答で返答します。範囲外のアドレスは、このコマンド応答で示されます。

SDメディアからの読み取りがエラーなしで終了すると、開始データトークンが送信され、その後固定数のデータバイトとCRC-16チェックサム用の2バイトが続きます。SDカードにハードウェア障害やメディア読み取りエラーが発生すると、開始データトークンは送信されません。代わりに、エラートークンが送信され、データ転送が中止されます。

SDカードへのデータブロックの書き込み

ホストは、SDカードブロック境界に整合されたバイトアドレスを提供する必要があり、データブロックの書き込みは読み取りと類似しています。書き込みブロックサイズはREAD_BL_LENと等しい必要があり、通常は512バイトです。書き込みはWRITE_BLOCK (24)コマンドの送付で開始され、このコマンドに対してSDカードはR1コマンド応答形式で応答します。書き込みが続行可能であるとコマンド応答が示すと、ホストはデータ開始トークン、続いて固定数のデータバイトを送信し、送信データのCRC-16チェックサムを終了します。SDカードは、書き込まれるデータの受諾または拒否を示すデータ応答トークンを返します。

データが受理されると、SDカードがビジーの間は、SDカードはDOラインを連続的にローに維持します。ホストは、ビジー期間にカードの選択をローに維持する必要はありませんが、CSがデアサートされると、SDカードはDOラインを解放します。複数のデバイスがSPIバスに接続されている場合に、このプロセスは便利です。ホストは、SDカードがビジー表示を解放するのを待機したり、またはチップ選択を定期的のアサートしてカードをチェックすることができます。カードが引き続きビジーの場合は、DOラインをローにプルしてこの状態を表示します。ビジー状態でない場合は、カードはDOラインをアイドル状態に戻します(図6を参照)。

SPIコマンドおよびデータエラー検出

CRC-7およびCRC-16チェックサムを使って、ホストとSDカード間の通信のエラーを検出することができます。エラー検出によって、着脱時の接点のはね返りや、着脱可能なメディアに特有の理想的でない接触保持状態などの物理的に誘発されたエラーが発生した場合のエラー回復が強化されます。最下位ビットを引数に設定したCRC_ON_OFF (59)コマンドを送出して、チェックサムを利用することを強く推奨します。

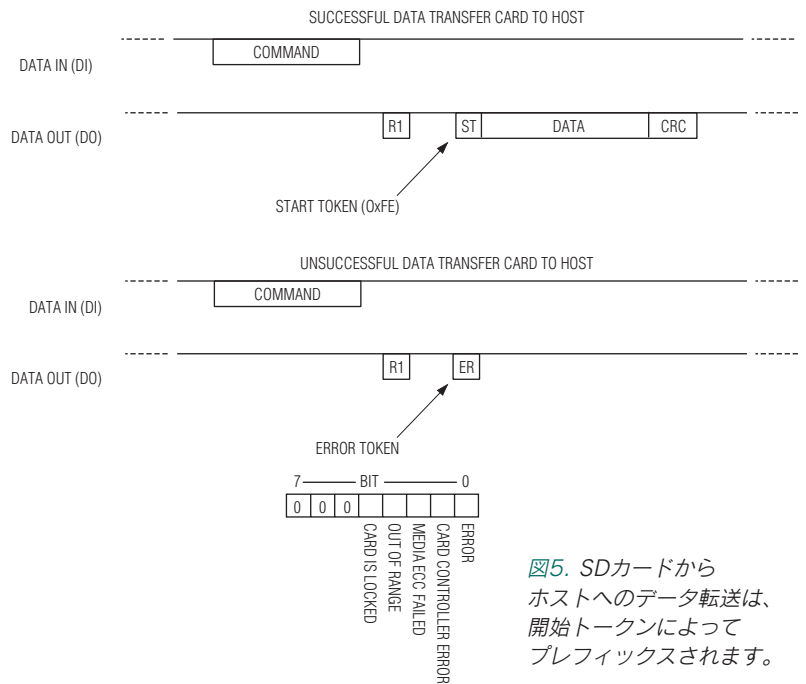


図5. SDカードから
ホストへのデータ転送は、
開始トークンによって
プレフィックスされます。

クロックレートは、
識別フェーズ時には
最高400kHzに制限する
必要がありますが、
SDカードが識別されると
最高25MHzまで上げる
ことができます。

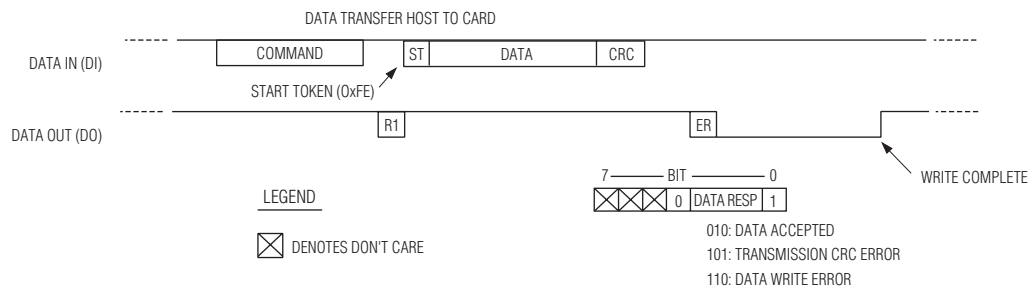


図6. ホストからSDカードへのデータ転送には、より複雑なハンドシェイクが必要です。

```

/* Enable CRC to protect against error */

printf("-> Send CMD59_CRC_ON_OFF\r\n");
arg[3] = 0x01; /* LSB set to 1 enables CRC verification */
xmitcmd(CMD59_CRC_ON_OFF, arg);
CLEAR_ARGS(arg);
if (waitForR1(&rxdata, 0) < 0) {
    printf("ERROR: Timeout on CMD59_CRC_ON_OFF\r\n");
    return -1;
}
if (rxdata != 0x00) {
    printf("WARNING: R1 status 0x%02x, expecting 0x00\r\n", rxdata);
}

```

MAXQ2000マイクロコントローラから提供されるハードウェアSPIのサポートによって、ほんの少しのオーバーヘッドでSDメディアカードにアクセスすることができます。

まとめ

SDメディアカードフォーマットは、組み込みシステム用の小型で低電力の不揮発性メモリソリューションです。MAXQ2000マイクロコントローラで提供されるハードウェアSPIのサポートによって、ほんの少しのオーバーヘッドでSDメディアカードにアクセスすることができます。japan.maxim-ic.com/MAXQ2000_SD (英文)でMaximから提供されるリファレンスソフトウェアは、最小限の実装を検証し、これには、SDカードとの間のブロックの読み取りと書き込みに不可欠な動作が含まれています。

参考資料

SDメディアフォーマットの詳細については、Secure Digital Association (www.sdcard.com/japan)から入手することができます。このプロジェクトで使用されたSDメディアブレイクアウトボードをwww.sparkfun.com/commerce/product_info.php?products_id=204 (英文)に注文することができます。

この記事の詳述版については、以下を参照してください：japan.maxim-ic.com/AN3969 (英文)。

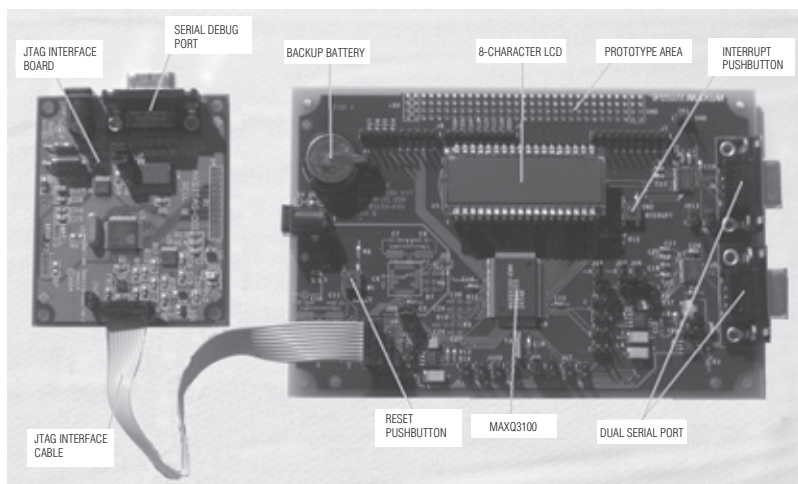
SPIはMotorola, Inc.の商標です。

Rowley CrossWorksとMAXQ3100*の評価キットによる温度ロギングアプリケーションの作成

この記事では、Rowley Associates社のCrossWorksとMAXQ3100の評価キット(EVキット)を使って、簡単な温度ロガーを作成します。MAXQ3100は温度センサを内蔵しているため、部品の追加は不要です。

CrossWorks for MAXQは、MAXQプラットフォームでCアプリケーション用のフル機能搭載の開発環境を提供します。CrossWorksは、ANSI準拠Cコンパイラ、MAXQアーキテクチャに最適化されたアセンブラ/リンカ、および大部分のMAXQマイクロコントローラに搭載されるJTAGインタフェースとハードウェアベースのデバッグエンジンとインタフェースするように設計されたデバッグを備えています。これらのすべての構成要素はプロジェクトベースの開発環境に統合され、この環境は包括的なヘルプシステム、組み込みサポート、およびMAXQ2000やMAXQ3100などのMAXQマイクロコントローラ用のサンプルを備えています。なお、Rowley CrossWorks for MAXQは現在、Windows®プラットフォーム用のみ提供されています。www.rowley.co.uk/maxq/index.htm(英文)から、30日間の評価ライセンスをダウンロードしてください。

CrossWorks for MAXQは、MAXQプラットフォームでCアプリケーション用のフル機能搭載の開発環境を提供します。



MAXQ3100のEVキットはMAXQ3100の主要機能を検証する部品を内蔵し(japan.maxim-ic.com/MAXQ3100を参照)、このキットを使って、カスタムハードウェア設計を進行しながら、ソフトウェア開発にすぐに着手することができます。このキットは以下の機能を備えています。

図1. MAXQ3100のキットとシリアル-JTAGボードが統合して、アプリケーション開発用の完全なシステムを構築します。

- 32kHz水晶付き、80ピンMQFPパッケージのMAXQ3100マイクロコントローラ
- PCホストからのデバッグおよびプログラミング用のシリアル-JTAGインタフェース
- JTAGインタフェースから直接MAXQ3100を駆動するための3.6Vリニアレギュレータ
- 1.8V~3.6Vを出力する手動調整可能なリニアレギュレータ
- MAXQ3100で直接駆動される8キャラクタ(1キャラクタ当たり14のセグメント)のLCD 1/4デューティディスプレイ
- DB9コネクタ付きの2個のレベルシフトシリアルポート
- リセットおよび外部割込み用プッシュボタン
- MAXQ3100ポート端子、コンパレータ入力、および電源にアクセス可能なプロトタイプ領域

シリアル-JTAGインタフェースボード(図1)と統合することで、このシステムはMAXQ3100のインシステムブートローダおよびデバッグ機能に完全アクセスすることができます。

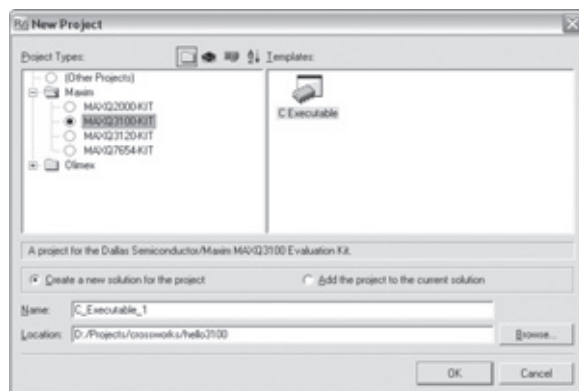


図2. 新規プロジェクトを作成する最初のステップは、対象デバイスおよび実行可能タイプを選択することです。

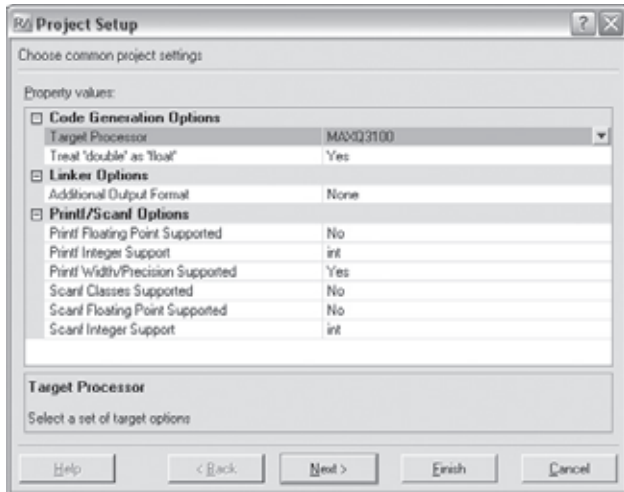


図3. MAXQ3100が、このプロジェクトの対象プロセッサです。

Rowley Crossworksでのプロジェクトの作成

製品のアクティベーションが完了したら(Rowley社のウェブサイト「サポート: CrossWorksの評価(Support: Evaluating CrossWorks)」を参照)、CrossWorksを起動します。新規プロジェクトを作成するには、メニューから[File]→[New]→[New Project]の順に選択します。[New Project]ダイアログで(図2)、[MAXQ3100 Kit]と[C Executable]を選択し、新規プロジェクトの名前と場所(保存先)を入力します。続く[Project Setup]ダイアログ(図3)で、MAXQ3100が「Target Processor」オプションに選択されていることを確認します。残りの設定をデフォルト値のままにすることができます。[Finish]をクリックして、新規プロジェクトを作成します。

アプリケーション概要

MAXQ3100の検証例は、周囲温度センサ、LCDコントローラ、リアルタイムクロック(RTC)、およびUARTインタフェースという4つのプロセッサの主要機能を示します。これらのペリフェラルを使って、温度レベルを測定し、LCDに時刻と温度を表示して、UARTインタフェースを通じてデータをPCに返送するアプリケーションを作成することができます。この例で使われる全コードは、japan.maxim-ic.com/MAXQ3100_temp_logger (英文)においてオンラインで入手可能です。

MAXQ3100のプログラムメモリはワードで再書き込み可能なEEPROMで実装されるため、このアプリケーションは残っている未使用プログラムメモリを使って、時刻および温度ログを格納することができます。MAXQ3100のユーティリティROMは、ソフトウェア制御のもとでEEPROM部に再書き込みし、消去可能なインアプリケーションプログラミングルーチンを備えています。

温度の測定

MAXQ3100の温度センサは、周囲温度を最低0.0625℃の分解能まで測定します。サンプル戻り値の幅は、10ビット(0.5℃分解能)から13ビット(0.0625℃分解能)まで選択可能です。

```
void
convertTemp(void)
{
    IC = 0;           // Disable interrupts

    TPCFG = 0x06;    // Set resolution to 13 bits
    TPCFG = 0x07;    // Start conversion
    while ((TPCFG & 0x01) == 0x01) {}    // Wait for conversion to complete
    g_lastTemp = TEMPR;    // Store temperature value

    IC = 1;         // Enable interrupts
}
```

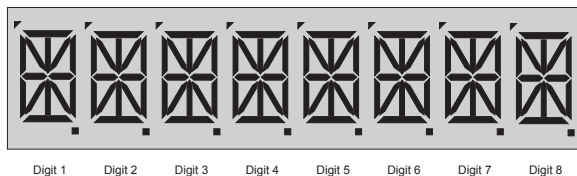
温度が測定されると、摂氏または華氏でその温度を浮動小数点値に変換することができます。

温度値の表示

MAXQ3100のEVキットは図4に示すように設定される8キャラクタのLCDを搭載しています。このLCDは、現在温度の読取り値を小数第2位まで示すのに十分過ぎる表示スペースを備えています。

LCDのキャラクタは14セグメントであるため、標準の0~9やアルファベット文字とともに、プラス、マイナス、および度などの特殊キャラクタを容易に表示することができます。最初のステップは、LCDコントローラの初期化です。

LCDが初期化されると、任意の8桁を、サポートされる任意のキャラクタ値に設定するコードを追加することができます。LCDセグメントはディスプレイレジスタLCD0~LCD15 (表1)にマッピングされるため、ディスプレイ上にキャラクタを表示することは、単に値を適切なレジスタ位置に書き込むことのみです。



2個のレジスタを使って各キャラクタのセグメント値を格納することができます。これには小数点およびインジケータアンシエータが含まれます。LCDコントローラの動作中に、これらのレジスタを変更することができます。

LCD上の小数点の設定とクリアを行う同様のルーチン(displayDP)が用意されています。これらのルーチンを導入すると、測定温度値を適切なディスプレイ桁に変換することができます。このアプリケーションは温度変換間の固定遅延を備えているため、ディスプレイは1秒当たり数回更新されます。

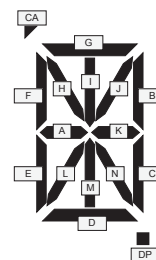


図4. MAXQ3100のEVキットは、アプリケーション用に8キャラクタ、14セグメントLCDを備えています。

クロックの開始と設定

このアプリケーションで、サンプルと測定された時刻と結合する(すなわちペアにすることができるようになると、記録された温度サンプルがより便利になります。MAXQ3100のRTCは32kHzの水晶発振器のクロックに基づいているため、このRTCは各温度サンプルをタイムスタンプするクロック値に適しています。アプリケーションは、要求されるとLCDディスプレイ上にクロックを表示し、1分に1回、時刻と温度のサンプルペアをEEPROMに記録します。

RTCが開始されると、このRTCを使って、プログラマブルな間隔で割り込みをトリガする周期的なアラームを生成することができます。マキシムのアプリケーションはこの間隔アラームによって、1秒に1回、高精度で割り込みを生成します。次に、この割り込みによって、アプリケーションでグローバル秒カウンタをインクリメントします。

CrossWorksでは、`__interrupt`キーワードを関数定義に追加することによって、どの関数も割り込みハンドラとして指定することができます。次に、`setIV()` 組込み関数を使って、割り込みハンドラ関数のエントリポイントをプログラマブルな割り込みベクトルレジスタ(IV)にロードすることができます。MAXQ3100の全割り込みはIVレジスタ内のアドレスへのベクトルとして処理されるため、1つの割り込みハンドラ関数を定義するのみです。このアプリケーションは割り込みを1つだけ使用しますが、他の割り込みを使用する必要がある場合は、割り込みハンドラ関数内で他の割り込みフラグをチェックして、関数呼出しをトリガした割り込みを識別することができます。

表1. LCDディスプレイメモリマップ(1/4デューティ)

レジスタ	ビット7 COM3	ビット6 COM2	ビット5 COM1	ビット4 COM0	ビット3 COM3	ビット2 COM2	ビット1 COM1	ビット0 COM0
LCD0	1D	1E	1F	1CA	1DP	1C	1B	1G
LCD1	1N	1K	1J	1I	1M	1L	1A	1H
LCD2	2D	2E	2F	2CA	2DP	2C	2B	2G
LCD3	2N	2K	2J	2I	2M	2L	2A	2H
LCD4	3D	3E	3F	3CA	3DP	3C	3B	3G
LCD5	3N	3K	3J	3I	3M	3L	3A	3H
LCD6	4D	4E	4F	4CA	4DP	4C	4B	4G
LCD7	4N	4K	4J	4I	4M	4L	4A	4H
LCD8	5D	5E	5F	5CA	5DP	5C	5B	5G
LCD9	5N	5K	5J	5I	5M	5L	5A	5H
LCD10	6D	6E	6F	6CA	6DP	6C	6B	6G
LCD11	6N	6K	6J	6I	6M	6L	6A	6H
LCD12	7D	7E	7F	7CA	7DP	7C	7B	7G
LCD13	7N	7K	7J	7I	7M	7L	7A	7H
LCD14	8D	8E	8F	8CA	8DP	8C	8B	8G
LCD15	8N	8K	8J	8I	8M	8L	8A	8H

CrossWorksでは、`__interrupt`キーワードを関数定義に追加することによって、どの関数も割り込みハンドラとして指定することができます。

RTCが開始されると、このRTCを使って、プログラマブルな間隔で割込みをトリガする周期的なアラームを生成することができます。

割込みハンドラ関数は、秒(0~59)、分(0~59)、および時(1~12)カウンタを概算でインクリメントします。秒カウンタがロールオーバーするごとに(1分に1回)、関数はasm_writeLogを使って、現在時刻と最後に測定された温度値を不揮発性EEPROMログに記録します。また、秒の割込みがトリガされるごとに、LCDディスプレイの左端のカレット記号(7桁目の左)が切り替わるため、ハートビートインジケータがもたらされます。現在時刻が表示されていなくても、この動作は行われます。

EEPROMのログの開始点は定数(LOG_START)で設定されます。この定数は、コンパイル済みアプリケーションの末尾を超えた領域に設定する必要があります。たとえば、このアプリケーションの現行バージョンは8972バイトのコードバイトを占有し、これはロードされたアプリケーションがプログラムメモリ内のワードアドレス0000h~1185hを実行することを意味します。LOG_STARTを1800hに設定すると、ログはアプリケーションの末尾をはるかに超え、ログ用に(2000h - 1800h) = 2048ワードのEEPROMが確保されます。2ワードがエントリごとに使用されるため、アプリケーションは1048組の時刻/温度値のペアを記録することができます。1分につき1つのログエントリで、約17時間連続の定期ログが得られます。また、新たなログエントリが書き込まれるごとに、最も古いエントリが削除されます。

通常は、CrossWorksは、割込みハンドラルーチンからもたらされるあらゆる潜在的な問題に対処します。割込みがトリガされると、現在実行中のルーチンの内容とワーキングレジスタが保存されます。この情報は、割込みハンドラが終了すると、復元されます。ただし、以下の場合に注意する必要があります。

- 割込みハンドラルーチンがアプリケーションの他の場所で使用されているグローバル変数にアクセスすると、こうした同じ変数にアクセスする他のコードのセクションは以下の2つのことを実行する必要があります。すなわち、変数の読み書きの前に割込みをディセーブルにして、処理が終了したら割込みをイネーブルすることです。
- MAXQ3100のローレベルレジスタで直接処理を実行するコードを実装する際には注意してください。割込みがトリガされたときに、CrossWorksはこれらのレジスタの状態を自動的に保存または復元しません。たとえば、割込みがシリアルポートへのprintf()中に発生した場合は、割込みハンドラ内に、printf()、すなわちシリアルポートレジスタに直接書き込むコードの呼出しが含まれてはいけません。
- 割込み呼出しをトリガしたどの割込みフラグも、割込みハンドラルーチンが終了する前にクリアする必要があります。そうでない場合は、割込みはすぐに再びトリガされます(たとえば、incr_seconds()ルーチンでは、アラームサブセカンドフラグRCNT.7をクリアする必要があります)。割込みフラグは、ハードウェアで自動的にクリアされません。

アセンブリルーチンとの接続

CrossWorksを使って、CコードとともにネイティブなMAXQアセンブリルーチンをアプリケーションにリンクさせることができます。このリンクによって、特化されたアセンブリルーチンをアプリケーションに組み込み、コンパイル済みCコードで実行するのが困難なタスクを実行することができます。マキシムのアプリケーションでは、アセンブリ関数を使って、プログラム空間内の各場所から読み取り、EEPROM内の各場所に書込んだり、消去するMAXQ3100のユーティリティROMルーチンとインタフェースすることができます。

```
int asm_readLog(int addr);
int asm_writeLog(int addr, int wval);
int asm_erasePage(int addr);
```

Cコードから呼び出されるアセンブリルーチンを記述するときには、パラメータと戻り値をどのように受け渡しをするかをまず決定する必要があります。CrossWorksでは、アキュムレータレジスタA[7]~A[4]を使って、最大4つの整数パラメータを関数に渡すことができます。たとえば、関数が以下のように定義されている場合は、

```
int asm_func1(int foo, int bar, int frob, int nitz)
```

この場合、関数へのエントリ時に、A[7]はfooの値に、A[6]はbarの値に、A[5]はfrobに、A[4]はnitzに設定されます。(ロングワードなどの大きな値はレジスタペアに渡されます。A[7]~A[4]に渡すには関数の入力パラメータが多すぎる場合は、パラメータはソフトスタックに渡されます。詳細については、CrossWorksのヘルプシステムを参照してください)。関数からの整数戻り値は、呼出しCコードに戻すためにA[7]に格納されます。

asm_writeLogルーチンには、プログラムメモリに書き込む場所と、そこに書き込むデータワードの2つのパラメータが必要です。アセンブリコードは、これらのパラメータをUROM_eepromWriteWordと呼ばれるユーティリティROM内のアセンブリ関数に渡します。この関数呼出しに続いて、アセンブリコードは終了前に、Cコードで使用されるレジスタが確実に復元されるようにする必要があります。CrossWorksでは、このレジスタセットは、アキュムレータレジスタA[0]~A[3]とA[8]~A[15]を含みます。これらのレジスタのいずれかがアセンブリルーチンで変更されると、ルーチン終了前にそのレジスタを復元する必要があります。また、以下のように値18hにDPCを復元する必要があり、DP[0]がバイトモードに、DP[1]およびBP[Offs]がワードモードに設定されます。

```
urom_eepromWriteWord EQU (0x874E << 1)

public _asm_writeWord

code                ; Code segment
even                ; Align to word boundary
_asm_writeWord:
move    DPC, #0x1C    ; Set all data pointers to word mode
move    DP[0], A[7]   ; DP[0] is the address that will be written
move    A[4], A[0]    ; Save off A[0] (used by C code)
move    A[0], A[6]    ; A[0] is the word value that will be written
move    A[5], A[1]    ; Save off A[1] (used by C code)
move    A[6], A[2]    ; Save off A[2] (used by C code)

call    urom_eepromWriteWord ; Writes word, destroys A[0],A[1],A[2]

move    DPC, #0x18    ; Reset data pointer modes for C code
move    A[2], A[6]    ; Restore A[2]
move    A[1], A[5]    ; Restore A[1]
move    A[0], A[4]    ; Restore A[0]
ret
```

なお、CrossWorksアセンブラでは、UROM_eepromWriteWordルーチンのアドレスはバイトアドレスによって定義されます(874Ehを1だけ左シフト)。これは、プログラム空間アドレスはすべてワードアドレスであると想定するMAXQアセンブラとは異なります。

記録された値をEEPROMから読み戻し(asm_readWord)、32ワードのページをEEPROMから消去する(asm_erasePage)同様の関数が実装されています。これらの関数は同じように実装され、ユーティリティROM関数UROM_moveDP0およびUROM_eepromErasePageを使用します。

ハードウェアスタックは、アセンブリルーチンの実行時に発生するもう1つの問題です。通常、CrossWorksは、データメモリに実装されるソフトCスタックを使って関数呼出し用のパラメータとローカル変数を格納して、ハードウェアスタックの深度を追跡します。マキシムのアプリケーションがCコードのみを使用している限り、ハードウェアスタックを確実にオーバーフローしないようにする責務は、すべてコンパイラにあります。

ただし、アセンブリルーチンが呼ばれても、アセンブリルーチンがどれだけのスタックレベルを必要とするかコンパイラは認識することができません。アセンブリコードのCALL、PUSH/POP、または割込みエントリにはそれぞれ1ワードのスタックワードが必要であり、ユーティリティROMルーチンにもスタックワードが必要です。この場合、スタックオーバーフローの発生を示す割込みやハードウェアインジケータがないため、ユーザの責任でスタックオーバーフローを回避してください。

アセンブリルーチンが使用するスタックレベルの数を決定すると、コンパイラが使用する数を決定することができます。通常、main()からの各サブルーチン呼出しには1つのスタックレベルが必要で、デバッグがアクティブの場合は、デバッグエンジンの呼出しのためにスタックレベルを追加する必要があります。また、スタックレベルの追加が許可されている場合は、コンパイラは追加のスタックレベルを使って、最適化するために反復コードを含むサブルーチンを作成します。この使用を指定数のスタックレベルに制限するには、メニューから[Project]→[Properties]を順に選択します。次に、[Linker]タブを選択し、[Additional Linker Options]フィールドに[-Oxcp=n]を入力します(図5)。これによって、コードファクタリング最適化パスの数をnに制限するようにコンパイラが指示され、使用スタックレベルの最大数は、以下のようになります。

CrossWorksを使って、CコードとともにネイティブなMAXQアセンブリルーチンをアプリケーションにリンクさせることができます。

CrossWorksでは、アキュムレータレジスタA[7]~A[4]を使って、最大4つの整数パラメータを関数に渡すことができます。関数からの整数戻り値は、呼出しCコードに戻すためにA[7]に格納されます。

(main()からのサブルーチン呼出しの最大深度) + (デバッガが使用される場合は、1) + (アセンブリで使用されるスタックレベル) + n

また、このダイアログは、hex出力ファイルをCrossWorksから生成する設定も備え(Additional Output Format = hexに設定)、MTKまたはMAX-IDEを使ってロード可能なコンパイル済みアプリケーションのバージョンが生成されます。

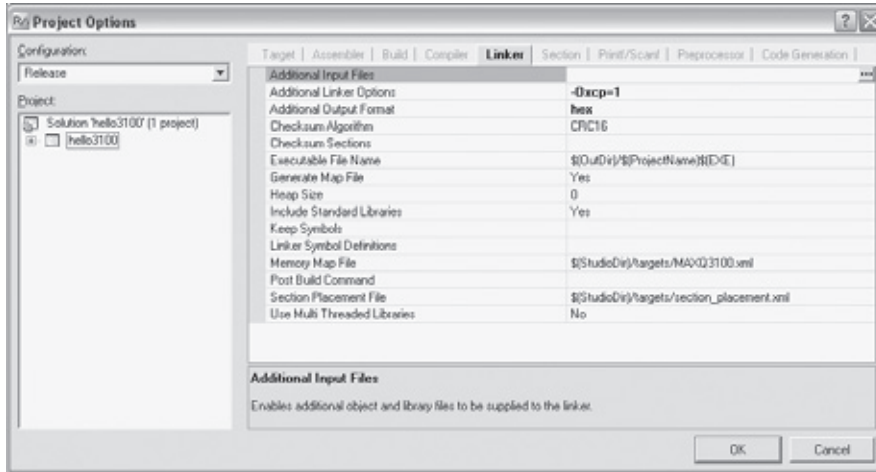


図5. コード最適化に使用されるスタックレベルの最大数を[Project Options]ダイアログで指定することができます。

CrossWorksはC標準のprintf()関数を標準ライブラリセットの一部として備えているため、時刻および温度値をASCIIに変換するのに必要なコードの大部分を実装済みです。

CrossWorksはC標準のprintf()関数を標準ライブラリセットの一部として備えているため、時刻および温度値をASCIIに変換するのに必要なコードの大部分を実装済みです。残されたステップは、データを送信するシリアルポートの初期化と制御を行うことのみです。__putchar関数を定義して、シリアルポート0を通じてキャラクタを出力することによって、インタフェースが処理されます。これが実行されると、printf()は必要に応じてputchar()関数を自動的に呼び出します。

なお、ログ出力コードを正常に機能させるには、printf()の浮動小数点数のサポートを明示的にイネードする必要があります。これをイネードすると、大量のコード(このアプリケーションビルドの場合は、約3500バイト)が追加されるため、このサポートはデフォルトではディセーブルされています。浮動小数点のprintf()のサポートをオンにするには、[Project Options]ダイアログに移動し、[Printf]タブを選択して、[Printf Floating Point Supported]フィールドを「Yes」に設定します。

また、このアプリケーションでは、シングルキャラクタをPCホスト側のターミナルエミュレータに入力することによって、ユーザが簡単な設定および制御操作をすることができます。これらのキャラクタがシリアルポート0を通じて受信されると、アプリケーションは以下の操作を実行します。

- d—クロック、現在の摂氏温度(C)および華氏温度(F)間でディスプレイモードをローテート
- m—クロックの分カウントを1ずつインクリメント
- h—クロックの時カウントを1ずつインクリメント
- F—華氏温度で時刻/温度ログに現在記録されているすべての非ゼロ値を出力
- C—摂氏温度で時刻/温度ログに現在記録されているすべての非ゼロ値を出力
- Z—ログに現在記録されているすべての値を消去

ホストシステムとの通信

記録された値をログからダウンロードし、温度ロギングシステムを設定するために、このアプリケーションは、MAXQ3100のEVキット上に搭載されたレベルシフトされた2個のシリアルポートのいずれか1個を使用します。これらのポート(MAXQ3100のシリアルポート0およびシリアルポート1)からのDB9コネクタをPCのCOMポートに直接接続することができます。このため、MTKのDumb TerminalモードまたはTeraTermなどの同様のターミナルエミュレータアプリケーションで温度ロギングデモとインタフェースすることができます。アプリケーションからのシリアルポート出力は標準の10ビット非同期形式の出力で(1スタートビット、8データビット、1ストップビット)、9600ボーで伝送されます。

表2は、華氏温度でのサンプルログ出力を示しています。

表2. サンプルログ出力

時刻	温度(°F)
10:31	77.34
10:32	84.20
10:33	79.25
10:34	75.20
10:35	78.12
10:36	98.37

Rowley Associates社のCrossWorksの包括的なツールセットと開発環境を追加すると、誰でもデータキャプチャおよび処理アプリケーションを標準ANSI準拠Cで迅速に開発し、デバッグすることができます。

まとめ

MAXQ3100は、温度センサ、デュアルUARTシリアルポート、および複雑なCアプリケーションを実行するのに十分な処理能力を備えるLCDコントローラを内蔵しています。Rowley Associates社のCrossWorksの包括的なツールセットと開発環境を追加すると、誰でもデータキャプチャおよび処理アプリケーションを標準ANSI準拠Cで迅速に開発し、デバッグすることができます。

この記事の詳述版については、以下を参照してください：japan.maxim-ic.com/AN3975 (英文)。

*開発中。入手可能性についてはお問い合わせください。
WindowsはMicrosoft Corporationの登録商標です。