MAXQ命令セットアーキテクチャとRISCのベンチマーク比較

本稿では、MAXQ命令セットと他のマイクロコントローラとの比較を行います。比較対象は、PIC16CXXX(ミッドレンジデバイス)とAVR、MSP430です。各命令セットとアーキテクチャの長所と短所を表にまとめて紹介します。また、いくつかのコードアルゴリズムと動作を基準として、コード密度とコード性能の比較も行います。本稿の最後では、各コード例のMIPS/mA(MIPS：100万命令／秒)を比較します。

MAXQ命令セットの概要

MAXQ命令セットは、転送トリガというコンセプトをベースにしています。命令ワードは、ソースオペランドとデスティネーションオペランドのみで構成されます。ソースオペランドとデスティネーションオペランドは物理レジスタを表したり、エンコーディングによってデータメモリやスタックメモリ、ワーキングアキュムレータなどへの間接アクセスポイントを表したり、暗黙にハードウェア演算をトリガすることもできます。MAXQの転送トリガアーキテクチャについては、本ジャーナルの以前の記事に詳しく説明されています。各MAXQデバイスのソースエンコーディングとデスティネーションエンコーディングがどのようになっているかは、デバイス添付のMAXQユーザガイドをご覧ください。なお、ソースエンコーディングとデスティネーションエンコーディングの一部は周辺ハードウェア機能など、デバイス固有の機能に割り当てられますが、一部はMAXQの基本命令セットを構成する固定したエンコーディングとなっています。図1に、MAXQの命令ワードと命令セットニーモニックを示します。

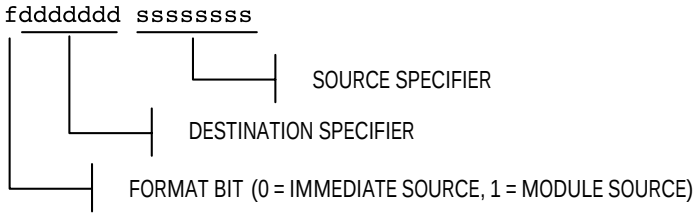


*図1. **MAXQの命令ワードで表現されるソース/デスティネーション転送によって、小規模でありながら非常に強力な命令セットが完成しました。***

MNEMONIC	DESCRIPTION	MNEMONIC	DESCRIPTION
BIT MANIPULATION		LOGICAL	
MOVE C, #0/#1	Clear/Set Carry	AND	Logical AND
CPL C	Complement Carry	OR	Logical OR
AND Acc.<b>	Logical AND Carry with Accumulator Bit	XOR	Logical XOR
OR Acc.<b>	Logical OR Carry with Accumulator Bit	CPL,NEG	One's, Two's Complement
XOR Acc.<b>	Logical XOR Carry with Accumulator Bit	SLA,SLA2, SLA4	Shift Left Arithmetically 1,2,4
MOVE C, Acc.<b>	Move Accumulator Bit to Carry	SRA,SRA2,SRA4	Shift Right Arithmetically 1,2,4
MOVE Acc.<b>,C	Move Carry to Accumulator Bit	SR	Logical Shift Right
MOVE C, src.<b>	Move Register Bit to Carry	RR,RRC	Rotate Right Carry (Ex/In)clusive
MOVE dst.<b>, #0/#1	Clear/Set Register Bit	RL,RLC	Rotate Left Carry (Ex/In)clusive
MATH		DATA TRANSFER	
ADD, ADDC	Add Carry (Ex/In)clusive	XCHN	Exchange Accumulator data nibbles
SUB, SUBB	Subtract Carry (Ex/In)clusive	XCH *(MAXQ20)*	Exchange Accumulator data bytes
FLOW CONTROL AND BRANCHING		MOVE dst, src	Move source to destination
JUMP {C/NC/Z/NZ/E/NE/S}	Jumps - unconditional or conditional, relative or absolute	PUSH/POP	Push/Pop stack
DJNZ LC[n], src	Decrement Counter, Jump Not Zero	POPI	Pop stack and enable interrupts (INS≤0)
CALL	Call – relative or absolute	Other	
RET {C/NC/Z/NZ/S}	Return – unconditional or conditional	NOP	No Operation
RETI {C/NC/Z/NZ/S}	Return from Interrupt – unconditional or conditional	CMP	Compare with Accumulator

...命令セットとデバイスアーキテクチャ(命令サイクル、メモリモデル、レジスタセット、アドレッシングモードなど)は一体であり、一体のものとして評価しなければなりません。

表1．命令セットの比較

ISA	長所	短所
AVR	• 32本の汎用ワーキングレジスタ(アキュムレータ) • データポインタも直接アクセスが可能なワーキングレジスタであり、ハイ/ローポインタバイトのマスキングやビット演算が容易 • ポインタ+変位で読み出し(変位は0～63バイト) • スタックは内蔵RAMによってのみ制限(90S1200はRAMを持たず、スタック深さが3で固定) • シングルサイクル動作 • 相対ジャンプ±2k (2サイクル) • すべてのAVR製品がEEPROMを持つ • レジスタフラグをセット/クリアする明示的命令を持つ。ビット演算命令が豊富。 • 割込ベクトルは独立	• パイプライン命令フェッチ • 32本のレジスタ以外では、ロード(LD)/ストア(ST)のオーバヘッドがLD/ST @X, Y, Z = 2サイクルとなる • LPM = 3サイクル • リテラル演算のサポート/スコープが小さい(ADDCもEORIもなし。R16～R31ではCPIとORI、ANDI、SUBI、SBCI、LDIしか使えない) • 繰り上がり専用のローテート命令がない • 条件ジャンプが+63/-64に制限(2サイクル) • CALL/RET/RETI = 4サイクル
PIC16CXXX	• ソース、デスティネーションビットがALU演算にエンコードされている • 直接データアクセス(シンボリックアドレッシングモード)により高密度のコードが生成されるとともに、データオーバレイを助長	• コアが4クロックで実行速度が遅い • パイプライン命令フェッチ • ページングによるアッパーデータメモリバンクへのアクセス(RPI:0バンクセレクト) • 間接データアクセスにはINDF、FSRレジスタが必要 • W(アキュムレータ)を直接ロードできない • ADDC、SUBBがない • スタック深さ = 8 • 相対ジャンプ/分岐がない－絶対(CALL、GOTO)か条件スキップ(BTFSx)のみ。 • コードメモリ読出しはRETLW = コード空間が浪費され、コード空間にCRCがない • CALL/GOTO/RET/RETFIE/RETWは8クロックサイクル必要(2命令サイクル) • 割込ベクトルが1本

MAXQと他の命令セットアーキテクチャとの比較

各アーキテクチャの命令セットは特定のデバイスリソースとアドレッシングモードを前提条件として構築されているため、MAXQの命令ニーモニックと他のアーキテクチャとの比較は困難なだけでなく不正確なものにしかなりません。このように命令セットとデバイスアーキテクチャ(命令サイクル、メモリモデル、レジスタセット、アドレッシングモードなど)は一体であり、一体のものとして評価しなければならないのです。表1**に、今回比較する各命令セットアーキテクチャの長所と短所をまとめました。**

コード例

命令セットアーキテクチャを比較するもっともよい方法は、一定のタスクを決め、そのタスクを処理するコードを書いてみることです。このセクションでは、何種類かのタスクについて、各命令セットアーキテクチャのコード密度とコード性能がどのようになったかを見てみます。最初の項目についてはコード例を示しますが、残りについてはグラフと文章による解説のみとします。その他の比較についても、ダラスセミコンダクタにお問い合わせくだされば、使用したコードをお渡しします。

表1．命令セットの比較(続き)

ISA	長所	短所
MSP430	• 豊富なソース、デスティネーションのアドレッシングモードが操作符号にエンコードされており、高密度のコードが生成される • 内部データパスが16ビット • 内部メモリをワードあるいはバイトとしてアクセス可能 • 定数発生器(CG)を持ち、-1、0、1、2、4、8を生成可能 • シングルサイクル動作 • スタックは内蔵RAMによってのみ制限 • 条件/相対ジャンプのデスティネーションレンジ = ±512 (2サイクル) • 割込ベクトルは独立しており、シングルソースフラグは自動的にクリアされる	• フォン・ノイマン型メモリマップ + 豊富なアドレッシングモード = サイクル数が多い。シングルサイクル命令は、Rnを排他的に処理する命令「のみ」。周辺レジスタアクセス = 3～6サイクル • CGがリテラルをサポートしておらず、余分なワードが必要 • デスティネーションオペランドでレジスタ間接やレジスタ間接自動増分が使えない • レジスタ間接動作で自動減分がサポートされていない • 記号アドレス指定のため、コードルーチンの再利用性が低い
MAXQ	• システムレジスタと周辺レジスタに対し、同一の論理メモリ空間のソースあるいはデスティネーションとしてアクセス可能なため、データ転送が高速 • パイプラインなしのシングルサイクル動作 • シングルサイクル条件ジャンプ(+127/-128)または2サイクルの絶対ジャンプ(0～65,535)。 • シングルサイクルのCALL/RET/RETI • 自動減分ループカウンタレジスタを持ち、通常ならカウンタ操作に浪費されるオーバヘッドがない • 自動増分/減分をサポートしたデータポインタを3個持つ。その1つ、FPはベースポインタ + オフセットアドレッシングをサポート(すなわち、BP[Offs]) • アキュムレータ(ワーキングレジスタ)ファイルの自動増分/自動減分/自動モジュロ制御 • 各データポインタに対し、アクセスモードをワード/バイトで選択可能 • プリフィックス可能な操作符号を持ち、命令セットの拡張や強化が簡単	• ALU演算のデスティネーションが、常に暗黙的にアクティブアキュムレータ • シングルポートのシンクロナスSRAMデータメモリのため、データポインタを使う前にアクティブにする(選択する)必要がある • デフォルトスタック深さ = 16。ただし、データメモリにソフトウェアスタックを実現しやすいデータポインタハードウェアを持つ

メモリコピー(*MemCpy64*)

メモリコピー操作は、データメモリブロックをマイクロコントローラが間接的に操作する能力を評価する例です。実行させるタスクは、データメモリソース位置からオーバーラップしていない別のデータメモリデスティネーションに64バイトをコピーするというものです。各マイクロコントローラのコードルーチンと、コピー操作のサイクルカウントとバイトカウントを示すグラフを次ページ以降に示します。前提条件は、コピー操作を始める前にポインタとバイトカウントがセットされていること、また、コピーするバイトはメモリ上にワードとして並んでおり、MSP430とMAXQ20でワードアクセスモードが利用できるということです。

システムレジスタと周辺レジスタに対し、同一の論理メモリ空間のソースあるいはデスティネーションとしてアクセス可能なため、データ転送が高速。

```
;=====================================AVR=======================================
; ramsize=r16          ;size of block to be copied
; Z-pointer=r30:r31    ;src pointer
; Y-pointer=r28:r29    ;dst pointer
; USES:
; ramtemp=r1           ;temporary storage register
loop:                          ; cycles
       ld     ramtemp,Z+       ; 2 @src => temp
       st     Y+,ramtemp       ; 2 temp => @dst
       dec    ramsize          ; 1
       brne   loop             ; 2/1
       ret                     ;  4/5
                               ;---------
                               ;(7*bytecount) + return –1(last brne isn't taken).
; WORD COUNT = 5 ; CYCLE COUNT = 451


;====================================MAXQ10=====================================
; DP[0] ; src pointer (default WBS0=0)
; DP[1] ; (dst-1) pointer (default WBS1=0)
; LC[0] ; byte count (Loop Counter)
loop:                                  ;words & cycles
       move   DP[0], DP[0]             ; 1  implicit DP[0] pointer selection
       move   @++DP[1],@DP[0]++        ; 1
       djnz   LC[0], loop              ; 1
       ret                             ; 1
                                       ;----------
                                       ; 4 / (3*bytecount) +1
; WORD COUNT = 4 ; CYCLE COUNT = 193


;====================================MAXQ20=====================================
; Assuming bytes are word aligned (like MSP430 code) for comparison
; DP[0] ; src pointer (default WBS0=1)
; DP[1] ; (dst-1) pointer (default WBS1=1)
; LC[0] ; byte count  (Loop Counter)
loop:                                  ;words/cycles
       move   DP[0], DP[0]             ; 1  implicit DP[0] pointer selection
       move   @++DP[1],@DP[0]++        ; 1
       djnz   LC[0], loop              ; 1
       ret                             ; 1
                                       ;----------
                                       ; 4 / (3*bytecount/2) +1
; WORD COUNT = 4 ; CYCLE COUNT = 97


;====================================MSP430=====================================
; MSP430 has a 16-bit data bus
; assuming bytes are word aligned, only requires (blocksize/2 transfers).
; R4   ;src pointer
; R5   ;dst pointer
; R6   ;size of block to copy
loop:                          ;words/cycles
       mov    @R4+, 0(R5)      ;2 / 5  @src++ => dst
       add    #2, R5           ;1 / 1  const generator makes this 1/1
       decd.b R6               ;1 / 1  really sub #2, R6
       jz     loop             ;1 / 2
       ret                     ;1 / 3
                               ;----------
                               ;6 / (9*(bytecount/2)) + return
; WORD COUNT = 6 ; CYCLE COUNT = 291
;==================================PIC16CXXX====================================
; a     ; src pointer base
; b    ; dst pointer base
; i    ; byte count held in reg file
; USES:
; temp  ; temp data storage
loop:                                  ; cycles
        decf    i, W                   ; 1 i-- => W
        addlw   a                      ; 1 (a+i--) => W starting at end
        movwf   FSR                    ; 1 W => FSR
        movfw   INDF                   ; 1 W <= @FSR get data
        movwf   temp                   ; 1 W => temp
```

```
        movlw   (b-a)               ; 1 diff in dest-src
        addwf   FSR, F              ; 1 (b+i--) => W
        movfw   temp                ; 1 temp => W
        movwf   INDF                ; 1 W => @FSR store data
        decfsz  i, F                ; 2/1 i--
        goto    loop                ; 2
        return                      ;   2
                                    ;----------
                                    ;11 / (12*bytecount) +1 (ret instead of
goto, +1 on decfsz)
; WORD COUNT = 12 ; CYCLE COUNT = 769 (*4clks/inst cycle = 3076)
```

メモリコピーの例で明らかとなったMAXQの長所は、データメモリで入出力バッファリングを頻繁に行わなければならないアプリケーションで大きな役割を果たします。

*MemCpy64*のサイクルカウント/バイトカウントの比較

コード密度はMAXQデバイスが一番高く、実行速度も他を圧倒しました。MAXQ10はデフォルトのバイトアクセスモードによるデータポインタを操作するため、MAXQ20よりもコピー操作が遅くなります。MAXQ10アプリケーションでコード密度よりも実行速度が重要で、かつ、コピーするデータメモリがワード整列している場合には(MSP430とMAXQ20の例でも置いた仮定)、ソース/デスティネーションデータポインタに対しワードアクセスモードを使用します。ワードモードを使うとMAXQ10のコピーループが半分になりますが、ワードアクセスモードのイネーブル/ディセーブル命令が必要になります。コピー操作でMAXQデバイスが他のデバイスを圧倒する理由は、MAXQのアーキテクチャに以下のような長所があるからです。

1) **パイプラインを使わない――他のデバイスと異なり、分岐時に命令プリフェッチのハッシュによるオーバヘッドが発生しない。**

2) **自動減分ループカウンタ――ループカウンタをマニュアルでディクリメントする必要がない。**

3) **ハーバードメモリマップ――プログラムとデータが物理空間を共有しないため、プログラムのフェッチとデータアクセスを同時に行うことができる。**

4) **ポストインクリメント/ディクリメント間接デスティネーションポインタ――デスティネーションポインタの移動を簡単かつ高速に行うことができる。MSP430では0(R5)によって@R5を示し、次の命令でデスティネーションポインタを進めなければならない点が弱点となっている。**

メモリコピーの例で明らかとなったMAXQの長所は、データメモリで入出力バッファリングを頻繁に行わなければならないアプリケーションで大きな役割を果たします。MAXQに次ぐ性能を持つのはMSP430でした。データメモリバッファリングを行う例として、MSP430デバイスに16ビット出力レジスタを持つADC周辺を組みあわせたものを考えてみましょう。周辺出力レジスタからデータメモリにデータを転送し、ポインタをインクリメントして次のADC出力に備えるためには、次のようなコードが必要になります。

```
                                ; words/cycles
        mov.w   &ADAT,0(R14)    ; 3 / 6       Store output word
                incd.w  R14     ; 1 / 1       Increment pointer
                                ; 4 / 7
```

同じ転送操作をMAXQ20で行うと、次のようになります。

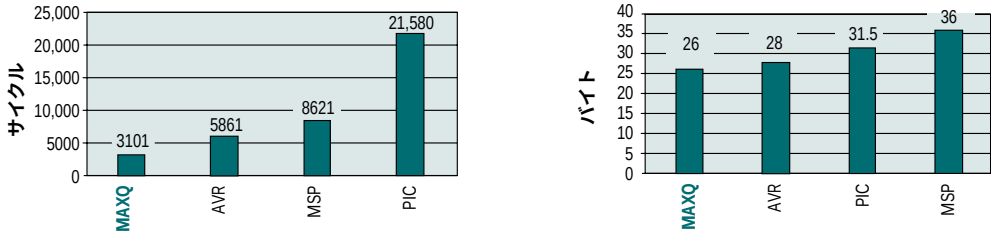
```
        move    @DP[0]++, ADCOUT    ; 1 / 1
```

バブルソートというルーチンは、データメモリのアクセス効率だけでなく、データバイト間の算術演算や比較演算を行い、条件に応じてバイトの並び順を変更する能力も評価することができます。

バブルソート(*BubbleSort*)

バブルソートというルーチンは、データメモリのアクセス効率だけでなく、データバイト間の算術演算や比較演算を行い、条件に応じてバイトの並び順を変更する能力も評価することができます。今回のコードでは、32バイトのデータメモリを昇順あるいは降順にソートします。サイクルカウントでは、隣接バイトの比較結果の半分の時間でバイトの並べ替えが行うことができると仮定しました。サイクルカウントとバイトカウントの比較結果は、次のグラフのようになりました。

*BubbleSort*のサイクル/バイトカウントの比較

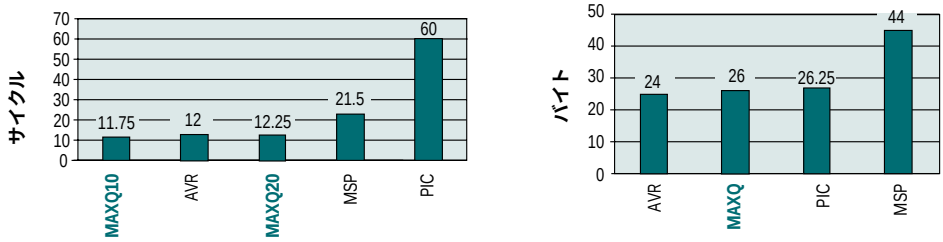


ここでも、MAXQデバイスがコード密度でも実行速度でも一番でした。MAXQが優れている理由は、メモリコピーの例で挙げたアーキテクチャ上の長所と同じです。

16進からASCIIへの変換(*Hex2Asc*)

この変換ルーチンでは、マイクロコントローラの算術演算と論理演算の能力を測ります。シングルバイトのデータを変換及び展開する際、リテラルバイトデータがサポートされているかも評価することになります。サイクルカウントは平均値です。これは、各ニブルが0～9、A～Fという16進値のいずれかであるためです。サイクルカウントとバイトカウントの比較結果は、次のグラフのようになりました。

*Hex2Asc*のサイクル/バイトカウントの比較



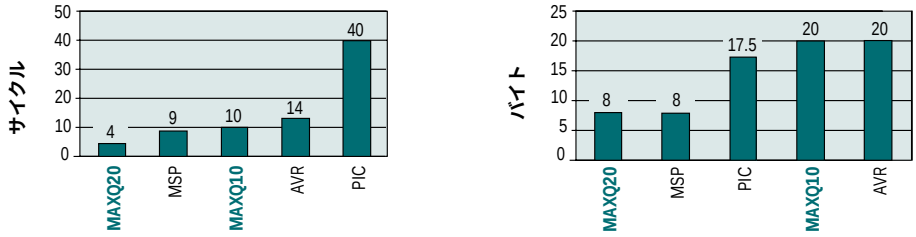
今回のテストルーチンでは、ワーキングレジスタに直接アクセス可能なAVRのワード数が1つ少ないという結果になりました。MAXQでは、効率的な変換を行おうとすると、アキュムレートポインタをマニュアルで更新しなければなりません。MSPのコード密度が低い理由は、ニブルを処理する演算を持っていないこと、また、定数発生器がサポートしていないリテラル(#nnnnh)をワードでエンコードしなければならないことです。性能面では、他のデバイスを引き離し、MAXQとアトメルAVRがほぼ同等でした。MSP430の性能が悪かった理由は、この操作をするために必要なコード量が多かったためです。

2ビット算術右シフト(*ShRight*)

このルーチンでは、16ビットワードのデータメモリアクセスとALU演算の能力が表れます。処理は、データメモリ上にある16ビットワードを算術シフトする(つまり、最上位ビットは保持する)というものです。処理ワードはデータメモリの先頭256バイト内にあり、かつ、マイクロコントローラからワードとしてアドレスできる位置にあると仮定します。サイクルカウントとバイトカウントの比較結果は、次のグラフのようになりました。

16ビットALU演算をサポートするMAXQ20とMSP430が、コード密度に優れるという結果になりました。8ビットマシンはすべて、同じ算術シフト演算をするために少なくとも2倍のコードワード数が必要でした。性能面ではMAXQ20が一番でした。また、8ビットALU演算しかサポートしていないMAXQ10も、16ビットのMSP430に近い性能を発揮しました。

*ShRight*のサイクル/バイトカウントの比較

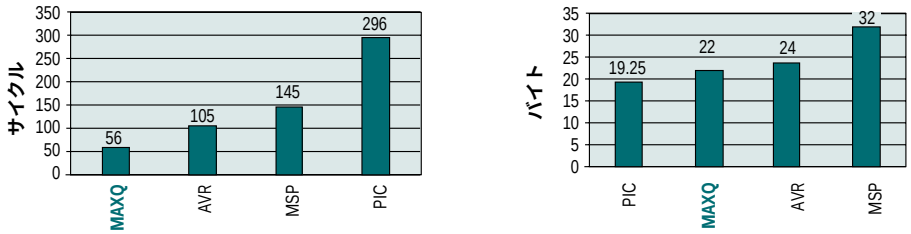


MAXQ20とMSP430のコード密度が高かった理由は、8ビットマシンよりも16ビットデータを効率的に処理することができるからです。ただし、処理方法には若干の違いがあります。MAXQ20ではシフトする16ビットワードをワーキングレジスタ(アキュムレータ)に転送し、マルチビット算術シフトを行います。MSP430は、レジスタ間接アドレッシングモード(RRA @R5)によりシングルビット算術シフトを複数回行い、処理ワードをメモリから明示的に転送することはしません。16ビットワードの算術シフトではマルチビット算術シフト操作符号(SRA2、SRA4、SLA2、SLA4)を使うことができる点が、MAXQ20がMSP430と同等以上のコード密度で、より高い性能を発揮した理由です。

ポートピンへのビットバンギング(*BitBang*)

この例では、直接ビット演算かシフト/ローテート演算によりバイトデータを分解し、各ビットをポートピンに分配する能力(「ビットバンギング」)を評価します。ポートピン出力はクロックとデータで、データはクロックの立上りエッジにおいて有効でなければなりません。このコードではポートピンを直接操作するため、I/Oポートレジスタへのアクセスがやりやすいかどうかも評価されます。ポートピンへのビットバンギング操作に関するサイクルカウントとバイトカウントの比較結果は、次のグラフのようになりました。

*BitBang*のサイクル/バイトカウントの比較



ここでも、群を抜いて性能のよいデバイスがMAXQでした。PICの性能が悪いのは、(他の例と同様に)、コアアーキテクチャが4サイクルであるためです。それよりさらにMSP430の成績が悪かった原因は、メモリアーキテクチャがフォン・ノイマン型であることと、ポート出力レジスタへのアクセスで絶対アドレッシングが必要だったためと考えられます。

コード密度では、MAXQとPICが同じワード数になりました。RISCマシンの中でPICがMAXQと同等の成績を収めることができたのは、MAXQのプログラムワードが16ビットでPICが14ビットだったためです。MSP430のコード密度が低かった理由は、周辺レジスタに絶対アドレッシングモード(&register)でアクセスするか、あるいは、定数発生器で分解することができないリテラル(#3h)を使う際に、少なくとも2ワードが必要になるためです。

BitBangテストでは、直接ビット演算かシフト/ローテート演算によってバイトデータを分解し、各ビットをポートピンに分配する能力(「ビットバンギング」)を評価します。

マイクロコントローラのMIPS/mAを測ると、消費電流も考慮に入れたコード効率を評価することができます。

MSP430の周辺レジスタアクセス方法については、もう少し説明しておく必要があるでしょう。マイクロコントローラの基本的な仕事は、外界と何らかの方法でインタフェースをとることだといえます。つまり、I/Oピンで発生する活動を制御、監視、処理しなければならないのです。マイクロコントローラに組みこむ周辺ハードウェアモジュールの数が非常に少ないときには、この処理をソフトウェアで処理することが可能です。ソフトウェアで意味のあることをしようとすれば、ポートピンの読み書きが必要になります。MSP430では、ポートピンレジスタが周辺レジスタ空間にあり、絶対アクセスモードでないとアクセスできません。このとき、マイクロコントローラに「スマート」な周辺機器をたくさんつなぐとどうなるでしょうか。オンチップの専用ハードウェアが必要な機能を発揮するために、構成し、制御し、アクセスしなければならない周辺レジスタの数が多くなります。MSP430では、これらのレジスタが周辺レジスタ空間にあり、絶対アクセスモードでないとアクセスできません。そのため、絶対アドレッシングモードによるコード密度と性能の低下を避けることができないのです。

MIPS/mA

プロセッサやコアのアーキテクチャを選定する際、消費電力が大きな考慮対象となることがよくあります。あるシステムの総消費電力は、電源電圧や動作周波数、可能な場合に低電力モードに落とすことができるかどうかなど、さまざまな要因によって左右されます。電源電圧や動作周波数を落とし、こまめに低電力モードにすれば、システム全体の消費電力を大幅に引き下げることが可能です。最低電源電圧を決めるのは主にマイクロコントローラの製造技術であり、一方、動作周波数の引き下げと低電力モードの活用はシステムデザイナーによって決められたアプリケーションの要求仕様に大きく依存します。マイクロコントローラのMIPS/mAを測ると、消費電流も考慮に入れたコード効率を評価することができます。MIPS/mAによって異なるデバイスの比較をするときには、電源電圧を共通にする必要があります。今回の比較では、電源電圧を3Vにしました。比較対象命令セットアーキテクチャ(AVR、MSP430、PIC16、MAXQ)の差異や効率を考慮するため、それぞれのコードに対し、独立のMIPS/mA比を生成する必要もあります。

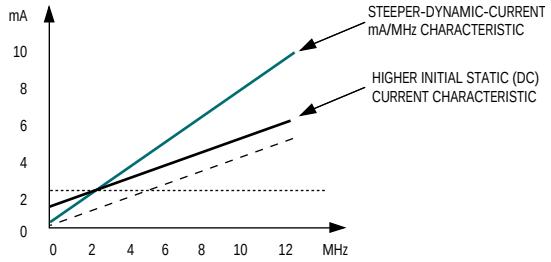


図2. IccActiveとMHzの関係を示す例で、スタティック電流やダイナミック電流が増えたときの影響がわかります。

まず、MIPS/mA比の「mA」部分を求めるため、各デバイスのデータシートをチェックします。ほとんどのマイクロコントローラのベンダは、最大動作周波数における消費電流の標準値と最大値を記載しています。スタティックな(DC)電流が非常に小さいと仮定すれば、これらのデータ点からmA/MHz値の標準値と最大値を予想し、ここから外挿により、所定のクロック周波数におけるアクティブ電流を求めることができます。ベンダからアクティブ電流対温度/周波数特性のデータが提供されていれば、個別のシステム環境におけるmA/MHz比をより正確に求めることができます。そのようなデータがなければ、何点かのデータとスタティック電流が小さいという仮定から推算するしか方法がありません。なお、スタティックな(DC)電流が多くなると、mA 対 MHzの特性曲線の始点が移動し、その分、クロック周波数を下げる(ダイナミック電流を減らす)ことによって得られる利点が減ることになります。図2は、IccActive 対 MHzの関係を示すグラフの例です。図2は、各種コアのmA/MHz値を比較したもので、情報ソースも記載しています。この記事の例では、表2で強調表示したmA/MHz値を使います。

MIPS/mA比の「MIPS」は、性能の違いを定量化にするために使用されます。ごく簡単なMIPSの算出方法を図3に示します。

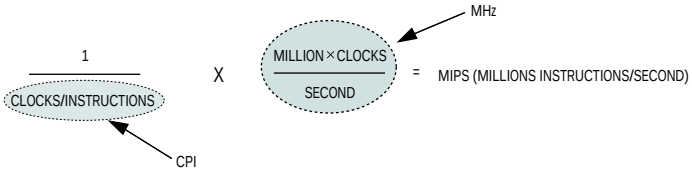


図3. MAXQアーキテクチャでは、ほとんどすべての命令を1クロック/命令で実行することによって、高いMIPS性能を実現しています。

あるアーキテクチャのMIPSを評価する場合、クロック/命令(CPI)値がとても重要です。たとえば、Microchip PICなどのアーキテクチャでは、1命令サイクルあたり複数クロックを必要とします。特定の命令を実行するときに複数の命令サイクルが必要だったり、ジャンプ/分岐時に命令パイプラインをハッシュするためのサイクルが必要だったりするアーキテクチャもよくあります。アーキテクチャの比較では、MIPSの平均性能は多くの場合、ピーク性能(MIPS)よりもかなり低くなり、また、どのような組み合わせの命令を対象にするかによっても異なります。

表2．コアによるmA/MHz値の違い

DEVICE	TYPICAL mA/MHz	MAX mA/MHz	SOURCE
PIC16C55X	0.7	1.25	PIC16C55X data sheet: DC Table 10.1, D010 (V_{CC} = 3V, 2MHz); XT or RC
PIC16C62X	0.7	1.25	PIC16C62X data sheet: DC Table 12.1, D010 (V_{CC} = 3V, 2MHz); XT or RC
PIC16LC71	0.35	0.625	PIC16C71X data sheet: DC Table 15.2, D010 (V_{CC} = 3V, 4MHz); XT or RC
PIC16F62X	0.15	0.175	PIC16F62X data sheet: DC Table 17.1, D010 (V_{CC} = 3V, 4MHz)
PIC16LF870/1	0.15	0.5	PIC16F870/1 data sheet: DC Table 14.1, D010 (V_{CC} = 3V, 4MHz); XT or RC
AT90S1200	0.33	0.75	AT90S1200 data sheet: EC Table (3V, 4MHz), Figure 38, 4mA/12MHz (typ)
AT90S2313	0.50	0.75	AT90S2313 data sheet: EC Table (3V, 4MHz), Figure 57, 7.5mA/15MHz (typ)
MSP430F1101	0.30	0.35	MSP430x11x1 data sheet: DC specs IccActive (V_{CC} = 3V, FMCLK = 1MHz)
MSP430C11X1	0.24	0.30	MSP430x11x1 data sheet: DC specs IccActive (V_{CC} = 3V, FMCLK = 1MHz)
MSP430Fx12x	0.30	0.35	MSP430x12x data sheet: DC specs (V_{CC} = 3V, FMCLK = 1MHz, FACLK = 32kHz)
MAXQ10	0.30		**Simulations**
MAXQ20	0.30		**Simulations**

より有益なインジケータとして、最終的なMIPS/mAの算出に使える値とするため、MIPSをMHzで割ります。このMIPS/MHz比は、(ある特定のコード例における)シングルクロックで実行される命令の平均数だと考えることができます。このMIPS/MHz値を前述のmA/MHz値と計算すれば、MIPS/mA比が得られます。次の表に、今まで検討したコードについて算出したMIPS/MHzとMIPS/mAを示します。

表3．各種コードアルゴリズムにおけるMIPS/MHzとMIPS/mAの比較

CORE	MIPS/MHz					
	MemCpy64	BubbleSort	Hex2Asc	ShRight	BitBang	*Peak*
MAXQ10	**1.00**	**0.99**	**1.00**	**1.00**	**1.00**	*1*
MAXQ20	**1.00**	**0.99**	**1.00**	**1.00**	**1.00**	*1*
PIC	0.23	0.20	0.23	0.23	0.21	*0.25*
MSP	0.44	0.39	0.64	0.33	0.38	*1*
AVR	0.57	0.62	0.90	0.71	0.61	*1*

CORE	MIPS/mA				
	MemCpy64	BubbleSort	Hex2Asc	ShRight	BitBang
MAXQ10	**3.33**	**3.30**	**3.33**	**3.33**	**3.33**
MAXQ20	**3.33**	**3.30**	**3.33**	**3.33**	**3.33**
PIC	1.53	1.35	1.53	1.50	1.40
MSP	1.85	1.62	2.66	1.39	1.55
AVR	1.71	1.86	2.69	2.14	1.83

ここからさらに分析を進めるためには、各コードで実際に実行された命令の数でMIPS/mA比を割り、コアアーキテクチャと命令セットの効率の違いを計算に入れる必要があります。このような計算を行う根拠は、3つのシングルサイクル命令(最大MIPS/MHz比 =1)と3サイクル命令(MIPS/MHz比 = 0.33)とは現実的に違いがないからです。このとき、算出されるMIPS/mA比は大きく異なります。そして実際は、同じタスクを行うなら、3命令よりも1命令のほうが

このMIPS/MHz比は、(ある特定のコード例における)シングルクロックで実行される命令の平均数だと考えることができます。

好まれます。MIPS/mA比を実行命令数で割れば、所定のタスクを実行するために各マイクロコントローラが使用する各種命令に対し、MIPS/mA比を調節することができます。この値をもっとも成績のよかったデバイスの値で正規化した結果が次の表です。

表4．正規化したMIPS/mA値の比較

CORE	NORMALIZED (MIPS/mA)				
	MemCpy64	BubbleSort	Hex2Asc	ShRight	BitBang
MAXQ10	0.50	**1.00**	**1.00**	0.40	**1.00**
MAXQ20	**1.00**	**1.00**	0.96	**1.00**	**1.00**
PIC	0.06	0.29	0.39	0.20	0.38
MSP	0.42	0.45	0.68	0.56	0.48
AVR	0.19	0.48	0.88	0.26	0.48

結論

正規化したMIPS/mA値によって、アーキテクチャと命令セット、電流消費特性が異なるマイクロコントローラについて、性能対電流比を相対的に比較することが可能になりました。正規化MIPS/mA値が高いほど、基本的に以下のような利点があります。(1)システムクロック周波数の低減が可能です。(2)低電力モードやスリープモードを長くすることができます。いずれもシステム全体の消費電力を下げることができる可能性を秘めた利点です。所定の電流/電力仕様の範囲で、より高いシステム性能を実現することができるともいえます。いずれにしても、高いMIPS/mA比は、MAXQアーキテクチャが高効率であることの確たる証です。