USBM3069-HS/USBM3069-HSL
ユーザーズマニュアル

テクノウェーブ株式会社

目次

図表目次

1. はじめに

このたびはマイコンボード『USBM3069-HS』/『USBM3069-HSL』をご購入頂き、まことにありがとうございます。以下をよくお読みになり、安全にご使用いただけますようお願い申し上げます。

□ 安全にご使用いただくために

製品を安全にご利用いただくために、以下の事項をお守りください。

危険　これらの注意事項を無視して誤った取り扱いをすると人が死亡または重傷を負う危険が差し迫って生じる可能性があります。

- 引火性のガスがある場所では使用しないでください。爆発、火災、故障の原因となります。

警告　これらの注意事項を無視して誤った取り扱いをすると人が死亡または重傷を負う可能性があります。

- 水や薬品のかかる可能性がある場所では使用しないでください。火災、感電の原因となります。
- 結露の発生する環境では使用しないでください。火災、感電の原因となります。
- 定格の範囲内でご使用ください。火災の原因となります。

注意　これらの注意事項を無視して誤った取り扱いをすると人が傷害を負う可能性があります。また物的損害の発生が想定されます。

- 製品のコネクタには尖った部分がありますので、取り扱いの際には十分ご注意ください。
- 本製品は製品の性質上、電源も含めて信号線が露出している部分があります。信号線同士がショートしないように注意してください。製品、接続したパソコンやその他の機器などが故障する恐れがあります。
- 濡れた手で製品を扱わないでください。故障の原因となります。
- 異臭、過熱、発煙に気がついた場合は、ただちに USB ケーブルを抜き電源を切断してください。
- 製品を改造しないでください。

□ その他の注意事項

- 本製品は一般民製品です。特別に高い品質・信頼性が要求され、その故障や誤動作が直接人命を脅かしたり、人体に危害を及ぼす恐れのある機器に使用することを前提としていません。本製品をこれらの用途に使用される場合は、お客様の責任においてなされることになります。
- お客様の不注意、誤操作により発生した製品、パソコン、その他の故障、及び事故につきましては弊社は一切の責任を負いませんのでご了承ください。
- 本製品または、付属のソフトウェアの使用による要因で生じた損害、逸失利益または第三者からのいかなる請求についても、当社は一切その責任を負えませんのでご了承ください。

□ マニュアル内の表記について

本マニュアル内では特に区別の必要がない限り、対応製品『USBM3069-HS』、『USBM3069-HSL』を『USBM3069-HS(L)』と表記します。

また、『USBM3069-HS』および『USBM3069-HSL』を、単に「製品」または「デバイス」と表記する場合があります。

本マニュアル内でハードウェアの電気的状態について記述する必要がある場合には、下記のように表記します。

表 1　電気的状態の表記方法

表記	状態
“ON”	電流が流れている状態、スイッチが閉じている状態、オープンコレクタ(オープンドレイン)出力がシンク出力している状態。
“OFF”	電流が流れていない状態、スイッチが開いている状態、オープンコレクタ(オープンドレイン)出力がハイインピーダンスの状態。
“Hi”	電圧がロジックレベルのハイレベルに相当する状態。
“Lo”	電圧がロジックレベルのローレベルに相当する状態。
“Z”	端子がハイインピーダンスの状態。

数値について「0x」、「&H」 、「H’」はいずれもそれに続く数値が 16 進数であることを表します。“0x10”、“&H1F”、“H’20”などはいずれも 16 進数です。同様に「B’」に続く数値は 2 進数であることを表します。例えば“B’01000001”のように表記されます。数値の最初に特別な表記が無い場合は 10 進数です。

2. 製品概要

□ 特徴

- 『USBM3069-HS(L)』は、ルネサステクノロジ社の H8 マイコンと USB インタフェースを搭載したマイコンボードです。『USBM3069-HS』には「H8/3069RF」、『USBM3069-HSL』には「H8/3029」を搭載しています。
- マイコン用のファームウェア、パソコン用のドライバソフトなどを開発すること無く、USB インタフェースを持ったパソコン用周辺機器を簡単に作製することができます。
- USB インタフェースは Cypress 社の CY7C68013A を採用しています。ハイスピードモードに対応し、最大 9.5Mbytes/sec のデータ転送が可能です。
- 搭載されたマイコンには、あらかじめ内蔵の周辺機能を簡単に利用するためのファームウェアが書き込まれ、すぐに USB-I/O ボードとして使用できます。そのため高価なエミュレータや開発環境、マイコンの知識などは必ずしも必要ではありません。
- USB-I/O ボードとして利用するための専用ライブラリが付属します。パソコン上のアプリケーションソフトからライブラリ関数を呼び出すことで、簡単に『USBM3069-HS(L)』の I/O 機能を利用できます。
- デフォルトのファームウェアに追加するかたちで、ユーザー独自のマイコンプログラムを作成し、ボード上のマイコンにダウンロードすることができます。I/O 機能はそのまま利用できますので、タイムクリティカルな処理や、オリジナルの機能だけをプログラミングして追加することができます。
- 弊社製品『USBM3069F』と比較し、基板サイズが約 1/3 と小型です。
- 電源供給方式として、バスパワー、セルフパワーが選択可能ですので、周辺回路に応じて自由に電源を選択できます。
- 『USBM3069-HS』は 5V 電源、『USBM3069-HSL』は 3.3V 電源で動作します。
- GUI で操作できる各種設定ツールが付属しています。
- Visual C++®、Visual Basic®、Visual Basic for Applications(VBA)、LabVIEW™ に対応しています。

□ 『USBM3069-HS(L)』の I/O 機能

- デジタル I/O
- バス(1M バイト×4 のアドレス空間)
- AD コンバータ(10 ビット)
- DA コンバータ(8 ビット)
- 16 ビットタイマ(PWM)
- 16 ビットタイマ(チャネル 2)を利用した 2 相エンコーダのパルス計数
- 32 ビットパルスカウンタ
- SCI(調歩同期シリアル通信、300～38400bps)
- ユーザーに開放された RAM 領域
- 8 ビットタイマのコンペアマッチによる自動データ転送[1]

[1] タイマ内蔵のカウンタとコンペアレジスタの内容が一致する度に 1 バイトずつポートへデータを転送します。つまり、一定の周期でポートの出力値を更新することができます。

□ 製品の使用法について

ホストパソコンから制御する

製品には、あらかじめ専用のマイコンプログラムが用意されています。このプログラムのことを**システムファーム** と呼びます（パソコン上で動作するプログラムやソフトウェアと区別するために、マイコン用のプログラムのことをファームウェア、または単にファームと呼びます）。

システムファームの役割は、USB インタフェースを通じてホストパソコンから送られてくる命令（制御コマンド）を解釈し、I/O やタイマなどのマイコン機能を制御することです。

製品の最も基本的な使用方法は、このシステムファームを利用してハードウェアを制御することです。下の図はこの場合の階層図を示しています。システムファームに命令を送るには、ホストパソコン上で動作するアプリケーションプログラムを作成し、用意された専用ライブラリの API 関数を呼び出します。この専用ライブラリのことを *USBM ライブラリ* と呼びます。

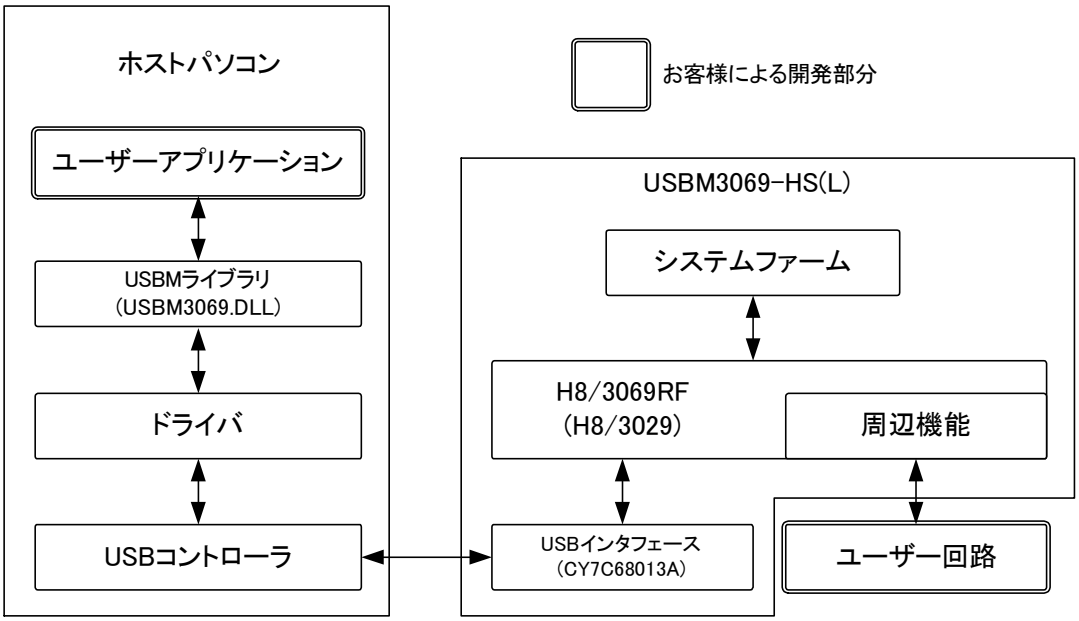


図 1　システムファームと USBM ライブラリによる制御

機能を追加する／単体で動作させる

ホストパソコンから制御する方法の他に、ボード上のマイコン用プログラムを効率よく開発できる仕組みも用意されています。そのため、マイコン上のプログラムでなければ実現が困難な複雑な制御や、リアルタイム性が要求される処理にも対応可能です。この、マイコン上で動作する追加プログラムのことを**ユーザーファーム** と呼びます。

ユーザーファームを利用することで、システムファームではサポートされない新しいコマンドを追加したり（図 2）、ホストパソコンと無関係に自律的に動作[2]させたりが可能になります。

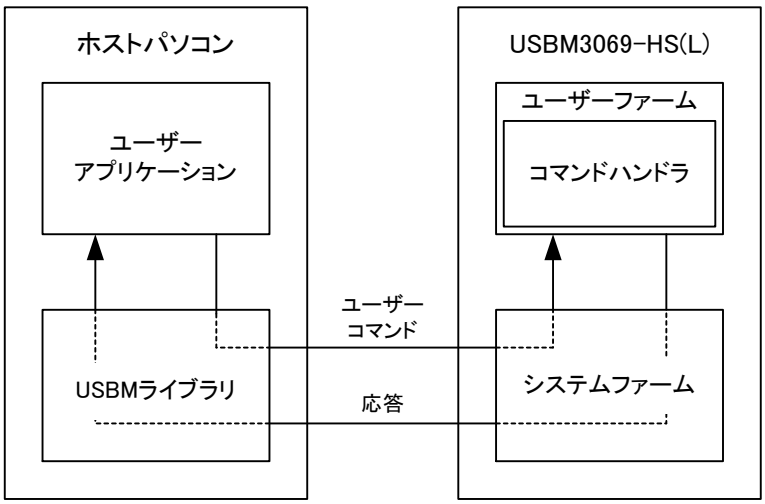


図 2　新しいコマンドの追加

ユーザーファームの開発言語は C 言語、開発環境は『YellowIDE(YCH8)』、『イエロースコープ(YSH8)』[3]をサポートしています。

- 製品では「H8/3069RF(H8/3029)」をモード 5 で利用しています。その他のモードには設定できません。
- 市販のフラッシュライティングツールを使用し、SCI 経由でフラッシュメモリを書き換えると、USB 経由でのプログラムのダウンロード及びファームウェアの更新ができなくなります。必ず付属ツールを使用してください。

2 セルフパワーでの動作が必要です。

3 『YellowIDE(YCH8)』及び『イエロースコープ(YSH8)』は有限会社イエローソフトの製品です。

□ 関連マニュアル

製品の使用方法に関して、以下のドキュメントを用意しております。合わせてご参照ください。

表 2　製品関連マニュアル

マニュアル名	内容
「USBM3069-HS/USBM3069-HSL ユーザーズマニュアル」(本マニュアル)	基本事項、ハードウェア、USBM ライブラリによるホストパソコンからの制御方法など
「USBM ライブラリ 関数リファレンス」	USBM ライブラリの各関数の説明
「M3069 マイコンボード ユーザーファーム開発マニュアル」	ユーザーファーム(マイコン用プログラム)の開発方法
「VI ライブラリヘルプファイル」(ヘルプファイル)	LabVIEW 用ライブラリの使用方法

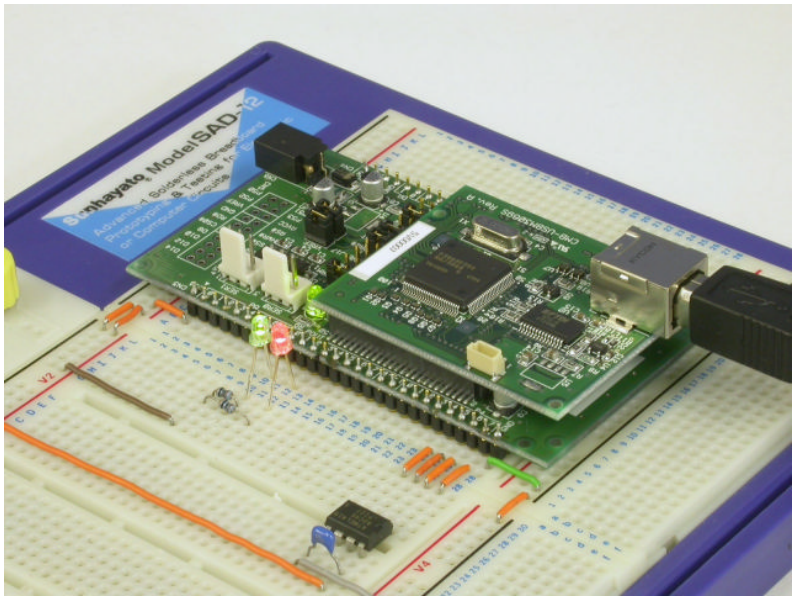
□ デバッグボードの利用

『USBM3069-HS(L)』の動作には必ず外部配線が必要になります。はじめてお使いになる場合や、試作を行う場合には専用のデバッグボード(別売り)のご利用をおすすめします。デバッグボードには以下のような特徴があります。

- 市販のブレッドボードが利用可能で、回路の実験や試作が簡単に行えます。
- 128Kbyte の SRAM を搭載し、大きなファームウェアのデバッグも可能です。
- RS-232C トランシーバを搭載していますので、パソコンと接続してのデバッグが容易です。
- バスパワー/セルフパワーを自由に設定できます。

表 3　製品に対応するデバッグボード

製品名	対応するデバッグボード
USBM3069-HS	M3069-S デバッグボード
USBM3069-HSL	M3069-SL デバッグボード



写真はデバッグボードと市販のブレッドボード(サンハヤト㈱ SAD-12)を利用した例です。

図 3　デバッグボードの利用

□ 『USBM3069-S』/『USBM3069-SL』との相違点

『USBM3069-HS』、『USBM3069-HSL』は、それぞれ弊社製品『USBM3069-S』、『USBM3069-SL』と高い互換性を持っていますが、以下の点が異なっています。

表 4 『USBM3069-HS(L)』と『USBM3069-S(L)』の主な相違点

相違点	USBM3069-S(L)	USBM3069-HS(L)
USB インタフェース	フルスピードモード(12Mbps)で動作します。ハイスピードには対応しません。	ハイスピード対応です。最大転送速度は約 9.5Mbytes/sec です。
CN2-24～CN2-26 端子の用途	P50-P52 の入力ポートとしても、アドレス出力としても利用できます。	アドレス出力に固定されます。入力ポートとしては利用できません。
CN1-11～CN1-18 端子の用途	P40-P47 の入出力ポート。	P40-P47 の入出力ポート、または、D0-D7 のデータバス。最大の転送能力を得るにはこれらの端子をデータバスとして使用し、16 ビットバス幅とする必要があります。
CN2-4 端子の用途	PUENB 端子をにより、P40-P47 のプルアップをコントロールできます。	予約端子となっています。P40-P47 のプルアップ設定はソフトウェアによって行います。

3. 製品仕様

□ 仕様概略

表 5　仕様概略

項目		仕様	備考
基板寸法		54×44 [mm]	コネクタなどの突起部含まず
電源電圧	USBM3069-HS	4.5～5.25[V]	
	USBM3069-HSL	3.0～3.6[V]	
消費電流(ボード単体、無負荷時)		100 [mA]	
動作温度範囲		0～70[℃]	
フラッシュメモリのプログラム保持年数		10 年	
インタフェース		USB	
I/O ポート数	入力専用ポート	最大 17 ピン	
	出力専用ポート	8 ピン	オープンコレクタ出力
	入出力兼用ポート	最大 16 ピン	
AD コンバータチャンネル数		4	入力範囲 0V～VREF
DA コンバータチャンネル数		2	出力範囲 0V～VREF
パルスカウンタ入力数		4	立下りのみカウント可能
シリアルチャンネル数		2	TTL 信号レベル
PWM 出力数		3	16 ビットタイマ機能を利用
2 相ロータリーエンコーダ接続用タイマチャンネル数		1	16 ビットタイマ機能を利用
通信速度	ライト(PC→ボード)	9.5 [MBytes/sec][4]	DMA 使用、16 ビットバス時
	リード(PC←ボード)	9.5 [MBytes/sec][4]	DMA 使用、16 ビットバス時
対応 OS		Windows XP, Vista, 7	

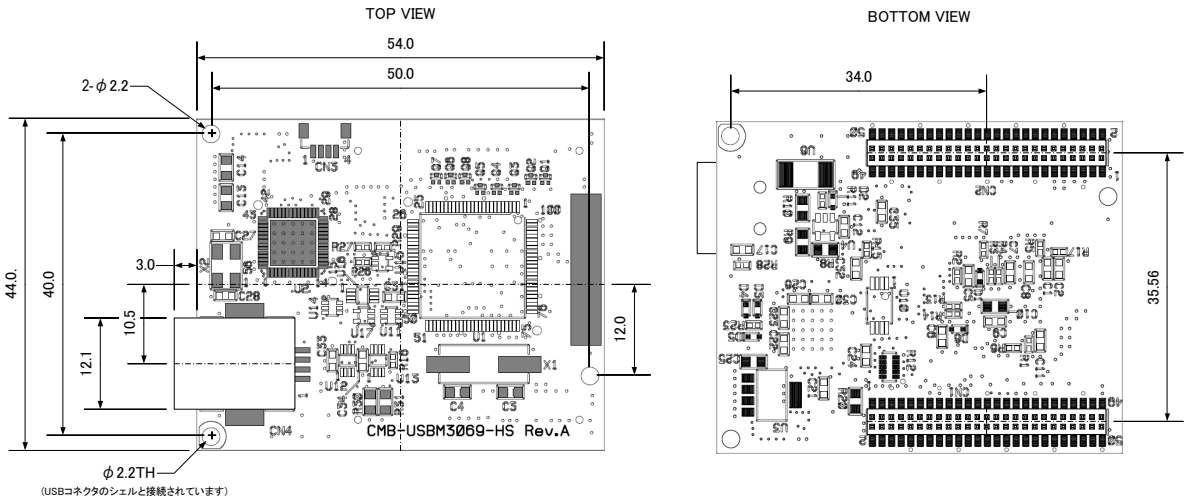


図 4　基板図

[4] 64KBytes のデータを DMA 転送機能を使用して入出力することで測定した結果です。マイコンや USB の使用状況により変化します。また、USB の性質上、小さなデータを入出力する場合は見かけのスループットが低下します。「命令実行までのレイテンシ」の節をご参照ください。

表 6　定格

項目		記号	Min	Max	単位	条件など
電源電圧	USBM3069-HS	VCC	−0.3	5.25	V	
	USBM3069-HSL		−0.3	3.6		
アナログ電源電圧	USBM3069-HS	AVCC	−0.3	5.25	V	
	USBM3069-HSL		−0.3	3.6		
入力電圧	AD0～AD3, RSV0,PUENB,VREF	Vin	−0.3	AVCC+0.3	V	
	USB_RES#	Vin	−0.5	15	V	25℃
	上記以外	Vin	−0.3	VCC+0.3	V	
出力電圧	POUT#	Vout	0	50	V	DC 出力時,25℃
出力 Lo レベル許容電流	P1,P2,P5,	I_{OL}		10	mA	
	POUT#			30		DC 出力時,25℃
	上記以外			2		
出力 Hi レベル許容電流	POUT#以外	$\lvert I_{OH} \rvert$		2	mA	
総和出力 Lo レベル許容電流	P1,P2,P5 の総和	$\lvert \Sigma I_{OL} \rvert$		80	mA	
	POUT#以外でP1,P2,P5を含む総和			120		
総和出力Hiレベル電流	POUT#以外の総和	$\lvert \Sigma I_{OH} \rvert$		40	mA	
動作温度		Topr	0	70	℃	

USB 端子は含まれません。

測定条件、本書に記載されない特性については「H8/3069R F-ZTAT™ ハードウェアマニュアル」または「H8/3029R F-ZTAT™ ハードウェアマニュアル」をご参照ください。

表 7　DC 特性(USBM3069-HS)

項目		記号	Min	Max	単位	測定条件
シュミットトリガ入力電圧	PA0～PA7, PC2#,PC3#	Vt^-	1.0		V	
		Vt^+		Vcc×0.7		
		$Vt^+ - Vt^-$	0.4			
入力 Hi レベル電圧	RES#,PA0～PA7, PC2#,PC3#以外	V_{IH}	2.0		V	
	RES#		Vcc−0.7			
入力 Lo レベル電圧	RES#,PA0～PA7, PC2#,PC3#以外	V_{IL}		0.8	V	
	RES#			0.5		
出力 Hi レベル電圧	POUT#以外	V_{OH}	3.5		V	I_{OH} = −1mA
出力 Lo レベル電圧	POUT#以外	V_{OL}		0.4	V	I_{OL} = 1.6mA
	POUT#			1.0		I_{OL} = 30mA

USB 端子は含まれません。

ADC、DAC、その他の機能別の特性はそれぞれのページをご参照ください。

測定条件、本書に記載されない特性については「H8/3069R F-ZTAT™ ハードウェアマニュアル」をご参照ください。

F-ZTAT は(株)ルネサステクノロジの商標です。

表 8　DC 特性(USBM3069-HSL)

項目		記号	Min	Max	単位	測定条件
シュミットトリガ入力電圧	PA0～PA7, PC2#,PC3#	Vt^-	Vcc × 0.2		V	
		Vt^+		Vcc × 0.7		
		$Vt^+ - Vt^-$	Vcc × 0.05			
入力 Hi レベル電圧	RES#,PA0～PA7, PC2#,PC3#以外	V_{IH}	Vcc × 0.7		V	
	RES#		Vcc × 0.9			
入力 Lo レベル電圧	RES#,PA0～PA7, PC2#,PC3#以外	V_{IL}		Vcc × 0.2	V	
	RES#			Vcc × 0.1		
出力 Hi レベル電圧	POUT#以外	V_{OH}	Vcc−1.0		V	I_{OH} = −1mA
出力 Lo レベル電圧	POUT#以外	V_{OL}		0.4	V	I_{OL} = 1.6mA
	POUT#			1.0		I_{OL} = 30mA

USB 端子は含まれません。

ADC、DAC、その他の機能別の特性はそれぞれのページをご参照ください。

測定条件、本書に記載されない特性については「H8/3029R F-ZTAT™ ハードウェアマニュアル」をご参照ください。

表 9　USB ID

ベンダーID	0x1237
プロダクト ID	0xF003

□ 端子説明

表 10　CN1 端子

USBM3069-HS(L)での番号/名称/機能/方向				H8/3069RF または H8/3029 の番号/名称	
コネクタ-ピン番	信号名	説明	方向	番号	信号名
CN1-1	VCC	電源			
CN1-2	GND	電源			
CN1-3	D15	データバス(PU)	I/O	34	D15/P37
CN1-4	D14	データバス(PU)	I/O	33	D14/P36
CN1-5	D13	データバス(PU)	I/O	32	D13/P35
CN1-6	D12	データバス(PU)	I/O	31	D12/P34
CN1-7	D11	データバス(PU)	I/O	30	D11/P33
CN1-8	D10	データバス(PU)	I/O	29	D10/P32
CN1-9	D9	データバス(PU)	I/O	28	D9/P31
CN1-10	D8	データバス(PU)	I/O	27	D8/P30
CN1-11	D7/P47	データバス、デジタル入出力(PU)	I/O	26	D7/P47
CN1-12	D6/P46	データバス、デジタル入出力(PU)	I/O	25	D6/P46
CN1-13	D5/P45	データバス、デジタル入出力(PU)	I/O	24	D5/P45
CN1-14	D4/P44	データバス、デジタル入出力(PU)	I/O	23	D4/P44
CN1-15	D3/P43	データバス、デジタル入出力(PU)	I/O	21	D3/P43
CN1-16	D2/P42	データバス、デジタル入出力(PU)	I/O	20	D2/P42
CN1-17	D1/P41	データバス、デジタル入出力(PU)	I/O	19	D1/P41
CN1-18	D0/P40	データバス、デジタル入出力(PU)	I/O	18	D0/P40
CN1-19	TxD0	シリアル 0 出力	O	12	TxD0/P90
CN1-20	RxD0	シリアル 0 入力	I	14	RxD0/P92
CN1-21	TxD1	シリアル 1 出力	O	13	TxD1/P91
CN1-22	RxD1	シリアル 1 入力	I	15	RxD1/P93
CN1-23	PC1#	パルスカウンタ 1 入力	I	17	IRQ5#/P95/SCK1
CN1-24	PC0#	パルスカウンタ 0 入力	I	16	IRQ4#/P94/SCK0
CN1-25	NC	予約			
CN1-26	NC	予約			
CN1-27	POUT7#	デジタル出力、LED 駆動可(OC)	O	60	P62#
CN1-28	POUT6#	デジタル出力、LED 駆動可(OC)	O	59	P61#
CN1-29	POUT5#	デジタル出力、LED 駆動可(OC)	O	58	P60#
CN1-30	POUT4#	デジタル出力、LED 駆動可(OC)	O	9	RxD2/PB7/TP15
CN1-31	POUT3#	デジタル出力、LED 駆動可(OC)	O	8	TxD2/PB6/TP14
CN1-32	POUT2#	デジタル出力、LED 駆動可(OC)	O	7	LCAS#/PB5/TP13/SCK2
CN1-33	POUT1#	デジタル出力、LED 駆動可(OC)	O	6	UCAS#/PB4/TP12
CN1-34	POUT0#	デジタル出力、LED 駆動可(OC)	O	87	P80
CN1-35	PA7/TIOCB2	デジタル入出力、TIOCB2 入出力(SH)	I/O	100	PA7/A20/TP7/TIOCB2
CN1-36	PA6/TIOCA2	デジタル入出力、TIOCA2 入出力(SH)	I/O	99	PA6/A21/TP6/TIOCA2
CN1-37	PA5/TIOCB1	デジタル入出力、TIOCB1 入出力(SH)	I/O	98	PA5/A22/TP5/TIOCB1
CN1-38	PA4/TIOCA1	デジタル入出力、TIOCA1 入出力(SH)	I/O	97	PA4/A23/TP4/TIOCA1
CN1-39	PA3/TIOCB0	デジタル入出力、TIOCB0 入出力(SH)	I/O	96	PA3/TCLKD/TIOCB0/TP3
CN1-40	PA2/TIOCA0	デジタル入出力、TIOCA0 入出力(SH)	I/O	95	PA2/TCLKC/TIOCA0/TP2
CN1-41	PA1/TCLKB	デジタル入出力、TCLKB 入力(SH)	I/O	94	PA1/TP1/TCLKB/TEND1#
CN1-42	PA0/TCLKA	デジタル入出力、TCLKA 入力(SH)	I/O	93	PA0/TP0/TCLKA/TEND0#
CN1-43	CS5#	CS5#出力	O	4	CS5#PB2/TMO2/TP10
CN1-44	CS0#	CS0#出力	O	91	P84/CS0#
CN1-45	ADTRG#	AD トリガ入力	I	90	P83/CS1#/IRQ3#/ADTRG#
CN1-46	PC3#/CS2#	パルスカウンタ 3 入力(SH)、CS2#出力	I/O	89	P82/CS2#/IRQ2#
CN1-47	PC2#/CS3#	パルスカウンタ 2 入力(SH)、CS3#出力	I/O	88	P81/CS3#/IRQ1#
CN1-48	NC	予約			
CN1-49	VCC	電源			
CN1-50	GND	電源			

POUT0#～POUT7#についてはトランジスタを介してオープンコレクタ出力となっています。

#は負論理の信号、OC はオープンコレクタ、SH 入力時にシュミットトリガ入力、PU は内部でプルアップされることを示しています。

表 11　CN2 端子

USBM3069-HS(L)での番号/名称/機能/方向				H8/3069RF または H8/3029 の番号/名称	
コネクタ-ピン番	信号名	説明	方向	番号	信号名
CN2-1	DA1	アナログ出力	O	85	P77/AN7/DA1
CN2-2	DA0	アナログ出力	O	84	P76/AN6/DA0
CN2-3	UFIRM#	ユーザーファーム起動	I	82	P74/AN4
CN2-4	NC	予約	I	83	P75/AN5
CN2-5	AD3	アナログ入力	I	81	P73/AN3
CN2-6	AD2	アナログ入力	I	80	P72/AN2
CN2-7	AD1	アナログ入力	I	79	P71/AN1
CN2-8	AD0	アナログ入力	I	78	P70/AN0
CN2-9	AVCC	アナログ電源		76	AVCC
CN2-10	VREF	リファレンス		77	VREF
CN2-11	VCC	電源			
CN2-12	GND	電源			
CN2-13	FWE	フラッシュ書き込み許可(PD)	I	10	FEW
CN2-14	MD2	モード設定(PU)	I	75	MD2
CN2-15	HWR#	上位バイトライトストローブ	O	71	P65/HWR#
CN2-16	LWR#	下位バイトライトストローブ	O	72	P66/LWR#
CN2-17	AS#	アドレスストローブ	O	69	P63/AS#
CN2-18	RD#	リードストローブ	O	70	P64/RD#
CN2-19	RES#	リセット(PU)	I	63	RES#
CN2-20	NMI#	モード設定(PU)	I	64	NMI
CN2-21	CLK	25MHz クロック出力	O	61	CLK
CN2-22	STBY#	未使用	I	62	STBY#
CN2-23	P53/A19	デジタル入力/アドレスバス(PU)	I/O	56	A19/P53
CN2-24	A18	アドレスバス	O	55	A18/P52
CN2-25	A17	アドレスバス	O	54	A17/P51
CN2-26	A16	アドレスバス	O	53	A16/P50
CN2-27	P27/A15	デジタル入力/アドレスバス(PU)	I/O	52	A15/P27
CN2-28	P26/A14	デジタル入力/アドレスバス(PU)	I/O	51	A14/P26
CN2-29	P25/A13	デジタル入力/アドレスバス(PU)	I/O	50	A13/P25
CN2-30	P24/A12	デジタル入力/アドレスバス(PU)	I/O	49	A12/P24
CN2-31	P23/A11	デジタル入力/アドレスバス(PU)	I/O	48	A11/P23
CN2-32	P22/A10	デジタル入力/アドレスバス(PU)	I/O	47	A10/P22
CN2-33	P21/A9	デジタル入力/アドレスバス(PU)	I/O	46	A9/P21
CN2-34	P20/A8	デジタル入力/アドレスバス(PU)	I/O	45	A8/P20
CN2-35	P17/A7	デジタル入力/アドレスバス	I/O	43	A7/P17
CN2-36	P16/A6	デジタル入力/アドレスバス	I/O	42	A6/P16
CN2-37	P15/A5	デジタル入力/アドレスバス	I/O	41	A5/P15
CN2-38	P14/A4	デジタル入力/アドレスバス	I/O	40	A4/P14
CN2-39	P13/A3	デジタル入力/アドレスバス	I/O	39	A3/P13
CN2-40	P12/A2	デジタル入力/アドレスバス	I/O	38	A2/P12
CN2-41	P11/A1	デジタル入力/アドレスバス	I/O	37	A1/P11
CN2-42	P10/A0	デジタル入力/アドレスバス	I/O	36	A0/P10
CN2-43	GATE	USB からの電源入力調節用	O		
CN2-44	USB_RES#	USB インタフェースのリセット入力(PU)	I		
CN2-45	NC	予約			
CN2-46	NC	予約			
CN2-47	NC	予約			
CN2-48	VUSB	USB インタフェース部用電源			
CN2-49	VBUS_IN	USB のバス電源入力			
CN2-50	VBUS_OUT	USB のバス電源出力			

#は負論理の信号、PU は内部でプルアップ、PD は内部でプルダウンされることを示しています。

- システムファームを利用する場合、予約、未使用端子は接続しないでください。
- 製品には、CN1,CN2 の推奨勘合コネクタ「CBC1502M120-20」(CVILUX)が付属しています。

表 12　CN3 端子

コネクタ-ピン番号	信号名	説明
CN3-1	VCC	
CN3-2	TxD1	CN1-21 と同じ
CN3-3	RxD1	CN1-22 と同じ
CN3-4	GND	

コネクタ型式 BM04B-SRSS(日本圧着端子製造株式会社)

□　メモリ空間

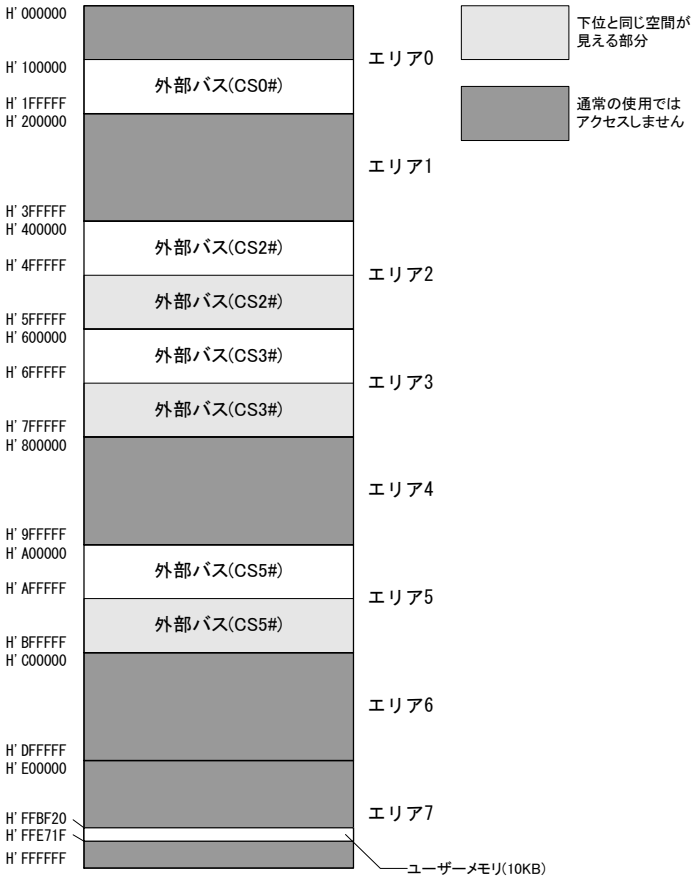


図 5　メモリマップ

図 5 は搭載マイコンのメモリ空間を示したメモリマップです。白い四角の中はユーザーが利用できる外部バス空間です。H8/3069RF(H8/3029)ではアドレスバスは 24 ビットですが、『USBM3069-HS(L)』では通常、上位 4 ビットを出力しませんので下位 20 ビット(1M バイト分)だけがアドレスバスに出力されます[5]。そのため、各エリアの上位 1M バイトは下位 1M バイトと同じアドレスとみなされます。つまり、H'400000 番地と H'500000 番地は同じアドレスと扱われます。エリア 2、3、5 については上位 1M バイト、下位 1M バイトどちらでアクセスしても構いませんが、エリア 0 については下位 1M バイトに外部アドレスとして扱われないフラッシュメモリのエリアが存在するため、H'100000～H'1FFFFF のアドレス範囲でアクセスするようにしてください。エリアの区別は出力される CS 信号

[5] アドレスを出力するためには P1,P2,P5 ポートを出力に切替える必要があります。

によって行います。

ユーザーメモリはマイコンの内蔵 RAM のうちユーザーに開放されているエリアで、一時的にデータを格納するのに利用できます。容量は小さいですが、ホストパソコンのメモリにデータを転送する場合と比較して、マイコンのローカルバス同士でのデータ転送は高速に行えます。

□ フラッシュメモリ

メモリ空間の H'000000～H'07FFFF の領域はマイコン内蔵のフラッシュメモリに割り当てられています。図 6 はフラッシュメモリ領域を詳しく示した図です。フラッシュメモリは全体で 512K バイト搭載されており、EB0～EB15 の 16 ブロックに分けて管理されます。図のように EB0、EB4～EB11 はシステムファームで利用される領域です。EB12～EB15 はユーザーファームを書き込むための領域として予約されています。EB1～EB3 の 12Kbyte の領域はユーザーに開放されており、ボード固有の設定情報やキャリブレーションデータの保存などに利用できます。

フラッシュは各ブロック単位に消去可能で、128 バイト単位での書き込みを行います。書き込みを行う際は、その領域を必ず消去する(全てのビットが"1"となる)必要があります。

フラッシュメモリの書換え可能回数の目安は 100 回、データ保持年数は 10 年です。

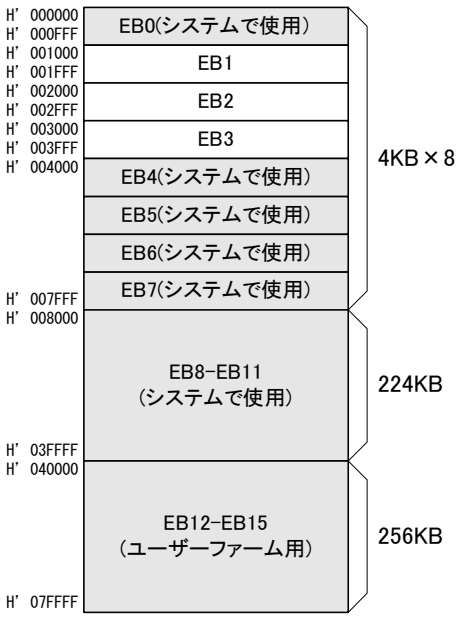


図 6　フラッシュメモリマップ

4. ハードウェア接続

□ 電源の設定

USB 機器では、内蔵もしくは AC アダプタなどの電源を使用し、ホストパソコンからの電源供給を必要としない機器をセルフパワーデバイスと呼び、USB ケーブルを通じてホストパソコン、または、USB ハブから電源の供給を受ける機器をバスパワーデバイスと呼びます。

以下では、電源設定に応じた『USBM3069-HS(L)』の端子接続方法を説明します。

バスパワーで使用する

図 7 はバスパワーデバイスとして使用する場合の接続方法です。バスパワーデバイスは USB バスから 500mA までの電源供給を受けることができますが、ホストからのコンフィギュレーションが終了するまで使用できる電流は 100mA に制限されています。そのため、図のように P チャンネルの MOS FET をスイッチとして接続します。図の GATE 信号(CN2-43)はコンフィギュレーションが正常に終了すると、"Lo"レベルを出力することで MOS FET を"ON"状態にし、マイコン、その他の回路に電流を供給します。また、規定以上の電流が回路に流れないように電流を制限する機能もあります。

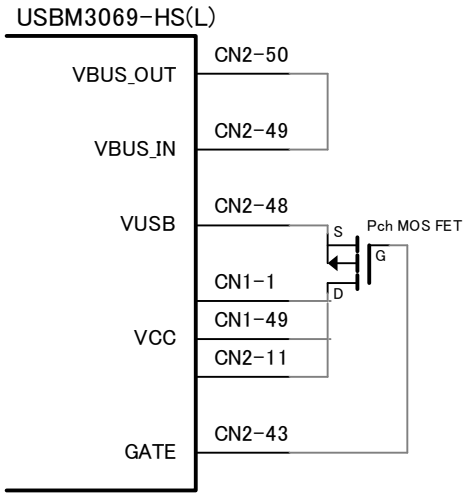


図 7　バスパワーの接続

セルフパワーで使用する

図 8 にセルフパワーで使用する場合の接続を示します。

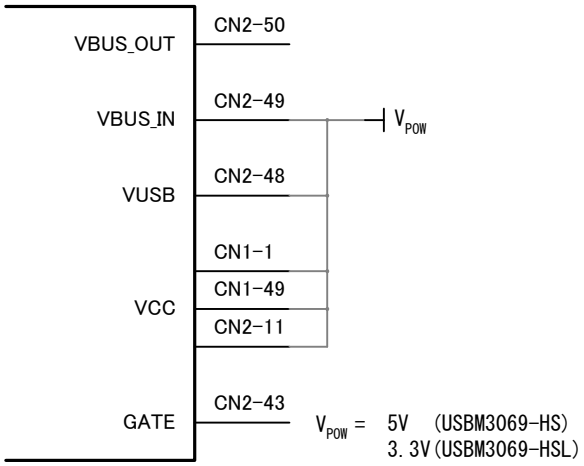


図 8 セルフパワーの接続

『USBM3069-HS(L)』のコンフィギュレーション情報

USB デバイスは自分がどのようなデバイスであるかをホストに知らせるためにディスクリプタと呼ばれるデータを保持しています。『USBM3069-HS(L)』ではハードの構成によって、ディスクリプタ内の消費するバス電力の情報を自動的に変更します。

『USBM3069-HS(L)』をバスパワーデバイスとして構成した場合、バスからの電流として 500mA をホストパソコンに要求します。そのため、バスパワー動作する USB ハブに接続した場合は、“電力供給能力の不足”として使用できない場合があります（バスパワー動作の USB ハブが 1 つのポートに供給できる電流は 100mA です）。しかし、パソコンや、セルフパワーの USB ハブからは 500mA まで電流供給を受けることができますので、外付けの回路では VCC 端子を通してこれを利用することができます。外付け回路では概ね 400mA 程度まで使用可能ですが、使用状況によりボードの消費電流が変動しますので、全体の消費電流が 500mA を超えないことを確認してください。

『USBM3069-HS(L)』をセルフパワーデバイスとして構成した場合、バスからの電流として 100mA をホストパソコンに要求します。そのため、厳密にはセルフパワーデバイスとはなりませんが、USB ポートは 100mA の電流を供給できることになっていますので実用上の問題はほとんどありません。また、セルフパワー構成にした場合、実際にはバスからの電流は消費しません。

□ 動作モード設定

製品の動作モードは、FWE、MD2、NMI の各端子の状態で決定します。表 13 に動作モードと端子設定の関係を示します。FWE 端子は製品内部で 10KΩの抵抗によりプルダウンされていますので、開放状態では”Lo”レベルになります。同様に MD2、NMI 端子は製品内部でプルアップされ、開放状態では”Hi”レベルになります。

通常モード、フラッシュ書換えモード、ブートモードなどの間でモードを切り替える場合は、一旦電源を切って端子設定後に再起動するか、端子設定後にリセット(RES#端子に 20msec 以上”Lo”を入力)する必要があります。

通常モードとユーザープログラムモードを切り替える場合は、単に FWE 端子の入力を変更することで可能です。

表 13　動作モードと端子設定

モード	端子設定			説明
	FWE	MD2	NMI	
通常モード (ユーザーモード)	Lo	Hi	Hi	ライブラリで I/O 制御可能な通常の動作モードです。フラッシュメモリの書換えはできません。
ユーザープログラムモード	Hi	Hi	Hi	通常動作しますが、フラッシュメモリの書換えが可能なモードです。ユーザープログラムモードでの書換えについては「USBM ライブラリ関数リファレンス」を参照してください。
フラッシュ書換えモード (ユーザーブートモード)	Hi	Lo	Lo	システムファームの更新、ユーザーファームの書き込み、製品情報の書き込みなどを行うモードです。通常動作はしません。
ブートモード	Hi	Lo	Hi	シリアル通信によりフラッシュメモリを書き換えるモードです。このモードで書換えを行うとシステムファームが消去され、復帰できなくなります。

( )内はマイコンのマニュアル記載のモード名

- 一般のフラッシュ書き込みツールはブートモードを使用しますが、このモードで書き込みを行うと、製品独自のプログラムも含めて全てのフラッシュメモリの内容が一旦消去されてしまいます。そのため、付属のフラッシュライティングツールが使用できなくり、システムファームの復帰もできなくなりますのでご注意ください。
- フラッシュ書換えモードは、システムファームの更新や製品情報の書き込みなどの際に必要になります。装置に組み込まれる場合には、ジャンパースイッチやディップスイッチを接続し、フラッシュ書換えモードを利用可能な構成にされることを推奨します。

デバッグボードをご利用の場合、表 14 に従ってジャンパースイッチを設定してください。

表 14　デバッグボードのジャンパー設定

モード	ジャンパー設定		
	FWE	UPRG#	UBOOT
通常モード (ユーザーモード)	OFF	ON	ON
ユーザープログラムモード	ON	OFF	ON
フラッシュ書換えモード (ユーザーブートモード)	ON	ON	ON
ブートモード	ON	ON	OFF

□ ユーザーファームの起動

上記、通常モード、または、ユーザープログラムモードで起動した場合（または、リセットが解除された場合）、UFIRM#端子（CN2-3）が”Lo”であれば、ユーザーファームが起動され、ユーザー独自のマイコンプログラムにより動作を開始します。UFIRM#端子が”Hi”の場合には、システムファームが起動し、通常動作を行います。

□ 未使用端子の処理

RES#、NMI#、MD2、FWE、USB_RES#を除いた各入力専用端子は、外部で接続されない場合はフローティング状態になります。また、PA0-PA7 の入出力端子も、初期状態では入力に設定されており同様です。

入力端子をフローティング状態のまま使用すると、消費電流の増大や、不安定動作の原因になる場合があり、推奨されません。使用しない端子は、入力を固定するか、出力に切り替えてご利用ください。

付属の設定ツール(「USBMTools」)には、I/O 用端子の初期状態を変更するソフトウェアが含まれており、P10-P17、PA0-PA7 の各端子を出力に変更することが可能です(P10-P17 の各端子は出力に設定するとアドレス出力となりますので、出力ポートとしては利用できません)。

□ RS-232C トランシーバ

ユーザーファームのデバッグを行う場合は、RS-232C トランシーバ IC を接続し、TxD1、RxD1 の信号をRS-232C の信号レベルに変換する必要があります。トランシーバ IC の接続方法は、図 40 の回路例（86 ページ）を参照してください。

5. 使用準備

□ ドライバのインストール

ドライバは付属 CD-ROM に納められています。

表 15　ドライバファイルの格納フォルダ

使用 OS	ドライバファイルの格納フォルダ
Windows XP, Vista, 7	CD の「¥HS_DRIVER¥WIN5」フォルダ

管理者のアカウントでログオンし、上記のフォルダから「setup.exe」を起動してください。

- 必ず「setup.exe」によるインストールを行ってください。ハードウェアウィザードで CD-ROM 内のフォルダを指定、または、検索してインストールを行った場合、必要なファイルがコピーされません。

Windows 7 の場合

① 「setup.exe」を起動すると、次のようなウィンドウが表示されますので「はい」を選択します。

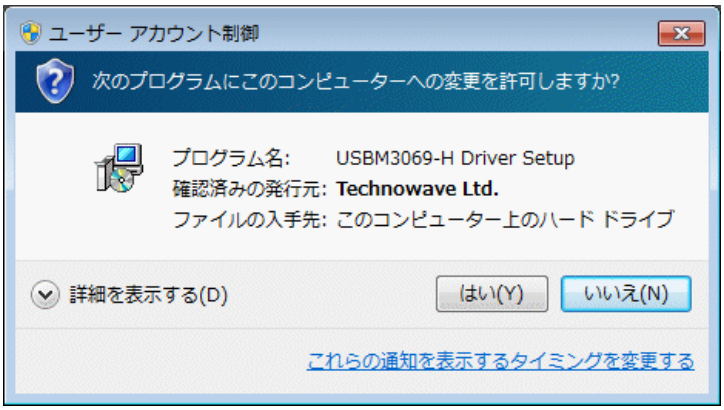


図 9　Windows 7 のドライバインストール画面（1）

② インストールプログラムが起動しますので、画面の指示に従ってインストールを行います。

③ 次のような画面が表示されますので「完了」ボタンを押してください

図 10　Windows 7 のドライバインストール画面（2）

④ デバイスを USB ケーブルでパソコンに接続します。図 11 のように「デバイス マネージャ」の画面に「USBM3069-H USB Device」と表示されれば、ドライバが正しくインストールされています。

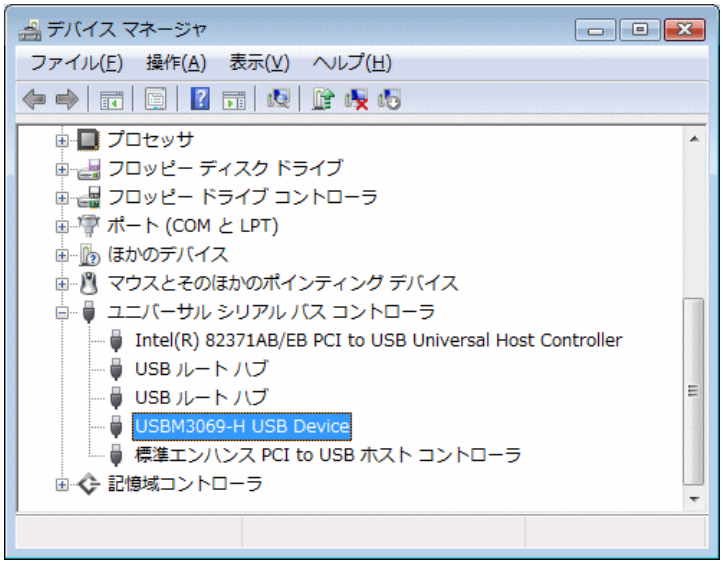


図 11　Windows 7 のドライバインストール確認

- 「デバイスマネージャ」を表示するには[スタート]メニューの[コンピュータ]を右クリックし、[プロパティ]を選択します。「システム」画面が表示されますので、「コントロールパネルホーム」中の[デバイスマネージャ]をクリックしてください。

Windows XP の場合

① 「setup.exe」を起動し、画面の指示に従ってインストールを行います。

② インストールが終了すると、次のような画面が表示されますので「完了」ボタンを押してください。

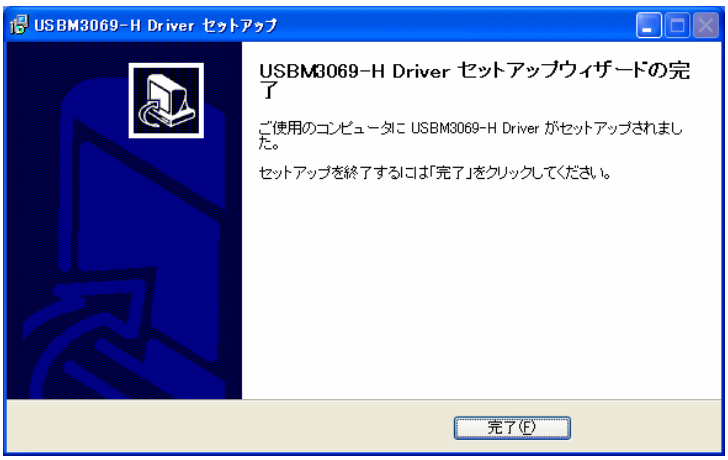


図 12　Windows XP のドライバインストール画面

③ デバイスを USB ケーブルでパソコンに接続します。図 13 のように「デバイス マネージャ」の画面に「USBM3069-H USB Device」と表示されれば、ドライバが正しくインストールされています。

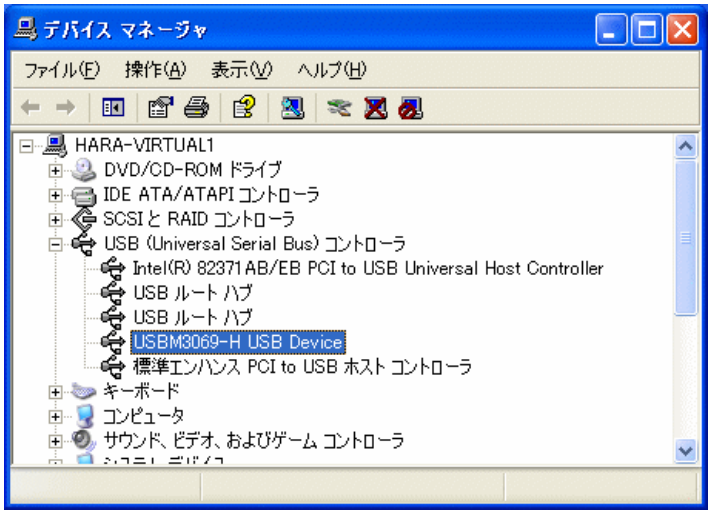


図 13 Windows XP のドライバインストール確認

- 「デバイスマネージャ」を表示するには[マイ コンピュータ]を右クリックし、[プロパティ]を選択します。「システムのプロパティ」画面が表示されますので、[ハードウェア]タブから[デバイスマネージャ]をクリックしてください。

□ VI ライブラリのインストール

開発環境として LabVIEW をご利用の場合は、付属 CD の「¥VI¥USBM3069VI¥setup.exe」を実行して、VI ライブラリをインストールします。使用方法に関してはライブラリに付属するヘルプファイルを参照してください。

□ 設定ツール

設定ツールのインストール

付属 CD の「¥TOOL¥USBMTools¥Setup.exe」を実行して、設定ツール(「USBMTools」)をインストールします。

設定ツールについて

標準のインストールでは、[スタート]メニュー→[すべてのプログラム]([プログラム])→[テクノウェーブ]→[USBMTools]を選択すると、「USBMTools」(図 14)を起動することができます(画面イメージはバージョンや OS によって異なる場合があります)。

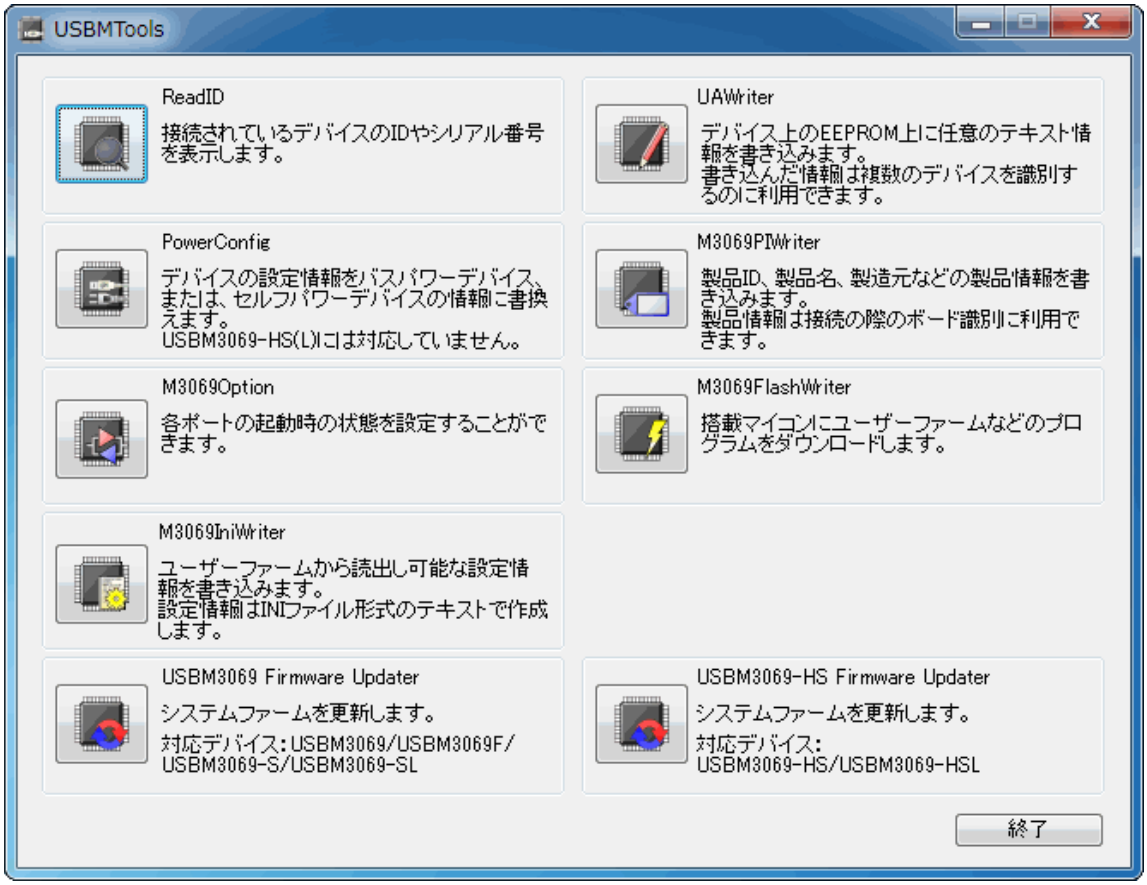


図 14　設定ツールのメニュー画面

表 16　設定ツールの機能説明

プログラム名	機能説明
ReadID	パソコンに接続されている製品のベンダーID、プロダクト ID、USB シリアル番号を表示します。
UAWriter	ユーザーエリアと呼ばれる領域にテキストデータを書き込みます。書き込んだデータは製品識別に利用できます。
PowerConfig	この製品では使用しません。
M3069PIWriter	製品情報を書き込みます。製品情報については後述の説明を参照してください。
M3069Option	起動時の入出力ポートの方向、出力データ、プルアップ機能の許可／禁止を指定します。
M3069FlashWriter	主に製品のフラッシュメモリにユーザーファームウェアをダウンロードする場合に使用します。
M3069IniWriter	ユーザーファームに動作パラメータを与えたい場合に使用します。
USBM3069 Firmware Updater	この製品では使用しません。
USBM3069-HS Firmware Updater	製品のシステムファームを更新します。

各設定ツールの使用方法については、オンラインヘルプまたは画面の説明を参照してください。

□ 製品情報の設定

『USBM3069-HS(L)』では搭載マイコンのフラッシュメモリを利用して、製品に関する情報を記憶することができます。このフラッシュメモリ領域を PI エリアと呼びます。付属ライブラリでは PI エリアに予め書き込まれた製品情報を指定して、特定のデバイスを操作することができるようになっていますので、『USBM3069-HS(L)』を組み込んだ装置の種類を特定したり、複数の製品を操作したりが簡単に行えます。

また、PI エリアを利用した製品識別は LAN インタフェース製品と共通に利用できますので、USB とネットワークの両方に対応したプログラムを作成する場合にも有効です。

特に『USBM3069-HS(L)』を組み込んだ製品を販売される場合は、誤って他の製品を操作することが無いように PI エリアに書き込んだ製品情報を指定してデバイスに接続するようにしてください。

製品情報は「USBMTools」の「M3069PIWriter」で設定します。表 17 は各設定項目の説明です。

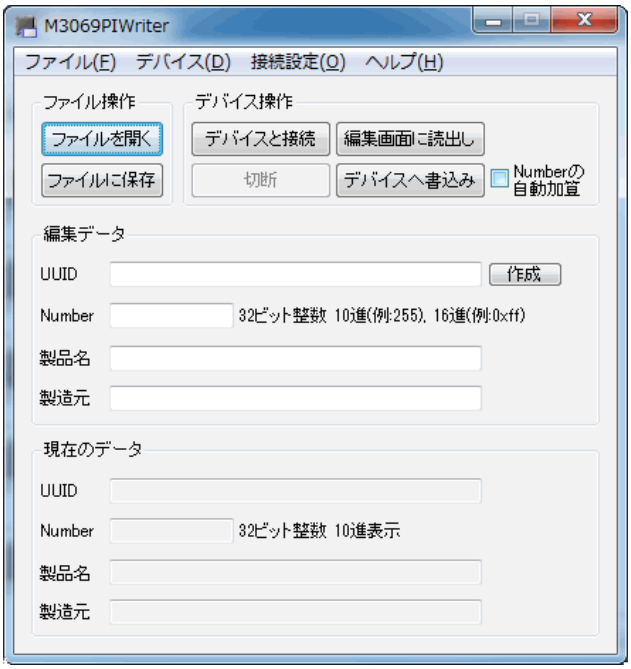


図 15 「M3069PIWriter」の画面

表 17 「M3069PIWriter」の設定項目

項目	説明
UUID	世界中で重複することのない番号です。これを製品の ID として使用することで誤った製品の操作を防ぐことができます。
Number	32 ビットの整数を記録することができます。この値はデバイスに接続するときの識別用に使用することができます。ボード毎に違う番号を書き込んでおくと複数のデバイスを同時に操作する際に便利です。
製品名	お客様の製品名を格納することを想定しています。32 バイトまでの文字列を格納できます。
製造元	お客様の会社名を格納することを想定しています。32 バイトまでの文字列を格納できます。

□ プログラミングの準備

Visual C++ の場合

●必要なファイルをコピーします。コピーするファイルとコピー先を以下に示します。

表 18　C 言語用ファイルのインストール

作成するプログラム	ファイル名	コピー元	コピー先
32bit プログラム	USBM3069.H	CD の「¥DLL」フォルダ	お客様のプロジェクトフォルダ
	USBM3069.LIB		
64bit プログラム	USBM3069.H	CD の「¥DLL」フォルダ	お客様のプロジェクトフォルダ
	USBM3069.LIB	CD の「¥DLL¥x64」フォルダ	

●「USBM3069.LIB」をお客様のプロジェクトに追加します。

●API 関数を呼び出す必要がある場合、適宜「USBM3069.H」をインクルードしてください。

Visual Basic 6.0 の場合

●必要なファイルをコピーします。コピーするファイルとコピー先を以下に示します。

表 19　Visual Basic 6.0 用ファイルのインストール

ファイル名	コピー元	コピー先
USBM3069.BAS	CD の「¥DLL」フォルダ	お客様のプロジェクトフォルダ

●「USBM3069.BAS」をお客様のプロジェクトに追加します。

Visual Basic .NET の場合

●必要なファイルをコピーします。コピーするファイルとコピー先を以下に示します。

表 20　Visual Basic .NET 用ファイルのインストール

ファイル名	コピー元	コピー先
USBM3069.VB	CD の「¥DLL」フォルダ	お客様のプロジェクトフォルダ

●「USBM3069.VB」をお客様のプロジェクトに追加します。

- 上記の開発用ファイルは「USBMTools」をインストールすることで、インストール先のドライブにコピーが作成されます。コピー先のフォルダは通常、[スタート]メニュー→[全てのプログラム]([プログラム])→[テクノウェーブ]→[ライブラリ]で開くことができます。

□ Visual Basic 用のプログラム例について

本マニュアルには Visual Basic 6.0 を対象としたサンプルを記載しています。Visual Basic の.NET 以降のバージョンでは、以下の点が変更になっていますのでご注意の上、ご参照ください。

変数のサイズは以下のように変更になっています。

表 21　Visual Basic 6.0 と Visual Bsic .NET の変数

ビット数	Visual Basic 6.0	Visual Basic .NET
16 ビット	Integer	Short
32 ビット	Long	Integer
64 ビット	なし	Long

また、関数を呼び出す場合、戻り値を必要としないときにも引数を“()”で囲む必要があります。以下に例を示します。

Visual Basic 6.0 の場合

```
Dim ADData(3) As Integer

USBM_ADRead hDev, ADData, 3, 1
```

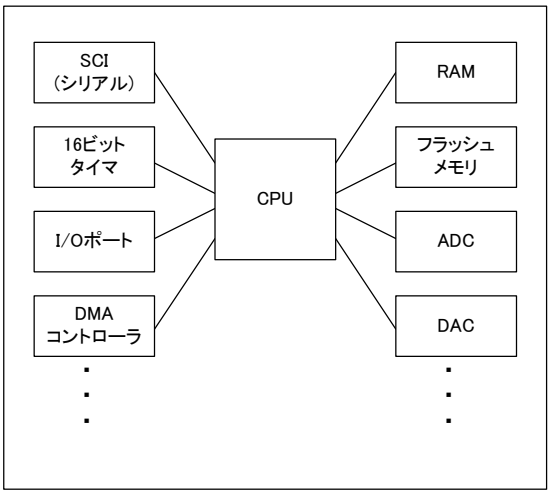
Visual Basic .NET の場合

```
Dim ADData(3) As Short

USBM_ADRead(hDev, ADData, 3, 1)
```

搭載マイコンについて

『USBM3069-HS』には「H8/3069RF」、『USBM3069-HSL』には「H8/3029」というマイクロコントローラが搭載されています(ともに株式会社ルネサステクノロジの製品です)。「H8/3029」は「H8/3069RF」の低電圧版で機能的にはほぼ同等です。これらのマイコンチップ内部には、CPU コアの他に、非常に豊富な周辺回路が内蔵されています。そのため、わずかな外付け回路と組み合わせることで、様々な製品に応用可能となっています。

H8/3069RF(H8/3029) マイコン



『USBM3069-HS(L)』に付属の USBM ライブラリとシステムファームは、マイコンが持つ多くの機能を、ホストパソコン上のプログラムから呼び出すことを可能とし、できる限り簡単にハードウェア制御を行っていただく目的で作成されています。単純な I/O 製品と比較してライブラリ関数が多いため、最初は戸惑われるかもしれませんが、必要な機能を絞り込んで動作をご理解いただければ、プログラミングは決して難しいものではありません。是非、お客様のハードウェア開発にお役立てください。

付属の CD-ROM には「H8/3069R F-ZTAT™ ハードウェアマニュアル」、および、「H8/3029 F-ZTAT™ ハードウェアマニュアル」を収録しております。本マニュアルと合わせてご参照ください。

6. 使用方法

以下の章では、USBM ライブラリを使用してハードウェアを制御する方法を説明していきます。本文中で使用方法を解説している *USBM_* で始まるライブラリ関数については別紙「USBM ライブラリ 関数リファレンス」に詳しい説明がありますので合わせてご参照ください。

□ デバイスのオープン

『USBM3069-HS(L)』の各機能を使用する前にデバイスをオープンする必要があります。オープンに成功すると関数はデバイスへのハンドルを返しますので、以後そのハンドルを使用してデバイスにアクセスすることになります。表 22 にデバイスのオープン及びクローズ時に使用する関数をあげます。

表 22 デバイスのオープン/クローズで使用する関数

関数名	説明
USBM_ListDevicesA()	オープンできるデバイスの数と USB シリアル番号を調べます。
USBM_OpenA()	ホストインタフェースや USB シリアル番号を指定してデバイスをオープンします。
USBM_OpenByPI()	PI エリア内の製品情報を指定してデバイスをオープンします。
USBM_Open()	指定した USB シリアル番号のデバイスをオープンします。
USBM_OpenByUA()	「ユーザーエリア」の情報が指定のものと一致したデバイスをオープンします。
USBM_Close()	ハンドルで指定したデバイスへのアクセスを終了します。
USBM_CloseAll()	現在のプロセスが接続しているデバイス全ての操作を終了します。
USBM_Initialize()	デバイスを初期化します。
USBM_InitializeA()	機能を指定してデバイスを初期化できます。

デバイスを探す

『USBM3069-HS(L)』はボードそれぞれに固有の USB シリアル番号（製品のシリアル番号とは異なります）を持っています。この番号はボード 1 枚毎に違ったものとなっていますので、複数デバイスを同時に操作する場合の識別に利用できます。*USBM_ListDevicesA()* 関数を使用すると、現在オープンできるデバイスのシリアル番号のリストを取得できます。

デバイスの USB シリアル番号を調べるには「USBMTools」(28 ページ)を使用してください。

シリアル番号を指定してデバイスをオープンする

USBM_OpenA() 関数を使用すると。USB シリアル番号を指定してデバイスをオープンすることができます。シリアル番号を指定しないで呼び出した場合は最初に見つかったデバイスをオープンします。

デバイスを閉じる場合は *USBM_Close()* 関数を使用します。

C 言語の例

```
TW_HANDLE hDev;

/*シリアル番号を指定してオープン*/
USBM_OpenA(&hDev, "H5000001", USBM_IF_HS);

if(hDev){
  USBM_Initialize(hDev);  /*デバイスの初期化*/

  /*...*/

  USBM_Close(hDev);  /*デバイスを閉じる*/
}
```

VisualBasic6.0 の例

```
Dim hDev As Long

'シリアル番号を指定してオープン
USBM_OpenA hDev, "H5000001", USBM_IF_HS

If hDev <> 0 Then
  USBM_Initialize hDev 'デバイスの初期化

  '...

  USBM_Close hDev 'デバイスを閉じる
End If
```

製品情報を指定してデバイスをオープンする

USB シリアル番号により複数のデバイスを識別して同時に操作することは可能ですが、値を自由に設定することができないため、番号からそのデバイスがどのような応用製品に利用されているかを知ることはできません。PI エリアに製品情報を書き込んでおくと、デバイス毎に独自の製品 ID やシリアル番号をつけて管理できるので、複数のデバイスを簡単に識別できます。

以下は製品情報の「Number」にそれぞれ“1”と“2”を設定した 2 つのデバイスを同時にオープンする例です。

C 言語の例

```
TW_HANDLE hDev1, hDev2;
char id[] = "13018f09-790c-439a-ba36-4e64e06b2472";

/*製品情報を指定してオープン*/
USBM_OpenByPI(&hDev1, id, 1, USBM_IF_HS);
USBM_OpenByPI(&hDev2, id, 2, USBM_IF_HS);
```

VisualBasic6.0 の例

```
Dim hDev1 As Long
Dim hDev2 As Long
Dim id As String

id = "13018f09-790c-439a-ba36-4e64e06b2472"

'製品情報を指定してオープン
USBM_OpenByPI hDev1, id, 1, USBM_IF_HS
USBM_OpenByPI hDev2, id, 2, USBM_IF_HS
```

ユーザーエリアのテキストを指定してデバイスをオープンする

任意のテキスト情報を、デバイスのコンフィギュレーション領域に書き込むことで、デバイスを識別することができます。コンフィギュレーション領域は本来デバイスの設定情報を記録するメモリエリアですが、一部がユーザーに開放されています。このユーザーに開放された部分をユーザーエリア[6]といいます。

例として、あるデバイスのユーザーエリアに“TW01”と予め書き込んだ場合、*USBM_OpenByUA()* 関数を使用して *USBM_OpenByUA(“TW01”, 4)* とすることでそのデバイスを指定してオープンすることができます。

ユーザーエリアへテキスト情報を書き込むには「USBMTools」(28 ページ)を使用します。

6 マイコンのメモリ上にあるユーザーメモリとは別です。『USBM3069-HS(L)』では搭載の EEPROM 内の領域です。

USB インタフェースとの接続バス幅設定

『USBM3069-HS(L)』では、USB インタフェース IC との接続バス幅を 8 ビットと 16 ビットから選択することができます。16 ビットを選択すると、*USBM_PortBWrite()*、*USBM_PortBRead()* 関数を使用したパソコン、デバイス間の最大データ転送速度が約 2 倍になり、より高速な通信が可能になります。

USB インタフェース IC との接続バス幅を 16 ビットとするためには、デバイスと接続する際に以下のように *USBM_MODE_BUS16* オプションを指定します。

C 言語の例

```
TW_HANDLE hDev;

USBM_OpenA(&hDev, NULL, USBM_IF_HS | USBM_MODE_BUS16);
```

VisualBasic6.0 の例

```
Dim hDev As Long

USBM_OpenA hDev, vbNullString, USBM_IF_HS Or USBM_MODE_BUS16
```

- USB インタフェース IC との接続バス幅を 16 ビットに設定すると、CN1-11 から CN1-18 までの端子はデータバスとして使用されます。P40-P47 の入出力ポートとしては使用できなくなりますのでご注意ください。

□ デジタル I/O（入出力ポート）

『USBM3069-HS(L)』では以下の入出力ポートが利用できます。ポートの中には他の周辺機能と端子を共用しているものがあります。共用端子を周辺機能で利用した場合にはポートとしての使用はできなくなりますのでご注意ください。表 23、表 24 に入出力ポートの種類と使用する関数をあげます。

表 23　入出力ポート

ポート名	ビット数	入出力	説明
P1	8	入力専用	アドレスバス(A0-A7)と共用
P2	8	入力専用	アドレスバス(A8-A15)と共用、プルアップ MOS によるプルアップ
P4	8	入出力	プルアップ MOS によるプルアップ(CN2-4 によってディセーブル可) データバス(D0-D7)と共用
P5	1	入力専用	P53 のみアドレスバス(A19)と共用で利用可能。 プルアップ MOS によるプルアップ
PA	8	入出力	16 ビットタイマと共用
POUT	8	出力専用	反転出力、UNR32AN(松下電器産業㈱)トランジスタによるオープンコレクタ出力、LED 駆動可能

表 24　デジタル I/O で使用する関数

関数名	説明
USBM_PortSetDir()	ポートの入力、出力の方向を切替えます。
USBM_PortWrite8A()	出力ポートの値を変更します。
USBM_PortRead8()	入力ポートの値を読み取ります。また出力ポートの出力値を読むこともできます。
USBM_PortCopy8()	任意のポート間でデータをコピーします。

各ポートは最大 8 本の端子で構成されています。例えば P1 の場合、P10～P17 までの 8 本の端子で構成され、表 11 の端子名と対応しています。

入力専用ポートは出力に設定した場合、自動的に外部バス用のアドレス出力になりますので、出力ポートとしては使用できません。入出力ポートは *USBM_PortSetDir()* 関数を使用して、ビット毎に入力、または、出力に設定することができます。通常、起動時には全て入力に設定されています。

デフォルトの設定では、P2、P4、P5 の各ポートはプルアップ MOS[7]によって VCC にプルアップされます。

出力専用ポート(POUT0#-POUT7#)は直接 LED を駆動できる端子で、トランジスタによるオープンコレクタ出力です(図 16 参照)。データとして“1”を書き込んだビットに対応する端子は、トランジスタがオンになり、“Lo”レベル出力となります。

7 マイコンの内部で抵抗と同じ働きをする機能で 16kΩ～100kΩ程度となります。レジスタが初期化されるまで有効になりませんので、システムファームが起動するまではプルアップされていない状態です。

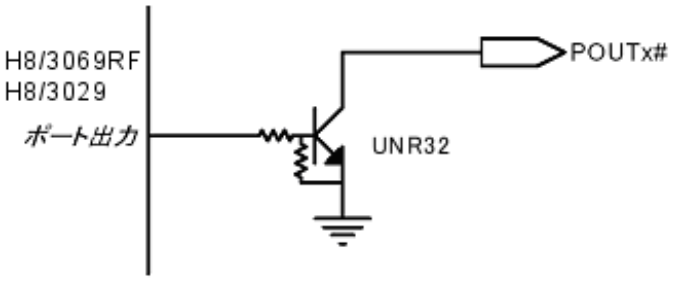


図 16　POUTx#端子の出力回路

ポートから入力する

USBM_PortRead8() 関数を使用することでポートからデータを読むことができます。読み出しは 8 ビット単位で行います。例えばP4を読み出した場合、読み取ったデータの各ビットは下の表のように各端子の入力値と対応しています。

表 25　データビットと端子の関係

ビット	7(MSB)	6	5	4	3	2	1	0(LSB)
対応端子	P47	P46	P45	P44	P43	P42	P41	P40

出力に設定されているビットを読み出すと、現在の出力値を読むことができます。また、P5 ポートの上位 4 ビットは常に“1”、下位 3 ビットは不定です。

ポートに出力する

USBM_PortWrite8A() 関数を使用することでポートから出力されるデータを変更できます。入力と同様に 8 ビット単位でデータを書き込むことができます。書き込んだデータビットと端子との関係も入力の場合と同様です。

入力に設定されている端子へ書き込みを行った場合、出力用のデータとしてレジスタに保持されますが、その端子を出力に設定するまでは端子には反映されません。

USBM_PortWriteA() 関数の *Mask* 引数に H’FF 以外を指定した場合は、*Mask* バイトのうち“0”となっているビットと対応する端子は影響を受けません。図 17 は H’55 というデータを、*Mask* を H’0F として出力した例です。

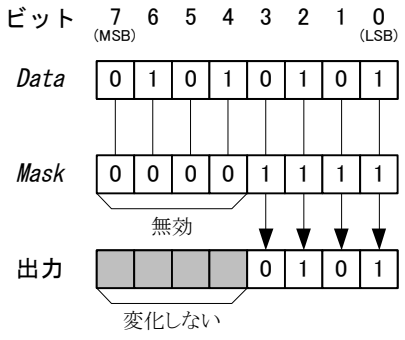


図 17　出力のマスク

入出力ポートの方向を変更する

起動時は全ての入出力ポートは入力となっています。方向を切替えるには *USBM_PortSetDir()* 関数を使用してください。設定用データの各ビットと端子の関係は表 25 の場合と同様です。

C 言語の例

```
BYTE Data;

USBM_PortRead8(hDev, USBM_P2, &Data); /*ポート 2 からリード*/
USBM_PortSetDir(hDev, USBM_P4, 0x0f); /*ポート 40～43 を出力に設定*/
USBM_PortWrite8A(hDev, USBM_P4, Data, 0xff); /*ポート 40～43 に出力*/
```

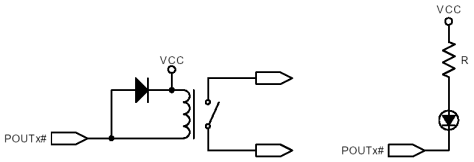
VisualBasic6.0 の例

```
Dim Data As Byte

USBM_PortRead8 hDev, USBM_P2, Data  'ポート 2 からリード
USBM_PortSetDir hDev, USBM_P4, &HF 'ポート 40～43 を出力に設定
USBM_PortWrite8A hDev, USBM_P4, Data, &HFF 'ポート 40～43 に出力
```

- 出力専用ポート以外の出力ピンは LED を駆動できません。出力専用ポート以外の 1 ピンあたりのドライブ能力は 2mA です。マイコンチップ全体でも 40mA に制限されます。
- POUT は H8/3069RF(H8/3029)の複数ポート(P6、P8、PB)を 1 種類のポートと見たててソフトウェアで制御しています。そのため POUT に書き込みを行った場合、ビット毎にデータが反映されるまでの時間に数μsec から 10 数μsec の差が生じますのでご注意ください。

POUTx# 端子の接続例

下の図は POUTx#端子にリレーや LED を接続する場合の例です。



リレーの接続例　　LEDの接続例

LED を接続する場合の直列抵抗(R) 値と抵抗の消費電力(P)は次の式で求めます。

$R = (Vcc - Vf) / If\ [\Omega]$

$P = R \times If^2\ [W]$

Vf ： LED の順方向電圧　If ： LED の順方向電流

リレーをご使用の場合には、リレーメーカーの注意事項をお守りください。

□ バス

『USBM3069-HS(L)』ではマイコンのバスに接続されたメモリタイプのデバイスを簡単な関数呼び出しで扱うことができます。

外部デバイスへは最大で 4 種類のチップセレクト信号(CSx#)を出力でき、それぞれについて 1M バイトのメモリ空間、合計で 4M バイトの外部アドレス空間を使用できます(図 5 参照)。チップセレクト信号は、接続するデバイス毎に使い分けることで外部デコード回路を簡略化できます。

表 26、表 27 ににバスアクセスに使用する端子と関数をあげます。

表 26　バスのアクセスに使用する端子

ピン番	信号名	説明
CN2-23～CN2-42	A0～A19	アドレスバス
CN1-3～CN1-18	D0～D15	データバス
CN1-44	CS0#	エリア 0 へのアクセスを示すチップセレクト信号
CN1-46	CS2#	エリア 2 へのアクセスを示すチップセレクト信号
CN1-47	CS3#	エリア 3 へのアクセスを示すチップセレクト信号
CN1-43	CS5#	エリア 5 へのアクセスを示すチップセレクト信号
CN2-15	HWR#	ライトストローブ。D8-D15 の値が有効である場合”Lo”になります
CN2-16	LWR#	ライトストローブ。D0-D7 の値が有効である場合”Lo”になります
CN2-18	RD#	リードストローブ
CN2-17	AS#	アドレスバス上のアドレス出力が有効であることを示すストローブ

表 27　バスアクセスで使用する関数

関数名	説明
USBM_AddressEnable()	アドレス出力を有効にします。
USBM_CSEnable()	CS2#、CS3# を有効にします。
USBM_BusSetWait()	外部バスのアクセスステートとウェイトを設定します。
USBM_PortWrite8()	任意のアドレスに 1 バイトのデータを書き込みます。
USBM_PortRead8()	任意のアドレスから 1 バイトのデータを読み出します。
USBM_PortCopy8()	任意のアドレス間で 1 バイトのデータをコピーします。
USBM_PortBWrite()	複数バイトのデータをまとめて書き込みます。
USBM_PortBRead()	複数バイトのデータをまとめて読み出します。
USBM_PortBCopy()	複数バイトデータをまとめてコピーします。
USBM_BusSetWidth()	指定のエリアのバス幅を変更します。

バスはエリア毎に 8 ビットアクセスとするか 16 ビットアクセスとするかを選択できます。

8 ビット空間へのアクセスはデータバスの上位 8 ビット(D8-D15)を使用して行います(図 18)。

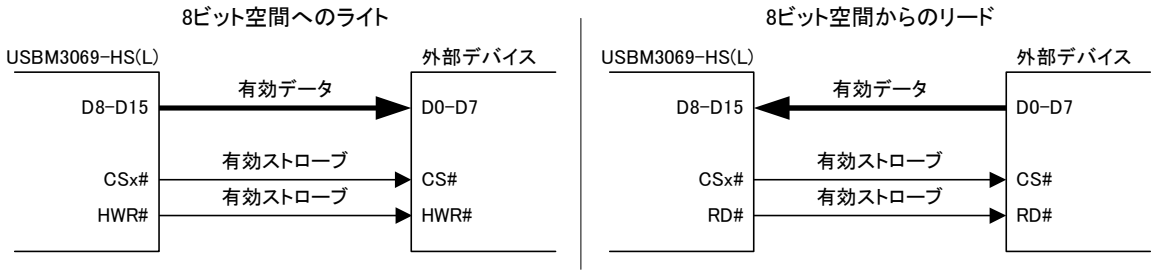


図 18　8 ビット空間へのアクセス

16 ビット空間へのワードアクセスではデータバスの全てのビット(D0-D15)が有効になります（図

19)。

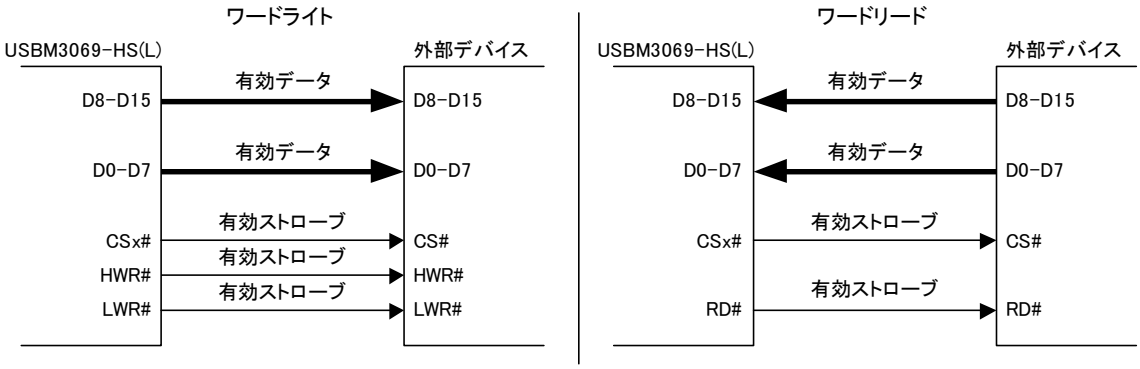


図 19　16 ビット空間へのワードアクセス

16 ビット空間へバイトアクセスする場合は、偶数アドレスへのアクセスではデータバスの上位 8 ビット(D8-D15)、奇数アドレスへのアクセスでは下位 8 ビット(D0-D7)が使用されます。また、ライトアクセスでは HWR#、LWR#によってデータバスの上位バイトと下位バイトのどちらが有効かを示します(図 20)。

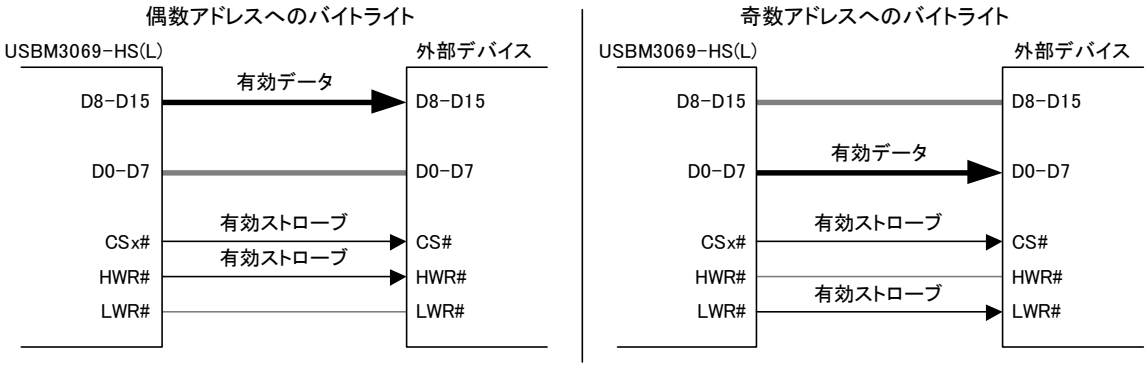


図 20　16 ビット空間へのバイトアクセス

- 16 ビット幅に選択した空間が 1 つでもある場合は、CN1-11 から CN1-18 までの端子はデータバスに設定されます。P40-P47 の入出力ポートとしては使用できなくなりますのでご注意ください。

バスへのアクセスタイミングは 2 ステートアクセス、3 ステートアクセスから選択することができます。また、3 ステートアクセスの場合には 3 ステートまでのソフトウェアウェイトを挿入することが可能です。図 21 に基本的な制御タイミングを示します。初期状態では、ユーザーに開放された全ての外部アドレス空間は 3 ステートアクセス、1 ソフトウェアウェイトに設定されています。

アクセスタイミングの詳細は H8/3069RF(H8/3029)のハードウェアマニュアルを参照してください。

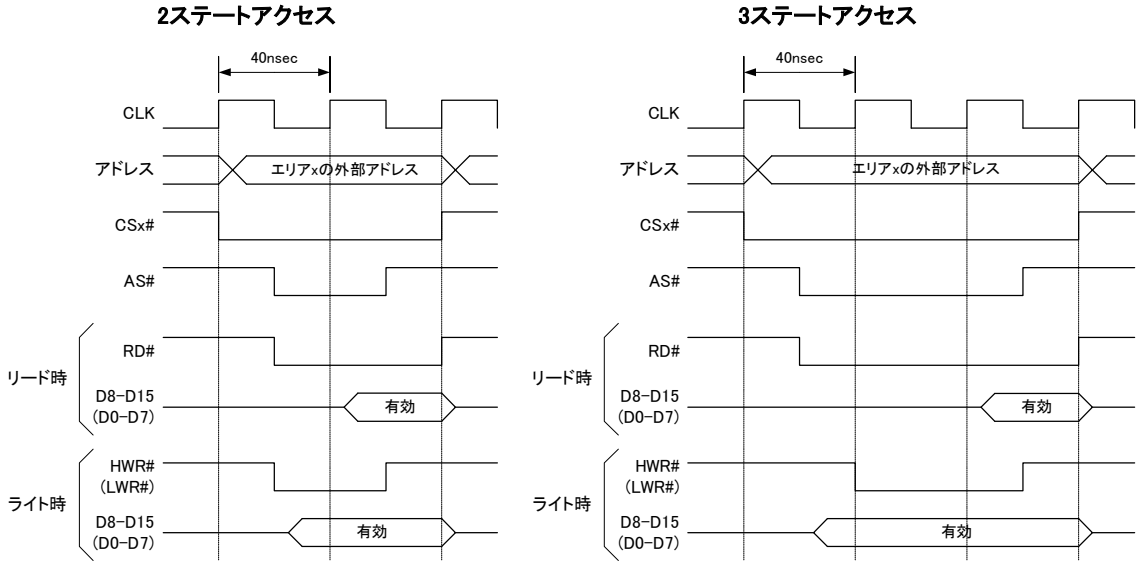


図 21　基本バス制御信号タイミング

バスの使用準備

① *USBM_AddressEnable()* 関数を使用し、アドレスを出力できるように設定します。

② エリア 2 またはエリア 3 にアクセスする場合は、*USBM_CSEnable()* 関数を使用し、対応する CSx# 信号を出力するように設定します。

③ 使用するエリアを 16 ビット空間とする場合は、*USBM_BusSetWidth()* 関数を使用します。

④ アクセスステート、または、ソフトウェアウェイトを設定する場合は、*USBM_BusSetWait()* 関数を使用します。

C 言語の例

```
USBM_AddressEnable(hDev, 16);                          //アドレスバスを下位 16 ビット出力
USBM_CSEnable(hDev, USBM_AREA2 | USBM_AREA3);          //CS2#と CS3#を出力
USBM_BusSetWidth(hDev, USBM_AREA2, TRUE);              //エリア 2 を 16 ビット空間に設定
USBM_BusSetWait(hDev, USBM_AREA2, USBM_BUS_2STATE); //エリア 2 を 2 ステートアクセスに設定
USBM_BusSetWait(hDev, USBM_AREA3, USBM_BUS_WAIT2);  //エリア 3 を 2 ウェイトに設定
```

VisualBasic6.0 の例

```
USBM_AddressEnable hDev, 16                        'アドレスバスを下位 16 ビット出力
USBM_CSEnable hDev, USBM_AREA2 Or USBM_AREA3       'CS2#と CS3#を出力
USBM_BusSetWidth hDev, USBM_AREA2, True            'エリア 2 を 16 ビット空間に設定
USBM_BusSetWait hDev, USBM_AREA2, USBM_BUS_2STATE 'エリア 2 を 2 ステートアクセスに設定
USBM_BusSetWait hDev, USBM_AREA3, USBM_BUS_WAIT2   'エリア 3 を 2 ウェイトに設定
```

バスへのアクセス

ホストパソコンからマイコンのアドレス空間に書き込みを行うには主に *USBM_PortBWrite()* 関数を、読み出しには *USBM_PortBRead()* 関数を使用します。また、マイコンのメモリ間でのデータ転送に

は *USBM_PortBCopy()* 関数を使用します。

これらの関数ではデータ転送にDMAを使用することができます。DMAによるデータ転送は、マイコンの CPU によるデータ転送よりも高速です。マイコンには DMA チャンネルが 2 チャンネル内蔵されていますが、関数により使用されるチャンネルが違います。表 28 に各関数に使用される DMA チャンネルを示します。

表 28　ポート・バスのアクセス時に使用される DMA チャンネル

関数	DMA チャンネル
USBM_PortBWrite()	0
USBM_PortBRead()	1
USBM_PortBCopy()	0

C 言語の例

```
char Data[65536];

//エリア 2 の先頭から DMA を使用して 64K バイトリード
USBM_PortBRead(hDev, USBM_AREA2_TOP, Data, 65536, 1, TRUE);

//エリア 3 の先頭に 64K バイトライト
USBM_PortBWrite(hDev, USBM_AREA3_TOP, Data, 65536, 1, FALSE);

//エリア 2 の先頭からユーザーメモリへ 8192 バイトコピー
USBM_PortBCopy(hDev, USBM_AREA2_TOP, USBM_USER_AREA, 8192, 1, 1);
```

VisualBasic6.0 の例

```
Dim Data(65535) As Byte

'エリア 2 の先頭から DMA を使用して 64K バイトリード
USBM_PortBRead hDev, USBM_AREA2_TOP, Data, 65536, 1, True

'エリア 3 の先頭に 64K バイトライト
USBM_PortBWrite hDev, USBM_AREA3_TOP, Data, 65536, 1, False

'エリア 2 の先頭からユーザーメモリへ 8192 バイトコピー
USBM_PortBCopy hDev, USBM_AREA2_TOP, USBM_USER_AREA, 8192, 1, 1
```

USB インタフェース IC との接続バス幅

デバイスと接続する際、*USBM_OpenA()* や *USBM_OpenByPI()* 関数の *Opt* 引数に"*USBM_MODE_BUS16*"オプションを指定することで、搭載マイコンと USB インタフェース IC を接続するバス幅が 16 ビットに拡張されます。これにより大きなのデータを転送する場合のスループットを向上させることができます。

接続方法の詳細は「デバイスのオープン」33 ページをご参照ください。

□ AD コンバータ

『USBM3069-HS(L)』では 10 ビットの AD コンバータを最大で 4 チャンネル利用できます。API 関数の呼び出しにより、AD 端子に入力されたアナログ電圧を 10 ビットのデジタルデータに変換して読み出すことができます。表 29、表 30 に AD コンバータで使用する端子、関数をあげます。

表 29　AD コンバータで使用する端子

ピン番	信号名	説明
CN2-8	AD0	チャンネル 0 アナログ入力
CN2-7	AD1	チャンネル 1 アナログ入力
CN2-6	AD2	チャンネル 2 アナログ入力
CN3-5	AD3	チャンネル 3 アナログ入力
CN2-9	AVCC	アナログ電源入力
CN2-10	VREF	リファレンス電圧入力
CN1-45	ADTRG#	外部トリガを使用する場合のトリガ入力

表 30　AD コンバータで使用する関数

関数名	説明
USBM_ADRead()	AD 変換を 1 回行い、結果を返します。
USBM_ADSetCycle()	連続して AD 変換を行う場合の変換周期を設定します。
USBM_ADBRead()	指定回数の AD 変換を連続して行い、結果を返します。
USBM_ADStart()	指定回数の AD 変換を連続して行います。この関数では変換と読み出しを非同期に行えます。
USBM_GetQueueStatus()	*USBM_ADStart()* で変換した結果が USB のリードバッファに何バイトあるか調べます。
USBM_Read()	*USBM_ADStart()* で変換した結果を USB のリードバッファから読み出します。
USBM_Abort()	*USBM_ADStart()* での変換を中止する場合や、*USBM_ADBRead()* がタイムアウトした場合に使用します。
USBM_Purge()	USB のリードバッファをクリアするのに使用します。
USBM_ReadStatus()	*USBM_ADBRead()*、または *USBM_ADStart()* による AD 変換中に、変換データが正しく転送されたかどうかを知るのに使用します。
USBM_ADCopy()	指定回数の AD 変換を連続して行い、デバイス上のメモリに結果を保存します。変換速度が他の関数よりも高速です。
USBM_ADReadCopyStatus()	*USBM_ADCopy()* の進行状況を読み出します。
USBM_ADReadBuffer()	*USBM_ADCopy()* で変換した結果を、デバイス上のメモリから読み出します。
USBM_ADStopCopy()	*USBM_ADCopy()* による変換を終了します。

AD/DA 共用のアナログ電源用端子 AVCC 端子とリファレンス入力用の VREF 端子があります。VREF 端子、AVCC 端子にはより精度の高いリファレンスを与えたい場合や、電源ノイズの影響を減らしたい場合に外部からアナログ用の電源を入力します。

『USBM3069-HS』では、AVCC 端子は 5V±10%、VREF 端子は 4.5V～AVCC としてください。

『USBM3069-HSL』では、AVCC 端子は 3.0～3.6V、VREF 端子は 3.0～AVCC としてください。

- AD コンバータ、DA コンバータを使用しない場合でも、AVCC および VREF 端子がオープンとならないようにしてください（VCC に接続してください）。

アナログ入力電圧と出力コードの関係を図 22 に示します。また、変換結果は図 23 のように 16 ビット変数の上位 10 ビットに納められ、下位 6 ビットは常に 0 となります。

変換結果は、符号無し整数として返されますので、Visual Basic で開発される場合は、ヘルパー関数を使用して *Long* 型変数に変換してください。

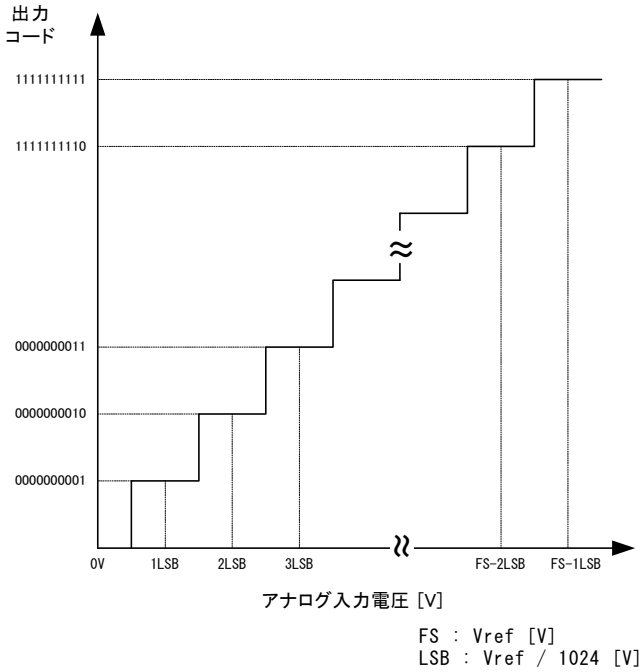


図 22　アナログ入力電圧と出力コードの関係

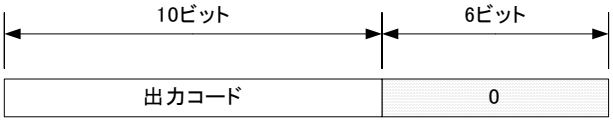


図 23　変換結果の格納方法

AD コンバータの特性を表 31 に示します。測定条件、特性項目の意味、その他仕様などは H8/3069RF(H8/3029)のハードウェアマニュアルを参照してください。十分な変換特性を得るために、同マニュアルの 15.6 節「使用上の注意」をご一読いただくことをお勧めいたします。

表 31　AD 変換特性

項目		min	max	単位
変換時間: 134 ステート(CKS=0)	アナログ入力範囲	0	VREF	V
	分解能	10	10	bit
	変換時間(単一モード)		5.36	μsec
	変換時間(連続変換中)	5.12	5.12	μsec
	アナログ入力容量		20	pF
	許容信号源インピーダンス		5	kΩ
	非直線性誤差		±3.5	LSB
	オフセット誤差		±3.5	LSB
	フルスケール誤差		±3.5	LSB
	量子化誤差		±0.5	LSB
	絶対精度		±4.0	LSB
変換時間: 70 ステート(CKS=1)	アナログ入力範囲	0	VREF	V
	分解能	10	10	bit
	変換時間(単一モード)		2.8	μsec
	変換時間(連続変換中)	2.64	2.64	μsec
	アナログ入力容量		20	pF
	許容信号源インピーダンス		3	kΩ
	非直線性誤差		±7.5	LSB
	オフセット誤差		±7.5	LSB
	フルスケール誤差		±7.5	LSB
	量子化誤差		±0.5	LSB
	絶対精度		±8.0	LSB

AD 変換結果を得る方法は大きく分けて 4 つの方法があります。

- 単純に命令発行時のアナログ電圧値を読み出す、*USBM_ADRead()* 関数を使用する方法。
- タイマまたは外部トリガに同期して連続で変換結果を得る *USBM_ADBRead()* 関数を用いる方法。
- タイマまたは外部トリガに同期して連続で変換した結果を、ホストパソコンでバッファリングし、逐次データを取り出せる *USBM_ADStart()* 関数を使用する方法。
- マイコンの最高速度で連続変換した結果を、DMA を用いてデバイス内のメモリにバッファリングする *USBM_ADCopy()* 関数を使用する方法。

表 32 は、それぞれの変換方法の特徴をまとめたものです。

表 32　AD 変換の方法と特徴

代表関数名	変換レート	プログラム	複数チャンネル*1	逐次読出し	特徴
USBM_ADRead()	低(数 msec)	容易	可	–	使い方が簡単ですが、変換レートが使用環境に依存します。直流向き。
USBM_ADBRead()	中(9μsec)	容易	不可	不可	使い方が簡単ですが、複数のチャンネルのスキャンができません。
USBM_ADStart()	中(9μsec/ch)	やや複雑	可	可	複数チャンネルのスキャンもでき、変換結果を逐次取り出せます。
USBM_ADCopy()	高(2.8μsec/ch)*2	やや複雑	可	不可	変換レートが最高で、変換中のデバイスアクセス可能です。

*1 全く同時に変換できるのは 1 チャンネルです。複数チャンネルの場合、スキャンモードを使用した順次スキャンになります。
*2 変換レートの設定はできません。常に最大のレートで変換します。

USBM_ADRead() を使用する(命令毎に変換)

USBM_ADRead() 関数を使用します。複数のチャンネルを同時に読み出すこともできます。関数を呼び出すと、ホストパソコンからデバイスに変換コマンドが送信され、デバイスは指定チャンネルの AD 変換を行い、ホストパソコンに変換結果を返します。

命令を呼び出して実際にサンプリングが行われるまでの時間は不定です(一般に数 msec のオーダーとなります)。繰り返し呼び出した場合の変換間隔も一定とはなりませんので、交流信号の変換には向きません。使い方が単純ですので直流信号を読み取るには適しています。

全ての変換方法に共通してマイコンの AD 変換回路は、完全に同時に複数のアナログ信号をサンプリングすることはできません。同時にサンプリングおよび変換が可能になるのは常に 1 チャンネルのみです。図 24 は 3 チャンネルの変換を行ったときの様子を示します。図中の変換時間 t_c は最初の 1 チャンネルが最大 5.34μsec、残りのチャンネルは 5.12μsec となります。

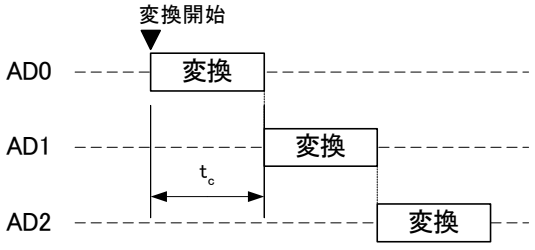


図 24　複数チャンネルの AD 変換の様子

C 言語の例

```
WORD ADData[4];

USBM_ADRead(hDev, ADData, 3, TRUE);          /*0-3 チャンネル全て読み出し*/
```

VisualBasic6.0 の例

```
Dim ADData(3) As Integer
Dim i As Integer

USBM_ADRead hDev, ADData, 3, 1  '0-3 チャンネル全て読み出し

Dim ADData32(3) As Long

For i = 0 To 3
  ADData32(i) = USBM_ToINT32(ADData(i)) '符号無し整数を正しく読むために変換
Next i
```

USBM_ADBRead() を使用する(連続で変換)

USBM_ADBRead() 関数を使用すると、予め設定した変換レートで連続サンプリングを行うことができます。変換タイミングはマイコンの 8 ビットタイマを利用して作られます(外部から入力することも可能)。変換周期の最小値は 9μsec です。

タイマコピー、パルスカウンタなどの割込みを利用する機能と同時に使用すると、変換が正しいタイミングで行われない可能性がありますのでご注意ください。また、1 回の変換毎に結果は USB を通じてホストパソコンに送られますが、接続された USB ポートの通信状態やご使用の環境により、変換データを全て送れない場合があります(図 25 参照)。最高レート近くでご使用の場合には、ホストパソコンの同一コントローラ上に他のデバイスを接続しないようにしてください。

上記の理由で変換や転送が正しく行われなかった可能性がある場合には、デバイスのステータスに *USBM_STS_TIMEOUT* のビットが立ちます。ステータスは *USBM_ReadStatus()* 関数で取得することができます。

USBM_ADRead() 関数では複数チャンネルをサンプリングすることはできません。

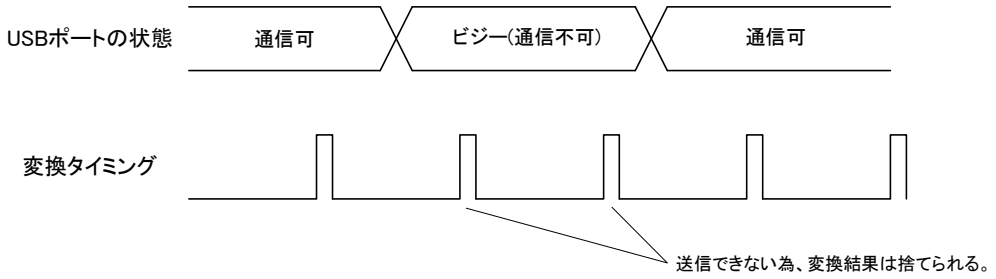


図 25　AD 変換結果の送信不能状態

① H8/3069RF(H8/3029)の 8 ビットタイマを使用して AD 変換のタイミングを作るには、*USBM_ADSetCycle()* 関数を使用します。変換周期は以下のようになります。

$$Tc = (Cmp+1) / f_{clk}\ [s] \quad (f_{clk}: \text{CLK で選択した周波数})$$

初期状態では外部トリガ信号(ADTRG#)によって変換を開始する設定となっています。

② *USBM_ADBRead()* 関数を呼び出すと 8 ビットタイマのコンペアマッチまたは ADTRG#の立下りの度に 1 回の変換を行い、指定回数終了した時点で関数からリターンします。連続変換で読み出すことのできるのは 0～3 チャンネルの中から指定した 1 チャンネルのみです。

C 言語の例

```
WORD Data[100];

USBM_ADSetCycle(hDev,99,USBM_TCLK390); /*約 3.9kHz で変換*/
USBM_ADBRead(hDev,Data,100,0,NULL);    /*チャンネル 0 を 100 回サンプリング*/
```

VisualBasic6.0 の例

```
Dim Data(99) As Integer
Dim i As Integer

USBM_ADSetCycle hDev, 99, USBM_TCLK390 '約 3.9kHz で変換
USBM_ADBRead hDev, Data, 100, 0, 0     'チャンネル 0 を 100 回サンプリング

Dim Data32(99) As Long

For i = 0 To 99
  Data32(i) = USBM_ToINT32(Data(i))  '符号無し整数を正しく読むために変換
Next i
```

USBM_ADStart() を使用する(変換しながらデータを取り出す)

USBM_ADStart() 関数も *USBM_ADBRead()* 関数の場合と同様に、予め設定した変換レートで連続してサンプリングを行うことができます。変換データを 16 ビットで取り出す場合には、最小変換時間は 9 μ sec/チャンネルとなりますので、4 チャンネル全てを同時に使用する場合には、1 回の変換に 9 ×4 = 36 μ sec 必要になります。

USBM_ADStart() を呼び出すとデバイスは、AD 変換を開始しますが、関数自体はすぐにリターンします。変換結果はデバイスからホストパソコンへ送られ、パソコン上のメモリにバッファリング[8]されます。変換中ホストパソコンのプログラムはブロックされませんので、他の非同期関数を呼び出した場合と同様に、メッセージ処理や画面描画などを行うことができます。ただし、デバイス自身は *USBM_Abort()* による中断コマンド以外を受け付けませんのでご注意ください。

バッファに溜まったデータのバイト数を知るには *USBM_GetQueueStatus()* 関数を、データを取り出すには *USBM_Read()* 関数を呼び出してください(これらの関数はデバイスにコマンドを送らないので呼び出し可能です)。

8ビットタイマで変換周期を作る場合で、*USBM_ADStart()* 呼び出し時に *Trig* 引数に *TRUE*を指定すると、ADTRG#入力により、タイマが起動される設定となります(図 26 参照)。この場合、ADTRIG# 信号は 1 回だけで良く、AD コンバータ自体はタイマから起動されます。図中の t_d は ADTRG#が入力されてからタイマが起動されるまでの遅延時間で 0.8 μ sec～2 μ sec の間で変化します。T_C は *USBM_ADSetCycle()* で設定した変換周期です。

[8] 64K バイトまでバッファできます。

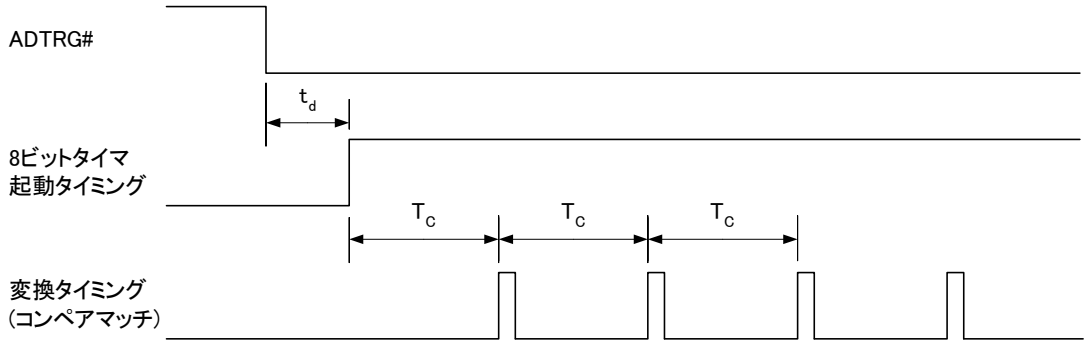


図 26　*USBM_ADStart()* の *Trig* 指定時の変換タイミング

USBM_ADBRead() の場合と同様、タイマコピー、パルスカウンタなどの割り込み、USB の使用状況、使用環境によって、影響を受けます。また、ステータスも同じように *USBM_ReadStatus()* 関数で取得することができます。

① H8/3069RF(H8/3029)の 8 ビットタイマを使用して AD 変換のタイミングを作るには、*USBM_ADSetCycle()* 関数を使用します。変換周期は以下のようになります。

$$Tc = (Cmp+1) / f_{clk} \text{ [sec]} \quad (f_{clk}: CLK \text{ で選択した周波数})$$

初期状態では外部トリガ信号(ADTRG#)によって変換を開始する設定となっています。

② *USBM_ADStart()* 関数を呼び出すと、デバイスは 8 ビットタイマのコンペアマッチまたは ADTRG#の立下りの度に 1 回の変換を行い、結果をホストパソコンに送信します。*Trig* を *TRUE* とした場合は、ADTRG#が入力されるまでタイマの起動を待機します。

③ *USBM_GetQueueStatus()* 関数を使用して受信バッファに蓄えられたデータのバイト数を取得します。

④ 必要な量のデータがバッファリングされたら、*USBM_Read()* 関数を使用して読み出します。複数チャンネルの変換の場合には、データは AD0、AD1、AD2、AD3、AD0、・・・のように順番に読み出されます。各データのサイズは *USBM_ADStart()* の *lByte* 引数の値により、1 バイト、または 2 バイトとなります。

⑤ 中断する場合には *USBM_Abort()* 関数を呼び出してください。受信バッファにデータが残っている場合は、*USBM_Read()* 関数で全て読み出すか、*USBM_Purge()* 関数で受信バッファをクリアしてください。

- 受信バッファ内に AD 変換データが残っていると、以降のデバイス制御が不能になりますので、必ずクリアするようにしてください。

C 言語の例

```
WORD wBuff[1024];
DWORD n;

USBM_ADSetCycle(hDev,38,USBM_TCLK390);     /* 約 10kHz で変換 */
USBM_ADStart(hDev,-1,3,TRUE,FALSE,FALSE); /* 停止するまで全チャンネル変換 */
while(1){
  USBM_GetQueueStatus(hDev,&n);  /* 変換されたデータ数を読み出す */
  if(n >= 2048) break;           /* 2048 バイト(1024 ワード)変換されたら抜ける*/
}
USBM_Read(hDev,wBuff,2048,&n); /* 変換結果の読み出し */
USBM_Abort(hDev);              /* 変換の終了 */
USBM_Purge(hDev,USBM_PURGE_RX);  /* 受信バッファをクリア */
```

VisualBasic6.0 の例

```
Dim wBuff(1023) As Integer
Dim n As Integer
Dim i As Integer

USBM_ADSetCycle hDev, 38, USBM_TCLK390    '約 10kHz で変換
USBM_ADStart hDev, -1 , 3, 1, 0, 0 '停止するまで全チャンネル変換

Do
  USBM_GetQueueStatus hDev, n  '変換されたデータ数を読み出す */
  If n >= 2048 Then Exit Do    '2048 バイト(1024 ワード)変換されたら抜ける
Loop

USBM_Read hDev, wBuff, 2048, n '変換結果の読み出し

USBM_Abort hDev               '変換の終了
USBM_Purge hDev, USBM_PURGE_RX  '受信バッファをクリア

Dim lBuff(1023) As Long

For i = 0 To 1023
  lBuff(i) = USBM_ToINT32(wBuff(i)) '符号無し整数を正しく読むための変換
Next i
```

USBM_ADCopy() を使用する(最大レートで変換する)

USBM_ADCopy() 関数は AD コンバータの最大の変換レートでサンプリングを行うことができます。サンプリングデータは主にマイコンの内蔵 RAM に蓄えられますが、外部バスにより大きなメモリを搭載すれば、それだけ多くのデータを一度にサンプリング可能になります。

常に最大レートで変換を行うため、*USBM_ADSetCycle()* の設定値は無視されます。変換開始のトリガ信号は ADTRG#端子より入力します。図 27 に *USBM_ADCopy()* を使用した場合の、変換の様子を示します。実際にはADTRG#信号はマイコン内部でサンプリングされ、マイコンの内部クロックに同期して変換が開始されます。

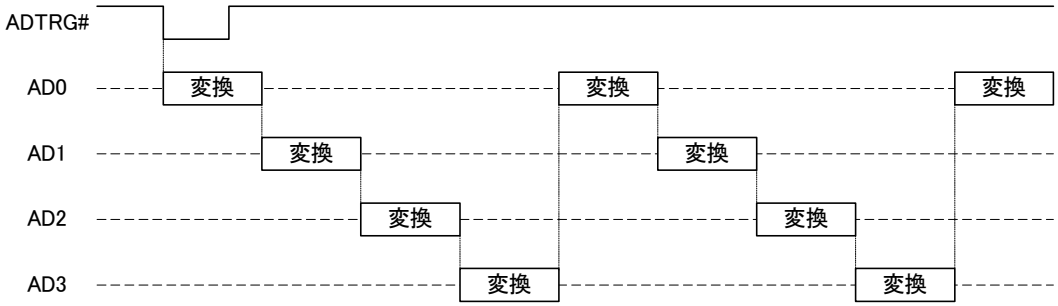


図 27 *USBM_ADCopy()* の変換の様子

また、変換は常に AD0 から開始され、指定の終了チャンネルまで行われます。1 チャンネルのみの変換を行う場合には、常に AD0 を使用する必要がありますのでご注意ください。

USBM_ADCopy() 関数を呼び出す際に *CKS* に *TRUE* を指定すると、変換ステート数を減らして、高速に変換ができるようになります。変換精度は低下しますが(表 31 参照)、精度よりも速度を優先する場合には有効になります。

変換データのメモリへの転送は DMA が使用されます。その為、変換動作中であっても、他の関数を呼び出して操作を行うことができます。ただし、他の用途で同一の DMA チャンネルが使用されないように注意が必要です。

指定された回数の変換が終了したかどうかを調べるためには *USBM_ADReadCopyStatus()* 関数を使用します。また、変換終了時、もしくは変換を中断する場合には *USBM_ADStopCopy()* 関数を呼び出してください。

① *USBM_ADCopy()* 関数を呼び出します。

② ADTRG#端子に信号が入力されると(“Lo”になると)、変換が開始されます。変換が終了したかどうかを調べるためには、*USBM_ADReadCopyStatus()* 関数を呼び出してください。

③ 指定回数の変換が終了したら *USBM_ADReadBuffer()* 関数を使用して結果を呼び出します。

④ *USBM_ADStopCopy()* 関数で終了処理をします。中断する場合にも、*USBM_ADStopCopy()* 関数を使用します。

C 言語の例

```
WORD wBuff[1024];
long n;

USBM_ADCopy(hDev,USBM_USER_AREA,256,3,FALSE,0,TRUE); /* 0-3 チャンネルを 256 回変換 */
while(1){ /* 変換が終わるまでループ(ADTRG#が入力されるまで変換開始されません) */
  USBM_ADReadCopyStatus(hDev,&n,0); /* 残りの変換数を調べる */
  if(n == 0) break; /* 変換が終わっていたら抜ける */
}
USBM_ADReadCopyBuffer(hDev,USBM_USER_AREA,wBuff,1024,TRUE,FALSE);
USBM_ADStopCopy(hDev,0);
```

VisualBasic6.0 の例

```
Dim wBuff(1023) As Integer
Dim n As Long
Dim i As Integer

USBM_ADCopy hDev, USBM_USER_AREA, 256, 3, 0, 0, 1 '0-3 チャンネルを 256 回変換
Do '変換が終わるまでループ(ADTRG#が入力されるまで変換開始されません)
  USBM_ADReadCopyStatus hDev, n, 0 '残りの変換数を調べる
  If n = 0 Then Exit Do '変換が終わっていたら抜ける
Loop
USBM_ADReadCopyBuffer hDev, USBM_USER_AREA, wBuff, 1024, 1, 0
USBM_ADStopCopy hDev, 0

Dim lBuff(1023) As Long

For i = 0 To 1023
  lBuff(i) = USBM_ToINT32(wBuff(i)) '符号無し整数を正しく読むために変換
Next i
```

アナログ入力端子の保護

アナログ入力端子に誤って GND～VREF の範囲外の電圧が加わる恐れがある場合には、ダイオードなどで保護回路を設けてください。図 28 に保護回路の例を示します。図の例で瞬間的に AVCC より大きな電圧が印加された場合は保護可能ですが、過電圧状態が長くなると AVCC が規定値以上になり破壊に至る可能性がありますので、入力信号は VREF を超えないようにしてください。

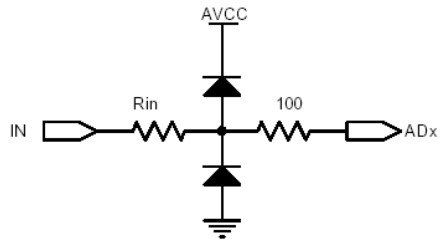


図 28　アナログ入力端子の保護回路例

□ DA コンバータ

『USBM3069-HS(L)』では 2 チャンネルの 8 ビット DA コンバータを利用できます。DAC レジスタを書き換えることにより、DA 出力端子の電圧を設定できます。また、16 ビットタイマと DMA を利用して、ハードウェアのみで高速に DAC レジスタを書き換えることができます。この方法は、予め設定した波形パターンを再生するような用途に向いています。データは DMA で自動的に転送されるため、デバイスは変換中であっても、他の命令を処理することが可能です。この機能を利用した場合、DA コンバータ 1 チャンネルにつき、16 ビットタイマと DMA を 1 チャンネルずつ使用します。16 ビットタイマと DMA は使用する DA と同一のチャンネルが使用されます。例えば、DA0 を利用する際は、タイマと DMA も 0 チャンネルが使用できなくなります。

表 33　DA コンバータで使用する端子

ピン番	信号名	説明
CN2-2	DA0	チャンネル 0 アナログ出力
CN2-1	DA1	チャンネル 1 アナログ出力
CN2-9	AVCC	アナログ電源入力
CN2-10	VREF	リファレンス電圧入力

表 34　DA コンバータで使用する関数

関数名	説明
USBM_PortWrite8()	DA コンバータに値を設定します。
USBM_PortBWrite()	連続して DA 変換する場合の変換データをデバイス上のメモリに転送します。
USBM_DASetCycle()	DA コンバータの変換周期を設定します。
USBM_DASetParm()	連続して DA 変換する場合のパラメータを設定します。
USBM_DAReadStatus()	連続して DA 変換する場合の未変換のデータ数を調べます。
USBM_DAStart()	連続 DA 変換を開始します。
USBM_DAStop()	連続 DA 変換を終了します。

DA コンバータのアナログ出力電圧は以下の式で計算できます。

$$Vout = DAC\ レジスタ設定値\ /\ 256\ \times\ VREF\ [V]$$

ボード上にはアナログ回路用の電源端子(AVCC)、リファレンス入力端子(VREF)があります。これらの端子の設定方法は AD コンバータの項を参照してください。表 35 に DA 変換の特性をあげます。測定条件などの詳細は H8/3069RF(H8/3029)のハードウェアマニュアルを参照してください。

表 35　DA 変換特性

項目	Min.	Typ.	Max.	単位	測定条件
分解能	8	8	8	bit	
変換時間	−	−	10	μsec	負荷容量 20pF
絶対精度	−	±1.5	±2.0	LSB	負荷抵抗 2MΩ
	−	−	±1.5		負荷抵抗 4MΩ

アナログ出力電圧を変更する

USBM_PortWrite8() 関数を使用し、レジスタ値を書き換えます。レジスタのアドレスは API のヘッダーファイル中で *USBM_DA0,USBM_DA1* で定義されています。命令を呼び出して実際にアナログ出力に反映されるまでの時間は不定です(一般に 1msec 程度です)。繰り返し呼び出した場合の変換間隔も一定とはなりませんので、決まった波形パターンを再生するような用途には向きません。使い方が単純ですので、直流値を出力するような場合に有効です。

DMA を使用して高速に変換する

① DA 変換するデータをユーザーメモリの任意の位置に *USBM_PortBWrite()* 関数で書き込んでおきます。

② *USBM_DASetCycle()* 関数を使用して DA コンバータの変換サイクルを決定します。

③ *USBM_DASetParm()* 関数を使用してパラメータを設定します。ソースアドレスには①でデータを書き込んだアドレスを指定してください。また、*lLoop* を *TRUE* とすることで中断を指示するまで、繰り返し変換を行うことが可能ですが、その場合、*nData* に指定できるデータ数が 255 に制限されますのでご注意ください。

④ *USBM_DAStart()* 関数を呼び出して変換を開始します。*USBM_DAReadStatus()* で残りのデータ数を読み出すことができます。

⑤ 終了処理のために *USBM_DAStop()* 関数を呼び出してください。また、中断する場合にもこの関数を用います。

C 言語の例

```
int i;
BYTE buff[255];

/* のこぎり波を出力します */
for(i=0;i<255;i++) buff[i] = (BYTE)i; /* 変数を初期化 */

USBM_PortBWrite(hDev,USBM_USER_AREA,buff,255,TRUE,FALSE);

USBM_DASetCycle(hDev,0,249,USBM_TCLK25000); /* 変換周期を 10us に設定 */
USBM_DASetParm(hDev,0,USBM_USER_AREA,255,TRUE); /* 255 バイトのデータを繰り返し変換 */
USBM_DAStart(hDev,0x01); /* 開始 */

/* ... */

USBM_DAStop(hDev,0x01); /* 終了 */
```

VisualBasic6.0 の例

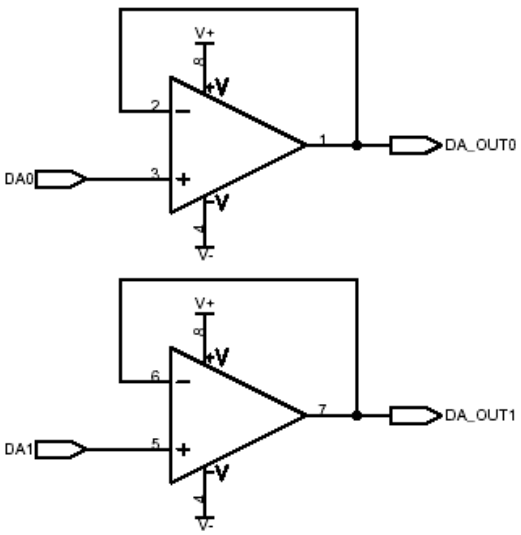
```
Dim i As Integer
Dim buff(254) As Byte

'のこぎり波を出力します
For i = 0 To 254
  buff(i) = i '変数を初期化
Next i

USBM_PortBWrite hDev, USBM_USER_AREA, buff, 255, 1, 0

USBM_DASetCycle hDev, 0, 249, USBM_TCLK25000 '変換周期を 10us に設定
USBM_DASetParm hDev, 0, USBM_USER_AREA, 255, 1 '255 バイトのデータを繰り返し変換
USBM_DAStart hDev, &H1 '開始

'...

USBM_DAStop hDev, &H1 '終了
```

DA 出力のバッファリング

DA コンバータは出力インピーダンスが高いため、負荷抵抗の値が小さくなると精度が悪化します。出力電流を増やすためにはバッファを追加してください。図はオペアンプのボルテージ・フォロアによるバッファ回路です。

図中の V+、V- はそれぞれオペアンプの正負の電源ですが、V+ = AVCC、V- = GND とすれば電源回路を省略できます。その場合オペアンプにはレール・トゥ・レール入出力のタイプ(AD8030、TS922 など)のものを選択してください。

0～AVCC 電圧まで完全に出力する場合には V+ > AVCC 、V- < GND となるように電源を別途用意する必要があります。



□ 16 ビットタイマ（PWM）

『USBM3069-HS(L)』では 3 チャンネルの 16 ビットタイマチャンネルを利用できます。16 ビットタイマは多くの機能を持っており、様々なアプリケーションで利用可能です。下に 16ビットタイマを利用した機能をあげます。

- 最大 3 相の PWM 出力
- 位相計数モードを使用した 2 相ロータリーエンコーダ出力のカウント
- インプットキャプチャ機能を利用した、時間測定
- 任意のパルス幅、セットアップ時間を指定できるシングルパルス出力
- DA コンバータの変換タイミングの生成(「DA コンバータ」参照)

また、パルスカウンタ(マイコンの割り込み)への信号入力によって、任意のタイマチャンネルのスタート、及び、ストップが可能です。この機能を利用することで、ホストパソコンを介することなく、16 ビットタイマをコントロール可能となり、リアルタイム性を要求される用途にも応用可能となります(「パルスカウンタ」参照)。表 36、表 37 に 16 ビットタイマで使用する端子と関数をあげます。

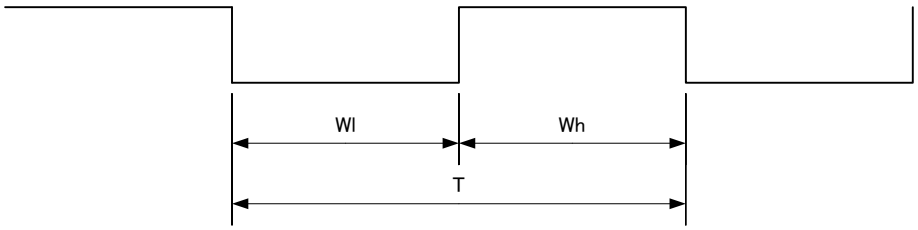
表 36　16 ビットタイマで使用する端子

ピン番	信号名	説明
CN1-40	TIOCA0	0 チャンネルのパルス出力、インプットキャプチャ入力、シングルパルス出力
CN1-39	TIOCB0	0 チャンネルのインプットキャプチャ入力
CN1-38	TIOCA1	1 チャンネルのパルス出力、インプットキャプチャ入力、シングルパルス出力
CN1-37	TIOCB1	1 チャンネルのインプットキャプチャ入力
CN1-36	TIOCA2	2 チャンネルのパルス出力、インプットキャプチャ入力
CN1-35	TIOCB2	2 チャンネルのインプットキャプチャ入力
CN1-42	TCLKA	外部クロック入力または位相計数モードでのパルス入力
CN1-41	TCLKB	外部クロック入力または位相計数モードでのパルス入力

表 37　16 ビットタイマで使用する関数

関数名	説明
USBM_TimerEnable()	指定チャンネルを PWM 出力モードに設定します。
USBM_TimerSetLevel()	タイマ出力端子を Lo または Hi に設定します。
USBM_TimerSetPulse()	PWM 出力の周期とデューティを設定します。
USBM_TimerSetClk()	タイマで使用する基準クロックを選択します。
USBM_TimerSetCnt()	タイマカウンタの初期値を設定します。
USBM_TimerReadCnt()	タイマカウンタの値を読み出します。
USBM_TimerSetCmp()	コンペアレジスタの値を設定します。
USBM_TimerSetCmpOut()	チャンネル 2 のコンペアマッチ時のポート操作を設定します。
USBM_TimerSetCmpClr()	チャンネル 2 のコンペアマッチ時にカウンタをクリアするように設定します。
USBM_TimerStartA()	指定チャンネルのタイマをスタートします。
USBM_TimerStop()	指定チャンネルのタイマをストップします。
USBM_TimerSetCapture()	インプットキャプチャの設定を行います。
USBM_TimerSetCaptureCnt()	インプットキャプチャの実行前にキャプチャ値を保存する GRA、GRB レジスタをクリアするのに使用します。
USBM_TimerReadCaptureCnt()	インプットキャプチャの結果を読み出します。
USBM_TimerSetSinglePulse()	シングルパルス出力の設定を行います。
USBM_TimerStart()	タイマをスタート、またはストップします。

PWM は最大 3 相の出力が可能です。*USBM_TimerEnable()* 関数で PWM 出力に設定されたチャンネルの TIOCAx 端子は自動的に出力端子になります。各チャンネルは 16 ビットカウンタとコンペアレジスタ A、コンペアレジスタ B という 16 ビットレジスタを持っており、選択されたクロックの入力によりカウンタの値がインクリメントされます。カウンタの値とコンペアレジスタ A の値が一致すると出力端子は 1 となります。また、さらにカウントアップされコンペアレジスタ B の値と一致すると出力端子は 0 となり、カウンタの値はリセットされて 0 に戻ります。つまり、クロックとコンペアレジスタ A、B の値を適当な値に設定することで PWM 出力を得ることができます。



$$T = (CmpB + 1) \times \frac{1}{CLK} \quad (S) \qquad Wl : Wh = CmpA + 1 : CmpB - CmpA$$

CmpA : USBM_TimerSetPulse()のLtoHの値
CmpB : USBM_TimerSetPulse()のHtoLの値
CLK : USBM_TimerSetClk()で設定したクロック値

図 29 PWM パルス

タイマチャンネル 2 は位相計数モードで使用可能です。位相計数モードとは 2 相エンコーダのパルスカウントを自動的に行うモードで、このモードに設定すると CN2-13 ピン、CN2-12 ピンがそれぞれ TCLKA、TCLKB 入力端子となり、この 2 つの端子に入力されるパルスの位相関係により、下の表のようにカウンタ値が自動的にアップ、またはダウンします。

表 38 位相計数モードのカウント方法

カウント方向	カウントダウン				カウントアップ			
TCLKA 端子	↑	Hi	↓	Lo	Lo	↑	Hi	↓
TCLKB 端子	Lo	↑	Hi	↓	↑	Hi	↓	Lo

↑:立上り ↓:立下り

さらにチャンネル 2 は *USBM_TimerSetCmpOut()* 関数で設定を行うと、コンペアレジスタ A、コンペアレジスタ B の 2 つのレジスタとカウンタとのコンペアマッチによりそれぞれ 1 つの 8 ビットポートに書き込み操作を行うことができます(コンペアアウト)。

インプットキャプチャは 0～2 全てのチャンネルで使用できます。インプットキャプチャの設定を行ったチャンネルは TIOCAx と TIOCBx 端子が自動的に入力端子となります。タイマをスタートするとタイマカウンタは、選択されたクロック入力に伴いインクリメントされます。このとき TIOCAx、または、TIOCBx 端子にパルスが入力されると、そのときのカウンタ値が入力端子に対応したレジスタに書き込まれます(図 30 参照)。インプットキャプチャを利用すると、パルス幅を測定したり、二つの信号の間隔を測定したりが可能です。TIOCAx、TIOCBx 端子は立上り、立下り、または、両エッジを検出するようにプログラム可能です。

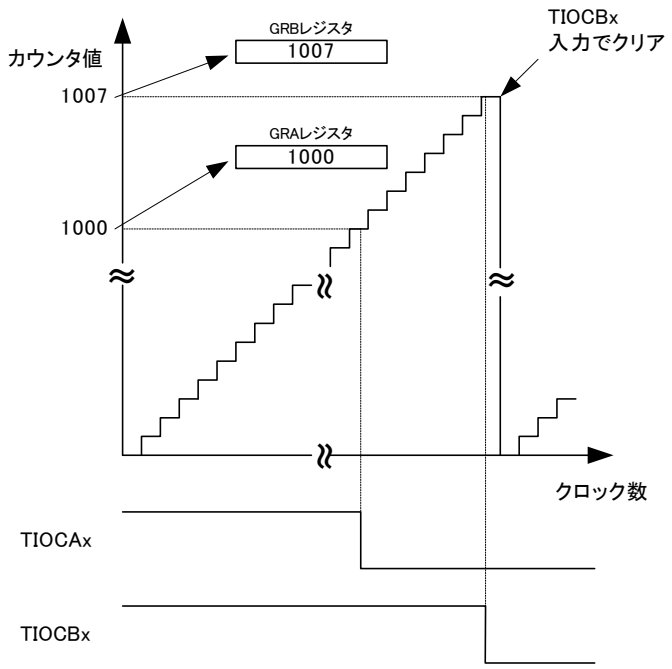


図 30　インプットキャプチャ

DMA コントローラ

「H8/3069RF(H8/3029)」は DMA(Direct Memory Access)コントローラを 2 チャンネル搭載しています。DMA コントローラを利用すると、メモリ、I/O ポート、内部レジスタなどの間で高速にデータ転送ができます。DMA を利用すると以下のメリットがあります。

- CPU よりも高速にデータ転送が行えます。
- データ転送による CPU への負荷を減らすことができます。

『USBM3069-HS(L)』ではホストパソコンとの通信、ポート間のデータコピー、AD コンバータ、DA コンバータなどで DMA を利用しています。これらの機能を同時に使用する場合には、チャンネルが制限されたり、使用不可能になったりする場合がありますのでご注意ください。詳しくは「複数機能の同時使用」をご覧ください。

シングルパルス出力はチャンネル 0 とチャンネル 1 で利用可能です。*USBM_SetSinglePulse()* 関数でパルス出力に設定されると、該当チャンネルの TIOCAx 端子は自動的に出力端子になります。*USBM_TimerStartA()* 関数を呼び出すと、設定したセットアップ時間とパルス幅で 1 回だけパルスが出力されます。パルス幅は 40nsec～20msec まで設定可能ですので、決まった時間のパルスが必要な場合にご使用ください。また、2 チャンネルを同時に使用すると、図 31 のように少しだけタイミングをずらした 2 つのパルスを得ることも可能です。

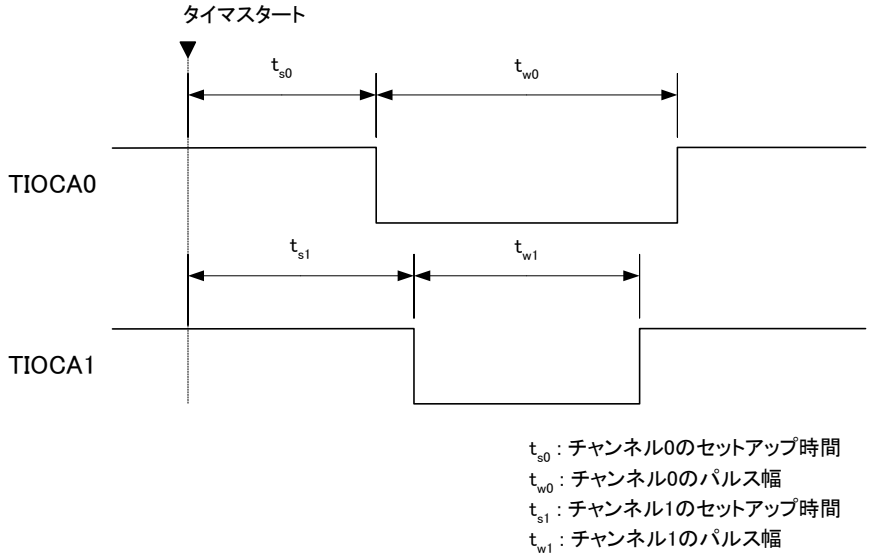


図 31　シングルパルス出力

PWM パルスを出力する

① *USBM_TimerSetClk()* 関数を使用してカウントに使用するクロックを選択します。

② *USBM_TimerSetPulse()* 関数を使用し、パルスの周期とデューティを決定します。

③ 必要な場合は *USBM_TimerSetLevel()* 関数でタイマ出力ピンの初期値を指定し、*USBM_TimerSetCnt()* 関数で開始時点での位相を設定します。

④ *USBM_TimerEnable()* 関数で使用するタイマチャンネルを指定します。この時点で指定されたチャンネルの TIOCA ピンは出力になります。

⑤ *USBM_TimerStartA()* 関数で使用するチャンネルをスタートさせるとパルスが出力されます。*USBM_TimerStartA()* 呼び出し時に *TrigPC* にパルスカウンタのチャンネルを指定すると、該当する PCx#信号の立下りを検出した時点でタイマがスタートされます。

⑥ 停止する場合は *USBM_TimerStop()* 関数を呼び出します。*USBM_TimerStop()* 呼び出し時に *TrigPC* にパルスカウンタのチャンネルを指定すると、対応する PCx#信号の立下りを検出した時点でタイマが停止されます。

⑦ タイマ出力を止めるには *USBM_TimerEnable()* 関数を、該当チャンネルのビットを 0 にして呼び出します。

⑧ パルスカウンタをトリガとして使用した場合は、使用したチャンネルを *USBM_PCStop()* 関数を用いて停止してください（パルスカウンタのスタートは自動的に行われます）。

C 言語の例

```
/*PWM 出力*/
USBM_TimerSetClk(hDev,0,USBM_TCLK12500); /*12.5MHz のクロックを選択*/
USBM_TimerSetClk(hDev,1,USBM_TCLK12500);
USBM_TimerSetPulse(hDev,0,4999,9999);  /*デューティ 50%,周波数 1.25kHz*/
USBM_TimerSetPulse(hDev,1,7499,9999);  /*デューティ 25%,周波数 1.25kHz*/
USBM_TimerSetCnt(hDev,0,0);
USBM_TimerSetCnt(hDev,1,8749);      /*位相を進める*/
USBM_TimerSetLevel(hDev,0x04);      /*チャンネル 1 の初期値を 1 にする*/
USBM_TimerEnable(hDev,0x03,FALSE); /*チャンネル 0,1 を出力*/
USBM_TimerStartA(hDev,0x03,-1,FALSE); /*チャンネル 0,1 をスタート*/

/*...*/

USBM_TimerStop(hDev,0x03);          /*タイマをストップ*/
USBM_TimerEnable(hDev,0x00,FALSE); /*タイマ出力を終了*/
```

VisualBasic6.0 の例

```
'PWM 出力
USBM_TimerSetClk hDev, 0, USBM_TCLK12500'12.5MHz のクロックを選択
USBM_TimerSetClk hDev, 1, USBM_TCLK12500
USBM_TimerSetPulse hDev, 0, 4999, 9999  'デューティ 50%,周波数 1.25kHz
USBM_TimerSetPulse hDev, 1, 7499, 9999  'デューティ 25%,周波数 1.25kHz
USBM_TimerSetCnt hDev, 0, 0
USBM_TimerSetCnt hDev, 1, 8749  '位相を進める
USBM_TimerSetLevel hDev, &H4'チャンネル 1 の初期値を 1 にする
USBM_TimerEnable hDev, &H3, 0   'チャンネル 0,1 を出力
USBM_TimerStartA hDev, &H3, -1, 0 'チャンネル 0,1 をスタート

'...

USBM_TimerStop hDev, &H3,  'タイマをストップ
USBM_TimerEnable hDev, &H0, 0   'タイマ出力を終了
```

位相計数カウンタを使用する

① *USBM_TimerEnable()* 関数を呼び出します。*MDF* 引数はTRUEとします。通常はチャンネル2をイネーブルにする必要はありません。

② コンペアマッチでカウンタをクリアする場合には *USBM_TimerSetClr()* 関数を使用します。

③ コンペアアウトを使用する場合には *USBM_TimerSetCmpOut()* 関数を使用します。

④ コンペアマッチによるクリア、もしくはコンペアアウト機能を使用する場合は *USBM_TimerSetCmp()* 関数でコンペアレジスタA、Bを設定します。

⑤ *USBM_TimerStartA()* 関数でタイマチャンネル2をスタートさせます。

⑥ TCLKA、TCLKB端子へのパルス入力でカウンタの値が変化します。*USBM_TimerReadCnt()* 関数で現在のカウンタ値を読み出します。

⑦ 計数を停止する場合は *USBM_TimerStop()* 関数でチャンネル2を停止します。

- タイマのコンペアマッチはクロック同期で発生します。そのため、コンペアアウト、コンペアマッチによるクリアは、タイマのカウンタとコンペアレジスタが一致した次のクロック入力(パルス入力)により行われます。例えばコンペアレジスタの値が 100 の場合、カウンタの値が 100 になった瞬間ではなく、次の 101 をカウントしたときにコンペアマッチが発生します。また、位相計数カウンタの場合ではカウンタ値が 100 になった時点でコンペアマッチが確定し、次のパルスでカウンタが 101 になる場合でも、99 になる場合でもコンペアマッチは発生します。

C 言語の例

```
short Count;

/*位相計数*/
USBM_TimerEnable(hDev,0,TRUE);    /*チャンネル 2 を位相計数モードに設定*/
USBM_TimerSetCmp(hDev,100,-100); /*100 と-100 でコンペアマッチ*/
USBM_TimerSetCmpOut(hDev,0,USBM_POUT,0xff); /*コンペアマッチ A で出力ポートに 0xff を出力*/
USBM_TimerSetCmpOut(hDev,1,USBM_POUT,0x00); /*コンペアマッチ B で出力ポートに 0x00 を出力*/
USBM_TimerSetCmpClr(hDev,0x03);  /*コンペアマッチが発生した場合にカウンタをクリア*/
USBM_TimerStartA(hDev,0x04,-1,FALSE);  /*カウントスタート*/

/*...*/

USBM_TimerReadCnt(hDev,2,&Count); /*カウンタの読み出し*/

/*...*/

USBM_TimerStop(hDev,0x04); /*カウント終了*/
```

VisualBasic6.0 の例

```
Dim Count As Integer

'位相計数
USBM_TimerEnable hDev, 0, 1 'チャンネル 2 を位相計数モードに設定
USBM_TimerSetCmp hDev, 100, -100'100 と-100 でコンペアマッチ
USBM_TimerSetCmpOut hDev, 0, USBM_POUT, &HFF 'コンペアマッチ A で出力ポートに FFh を出力
USBM_TimerSetCmpOut hDev, 1, USBM_POUT, &H0  'コンペアマッチ B で出力ポートに 00h を出力
USBM_TimerSetCmpClr hDev, &H3   'コンペアマッチが発生した場合にカウンタをクリア
USBM_TimerStartA hDev, &H4, -1, 0  'カウントスタート

'...

USBM_TimerReadCnt hDev, 2, Count'カウンタの読み出し

'...

USBM_TimerStop hDev, &H4   'カウント終了
```

インプットキャプチャを使用する

① *USBM_TimerSetClk()* 関数で使用するチャンネルのクロックを選択します。

② *USBM_TimerSetCapture()* 関数で使用するチャンネルをインプットキャプチャモードに設定します。

③ *USBM_TimerSetCaptureCnt()* 関数で現在の GRA、GRB レジスタ(キャプチャ値を保存するレジスタ)をクリアします。

④ *USBM_TimerStartA()* 関数でタイマをスタートします。

⑤ *USBM_TimerReadCaptureCnt()* 関数で GRA、GRB レジスタの値を読み出します。

⑥ *USBM_TimerStop()* 関数で、使用したチャンネルを停止します。

- 実際にインプットキャプチャを使用する際には、タイマ動作と同期していない信号のキャプチャは上手く行えない場合があります。使用しないチャンネルをシングルパルス出力などに設定し、トリガ信号として使用されることをお勧めします。

シングルパルス出力を使用する

① *USBM_TimerSetLevel()* 関数を用い、使用する TIOCAx 端子の出力レベルを設定します。正極性のパルス出力の場合は 0 に、負極正の場合は 1 に設定してください。

② *USBM_TimerSetSinglePulse()* 関数を呼び出し、パルスの設定を行います。

③ *USBM_TimerStartA()* 関数を呼び出して、使用するチャンネルをスタートするとパルスが出力されます。タイマの停止は自動的に行われます。

- パルス幅を 64000 カウント以上にすると、正しく動作しない場合があります。

C 言語の例

```
/* シングルパルスをトリガとして応答のパルス幅を測定する例 */
long CntA, CntB;

/* タイマ 0 をインプットキャプチャのトリガ出力として使用する */
USBM_TimerSetClk(hDev, 0, USBM_TCLK25000); /* カウントクロックを設定(25MHz); */
/* タイマ 0 が出力になったとき TIOCA0 端子が「Hi」となるように初期値を設定 */
USBM_TimerSetLevel(hDev, 0x01);
/* トリガパルスの設定(セットアップ時間 400nsec, パルス幅 200nsec) */
USBM_TimerSetSinglePulse(hDev, 0, 10, 5, FALSE);

/* タイマ 1 をインプットキャプチャモードに設定 */
USBM_TimerSetClk(hDev, 1, USBM_TCLK25000); /* カウントクロックを設定(25MHz) */
USBM_TimerSetCaptureCnt(hDev, 1, 0, 0); /* GRA,GRB レジスタのクリア */
/* GRA に TIOCA1 の立下りエッジを、GRB に TIOCB1 の立上りエッジをキャプチャ */
USBM_TimerSetCapture(hDev, 1, USBM_CAPT_FALL, USBM_CAPT_RISE);
USBM_TimerStartA(hDev, 0x03, -1, FALSE, 1); /* タイマをスタート */

/* 必要な場合 Sleep()などで時間を確保する */

USBM_TimerStop(hDev, 0x03, -1, FALSE, 1); /* タイマを停止 */
USBM_TimerReadCaptureCnt(hDev, 1, &CntA, &CntB); /* キャプチャ結果の読み出し */
```

VisualBasic6.0 の例

```
'シングルパルスをトリガとして応答のパルス幅を測定する例
Dim CntA As Long, CntB As Long

'タイマ 0 をインプットキャプチャのトリガ出力として使用する
USBM_TimerSetClk hDev, 0, USBM_TCLK25000 'カウントクロックを設定(25MHz)
'タイマ 0 が出力になったとき TIOCA0 端子が「Hi」となるように初期値を設定
USBM_TimerSetLevel hDev, &H1
'トリガパルスの設定(セットアップ時間 400nsec, パルス幅 200nsec)
USBM_TimerSetSinglePulse hDev, 0, 10, 5, 0

'タイマ 1 をインプットキャプチャモードに設定
USBM_TimerSetClk hDev, 1, USBM_TCLK25000 'カウントクロックを設定(25MHz)
USBM_TimerSetCaptureCnt hDev, 1, 0, 0 'GRA,GRB レジスタのクリア
'GRA に TIOCA1 の立下りエッジを、GRB に TIOCB1 の立上りエッジをキャプチャ
USBM_TimerSetCapture hDev, 1, USBM_CAPT_FALL, USBM_CAPT_RISE
USBM_TimerStartA hDev, &H3, -1, 0, 1 'タイマをスタート

'必要な場合、時間を確保する

USBM_TimerStop hDev, &H3, -1, 0, 1 'タイマを停止
USBM_TimerReadCaptureCnt hDev, 1, CntA, CntB 'キャプチャ結果の読み出し
```

□ パルスカウンタ

『USBM3069-HS(L)』はマイコンの割り込み端子を利用した 32 ビットパルスカウンタを最大 4 チャンネル利用できます(PC0～PC3)。また、単にパルス数をカウントするという使い方の他に、カウンタ値がある値に到達したときに、マイコン内の任意のアドレスに対して、予め設定した値を書き込むコンペアアウトという機能があります。書き込むアドレスには出力ポートは勿論、マイコンの内部レジスタを設定することもできます。この機能は、『USBM3069-HS(L)』でリアルタイム処理を行う上で重要な機能です。表 39、表 40 にパルスカウンタで使用する端子と関数をあげます。また、表 41 はパルスカウンタに入力可能なパルス周波数です。

表 39　パルスカウンタで使用する端子

ピン番	信号名	説明
CN1-24	PC0#	パルスカウンタ 0 入力
CN1-23	PC1#	パルスカウンタ 1 入力
CN1-47	PC2#	パルスカウンタ 2 入力
CN1-46	PC3#	パルスカウンタ 3 入力

- PC0#、PC1#はシュミットトリガ入力ではありません。入力信号の立上り、立下りが緩やかな場合、実際よりも多くカウントされる恐れがあります。必要に応じてシュミットトリガ入力のバッファなどを追加してください。

表 40　パルスカウンタで使用する関数

関数名	説明
USBM_PCSetCmp()	コンペアレジスタの値を設定します。
USBM_PCSetCnt()	カウンタの値を設定します。
USBM_PCReadCnt()	カウンタの値を読み出します。
USBM_PCSetControl()	パルスカウンタのカウントアップ/ダウンの方法、クリア条件を設定します。
USBM_PCSetCondBit()	パルスカウンタのアップ/ダウンを決定する条件ビットを指定します。
USBM_PCSetCmpOut()	コンペアマッチ時に出力を行うポートとデータを設定します。
USBM_PCStartA()	指定チャンネルのカウントをスタートします。
USBM_PCStop()	指定チャンネルのカウントをストップします。
USBM_PCStart()	パルスカウンタのカウントをスタートまたは停止します。

表 41　パルスカウンタに入力可能なパルス周波数

チャンネル	周波数(周期)
0	33.3KHz(30 μ sec)
1～3	40KHz(25 μ sec)

- 表 41 は 1 チャンネルだけ使用した場合の仕様です。パルスカウンタは割り込みを利用しソフトウェアで実装されていますので、複数のチャンネルを同時に使用するとその分、処理時間が必要になり入力可能なパルス周波数は低下します。また、タイマコピーなどの他の割り込み処理と併用すると、直ちにカウントが行われない場合がありますのでご注意ください。

パルスカウンタをスタートすると、チャンネルに対応する端子から入力される信号の立下りで、カウンタが 1 ずつ増加、または減少します。カウンタが増加するか減少するかは、*USBM_PCSetControl()*

関数で設定しますが、その際、*USBM_PCSetCondBit()* で指定されたコンディションビットが参照されます。コンディションビットには、マイコン内の任意アドレスの任意ビットが選択可能で、このビットの状態によって、カウンタが入力パルスにより、増加するのか、減少するのか、パルスを無視するのかを決定することができます。例えば、ポート 1 の P10 端子をコンディションビットとして指定した場合、この端子が“Lo”の場合はパルス入力でカウントアップ、“Hi”の場合は逆にカウントダウン、というような設定が可能になります。図 32 に *USBM_PCSetControl()* 関数の *Bits* 引数の意味を、表 42 に *Bits* 引数の設定値とカウンタの増減の関係を示します。

図 32　*USBM_PCSetControl()* の *Bits* 引数

表 42　*USBM_PCSetControl()* によるカウント方法の設定

USBM_PCSetControl() の引数(Bits)		コンディションビットによる増減	
ビット 5	ビット 4	コンディションビット = 0 のとき	コンディションビット = 1 のとき
0	0	カウントアップ	
0	1	カウントアップ	カウントダウン
1	0	カウントダウン	カウントアップ
1	1	カウントしない	カウントアップ

コンディションビットには、一般のポート入力以外に、別のパルスカウンタチャンネルの入力端子を選択することが可能です。これを、利用すると 16 ビットタイマのチャンネル 2 による位相計数カウントの様に 2 相ロータリーエンコーダ出力のカウントに使用できます(立上りはカウントできませんので、カウント数は 16 ビットタイマの場合の半分、角度精度は 1/3 になります)。図 33 にチャンネル 0 とチャンネル 1 を使用した 2 相ロータリーエンコーダ出力のカウントの様子を示します。

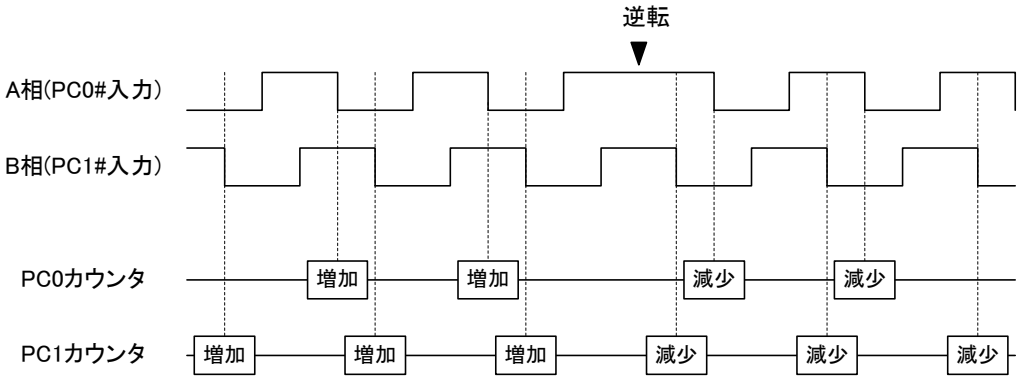


図 33　2 相ロータリーエンコーダ出力のカウント

- 上記の方法は信号の立下りのみカウントしますので、パルス発生位置でエンコーダが正転、逆転を繰り返すと誤ったカウントを行う場合があります。パルスカウンタにインクリメンタル式ロータリーエンコーダの 2 相パルス出力を接続する場合の回路例とサンプルプログラムが、添付 CD の「¥SAMPLE¥M3069_Samples¥EncoderSample」にございますのでご参照ください。
- パルスカウンタの機能は割込みを利用してソフトで実装されていますので、1 回のカウント処理に 1 チャンネルあたり最大 30μsec の時間を要します。周波数の高い信号のカウントには不向きです。16 ビットタイマの位相計数機能を優先してご利用ください。

全てのチャンネルには32ビットのコンペアレジスタが用意されており、カウンタ値とのコンペアマッチによりそれぞれ 1 つの 8 ビットポートに書き込み操作を行うことができます。この機能をコンペアアウトと呼びます。この機能を利用すると、外部からデバイスへの信号入力に対するフィードバックを素早く行うことが可能になります。図 34 にチャンネル 0 へのパルス入力に応答して、ある出力ポートの値を変化させる例を示します。図中の t_d はパルス入力から実際にポート操作が行われるまでの遅延時間で、8μsec～20μsec となります。同じ処理をホストパソコンを介して行った場合、カウンタ値の読み出しに数 msec、ポート出力に数 msec の合わせて 10msec 近い時間を要してしまいます。

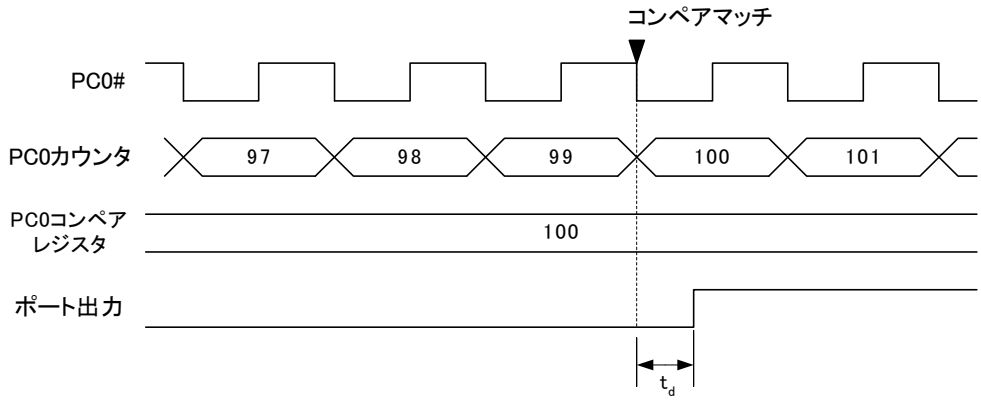


図 34　パルスカウンタのコンペアアウト

この機能を利用してパルスカウンタへの入力をトリガとして、16 ビットタイマの起動と終了、タイマコピー機能の起動と終了が簡単に行えるようになっています。

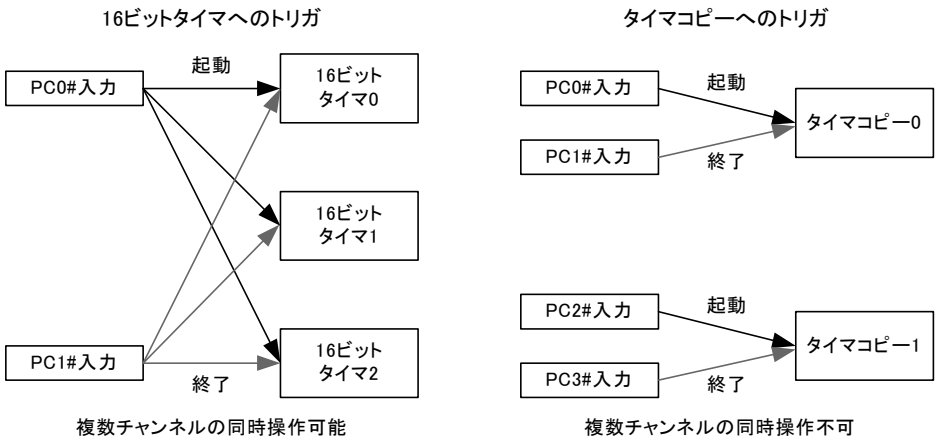


図 35　パルスカウンタ入力による 16 ビットタイマとタイマコピーの操作

単相パルスをカウントする

① カウンタに初期値を設定する場合には *USBM_PCSetCnt()* 関数を使用します。

② コンペアマッチによるリセットやコンペアアウト機能を使用する場合は *USBM_PCSetCmp()* 関数でコンペアレジスタを設定します。

③ パルス入力により、カウントアップするか、カウントダウンするか、パルスを無視するかを任意のポートの、任意のビット(コンディションビット)で指定することができます。この機能を使用するためには *USBM_PCSetCondBit()* 関数でポートとコンディションビットを抜き出すためのビットマスクを指定します。

④ 必要な場合には *USBM_PCSetControl()* 関数でクリアの条件とコンディションビットの状態でどのようにカウントするかを決定します。初期状態ではパルスを無条件でカウントし、クリアは自動的に行われない設定になっています。

⑤ *USBM_PCStartA()* 関数でカウントを開始します。

⑥ *USBM_PCReadCnt()* 関数で現在のカウント値を読み出します。

⑦ カウントを終了するには *USBM_PCStop()* 関数を呼び出します。

2 相パルスをカウントする

2 相のパルスをカウントするにはパルスカウンタを 2 チャンネル使用します。各チャンネルの設定の仕方は上の単相の場合と同様です。ただし、③と④で呼び出す *USBM_PCSetCondBit()* 関数と *USBM_PCSetControl()* 関数に特定の値を指定します。ここでは、PC0#、PC1#を使用する場合を例に説明します。

① *USBM_PCSetCondBit()* 関数を0チャンネルを対象に呼び出します。その際、ポートアドレスに定数 *USBM_PC1_ADDR*、をマスクに定数 *USBM_PC1_BIT* を指定します。C 言語での呼び出しは以下のようになります。

```
USBM_PCSetCondBit(hDev, 0, USBM_PC1_ADDR, USBM_PC1_BIT);
```

同様に 1 チャンネルを対象に *USBM_PCSetCondBit()* 関数を呼び出します。ポートアドレスに定数 *USBM_PC0_ADDR*、マスクに定数 *USBM_PC0_BIT* を指定します。これでお互いのパルス入力をコンディションビットに指定したことになります。

② 次に *USBM_PCSetControl()* 関数を 0 チャンネルを対象に呼び出します。その際、コントロールビットに H'10 を指定します。C 言語での呼び出しは以下のようになります。

```
USBM_PCSetControl(hDev, 0, 0x10);
```

同様に1チャンネルの設定ではコントロールビットに H'20 を指定します。これで 2 相用の設定は終わりです。カウント数を読む場合はチャンネル 0、1 両方の値を読み出し、足し合わせます。他の手順は単相の場合と同様です。

C 言語の例

```
long Count[4];
long Sum;

/*チャンネル 0 と 1 を使用した位相計数*/
USBM_PCSetCnt(hDev,0,0);  /*カウンタのクリア*/
USBM_PCSetCnt(hDev,1,0);
USBM_PCSetCondBit(hDev,0,USBM_PC1_ADDR,USBM_PC1_BIT); /*カウントの条件をそれぞれ設定*/
USBM_PCSetCondBit(hDev,1,USBM_PC0_ADDR,USBM_PC0_BIT);
USBM_PCSetControl(hDev,0,0x10); /*コンディションビットによる加算､減算の設定*/
USBM_PCSetControl(hDev,1,0x20);
USBM_PCStartA(hDev,USBM_PC0 | USBM_PC1); /*計数をスタート*/

/*...*/

USBM_PCReadCnt(hDev,USBM_PC_ALL,Count);  /*まとめて読み出すと高速*/
Sum = Count[0] + Count[1]; /*チャンネル 0 と 1 の値を足すとカウント数が得られる*/

/*...*/

USBM_PCStop(hDev,USBM_PC_0 | USBM_PC1); /*計数終了*/
```

VisualBasic6.0 の例

```
Dim Count(3) As Long
Dim Sum As Long

'チャンネル 0 と 1 を使用した位相計数
USBM_PCSetCnt hDev, 0, 0
USBM_PCSetCnt hDev, 1, 0
USBM_PCSetCondBit hDev, 0, USBM_PC1_ADDR, USBM_PC1_BIT 'カウントの条件をそれぞれ設定
USBM_PCSetCondBit hDev, 1, USBM_PC0_ADDR, USBM_PC0_BIT
USBM_PCSetControl hDev, 0, &H10 'コンディションビットによる加算、減算の設定
USBM_PCSetControl hDev, 1, &H20
USBM_PCStartA hDev, (USBM_PC0 Or USBM_PC1) '計数をスタート

'...

USBM_PCReadCnt hDev, USBM_PC_ALL, Count 'まとめて読み出すと高速
Sum = Count(0) + Count(1) 'チャンネル 0 と 1 の値を足すとカウント数が得られる

'...

USBM_PCStop hDev, (USBM_PC0 Or USBM_PC1) '計数終了
```

□ SCI(シリアル通信)

『USBM3069-HS(L)』はTTLレベルのシリアル通信チャンネルを2つ利用可能です。通信方式は調歩同期のみです。通信速度は 300bps～38400bps でフロー制御はありません。受信バッファは 127 バイトで、オーバーフローするとSCI用のステータスレジスタにエラーをセットし、オーバーフローしたデータは捨てられます。表 43、表 44 にSCIで使用する端子、関数をあげます。また、表 45 にSCIの仕様を示します。

表 43 SCIで使用する端子

ピン番	信号名	説明
CN1-19	TxD0	SCI0 送信
CN1-20	RxD0	SCI0 受信
CN1-21	TxD1	SCI1 送信
CN1-22	RxD1	SCI1 受信

表 44 SCIで使用する関数

関数名	説明
USBM_SCISetMode()	通信条件の設定を行います。
USBM_SCIReadStatus()	SCIのエラー、受信バイト数を読み出します。
USBM_SCIRead()	SCIから指定バイト数のデータを読み出します。
USBM_SCIWrite()	SCIからデータを送信します。
USBM_SCISetDelimiter()	デリミタ文字を指定します。

表 45 SCIの仕様

項目	仕様
チャンネル数	2
方式	調歩同期式(フロー制御なし)
ビットレート	300～38400bps
信号レベル	TTL

- SCI1 はユーザーファーム開発時に、デバッグ用通信チャンネル、または、標準入出力として利用しますが、USBM ライブラリを用いて SCI1 を操作すると、これらの機能が動作しなくなります。

デリミタ文字を指定しておくと、*USBM_SCIRead()* 呼び出したときにデバイス側でデリミタ文字をチェックし、発見した場合は受信データが指定バイト数に達していなくても、残りのデータを0で埋めて処理を戻します。

データを送信する

① *USBM_SCISetMode()* 関数でボーレート、データビット数、パリティ、ストップビットなどを設定します。

② *USBM_SCIWrite()* 関数で送信したいデータを送ります。

データを受信する

① *USBM_SCISetMode()* 関数でボーレート、データビット数、パリティ、ストップビットなどを設定します。

② 受信データ長が不定で、デリミタ文字によってパケットの区切りを識別する場合には、*USBM_SCISetDelimiter()* 関数を使用します。

③ *USBM_SCIRead()* 関数でデータを受信します。

- *USBM_SCIRead()* 関数を呼び出すと、デバイスは指定バイト数を受信するまで無限ループに入ります。中止方法、タイムアウトの設定方法については 75 ページ「タイムアウト設定」の節を参照してください。
- *USBM_SCIRead()* 関数を呼び出したときに、関数やデバイスが受信データ待ちの状態になるのを防ぐためには、*USBM_SCIReadStatus()* 関数を使用して予め受信バッファ中のデータ数を確認してください。

C 言語の例

```
char Data[6] = “Hello”;

USBM_SCISetMode(hDev,0,USBM_SCI_DATA8 | USBM_SCI_NOPARITY | USBM_SCI_STOP1,
                USBM_SCI_BAUD38400);  /*モードを設定*/

USBM_SCIWrite(hDev,0,Data,6); /*シリアルポートから出力*/
USBM_SCIRead(hDev,0,Data,6,NULL);  /*シリアルポートから読み出し*/
```

VisualBasic6.0 の例

```
Dim Data(5) As Byte

Data(0) = Asc("H")
Data(1) = Asc("e")
Data(2) = Asc("l")
Data(3) = Asc("l")
Data(4) = Asc("o")
Data(5) = Asc(vbNullChar)

USBM_SCISetMode hDev, 0, (USBM_SCI_DATA8 Or USBM_SCI_NOPARITY Or USBM_SCI_STOP1), _
USBM_SCI_BAUD38400  'モードを設定

USBM_SCIWrite hDev, 0, Data, 6  'シリアルポートから出力
USBM_SCIRead hDev, 0, Data, 6, 0   'シリアルポートから読み出し
```

□ タイマコピー

『USBM3069-HS(L)』は H8/3069RF(H8/3029)内蔵の 8 ビットタイマを利用したタイマコピーという機能を実装しています。8 ビットタイマも 16 ビットタイマと同様、各チャンネルにカウンタとコンペアレジスタを持っていますが、そのコンペアマッチにより発生する割り込みを利用してポート間で(あるアドレスからあるアドレスへ)1 バイトずつデータをコピーすることができます。マイコンの内蔵メモリに予めデータを転送しておき、タイマコピーを起動することで、出力ポートの値を一定の周期で更新したり、DA コンバータの出力電圧を自動的に更新したりが可能です。図 36 にパルスモーター用のパターンをポート 4 に出力する様子を示します。また、表 46、表 47 にタイマコピーで使用する端子、関数をあげます。

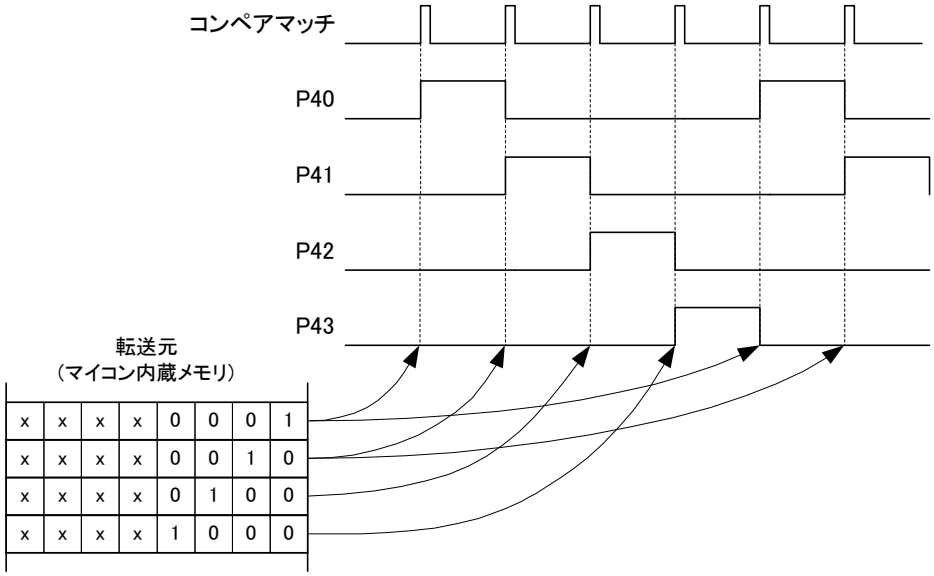


図 36　タイマコピーの動作の様子

表 46　タイマコピーで使用する端子

ピン番	信号名	説明
CN1-42	TCLKA	チャンネル 0 に外部クロックを入力する場合に使用
CN1-41	TCLKB	チャンネル 1 に外部クロックを入力する場合に使用

表 47　タイマコピーで使用する関数

関数名	説明
USBM_TCPYSetParm()	転送の設定を行います。
USBM_TCPYSetPatternCtrl()	出力ポートにパルスパターンを順次出力するよう設定します。
USBM_TCPYSetCycle()	転送周期を設定します。
USBM_TCPYSetTrig()	タイマコピーのトリガ信号を選択します。
USBM_TCPYReadStatus()	転送状況を読み出します。
USBM_TCPYStart()	タイマコピーを開始、終了します。

USBM_TCPYSetPatternCtrl() 関数を使用すると、用意した転送パターンの途中から転送を開始したり、停止させたりということが可能となっています。*USBM_TCPYSetPatternCtrl()* 関数の *SrcPort*

には転送パターンの先頭アドレス、*nSrc* は用意された転送パターンのバイト数です。*nCopy* は何回転送を行うかを設定します。*nCopy* 回の転送途中に転送元アドレスが転送パターンの数を超えると、自動的に転送元をパターンの先頭に戻して転送を続けます。逆に転送パターンの下限を超えた場合も同様に最後尾から転送を続けます。表 48 に *USBM_TCPYSetPatternCtrl()* 関数の引数の設定例、図 37 に、その場合の動作の様子を示します。転送先であるポート 4 の出力値はコンペアマッチが発生とともに、図 37 の右のように変化します。

表 48　*USBM_TCPYSetPatternCtrl()* の設定例

hDev	*CH*	*SrcPort*	*DstPort*	*nCopy*	*nSrc*	*SrcInc*	*Start*	*Mask*
xxx	xxx	0xffbf20	USBM_P4	5	4	1	2	0xff

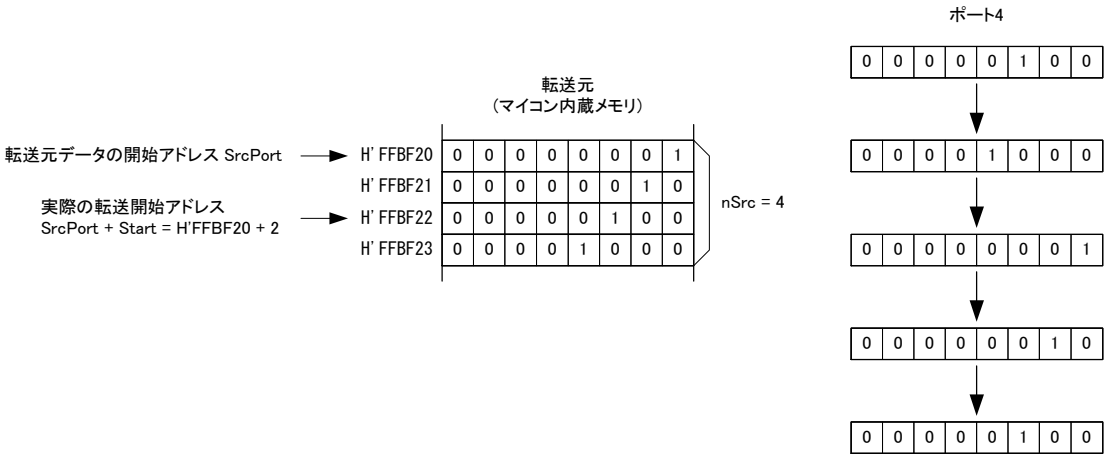


図 37　*USBM_TCPYSetPatternCtrl()* の設定例

また、*USBM_TCPYSetTrig()* 関数を使用すれば、パルスカウンタ(マイコンの割り込み)への信号入力によって、任意のタイマコピーチャンネルのスタート、及び、ストップが可能になります。この機能を利用することで、ホストパソコンを介することなく、タイマコピー機能をコントロール可能となり、リアルタイム性を要求される用途にも応用可能となります(「パルスカウンタ」参照)。

タイマコピー機能を使用する

① *USBM_TCPYSetCycle()* 関数でコピーの周期を指定します。

② *USBM_TCPYSetParm()* 関数、または *USBM_TCPYSetPatternCtrl()* 関数で転送に関するパラメータを設定します。

③ パルスカウンタを起動トリガ、終了トリガとして利用する場合は、*USBM_TCPYSetTrig()* 関数を使用します。直ぐに転送を開始する場合には *USBM_TCPYStart()* 関数で該当チャンネルをスタートさせます。

④ コピーを終了するには *USBM_TCPYStart()* 関数を、該当するビットを 0 にして呼び出します。

⑤ パルスカウンタをトリガとして使用した場合には、*USBM_PCStop()* 関数で使用したチャンネルを停止します。

- タイマコピーで1回の転送に要する時間は最大 25μsec です。ただし、タイマコピーは割り込みを利用しソフトウェアで実装されていますので、他の割り込みなどで直ちにコピーできない場合があります。そのため、複数のチャンネルを使用する場合、また他に割り込みを利用する機能を使用している場合にはコピーのタイミングがタイマの設定値どおりに行われないことがあります。また、最悪の場合には割り込みが無視され出力が更新されないことも考えられますのでご注意ください。

C 言語の例

```
BYTE Data[512];
int i;

/*DA0 に三角波を出力*/
for(i=0;i<256;i++){
  Data[i] = Data[511-i] = i & 0xff;
}
USBM_PortBWrite(hDev,USBM_USER_AREA,Data,512); /*ユーザーメモリにコピー*/
USBM_TCPYSetCycle(hDev,0,99,USBM_TCLK390); /*転送サイクルを約 3.9kHz に設定*/
USBM_TCPYSetParm(hDev,0,USBM_USER_AREA,USBM_DA0,512,1,0,TRUE); /*DA0 に繰り返し転送*/
USBM_TCPYStart(hDev,1); /*チャンネル 0 をスタート*/

/*...*/

USBM_TCPYStart(hDev,0); /*コピー終了*/
```

VisualBasic6.0 の例

```
Dim Data(511) As Byte
Dim i As Integer

'DA0 に三角波を出力
For i = 0 To 255
  Data(i) = i And &HFF
  Data(511 - i) = i And &HFF
Next i
USBM_PortBWrite hDev, USBM_USER_AREA, Data, 512 'ユーザーメモリにコピー
USBM_TCPYSetCycle hDev, 0, 99, USBM_TCLK390 '転送サイクルを約 3.9kHz に設定
USBM_TCPYSetParm hDev, 0, USBM_USER_AREA, USBM_DA0, 512, 1, 0, 1 'DA0 に繰り返し転送
USBM_TCPYStart hDev, 1 'チャンネル 0 をスタート

'...

USBM_TCPYStart hDev, 0 'コピー終了
```

□ タイムアウト設定

専用 API 関数は、そのほとんどが同期動作です。そのため、用途によってはタイムアウト時間の設定が必要になる場合があります。その場合、*USBM_SetTimeouts()* 関数を使用してください。初期状態ではデバイスへの書き込み、デバイスからの読み出し、共に 5 秒間でタイムアウトするように初期化されます。

また、*USBM_ADBRead()* 関数、*USBM_SCIRead()* 関数に関しては、マイコン内で指定数のデータが読み込まれるまで無限ループに入ります。そのため、何らかの理由で、前記関数がタイムアウトして戻った場合に、デバイス側は無限ループのまま、後の命令を受け付けなくなります。その場合、*USBM_Abort()* 関数を呼び出すことにより、読み出しループから抜け出し、通常のコマンドループに戻るよう指示する必要があります。

関数がタイムアウトした場合の復帰処理

① 関数の戻り値をチェックし、タイムアウトが発生したかどうかを調べます。

② タイムアウトした場合、*USBM_Abort()* 関数を呼び出し、マイコン内の読み出しループを中止させます。

③ *USBM_Purge()* 関数を呼び出し、リードバッファに溜まったデータ(ループを中止させる前にデバイスから送られたデータ)を破棄します。

C 言語の例

```
long nRead;
char Data[100];
TW_STATUS ret;

USBM_SetTimeouts(hDev, 2000, 1000); //リードタイムアウト 2 秒,ライトタイムアウト 1 秒

ret = USBM_SCIRead(hDev, 0, Data, 100, &nRead);
if(ret == TW_TIMEIOUT)){  //タイムアウトした場合
  USBM_Abort(hDev);
  USBM_Purge(hDev, USBM_PURGE_RX);  //リードバッファをクリア
  return;
}

/*...*/
```

VisualBasic6.0 の例

```
Dim nRead As Long
Dim Data(99) As Byte
Dim ret As Long

USBM_SetTimeouts hDev, 2000, 1000 '//リードタイムアウト２秒,ライトタイムアウト１秒

ret = USBM_SCIRead(hDev, 0, Data, 100, nRead)
If ret = TW_TIMEOUT Then 'タイムアウトした場合
  USBM_Abort hDev
  USBM_Purge hDev, USBM_PURGE_RX 'リードバッファをクリア
  Exit Sub
End If

'...
```

□ フラッシュメモリ

内蔵のフラッシュメモリの一部は、ライブラリ関数によって消去／書き込みを行うことができます。この領域を利用して、各デバイス固有の設定情報やキャリブレーション情報を保持することができます。

搭載マイコンは 512Kbyte のフラッシュメモリを内蔵していますが、このうち EB1～EB3 の 12Kbyte の領域をライブラリ関数で書き換えることができます（図 6 参照）。

表 49　書換え可能なフラッシュメモリ

ブロック	開始番地	終了番地
EB1	H'001000	H'001FFF
EB2	H'002000	H'002FFF
EB3	H'003000	H'003FFF

フラッシュメモリを書き換える

① フラッシュメモリを書き換えるためには、まず、デバイスを「フラッシュ書換えモード」で起動します。設定方法については、24 ページの「動作モード設定」を参照してください。

② *Opt* 引数に *USBM_MODE_FLASH* を指定して *USBM_OpenA()* 関数を呼び出し、デバイスに接続します。

③ フラッシュメモリを消去する場合には、*USBM_FlashEraseBlk()* 関数を呼び出します。消去はブロック単位で行われますので、*Blk* 引数には消去したいブロック番号(1～3)を指定します。消去が完了したブロックは全てのビットが"1"になります。

④ 書き込みを行う場合には、*USBM_FlashWrite()* 関数を使用します。書き込みは 128 バイト単位で行いますので、書き込みアドレスは必ず 128 バイト境界とし、書き込みバイト数は 128 の倍数を指定してください。書き込みを行うアドレスは消去済みの領域でなくてはいけません。

⑤ データを読み出す必要がある場合は、*USBM_FlashRead()* 関数を使用します。読み出すバイト数は 128 の倍数とします。

C 言語の例

```
BYTE WriteData[128]; //書き込み用バッファ。書き込む内容で初期化してください。
BYTE ReadData[128];
TW_HANDLE hDev;

//フラッシュ書換えモードのデバイスに接続
USBM_OpenA(&hDev, NULL, USBM_IF_USB | USBM_MODE_FLASH);
if(hDev == NULL){
  //接続に失敗したときの処理
}
USBM_FlashEraseBlk(hDev, 1); //EB1 を消去
USBM_FlashWrite(hDev, USBM_EB1_TOP, WriteData, 128); //EB1 の先頭へ書き込み
USBM_FlashRead(hDev, USBM_EB1_TOP, ReadData, 128);   //読み出し

USBM_Close(hDev);
```

VisualBasic6.0 の例

```
Dim WriteData(127) As Byte '書き込み用バッファ。書き込む内容で初期化してください。
Dim ReadData(127) As Byte
Dim hDev As Long

USBM_Open hDev, "", USBM_IF_USB Or USBM_MODE_FLASH
If hDev = 0 Then
  '接続に失敗したときの処理
End If

USBM_FlashEraseBlk hDev, 1 'EB1 を消去
USBM_FlashWrite hDev, USBM_EB1_TOP, WriteData, 128 'EB1 の先頭へ書き込み
USBM_FlashRead hDev, USBM_EB1_TOP, ReadData, 128   '読み出し

USBM_Close hDev
```

- フラッシュ書換えモードのデバイスと接続した場合は *USBM_Flash* で始まる関数以外は使用できません。

□ ハードウェアイベントの監視

『USBM3069-HS(L)』では、パルスカウンタのカウント値や AD 変換結果を閾値と比較し、指定の値になったときに、Windows のメッセージ機構を通じてアプリケーションに通知することができます。

表 50 ハードウェアイベントの監視に使用する端子

ピン番	信号名	説明
CN1-24	PC0#	パルス(負極性)の入力時、または、カウント数が指定条件になったときに通知します。
CN1-23	PC1#	パルス(負極性)の入力時、または、カウント数が指定条件になったときに通知します。
CN1-47	PC2#	パルス(負極性)の入力時、または、カウント数が指定条件になったときに通知します。
CN1-46	PC3#	パルス(負極性)の入力時、または、カウント数が指定条件になったときに通知します。
CN2-8	AD0	入力電圧が指定条件になったときに通知します。
CN2-7	AD1	入力電圧が指定条件になったときに通知します。
CN2-6	AD2	入力電圧が指定条件になったときに通知します。
CN3-5	AD3	入力電圧が指定条件になったときに通知します。

表 51 ハードウェアイベントの監視に使用する関数

関数名	説明
USBM_SetHwEvent()	ハードウェアイベントの監視を開始/終了します。
USBM_PCSetCnt()	カウンタの値を設定します。
USBM_PCSetControl()	パルスカウンタのカウントアップ/ダウンの方法、クリア条件を設定します。
USBM_PCSetCondBit()	パルスカウンタのアップ/ダウンを決定する条件ビットを指定します。
USBM_PCSetCmp()	カウンタを自動的にクリアする場合などの設定を行います。
USBM_PCStartA()	指定チャンネルのカウントをスタートします。
USBM_PCStop()	指定チャンネルのカウントをストップします。

ハードウェアイベントの監視を開始するには、*USBM_SetHwEvent()* 関数を呼び出します。この関数には引数として *USBM_HW_EVENT* 構造体を渡します。*USBM_HW_EVENT* 構造体の C 言語でのプロトタイプと使用する定数を図 38 に示します。

ハードウェアイベントの通知先がウィンドウの場合、*hRecvWindow* にウィンドウのハンドルを指定します。通知先がスレッドの場合には *idRecvThread* にスレッド ID を指定します。これらは必要の無い場合 0 としてください。

EventBits には監視するハードウェアイベントをビットで指定します。例えば、AD0 のアナログ入力を監視する場合、*EventBits* には *USBM_EVENT_AD0* を指定します。複数の監視を行う場合、ビットを OR で結合します。

Message にはメッセージの番号を指定します。パルスカウンタやアナログ入力について条件が成立すると、ここで設定した番号のメッセージが、ウィンドウ、または、スレッドにポストされます。その際、メッセージのパラメータとして渡される *wParam* は *EventBits* に指定したビットのうち 1 つが"1"となり、メッセージの原因となったハードウェアイベントを示します。また、*lParam* にはパルスカウンタに関するイベントの場合はそのカウント値が、アナログ入力に関するイベントの場合は AD 変換の結果がセットされます。

- アプリケーションで自由に使用できるメッセージ番号は 0x8000(*WM_APP*)～0xbfff の範囲です。

監視を終了するには *USBM_HW_EVENT* 構造体のポインタとして *NULL* を渡すか、*EventBits* の値を 0 としてください。

```
typedef struct tagHwEvent{
  HWND hRecvWindow;   //メッセージを受け取るウィンドウのハンドル
  DWORD idRecvThread; //メッセージを受け取るスレッドの ID
  LPVOID lpRsv;       //予約
  UINT Message;       //受け取るメッセージの番号(0x8000-0xbfff)
  DWORD EventBits;    //監視するイベントをビットで指定
  long PCCnt[4];      //PC0-PC3 のカウント値と比較する閾値
  long PCCmp[4];      //PC0-PC3 の比較方法/PCCnt の自動インクリメント
  long ADVal[4];      //AD0-AD3 の入力値と比較する閾値
  short ADCmp[4];     //AD0-AD3 の比較方法/ヒステリシス
} USBM_HW_EVENT;

#define USBM_EVENT_PC0  0x00000001  //パルスカウンタ 0(PC0)を監視
#define USBM_EVENT_PC1  0x00000002  //パルスカウンタ 1(PC1)を監視
#define USBM_EVENT_PC2  0x00000004  //パルスカウンタ 2(PC2)を監視
#define USBM_EVENT_PC3  0x00000008  //パルスカウンタ 3(PC3)を監視
#define USBM_EVENT_AD0  0x00000010  //アナログ入力 0(AD0)を監視
#define USBM_EVENT_AD1  0x00000020  //アナログ入力 1(AD1)を監視
#define USBM_EVENT_AD2  0x00000040  //アナログ入力 2(AD2)を監視
#define USBM_EVENT_AD3  0x00000080  //アナログ入力 3(AD3)を監視
#define USBM_EVENT_USER 0x80000000  //ユーザー定義
#define USBM_EVENT_PC   0x0000000f  //パルスカウンタ全て
#define USBM_EVENT_AD   0x000000f0  //アナログ入力全て

#define USBM_CMP_GE     0x7fffffff  //カウンタ値が≧PCCnt[]で通知
#define USBM_CMP_LE     0x80000000  //カウンタ値が≦PCCnt[]で通知
```

図 38　USBM_HW_EVENT 構造体と使用する定数

パルスカウンタ入力を監視する

パルスカウンタの入力を監視する場合、*USBM_HW_EVENT* 構造体の *PCCnt[]* と *PCCmp[]* に値を設定します。*PCCnt[0]* ～ *PCCnt[3]* には、それぞれ 0～3 チャンネルのカウンタ値と比較する閾値を設定します。*PCCmp[0]* ～ *PCCmp[3]* には閾値との比較方法、閾値の自動インクリメントを設定します。*PCCmp[]* の設定値と具体的動作を表 52 に示します。

表 52　PCCmp[]の設定値と動作の関係

PCCmp[x]の設定	ハードウェアイベントの検出条件	イベント検出後の PCCnt[x]の値
USBM_CMP_GE (0x7fffffff)	指定チャンネル(x)のカウンタ値が PCCnt[x]以上になった場合	変化なし
USBM_CMP_GE 以外の正の値	指定チャンネル(x)のカウンタ値が PCCnt[x]以上になった場合	PCCnt[x] = PCCnt[x] + PCCmp[x]
0	指定チャンネル(x)のカウンタ値が変化した場合	変化なし
USBM_CMP_LE 以外の負の値	指定チャンネル(x)のカウンタ値が PCCnt[x]以下となった場合	PCCnt[x] = PCCnt[x] + PCCmp[x]
USBM_CMP_LE (0x80000000)	指定チャンネル(x)のカウンタ値が PCCnt[x]以下となった場合	変化なし

PCCmp[] が 0 の場合、カウンタ値が変化する毎にアプリケーションに通知されます。*PCCnt[]* の値は無視されます。

PCCmp[] を正の値とすると、指定チャンネルのカウンタ値が *PCCnt[]* 以上となった場合にハードウェアイベントとして検出しアプリケーションに通知されます。負の値とすると、逆にカウンタ値が *PCCnt[]* 以下になった場合に通知されます。

また、*USBM_CMP_GE* (0x7fffffff)、*USBM_CMP_LE* (0x80000000)以外の値を指定した場合には、ハードウェアイベントとして検出された後に、*PCCmp[]* の値が *PCCnt[]* に自動的に加算されます。これによって、一定カウント毎にハードウェアイベントとしてアプリケーションに通知を行うことができます。例えば、*PCCnt[0]* の初期値が 100、*PCCmp[0]* の値が 100 の場合、0 チャンネルのカウンタが 100 になったとき、最初のメッセージが送信されます。このとき *PCCnt[0]* は *PCCmp[0]* の値が加算され 200 となります。その後、カウンタ値が 200 になると、2 番目のメッセージが送信されます。以下、同様に 100 カウント毎にメッセージが送信されます。

PCCmp[] が *USBM_CMP_GE* または *USBM_CMP_LE* の場合、一度ハードウェアイベントが発生し、メッセージが送信されると、その条件が解除されるまで再度通知されることはありません。例えば、*PCCnt[0]* が 100 で、*PCCmp[0]* が *USBM_CMP_GE* の場合、0 チャンネルのカウンタ値が 100 以上になった場合、メッセージが送られますが、以降カウンタ値が変化してもメッセージは送信されません。この場合、*USBM_PCSetCnt()* 関数の呼び出しなどで、カウンタ値が 100 未満になると、再びメッセージ送信可能な状態になります。

USBM_SetHwEvent() を呼び出しただけでは、パルスカウンタのカウント動作は開始されませんので、別途制御関数を呼び出す必要があります。

① *USBM_SetHwEvent()* 関数を使用し、ハードウェアイベントの監視を開始します。

② 必要であれば、*USBM_PCSetControl()* 関数などを使用し、パルスカウンタのカウント条件を設定します。カウント条件等の詳細はパルスカウンタ(65 ページ)を参照してください。

③ *USBM_PCStartA()* 関数を使用し、パルスカウンタの計数を開始します。

- 自動インクリメントでは *PCCnt[]* 値のオーバーフローは考慮されません。例えば *PCCnt[]* の値が正の最大値を超えた場合、負の値となってしまいますのでご注意ください。

アナログ入力を監視する

アナログ入力を監視する場合、前記の *USBM_HW_EVENT* 構造体の *ADVal[]* と *ADCmp[]* に値を設定します。*ADVal[0]* ～ *ADVal[3]* には、それぞれ 0～3 チャンネルのアナログ入力値と比較する閾値を設定します。*ADCmp[0]* ～ *ADCmp[3]* には閾値との比較方法とヒステリシスを設定します。

ADVal[] は 32 ビットの変数ですが、格納する値は図 23(45 ページ)と同様の 16 ビット値です(上位 16 ビットは常に 0 としてください)。設定したい閾値電圧を V_{TH} とすると、*ADVal[]* への設定値 C_{th} は、

以下の式で求めることができます。

$$C_{TH} \fallingdotseq (V_{TH} / VREF) \times 65536$$

ADCmp[] は0以上の場合、指定チャンネルのAD変換結果が *ADVal[]* 以上となる場合に、ハードウェアイベントとして検出しアプリケーションに通知されます。*ADCmp[]* が負の場合は、AD 変換結果が *ADVal[]* 以下となる場合に通知されます。

また、*ADCmp[]* の大きさはヒステリシスの大きさを表します。*ADCmp[]* が正の場合、対応するアナログ入力電圧が V_{TH}[V] 以上になることでハードウェアイベントが検出されますが、同じイベントが何度も通知されるのを避けるために、この時点で該当チャンネルの次のイベント検出は一旦禁止されます。この禁止状態は入力電圧が(V_{TH} 以下ではなく) V_{TH} - V_{HYST} [V] 以下となったときに解除されます(図 39)。このときの V_{HYST} をヒステリシス電圧と呼びます。ヒステリシス電圧が適切な大きさに設定されていないと、入力電圧が V_{TH} 付近のとき、ノイズなどによる微小な電圧変化でもハードウェアイベントが検出されてしまい、不要なメッセージが何度も送信される場合があります。

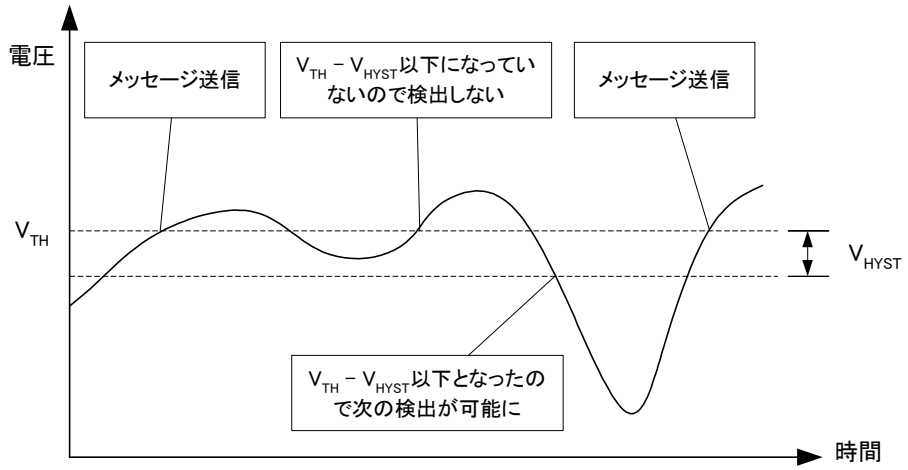


図 39　ヒステリシスが設定されている場合の動作

ADCmp[] の設定値 C_{CMP} も V_{HYST} 電圧から以下の式で求めることができます。

$$C_{CMP} \fallingdotseq (V_{HYST} / VREF) \times 65536$$

例えば、AD0 の V_{TH} を VREF/2 [V]、V_{HYST} を約 VREF/100 [V] とし、入力電圧 ≧ V_{TH} のときにメッセージを受け取るには、以下の計算から、*ADVal[0]* = 32768、*ADCmp[0]* = 655 とします。

$$C_{TH} = (VREF / 2) / VREF \times 65536 = 32768$$
$$C_{CMP} = (VREF / 100) / VREF \times 65536 = 655.36 \fallingdotseq 655$$

逆に入力電圧 ≦ V_{TH} のときにメッセージを受け取るには、*ADCmp[0]* = −655 とします。

- 0 ＜ *ADVal[]* − *ADCmp[]* ＜ 65535 となるようにしてください。

□ 複数機能の同時使用

『USBM3069-HS(L)』の機能には同時に動作可能なものがあります。これらの機能はマイコンの処理時間をシェアしながら動作するため、同時に使用することで、機能が制限されたり、性能が低下したりする場合があります。そのため、これらの機能の実装方法や処理にかかる時間などを把握しておくことが重要です。

表 53 に『USBM3069-HS(L)』の機能別の処理形態を示します。同期/非同期の欄が「同期」となっている機能は、呼び出すと命令が完了するまでデバイスが他の命令を受け付けなくなります。「非同期」となっている機能は、命令を実行すると動作は開始しますが、他の命令も受け入れられるものです。非同期の機能を複数使用することや、非同期の機能が動作中に同期の機能を呼び出すことは可能ですが、同期の機能が動作中は、他の同期機能の実行や非同期の機能の開始ができません。

表 53 各機能の処理形態

機能	処理の種類 (優先順位)	同期/ 非同期	注意事項、処理時間など
ポート、バスに対するアクセス (DMA を使用しない場合) *USBM_PortWrite8()* など	通常処理(1)	同期	−
ポート、バスに対するアクセス (DMA を使用する場合) *USBM_PortBWrite()* など	DMA 処理(3)	同期	バースト転送しますので、転送終了まで他の命令は実行されません。他の機能との併用はお薦めしません。1 バイトの転送に数 100nsec(アクセス対象による)を要します。
AD 変換 (*USBM_ADCopy()* 以外) *USBM_ADBRead()*など	通常処理(1)	同期	割込み処理と同時実行されると、指定したサイクル通りに変換が行われない可能性があります。
USBM_ADCopy() による AD 変換	DMA 処理(3)	非同期	DMA チャンネルを 1 つ使用します。1 回の変換結果を転送するのに 400nsec～1.6 μ sec(チャンネル数による)の時間を要します。
DA 変換 (DMA を使用する場合) *USBM_DAStart()*	DMA 処理(3)	非同期	16 ビットタイマと DMA チャンネルを 1 つ、または、2 つずつ使用します。1 回のデータ転送に 280nsec の時間を要します。
16 ビットタイマ (PWM 出力、インプットキャプチャ)	ハードウェア処理(−)	非同期	−
16 ビットタイマ(シングルパルス出力)	割込み処理(2)	非同期	シングルパルスではパルス出力終了時に割込み処理が行われます。処理時間は 10 μ sec です。
16 ビットタイマ(位相計数)	割込み処理(2)	非同期	位相計数のカウント自体はハードウェア処理ですが、コンペアマッチ時の処理は割込みです。処理時間は 25 μ sec です。
パルスカウンタ	割込み処理(2)	非同期	パルス入力時に割込み処理が行われます。処理時間は 30 μ sec、または 25 μ sec です(チャンネルにより変わります)。
タイマコピー	割込み処理(2)	非同期	コンペアマッチ時に割込み処理が行われます。処理時間は 25 μ sec です。
シリアル通信 *USBM_SCIWrite()* など	通常処理(1)/ 割込み処理(2)	同期/ 非同期	ポートからデータ受信時は割込み処理でバッファに格納されます。処理時間は 30 μ sec です。

割込みの処理時間は将来のバージョンで変更される場合があります。
DMA の転送所要時間は内部メモリアクセスを想定したものです。外部メモリを使用する場合は長くなります。
シリアル通信は *USBM_SCIWrite()*、*USBM_SCIRead()* などは通常処理ですが、シリアルポートで受信したデータをバッファリングする動作が割込み処理になります。

表 53 の処理の種類とは、その機能をデバイス上のマイコンで処理する際に、どのような形態で処理されるのかを表しています。表 54 にそれぞれの説明と優先順位を示します。優先順位の数字が大きいものほど優先的に処理が行われます。優先順位が高い処理と、低い処理が同時実行される場合、優先順位の高い処理が、CPU やメモリアクセスサイクルを開放しない限り、優先順位の低い処理は実行を待たされます。ハードウェア処理はマイコン内蔵の周辺回路だけで動作が完結するため、他の処理との優先順位はありません。

表 54　命令の実行方法

処理の種類	説明	優先順位
通常処理	通常の命令実行サイクルの中で処理されます。マイコンの CPU により処理されますが、優先順位が最低なので他の処理を邪魔することはありません。	1
割込み処理	割込みルーチンで処理されます。処理が終わるまで通常処理は実行できません。また、重複して割込みが発生すると、前の処理が終了するまで次の処理は行われません。割込み信号が発生してから、実際に割込みルーチンが開始されるまでは、数 μ sec の時間を要し、この時間は信号発生時の実行命令により変化します。	2
DMA 処理	メモリ間のデータコピーの際、DMA コントローラにより自動的に処理されます。DMA コントローラがメモリにアクセスしている間、マイコンの CPU は命令を実行できませんが、CPU にかかる負担は最小限です。	3
ハードウェア処理	マイコン内蔵の周辺回路だけで処理が完結するものです。マイコンの CPU は全く介在しません。メモリなどのリソースにもアクセスしませんので、他の処理と並列させても負荷になりません。	-

表 55　複数機能を同時に実行した場合の影響

	ポート、バスに対するアクセス（DMAを使用しない場合）	ポート、バスに対するアクセス（DMAを使用する場合）	AD変換(USBM_ADCopy() 以外)	USBM_ADCopy() によるAD変換	DA変換(DMAを使用する場合)	16ビットタイマ(PWM出力)	16ビットタイマ（インプットキャプチャ）	16ビットタイマ（シングルパルス出力）	16ビットタイマ(位相計数)	パルスカウンタ	タイマコピー	シリアル通信（ホストパソコンからのアクセス）	シリアル通信（受信データのバッファリング）
ポート、バスに対するアクセス（DMAを使用しない場合）	×	×	×	○	○	◎	◎	○	○	○	○	×	○
ポート、バスに対するアクセス（DMAを使用する場合）	×	×	×	◎ch	◎ch	◎	◎	◎	◎	◎	◎	×	◎
AD変換(USBM_ADCopy() 以外)	×	×	×	×	○	◎	◎	△	△	△	△	×	△
USBM_ADCopy() によるAD変換	◎	▲ch	×	×	○ch	◎	◎	◎	◎	◎	◎	◎	◎
DA変換(DMAを使用する場合)	◎	▲ch	◎	○ch	○ch	◎ch	◎ch	◎ch	◎	◎	◎	◎	◎
16ビットタイマ(PWM出力)	◎	◎	◎	◎	◎	◎ch	◎ch	◎ch	◎ch	◎	◎	◎	◎
16ビットタイマ（インプットキャプチャ）	◎	◎	◎	◎	◎	◎ch	◎ch	◎ch	◎ch	◎	◎	◎	◎
16ビットタイマ（シングルパルス出力）	◎	▲	◎	○	○	◎ch	◎ch	○ch	○	○	○	◎	○
16ビットタイマ(位相計数)	◎	▲	◎	○	○	◎ch	◎ch	△	×	△	△	◎	△
パルスカウンタ	◎	▲	◎	○	○	◎	◎	△	△	△ch	△	◎	△
タイマコピー	◎	▲	◎	○	○	◎	◎	△	△	△	△ch	◎	△
シリアル通信（ホストパソコンからのアクセス）	×	×	×	○	○	◎	◎	○	○	○	○	×	○
シリアル通信（受信データのバッファリング）	◎	△	◎	○	○	◎	◎	○	○	△	△	◎	×

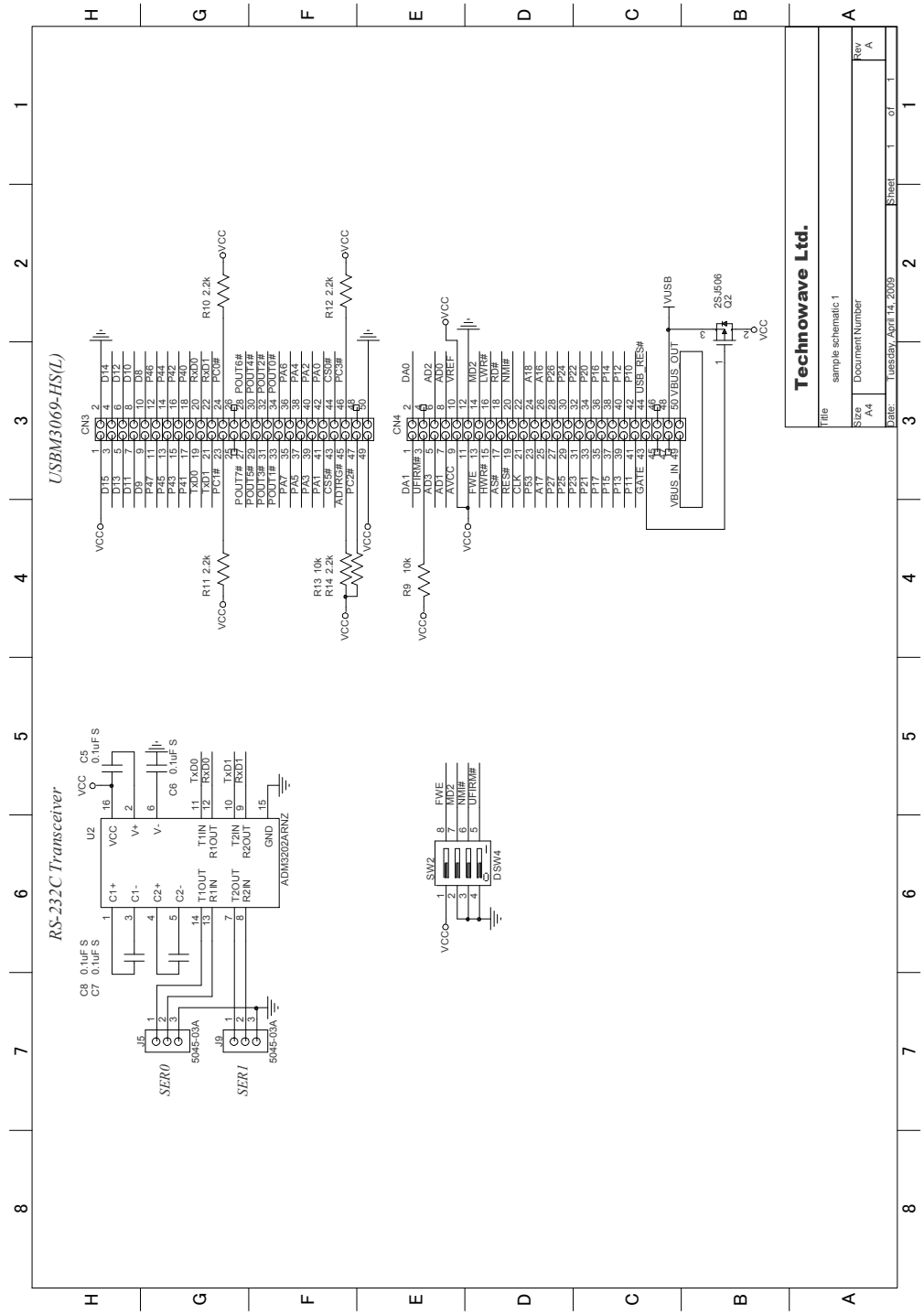
◎　同時に使用しても影響を受けない
○　同時に使用することによる影響はゼロではないが、ほとんどの場合、問題なく使用可能
△　性能は損なわれるが、用途によっては同時使用可能
▲　呼び出しは可能だが、著しく性能が損なわれる可能性がある
×　同時に呼び出せない
ch　使用チャンネルに制限を受ける

表 55 は複数の機能を同時に実行した場合の影響を示しています。ここでの実行とは機能が動作している状態を指し、動作開始までのパラメータ等のセッティングは含まれません。例えば、PWM 出力では *USBM_TimerStartA()* 関数を呼び出した後、自動的にパルスが出力されている状態が実行状態で、それ以前の *USBM_TimerSetPulse()* の呼び出しなどは、「ポート、バスに対するアクセス(DMA を使用しない場合)」にあたります。

この表の行は影響を受ける機能、列が影響を与える機能を表しています。例えば、「ポート、バスに対するアクセス(DMA を使用する場合)」(以下、「DMA アクセス」)と「パルスカウンタ」の影響を調べる場合、「DMA アクセス」の行と「パルスカウンタ」の列の交点は「◎」となっているので、「DMA アクセス」は「パルスカウンタ」には影響を受けないことがわかります。逆に「パルスカウンタ」の行と「DMA アクセス」の交点は「▲」となっているので、「パルスカウンタ」は「DMA アクセス」により、機能を著しく損なわれる可能性があることを示しています。この評価(特に「△」や「▲」)、はあくまで目安であり、絶対的な評価ではありません。例えば、先の例で言えば、5msec 間隔のパルスをカウント中に 100 バイト程度を「DMA アクセス」で転送しても、実用上ほとんど影響はありませんが、100 μ sec 間隔のパルスをカウント中に、数 10K バイトのデータを転送してしまうと、カウントに抜けが生じてしまいます。

Appendix

□ 回路例



図はバスパワーデバイスの例です。
シリアル通信を使用しない場合は、RxD0、RxD1 をそれぞれ 10kΩの抵抗でプルアップしてください。

図 40 回路例

□ データ転送速度

下表は Windows XP (32bit 版)の環境で、64K バイトのデータを連続して転送した場合の転送レート(参考値)です。

表 56　ハイスピード接続時の転送レート(参考値)

単位:MBytes/sec

USB インタフェースとの接続バス幅	ライト(パソコン→デバイス)		リード(デバイス→パソコン)	
	DMA なし	DMA 使用	DMA なし	DMA 使用
8 ビット	1.5	5.3	1.5	5.3
16 ビット	2.8	9.5	2.8	9.5

表 57　フルスピード接続時の転送レート(参考値)

単位:KBytes/sec

USB インタフェースとの接続バス幅	ライト(パソコン→デバイス)		リード(デバイス→パソコン)	
	DMA なし	DMA 使用	DMA なし	DMA 使用
8 ビット	881	881	892	892
16 ビット	834	834	961	962

□ 命令実行までのレイテンシ

USB を使用したデジタル/アナログ入出力モジュールで、しばしば問題となるのは命令実行までのレイテンシ(遅延時間)です。USB では小さなデータをランダムにアクセスするような用途では効率が悪くなるため、小さなデータを送受信するための時間は、単純に転送レートとの比で計算することはできません。下の図は *USBM_PortRead8()* 、*USBM_PortWrite8()* 関数を 1000 回ずつ呼び出し、応答時間を調べたものです。

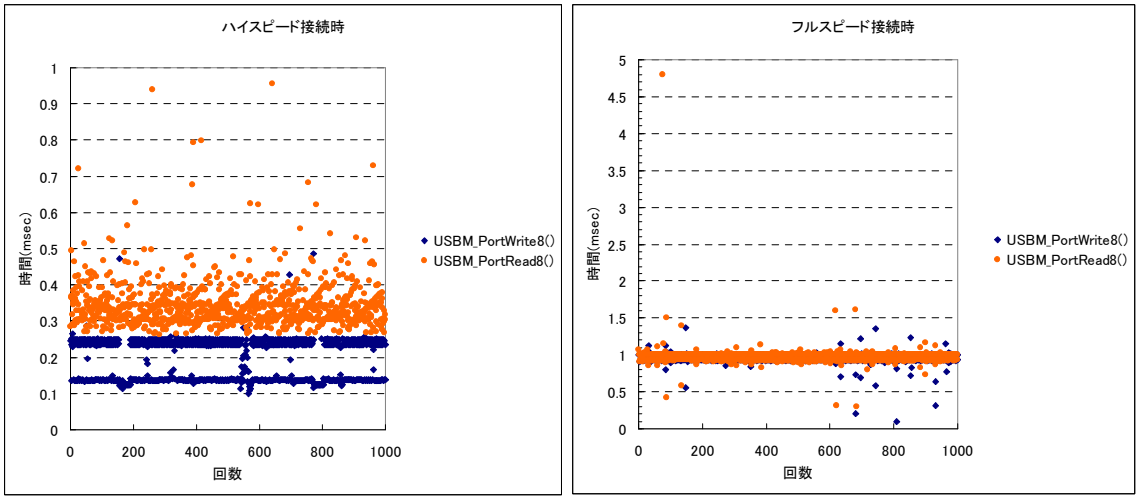


図 41　ポートアクセスの応答時間

このように USB インタフェース自体は、ホストからリアルタイムにハードウェアをコントロールするような用途には不向きです。そのため、『USBM3069-HS(L)』ではハードウェアからの入力に対する簡単なフィードバックを、ホストパソコンを介さず、デバイス上で処理できるようにパルスカウンタや、16 ビットタイマにコンペアアウトという機能を設けています(詳しくはそれぞれの節を参照してください)。

また、マイコン上で動作するユーザーファームを作成することで、より複雑な処理にも対応可能となっています。

保証期間

本製品の保証期間は、お買い上げ日より 1 年間です。保証期間中の故障につきましては、無償修理または代品との交換で対応させていただきます。ただし、以下の場合は保証期間内であっても有償での対応とさせていただきますのでご了承ください。

1. 本マニュアルに記載外の誤った使用方法による故障。
2. 火災、震災、風水害、落雷などの天災地変および公害、塩害、ガス害などによる故障。
3. お買い上げ後の輸送、落下などによる故障。

サポート情報

『USBM3069-HS(L)』に関する情報、最新のファームウェア、ユーティリティなどは弊社ホームページにてご案内しております。また、お問い合わせ、ご質問などは下記までご連絡ください。

テクノウェーブ(株)
URL : http://www.techw.co.jp
E-mail : support@techw.co.jp

(1) 本書、および本製品のホームページに掲載されている応用回路、プログラム、使用方法などは、製品の代表的動作・応用例を説明するための参考資料です。これらに起因する第三者の権利(工業所有権を含む)侵害、損害に対し、弊社はいかなる責任も負いません。

(2) 本書の内容の一部または全部を無断転載することをお断りします。

(3) 本書の内容については、将来予告なしに変更することがあります。

(4) 本書の内容については、万全を期して作成いたしましたが、万一ご不審な点や誤り、記載もれなど、お気づきの点がございましたらご連絡ください。

改訂記録

年月	版	改訂内容
2009 年 4 月	初	
2009 年 11 月	2	・Windows 7 に対応した内容に変更 ・64bit 版に対応した内容に変更
2010 年 6 月	3	・システムファーム Ver.4.2.1 以降に対応した記述に変更 ・USBM ライブラリ 3.6.0.1 以降に対応した記述に変更
2012 年 2 月	4	・対応 OS 仕様を変更 ・「USBMTools」に関する記述を変更 ・PC2#,PC3#信号の入力仕様の誤りを修正 ・AD 変換時間に関する記述を修正 ・その他

管理番号:TW-RE-MB36-4